



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ



Rezolvarea problemelor de căutare

Strategii de căutare neinformată

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Sumar

- Probleme
- Rezolvarea problemelor
 - Pași în rezolvarea problemelor
- Rezolvarea problemelor prin căutare
 - Pași în rezolvarea problemelor prin căutare
 - Tipuri de strategii de căutare

Materiale de citit și legături utile

- capitolele I.1, I.2, II.3 și II.4 din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolele 1 - 4 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- capitolele 2.1 – 2.5 din <http://www-g.eng.cam.ac.uk/mmg/teaching/artificialintelligence/>

Probleme



□ Două mari categorii de probleme:

- Rezolvabile în mod determinist
 - Calculul sinusului unui unghi sau a rădăcinii pătrate dintr-un număr
- Rezolvabile în mod stocastic
 - Probleme din lumea reală → proiectarea unui ABS
 - Presupun căutarea unei soluții → metode ale IA

Probleme

□ Tipologie

■ Probleme de căutare/optimizare

- Planificare, proiectarea sateliștilor



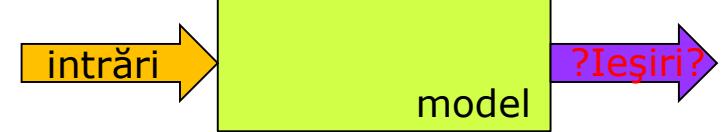
■ Probleme de modelare

- Predicții, clasificări



■ Probleme de simulare

- Teoria jocurilor economice



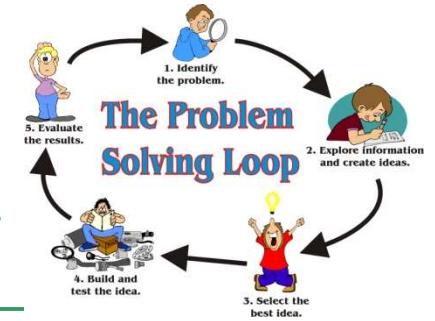
Rezolvarea problemelor



- Constă în identificarea unei soluții
 - În informatică (IA) → proces de căutare
 - În inginerie și matematică → proces de optimizare

- Cum?
 - Reprezentarea soluțiilor (partiale) → puncte în spațiul de căutare
 - Proiectarea unor operatori de căutare → transformă o posibilă soluție în altă soluție

Pași în rezolvarea problemelor



- Definirea problemei
- Analiza problemei
- Alegerea unei tehnici de rezolvare
 - căutare
 - reprezentarea cunoștințelor
 - abstractizare

Rezolvarea problemelor prin căutare



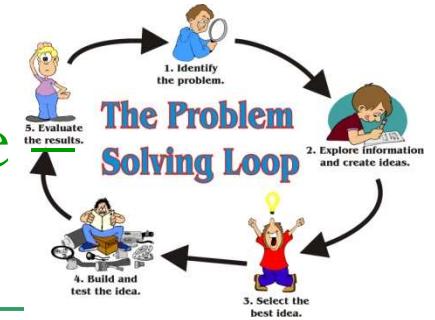
- bazată pe urmărirea unor obiective
- compusă din acțiuni care duc la îndeplinirea unor obiective
 - fiecare acțiune modifică o anumită stare a problemei
- succesiune de acțiuni care transformă starea inițială a problemei în stare finală

Pași în rezolvarea problemelor prin căutare

Definirea problemei

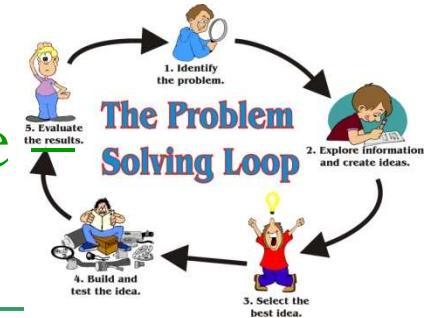
□ Definirea problemei implică stabilirea:

- unui spațiu de stări
 - toate configurațiile posibile, fără enumerarea obligatorie a tuturor configurațiilor
 - reprezentare
 - explicită – construirea (în memorie) a tuturor stărilor posibile
 - implicită – utilizarea unor structuri de date și a unor funcții (operatori)
- unei/unor stări inițiale
- unei/unor stări finale - obiectiv
- unui/unor drum(uri)
 - succesiuni de stări
- unui set de reguli (acțiuni)
 - funcții succesor (operatori) care precizează starea următoare a unei stări
 - funcții de cost care evaluatează
 - trecerea dintr-o stare în alta
 - un întreg drum
 - funcții obiectiv care verifică dacă s-a ajuns într-o stare finală



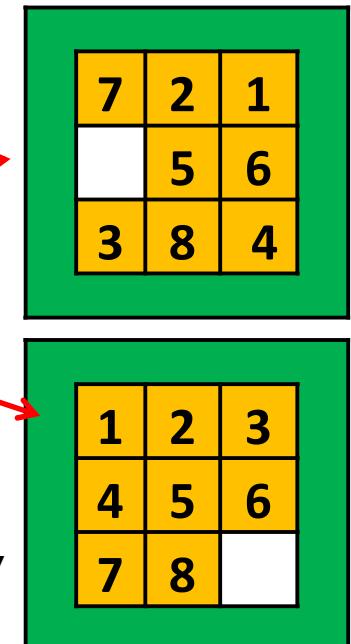
Pași în rezolvarea problemelor prin căutare

Definirea problemei



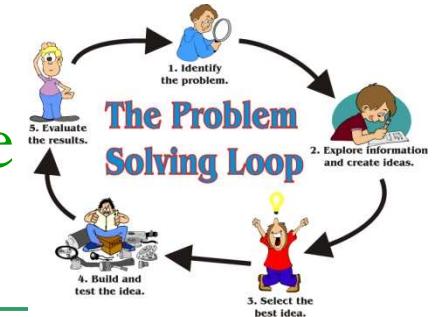
□ Exemple

- Joc puzzle cu 8 piese
 - Spațiul stărilor – configurații ale tablei de joc cu 8 piese
 - Starea inițială – o configurație oarecare
 - Starea finală – o configurație cu piesele aranjate într-o anumită ordine
 - Reguli → acțiuni albe
 - Condiții: mutarea în interiorul tablei
 - Transformări: spațiul alb se mișcă în sus, în jos, la stânga sau la dreapta
 - Soluția – secvența optimă de acțiuni albe



Pași în rezolvarea problemelor prin căutare

Definirea problemei



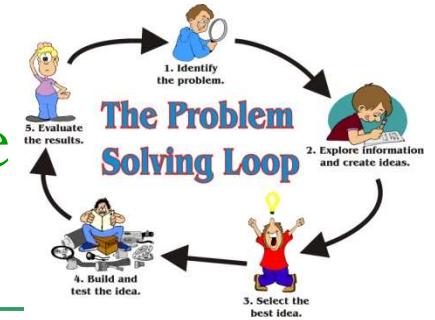
□ Exemple

- Joc cu n dame
 - Spațiul stărilor – configurații ale tablei de joc cu n regine
 - Starea inițială – o configurație fără regine
 - Starea finală – o configurație cu n regine care nu se atacă
 - Reguli → amplasarea unei regine pe tablă
 - Condiții: regina amplasată nu este atacată de nici o regină existentă pe tablă
 - Transformări: amplasarea unei noi regine într-o căsuță de pe tabla de joc
 - Soluția – amplasarea optimă a reginelor pe tablă

	a	b	c	d	e	f	g	h	
1									1
2									2
3									3
4									4
5									5
6									6
7									7
8									8
	a	b	c	d	e	f	g	h	

Pași în rezolvarea problemelor prin căutare

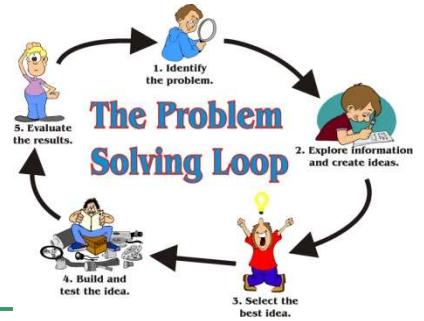
Analiza problemei



- Se poate descompune problema?
 - Sub-problemele sunt independente sau nu?
- Universul stărilor posibile este predictibil?
- Se dorește obținerea oricărei soluții sau a unei soluții optime?
- Soluția dorită constă într-o singură stare sau într-o succesiune de stări?
- Sunt necesare multiple cunoștințe pentru a limita căutarea sau chiar pentru a identifica soluția?
- Problema este conversațională sau solitară?
 - Este sau nu nevoie de interacțiune umană pentru rezolvarea ei?

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare

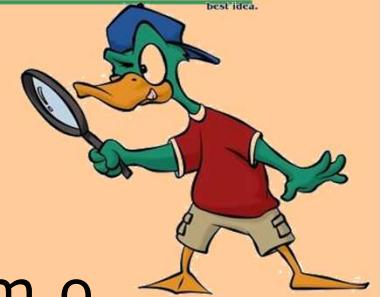
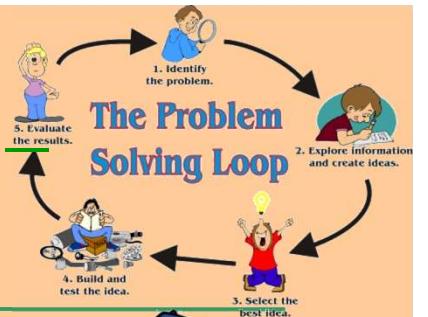


- Rezolvarea prin utilizarea regulilor (în combinație cu o strategie de control) de deplasare în spațiul problemei până la găsirea unui drum între starea inițială și cea finală
- Rezolvare prin căutare
 - Examinarea sistematică a stărilor posibile în vederea identificării
 - unui drum de la o stare inițială la o stare finală
 - unei stări optime
 - Spațiul stărilor = toate stările posibile + operatorii care definesc legăturile între stări



Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare

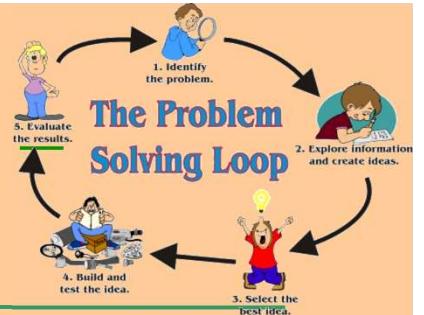


□ Rezolvare prin căutare

- Strategii de căutare multiple → cum alegem o strategie?
 - Complexitatea computațională (temporală și spațială)
 - Completitudine → algoritmul se sfârșește întotdeauna și găsește o soluție (dacă ea există)
 - Optimalitate → algoritmul găsește soluția optimă (costul optim al drumului de la starea initială la starea finală)

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare



□ Rezolvare prin căutare

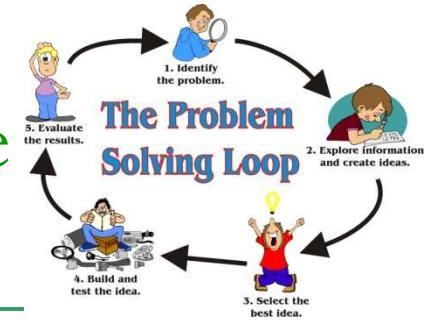
- Strategii de căutare multiple → cum alegem o strategie?

- Complexitatea computațională (temporală și spațială)
 - Performanța strategiei depinde de:
 - Timpul necesar rulării
 - Spațiul (memoria) necesară rulării
 - Mărimea intrărilor algoritmului
 - Viteza calculatorului
 - Calitatea compilatorului
 - Factori interni
 - Factori externi
- Se măsoară cu ajutorul complexității → Eficiență computațională
 - Spațială → memoria necesară identificării soluției
 - $S(n)$ – cantitatea de memorie utilizată de cel mai bun algoritm A care rezolvă o problemă de decizie f cu n date de intrare
 - Temporală → timpul necesar identificării soluției
 - $T(n)$ – timpul de rulare (numărul de pași) al celui mai bun algoritm A care rezolvă o problemă de decizie f cu n date de intrare



Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare



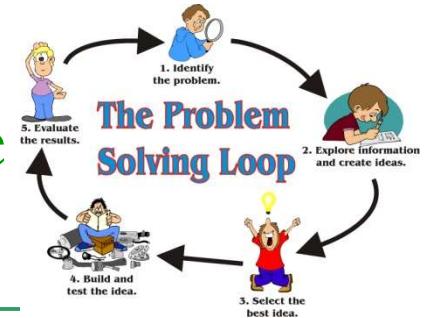
- Rezolvarea problemelor prin căutare poate consta în:

- Construirea progresivă a soluției
- Identificarea soluției potențiale optime

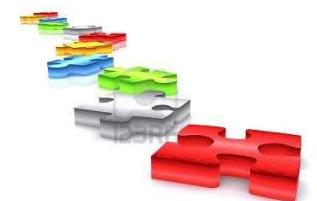


Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare



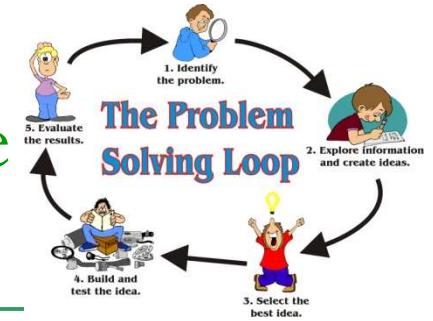
- Rezolvarea problemelor prin căutare poate consta în:
 - Construirea progresivă a soluției
 - Componentele problemei
 - Stare inițială
 - Operatori (funcții succesor)
 - Stare finală
 - Soluția = un drum (de cost optim) de la starea inițială la starea finală
 - Spațiul de căutare
 - Multimea tuturor stărilor în care se poate ajunge din starea inițială prin aplicarea operatorilor
 - stare = o componentă a soluției
 - Exemple
 - Problema comisului voiajor
 - Algoritmi
 - Ideea de bază: se începe cu o componentă a soluției și se adaugă noi componente până se ajunge la o soluție completă
 - Recursivi → se re-aplică până la îndeplinirea unei condiții
 - Istoricul căutării (drumul parcurs de la starea inițială la starea finală) este reținut în liste de tip LIFO sau FIFO
 - Avantaje
 - Nu necesită cunoștințe (informații inteligente)



Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare

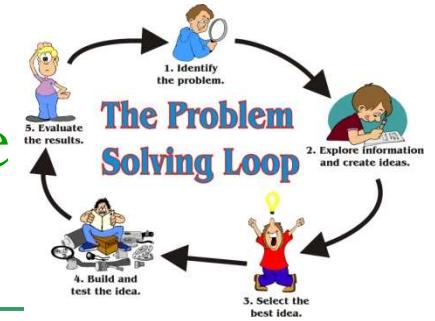
- Rezolvarea problemelor prin căutare poate consta în:
 - Identificarea soluției potențiale optime
 - Componentele problemei
 - Condiții (constrângeri) pe care trebuie să le satisfacă (parțial sau total) soluția
 - Funcție de evaluare a unei soluții potențiale → identificarea optimului
 - Spațiul de căutare
 - mulțimea tuturor soluțiilor potențiale complete
 - Stare = o soluție completă
 - Exemple
 - Problema celor 8 regine
 - Algoritmi
 - Ideea de bază: se începe cu o stare care nu respectă anumite constrângeri pentru a fi soluție optimă și se efectuează modificări pentru a elimina aceste violări
 - Iterativi → se memorează o singură stare și se încearcă îmbunătățirea ei
 - Istoricul căutării nu este reținut
 - Avantaje
 - Simplu de implementat
 - Necesară puțină memorie
 - Poate găsi soluții rezonabile în spații de căutare (continuе) foarte mari pentru care alți algoritmi sistematici nu pot fi aplicati



www.shutterstock.com - 36774760

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare



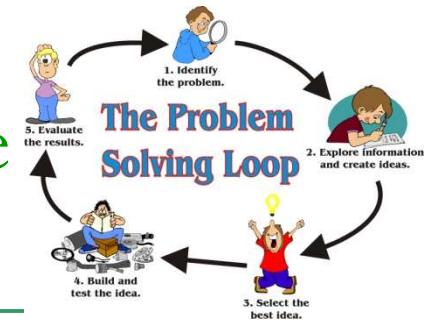
□ Rezolvarea problemelor prin căutare presupune

- algoritmi cu o complexitate ridicată (probleme NP-complete)
- căutarea într-un spațiu exponențial



Pași în rezolvarea problemelor prin căutare

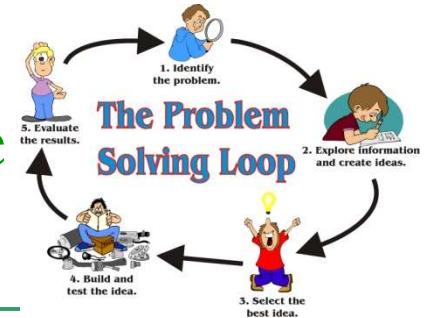
Alegerea unei tehnici de rezolvare



- Tipologia strategiilor de căutare
 - În funcție de modul de **generare** a soluției
 - Căutare **constructivă**
 - Construirea progresivă a soluției
 - Ex. TSP
 - Căutare **perturbativă**
 - O soluție candidat este modificată în vederea obținerii unei noi soluții candidat
 - Ex. SAT - Propositional Satisfiability Problem
 - În funcție de modul de **traversare** a spațiului de căutare
 - Căutare **sistemantică**
 - Traversarea completă a spațiului
 - Ideintificarea soluției dacă ea există → algoritmi compleți
 - Căutare **locală**
 - Traversarea spațiului de căutare dintr-un punct în alt punct vecin → algoritmi incompleți
 - O stare a spațiului poate fi vizitată de mai multe ori
 - În funcție de elementele de **certitudine** ale căutării
 - Căutare **deterministă**
 - Algoritmi de identificare exactă a soluției
 - Căutare **stocastică**
 - Algoritmi de aproximare a soluției
 - În funcție de stilul de **explorare** a spațiului de căutare
 - Căutare **secvențială**
 - Căutare **paralelă**

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare

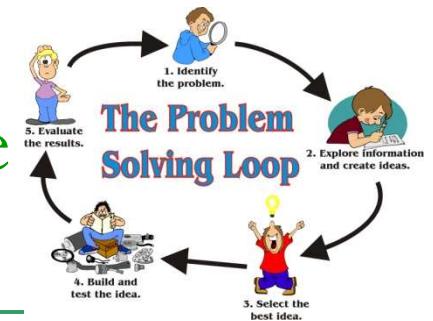


Tipologia strategiilor de căutare

- În funcție de **scopul** urmărit
 - Căutare **uni-obiectiv**
 - Soluția trebuie să satisfacă o singură condiție/constrângere
 - Căutare **multi-obiectiv**
 - Soluția trebuie să satisfacă mai multe condiții (obiective)
- În funcție de **numărul de soluții optime**
 - Căutare **uni-modală**
 - Există o singură soluție optimă
 - Căutare **multi-modală**
 - Există mai multe soluții optime
- În funcție de tipul de **algoritm** folosit
 - Căutare de-a lungul unui **număr finit de pași**
 - Căutare **iterativă**
 - Algoritmii converg către soluție
 - Căutare **euristică**
 - Algoritmii oferă o aproximare a soluției
- În funcție de **mecanismul** căutării
 - Căutare **tradițională**
 - Căutare **modernă**
- În funcție de **locul** în care se desfășoară căutarea în spațiul de căutare
 - Căutare **locală**
 - Căutare **globală**

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare



Tipologia strategiilor de căutare

□ În funcție de tipul (**linearitatea**) constrângerilor

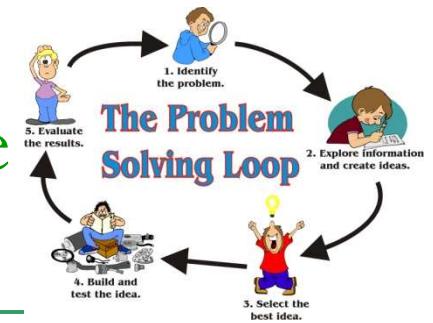
- Căutare **liniară**
- Căutare **neliniară**
 - Clasică (deterministă)
 - Directă – bazată doar pe evaluarea funcției obiectiv
 - Indirectă – bazată și pe derivata (I și/sau II) a funcției obiectiv
 - Enumerativă
 - În funcție de modul de stabilire a soluției
 - Ne-informată → soluția se știe doar când ea coincide cu starea finală
 - Informată → se lucrează cu o funcție de evaluare a unei soluții parțiale
 - În funcție de tipul spațiului de căutare
 - Completă – spațiu finit (dacă soluția există, ea poate fi găsită)
 - Incompletă – spațiu infinit
 - Stocastică
 - Bazată pe elemente aleatoare

□ În funcție de **agenții** implicați în căutare

- Căutare realizată de un **singur agent** → fără piedici în atingerea obiectivelor
- Căutare **adversarială** → adversarul aduce o incertitudine în realizarea obiectivelor

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare

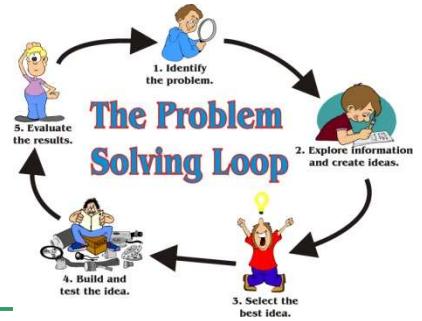


Exemplu

- Tipologia strategiilor de căutare
 - În funcție de modul de generare a soluției
 - **Căutare constructivă**
 - Căutare perturbativă
 - În funcție de modul de traversare a spațiului de căutare
 - **Căutare sistematică**
 - Căutare locală
 - În funcție de elementele de certitudine ale căutării
 - **Căutare deterministă**
 - Căutare stocastică
 - În funcție de stilul de explorare a spațiului de căutare
 - **Căutare secvențială**
 - Căutare paralelă

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare



Exemplu

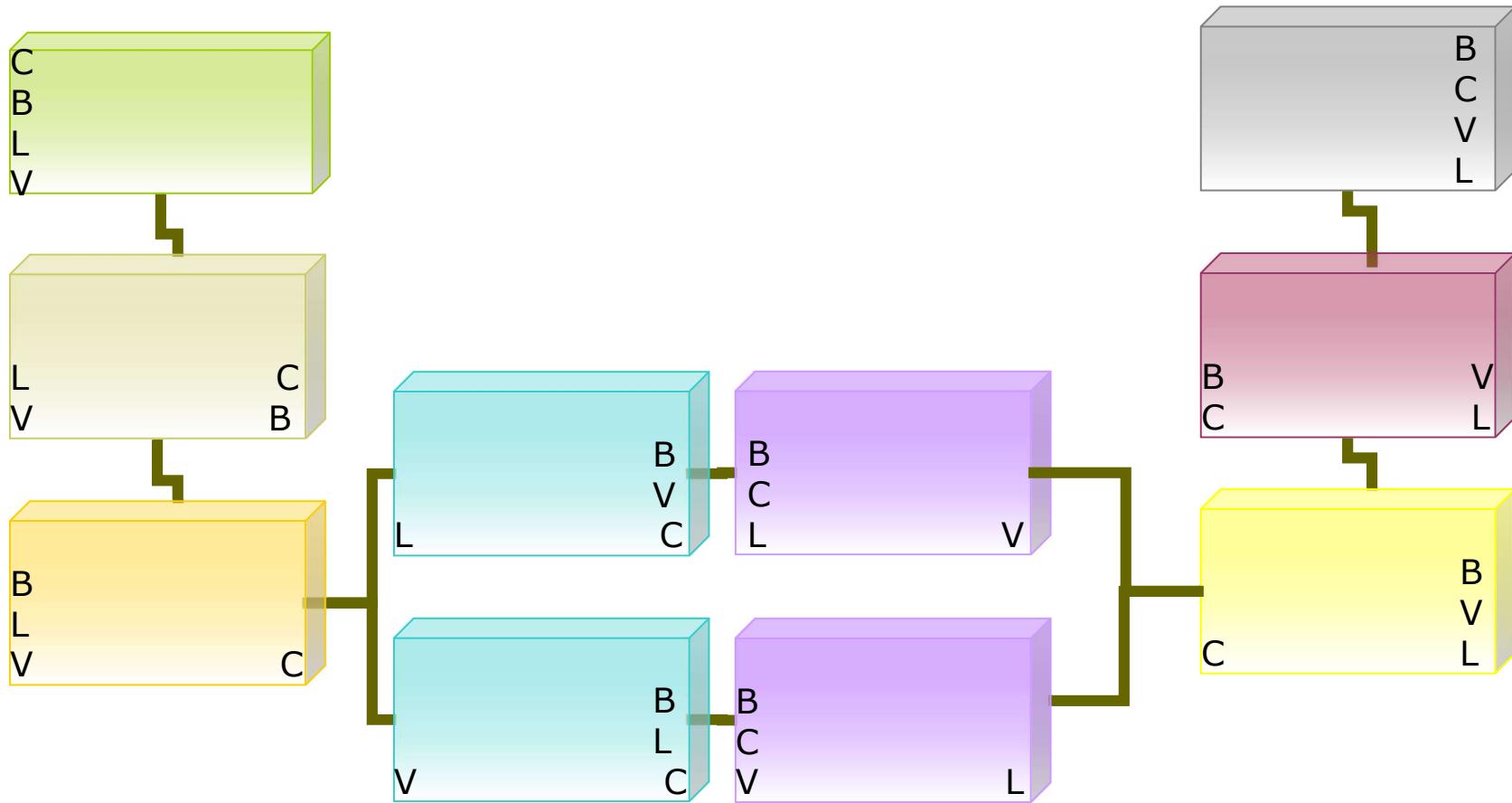
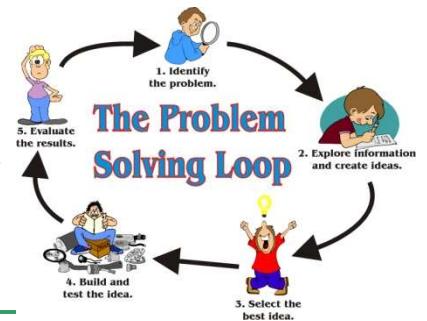
□ Problema "capra, varza și lupul"

- Se dau:
 - o capră, o varză și un lup pe malul unui râu
 - o barcă cu barcagiu

- Se cere
 - Să se traverseze toți pasagerii pe malul celălalt al râului
 - cu următoarele condiții
 - în barcă există doar 2 locuri
 - nu pot rămâne pe același mal
 - capra și varza
 - lupul și capra

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare



Strategii de căutare – Elemente fundamentale



- Tipuri abstracte de date (TAD)
 - TAD listă → structură liniară
 - TAD arbore → structură arborescentă (ierarhică)
 - TAD graf → structură de graf

- TAD
 - Domeniu și operații
 - Reprezentare

Strategii de căutare – Elemente fundamentale – TAD Listă



□ Domeniu

- $D = \{l \mid l = (el_1, el_2, \dots), \text{ unde } el_i, i=1,2,3\dots, \text{ sunt de același tip } TE \text{ (tip element) și fiecare element } el_i, i=1,2,3\dots, \text{ are o poziție unică în } l \text{ de tip } TP \text{ (TipPoziție)}\}$

□ Operații

- | | |
|---|--|
| <ul style="list-style-type: none">• Creare(l)• Prim(l)• Ultim(l)• Următor(l,p)• Anterior(l,p)• Valid(l,p)• getElement(l,p)• getPoziție(l,e)• Modifică(l,p,e)• AdăugareLaÎnceput(l,e) | <ul style="list-style-type: none">• AdăugareLaSfârșit(l,e)• AdăugareDupă(l,p,e)• AdăugareÎnainte(l,p,e)• Eliminare(l,p)• Căutare(l,e)• Vidă(l)• Dimensiune(l)• Distrugere(l)• getIterator(l) |
|---|--|

□ Reprezentare

- Vectorială
- Liste (simplu sau dublu) înlănțuite, etc

□ Cazuri particulare

- Stivă – LIFO
- Coadă – FIFO
- Coadă cu priorități



Strategii de căutare – Elemente fundamentale – TAD Listă



□ Domeniu

- $D = \{l \mid l = (el_1, el_2, \dots), \text{ unde } el_i, i=1,2,3\dots, \text{ sunt de același tip } TE \text{ (tip element) și fiecare element } el_i, i=1,2,3\dots, \text{ are o poziție unică în } l \text{ de tip } TP \text{ (TipPoziție)}\}$

□ Operații

- | | |
|---|--|
| <ul style="list-style-type: none">• Creare(l)• Prim(l)• Ultim(l)• Următor(l,p)• Anterior(l,p)• Valid(l,p)• getElement(l,p)• getPoziție(l,e)• Modifică(l,p,e)• AdăugareLaÎnceput(l,e) | <ul style="list-style-type: none">• AdăugareLaSfârșit(l,e)• AdăugareDupă(l,p,e)• AdăugareÎnainte(l,p,e)• Eliminare(l,p)• Căutare(l,e)• Vidă(l)• Dimensiune(l)• Distrugere(l)• getIterator(l) |
|---|--|

□ Reprezentare

- Vectorială
- Liste (simplu sau dublu) înlănțuite, etc

□ Cazuri particulare

- Stivă – LIFO
- Coadă – FIFO
- Coadă cu priorități



Strategii de căutare – Elemente fundamentale – TAD Listă



□ Domeniu

- $D = \{l \mid l = (el_1, el_2, \dots), \text{ unde } el_i, i=1,2,3\dots, \text{ sunt de același tip } TE \text{ (tip element) și fiecare element } el_i, i=1,2,3\dots, \text{ are o poziție unică în } l \text{ de tip } TP \text{ (TipPoziție)}\}$

□ Operații

- Creare(l)
- Prim(l)
- Ultim(l)
- Următor(l,p)
- Anterior(l,p)
- Valid(l,p)
- getElement(l,p)
- getPoziție(l,e)
- Modifică(l,p,e)
- AdăugareLaÎnceput(l,e)
- AdăugareLaSfârșit(l,e)
- AdăugareDupă(l,p,e)
- AdăugareÎnainte(l,p,e)
- Eliminare(l,p)
- Căutare(l,e)
- Vidă(l)
- Dimensiune(l)
- Distrugere(l)
- getIterator(l)

□ Reprezentare

- Vectorială
- Liste (simplu sau dublu) înlănțuite, etc

□ Cazuri particulare

- Stivă – LIFO
- Coadă – FIFO
- Coadă cu priorități



Strategii de căutare – Elemente fundamentale – TAD Graf



- Domeniu – container de noduri și legături între noduri
 - $D = \{nod_1, nod_2, \dots, nod_n, leg_1, leg_2, \dots, leg_m\}$, unde nod_i , cu $i=1, 2, \dots, n$ sunt noduri, iar leg_i , cu $i=1, 2, \dots, m$ sunt muchii între noduri}
- Operații
 - creare
 - creareNod
 - traversare
 - getIterator
 - distrugere
- Reprezentare
 - Lista muchilor
 - Lista de adiacență (Tradițională și Modernă)
 - Matricea de adiacență (Tradițională și Modernă)
 - Matricea de incidentă
- Cazuri particulare
 - Grafuri orientate și neorientate
 - Grafuri simple sau multiple
 - Grafuri conexe sau nu
 - Grafuri complete sau nu
 - Grafuri cu sau fără cicluri (aciclice \rightarrow păduri, arbori)

Strategii de căutare – Elemente fundamentale – TAD Arboare



- Domeniu – container de noduri si legături între noduri
 - $D = \{nod_1, nod_2, \dots, nod_n, leg_1, leg_2, \dots, leg_m\}$, unde nod_i , cu $i=1,2,\dots,n$ sunt noduri, iar leg_i , cu $i=1,2,\dots,m$ sunt muchii între noduri astfel încât să nu se formeze cicluri}
- Operații
 - creare
 - creareFrunză
 - adăugareSubarbore
 - getInfoRădăcină
 - getSubarbore
 - traversare
 - getIterator
 - distrugere
- Reprezentare
 - Vectorială
 - Liste înlántuite ale descendentilor
- Cazuri particulare
 - Arbori binari (de căutare)
 - Arbori n-ari

Strategii de căutare – Elemente fundamentale – parcurgerea grafelor



- Drumuri
 - drum (*path*)
 - nodurile nu se pot repeta
 - *trail*
 - muchiile nu se pot repeta
 - *walk*
 - fără restricții
 - drum închis
 - nodul inițial = nodul final
 - circuit
 - un *trail* închis
 - ciclu
 - un *path* închis

Strategii de căutare neinformate (SCnI)



- Caracteristici
 - nu se bazează pe informații specifice problemei
 - sunt generale
 - strategii oarbe
 - strategii de tipul forței brute
- Topologie
 - În funcție de ordinea expandării stărilor în spațiul de căutare:
 - SCnI în structuri liniare
 - căutare liniară
 - căutare binară
 - SCnI în structuri ne-liniare
 - căutare în lățime (breadth-first)
 - căutare de cost uniform (branch and bound)
 - căutare în adâncime (depth-first)
 - căutare în adâncime limitată (limited depth-first)
 - căutare în adâncime iterativă (iterative deepening depth-first)
 - căutare bidirecțională

SCnI în structuri liniare

Căutare liniară



□ Aspecte teoretice

- Se verifică fiecare element al unei liste până la identificarea celui dorit
- Lista de elemente poate fi ordonată sau nu

□ Exemplu

- Lista = (2, 3, 1, ,7, 5)
- Elemt = 7

□ Algoritm

```
bool LS(elem, list){  
    found = false;  
    i = 1;  
    while ((!found) && (i <= list.length)){  
        if (elem == list[i])  
            found = true;  
        else  
            i++;  
    } //while  
    return found;  
}
```

SCnI în structuri liniare

Căutare liniară



□ Analiza căutării

- Complexitate temporală
 - Cel mai bun caz: $elem = list[1] \Rightarrow O(1)$
 - Cel mai slab caz: $elem \notin list \Rightarrow T(n) = n + 1 \Rightarrow O(n)$
 - Cazul mediu: $T(n) = (1 + 2 + \dots + n + (n+1))/(n+1) \Rightarrow O(n)$
- Complexitate spațială
 - $S(n) = n$
- Completitudine
 - da
- Optimalitate
 - da

□ Avantaje

- Simplitate, complexitate temporală bună pentru structuri mici
- Structura nu trebuie sortată în prealabil

□ Dezavantaje

- complexitate temporală foarte mare pentru structuri mari

□ Aplicații

- Căutări în baze de date reale

SCnI în structuri liniare

Căutare binară



□ Aspecte teoretice

- Localizarea unui element într-o listă ordonată
- Strategie de tipul *Divide et Conquer*

□ Exemplu

- List = (2, 3, 5, 6, 8, 9, 13, 16, 18), Elem = 6
 - List = (2, 3, 5, 6, 8, 9, 13, 16, 18)
 - List = (2, 3, 5, 6)
 - List = (5, 6)
 - List = (6)

□ Algoritm

```
bool BS(elem, list){  
    found = false;  
    left = 1;  
    right = list.length;  
    while((left < right) && (!found)){  
        middle = left + (right - left)/2;  
        if (element == list[middle])  
            found = true;  
        else  
            if (element < list[middle])  
                right = middle - 1;  
            else  
                left = middle + 1;  
    } //while  
    return found;  
}
```

SCnI în structuri liniare

Căutare binară



□ Analiza căutării

- Complexitate temporală $T(n) = 1$, pt $n = 1$ și $T(n) = T(n/2) + 1$, altfel
Pp. că $n = 2k \Rightarrow k = \log_2 n$
Pp. că $2k < n < 2k+1 \Rightarrow k < \log_2 n < k + 1$
$$\begin{aligned} T(n) &= T(n/2) + 1 \\ T(n/2) &= T(n/2^2) + 1 \\ &\quad \dots \\ T(n/2^{k-1}) &= T(n/2^k) + 1 \\ \hline T(n) &= k + 1 = \log_2 n + 1 \end{aligned}$$
- Complexitate spațială – $S(n) = n$
- Completitudine – da
- Optimalitate – da

□ Avantaje

- Complexitate temporală redusă față de căutarea liniară

□ Dezavantaje

- Lucrul cu vectori (trebuie accesate elemente indexate) sortați

□ Aplicații

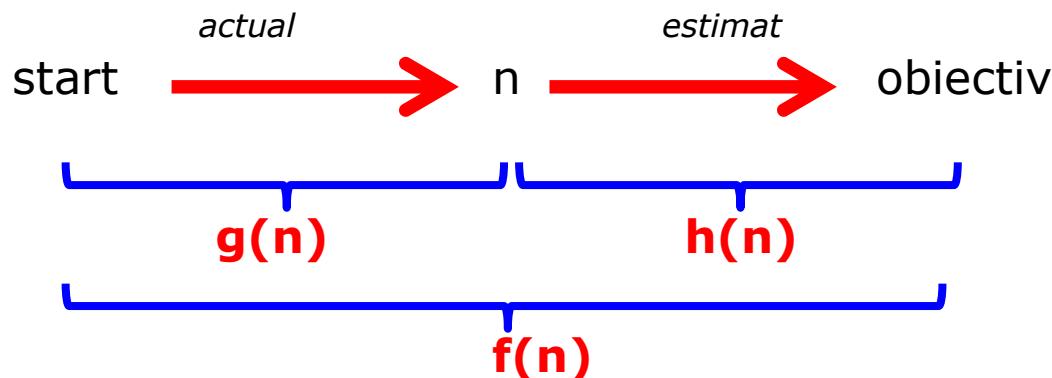
- Jocul ghicirii unui număr
- Căutare într-o carte de telefon/dicționar

SC în structuri arborescente



❑ Notiuni necesare

- $f(n)$ – funcție de evaluare pentru estimarea costului soluției prin nodul (starea) n
- $h(n)$ – funcție euristică pentru estimarea costului drumului de la nodul n la nodul obiectiv
- $g(n)$ – funcție de cost pentru estimarea costului drumului de la nodul de start până la nodul n
- $f(n) = g(n) + h(n)$



SCnI în structuri arborescente căutare în lățime (breadth-first search – BFS)



□ Aspecte teoretice

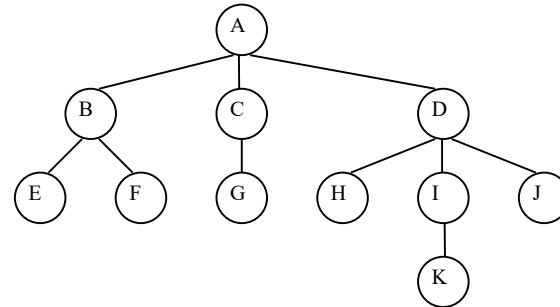
- Toate nodurile aflate la adâncimea d se expandează înaintea nodurilor aflate la adâncimea $d+1$
- Toate nodurile fii obținute prin expandarea nodului curent se adaugă într-o *listă* de tip *FIFO* (*coadă*)

□ Exemplu

- Ordinea vizitării: A, B, C, D, E, F, G, H, I, J, K

□ Algoritm

```
bool BFS(elem, list){  
    found = false;  
    visited = ∅;  
    toVisit = {start};           //FIFO list  
    while((toVisit != ∅) && (!found)){  
        node = pop(toVisit);  
        visited = visited U {node};  
        if (node == elem)  
            found = true;  
        else{  
            aux = ∅;  
            for all (unvisited) children of node do  
                aux = aux U {child};  
            }  
        toVisit = toVisit U aux;  
    } //while  
    return found;  
}
```



Vizitate deja	De vizitat
∅	A
A	B, C, D
A, B	C, D, E, F
A, B, C	D, E, F, G
A, B, C, D	E, F, G, H, I, J
A, B, C, D, E	F, G, H, I, J
A, B, C, D, E, F	G, H, I, J
A, B, C, D, E, F, G	H, I, J
A, B, C, D, E, F, G, H	I, J
A, B, C, D, E, F, G, H, I	J, K
A, B, C, D, E, F, G, H, I, J	K
A, B, C, D, E, F, G, H, I, J, K	∅

SCnI în structuri arborescente căutare în lățime (breadth-first search – BFS)



□ Analiza căutării:

- Complexitate temporală:
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d - lungimea (adâncimea) soluției
 - $T(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
- Complexitate spațială
 - $S(n) = T(n)$
- Completitudine
 - Dacă soluția există, atunci BFS o găsește
- Optimalitate
 - nu

□ Avantaje

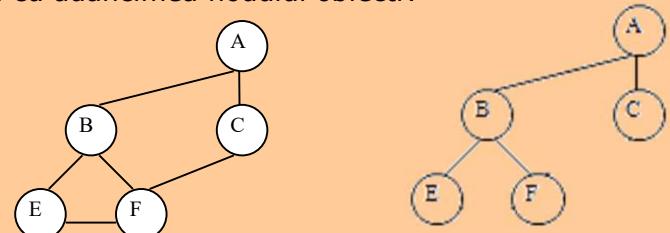
- Găsirea drumului de **lungime** minimă până la nodul obiectiv (soluția cea mai puțin adâncă)

□ Dezavantaje

- Generarea și stocarea unui arbore a cărui mărime crește exponențial cu adâncimea nodului obiectiv
- Complexitate temporală și spațială exponențială
- [Experimentul Russel&Norvig????](#)
- Funcțional doar pentru spații de căutare mici

□ Aplicații

- Identificarea tuturor componentelor conexe într-un graf
- Identificarea celui mai scurt drum într-un graf
- Optimizări în rețele de transport → algoritmul Ford-Fulkerson
- Serializarea/deserializarea unui arbore binar (vs. serializarea în mod sortat) permite reconstrucția eficientă a arborelui
- Copierea colecțiilor (garbage collection) → algoritmul Cheney



Vizitate deja	De vizitat
Φ	B
B	A, E, F
B, A	E, F, C
B, A, E	F, C
B, A, E, F	C
B, A, E, F, C	Φ

SCnI în structuri arborescente căutare de cost uniform (uniform cost search – UCS)



Aspecte teoretice

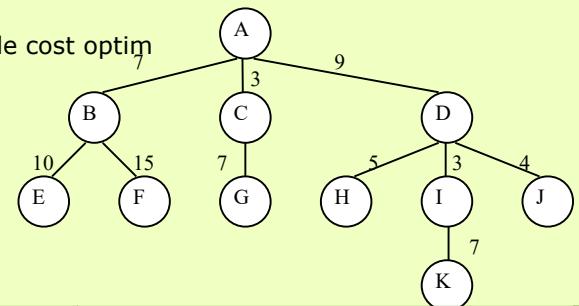
- BFS + procedură specială de expandare a nodurilor (bazată pe costurile asociate legăturilor dintre noduri)
- Toate nodurile de la adâncimea d sunt expandate înaintea nodurilor de la adâncimea $d+1$
- Toate nodurile fii obținute prin expandarea nodului curent se adaugă într-o **listă ORDONATĂ** de tip **FIFO**
 - Se expandează mai întâi nodurile de cost minim
 - Odată obținut un drum până la nodul țintă, acesta devine candidat la drumul de cost optim
- Algoritmul *Branch and bound*

Exemplu

- Ordinea vizitării nodurilor: A, C, B, D, G, E, F, I, H, J, K

Algoritm

```
bool UCS(elem, list){
    found = false;
    visited = {};
    toVisit = {start}; //FIFO sorted list
    while((toVisit != {}) && (!found)) {
        node = pop(toVisit);
        visited = visited U {node};
        if (node== elem)
            found = true;
        else
            aux = {};
            for all (unvisited) children of node do{
                aux = aux U {child};
            } // for
            toVisit = toVisit U aux;
            TotalCostSort(toVisit);
    } //while
    return found;
}
```



visited	toVisit
\emptyset	A
A	C(3), B(7), D(9)
A, C	B(7), D(9), G(3+7)
A, C, B	D(9), G(10), E(7+10), F(7+15)
A, C, B, D	G(10), I(9+3), J(9+4), H(9+5), E(17), F(22)
A, C, B, D, G	I(12), J(13), H(14), E(17), F(22)
A, C, B, D, G, I	J(13), H(14), E(17), F(22), K(9+3+7)
A, C, B, D, G, I, J	H(14), E(17), F(22), K(19)
A, C, B, D, G, I, J, H	E(17), F(22), K(19)
A, C, B, D, G, I, J, H, E	F(22), K(19)
A, C, B, D, G, I, J, H, E, F	K(19)
A, C, B, D, G, I, J, H, E, F, K	\emptyset

SCnI în structuri arborescente

căutare de cost uniform (uniform cost search – UCS)

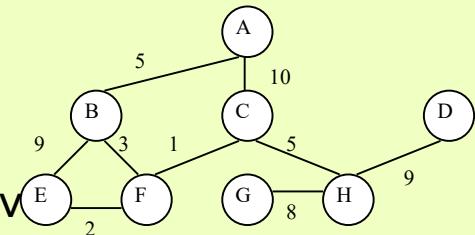


□ Analiza complexității

- Complexitate temporală:
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d - lungimea (adâncimea) soluției
 - $T(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
- Complexitate spațială
 - $S(n) = T(n)$
- Completitudine
 - Da – dacă soluția există, atunci UCS o găsește
- Optimalitate
 - Da

□ Avantaje

- Găsirea drumului de **cost** minim până la nodul obiectiv



□ Dezavantaje

- Complexitate temporală și spațială exponențială

□ Aplicații

- Cel mai scurt drum → algoritmul Dijkstra

Vizitate deja	De vizitat
∅	A(0)
A(0)	B(5), C(10)
A(0), B(5)	F(8), C(10), E(14)
A(0), B(5), F(8)	C(9), E(10)
A(0), B(5), F(8), C(9)	E(10), H(14)
A(0), B(5), F(8), C(9), E(10)	H(14)

SCnI în structuri arborescente căutare în adâncime (depth-first search – DFS)



□ Aspecte teoretice

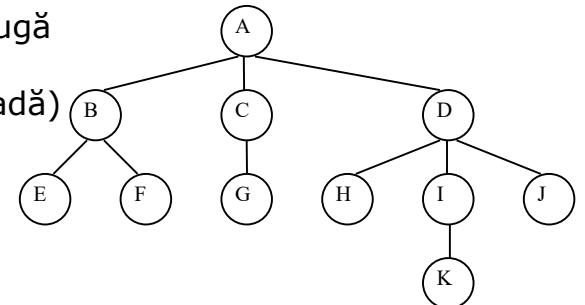
- Expandarea intr-un nod fiu și înaintarea în adâncime până când
 - Este găsit un nod ţintă (obiectiv) sau
 - Nodul atins nu mai are fii
- Cu revenirea în cel mai recent nod care mai poate fi explorat
- Toate nodurile fiu obținute prin expandarea nodului curent se adaugă într-o listă de tip *LIFO (stivă)*
- Similar cu BFS, dar nodurile se plasează într-o stivă (în loc de coadă)

□ Exemplu

- Ordinea vizitării nodurilor: A, B, E, F, C, G, D, H, I, K, J

□ Algoritm

```
bool DFS(elem, list){  
    found = false;  
    visited = ∅;  
    toVisit = {start};      //LIFO list  
    while((toVisit != ∅) && (!found)){  
        node = pop(toVisit);  
        visited = visited U {node};  
        if (node== elem)  
            found = true;  
        else{  
            aux = ∅;  
            for all (unvisited) children of node do{  
                aux = aux U {child};  
            }  
            toVisit = aux U toVisit;  
        }  
    } //while  
    return found;  
}
```



Vizitate deja	De vizitat
∅	A
A	B, C, D
A, B	E, F, C, D
A, B, E	F, C, D
A, B, E, F	C, D
A, B, E, F, C	G, D
A, B, E, F, C, G	D
A, B, E, F, C, G, D	H, I, J
A, B, E, F, C, G, D, H	I, J
A, B, E, F, C, G, D, H, I	K, J
A, B, E, F, C, G, D, H, I, K	J
A, B, E, F, C, G, D, H, I, K, J	∅

SCnI în structuri arborescente căutare în adâncime (depth-first search – DFS)



□ Analiza complexității

- Complexitate temporală:
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d_{max} – lungimea (adâncimea) maximă a unui arbore explorat
 - $T(n) = 1 + b + b^2 + \dots + b^{d_{max}} \Rightarrow O(b^{d_{max}})$
- Complexitate spațială
 - $S(n) = b * d_{max}$
- Completitudine
 - Nu → algoritmul nu se termină pt drumurile infinite (neexistând suficientă memorie pt reținerea nodurilor deja vizitate)
- Optimalitate
 - Nu → căutarea în adâncime poate găsi un drum soluție mai lung decât drumul optim

□ Avantaje

- Găsirea drumului de lungime minimă până la nodul obiectiv cu consum minim de memorie
 - versiunea recursivă

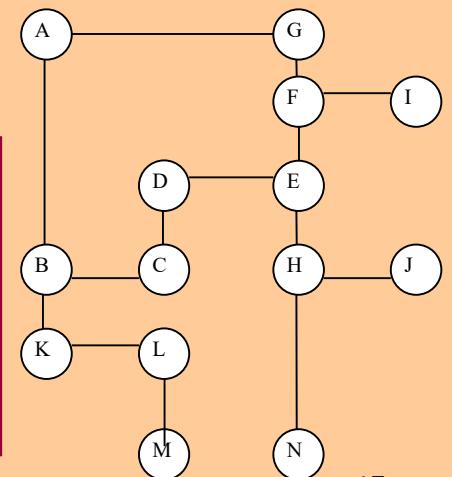
□ Dezavantaje

- Se poate bloca pe anumite drumuri greșite (nenorocoase) fără a putea reveni
 - Ciclu infinit
 - Găsirea unei soluții mai "lungi" decât soluția optimă

□ Aplicații

- Problema labirintului (maze)
- Identificarea componentelor conexe
- Sortare topologică
- Testarea planarității

A				G		
				F	I	
		D		E		
B		C		H	J	
K		L		M	N	



SCnI în structuri arborescente căutare în adâncime (depth-first search – DFS)



```

bool DFS_edges(elem, list) {
    discovered = Φ;
    back = Φ;
    toDiscover = Φ;      //LIFO
    for (all neighbours of start) do
        toDiscover = toDiscover U {(start, neighbour)}
    found = false;
    visited = {start};
    while((toDiscover != Φ) && (!found)) {
        edge = pop(toDiscover);
        if (edge.out !ε visited) {
            discovered = discovered U {edge};
            visited = visited U {edge.out}
            if (edge.out == end)
                found = true;
            else{
                aux = Φ;
                for all neighbours of edge.out do{
                    aux = aux U {(edge.out, neighbour)};
                }
                toDiscover = aux U toDiscover;
            }
        else
            back = back U {edge}
    } //while
    return found;
}

```

Muchia	Muchii vizitate deja	Muchii de vizitat	înapoi	Noduri vizitate
	Φ	AB, AF	Φ	A
AB	AB	BC, BK, AF	Φ	A, B
BC	AB, BC	CD, BK, AF	Φ	A, B, C
CD	AB, BC, CD	DE, BK, AF	Φ	A, B, C, D
DE	AB, BC, CD, DE	EF, EH, BK, AF	Φ	A, B, C, D, E
EF	AB, BC, CD, DE, EF	FI, FG, EH, BK, AF	Φ	A, B, C, D, E, F
FI	AB, BC, CD, DE, EF, FI	FG, EH, BK, AF	Φ	A, B, C, D, E, F, I
FG	AB, BC, CD, DE, EF, FI, FG	GA, EH, BK, AF	Φ	A, B, C, D, E, F, I, G
GA	AB, BC, CD, DE, EF, FI, FG	EH, BK, AF	GA	A, B, C, D, E, F, I, G
EH	AB, BC, CD, DE, EF, FI, FG	HJ, HN, BK, AF	GA	A, B, C, D, E, F, I, G, H
HJ	AB, BC, CD, DE, EF, FI, FG, HJ	HN, BK, AF	GA	A, B, C, D, E, F, I, G, H, J
HN	AB, BC, CD, DE, EF, FI, FG, HI, HN	BK, AF	GA	A, B, C, D, E, F, I, G, H, N

SCnI în structuri arborescente căutare în adâncime limitată (depth-limited search – DLS)



Aspecte teoretice

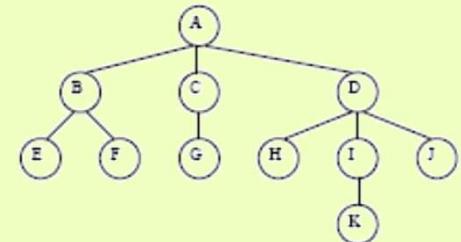
- DFS + adâncime maximă care limitează căutarea (d_{lim})
- Se soluționează problemele de completitudine ale căutării în adâncime (DFS)

Exemplu

- $d_{lim} = 2$
- Ordinea vizitării nodurilor: A, B, E, F, C, G, D, H, I, J

Algoritm

```
bool DLS(elem, list, dlim) {
    found = false;
    visited = {};
    toVisit = {start}; //LIFO list
    while((toVisit != {}) && (!found)) {
        node = pop(toVisit);
        visited = visited U {node};
        if (node.depth <= dlim) {
            if (node == elem)
                found = true;
            else{
                aux = {};
                for all (unvisited) children of node do{
                    aux = aux U {child};
                }
                toVisit = aux U toVisit;
            } //if found
        } //if dlim
    } //while
    return found;
}
```



Vizitate deja	De vizitat
{}	A
A	B, C, D
A, B	E, F, C, D
A, B, E	F, C, D
A, B, E, F	C, D
A, B, E, F, C	G, D
A, B, E, F, C, G	D
A, B, E, F, C, G, D	H, I, J
A, B, E, F, C, G, D, H	I, J
A, B, E, F, C, G, D, H, I	J
A, B, E, F, C, G, D, H, I, K, J	{}

SCnI în structuri arborescente căutare în adâncime limitată (depth-limited search – DLS)



■ Analiza complexității

- Complexitate temporală:
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d_{lim} – limita lungimii (adâncimii) permisă pentru un arbore explorat
 - $T(n) = 1 + b + b^2 + \dots + b^{d_{lim}} \Rightarrow O(b^{d_{lim}})$
- Complexitate spațială
 - $S(n) = b * d_{lim}$
- Completitudine
 - Da, dar $\Leftrightarrow d_{lim} > d$, unde d = lungimea (adâncimea) soluției optime
- Optimalitate
 - Nu \rightarrow căutarea în adâncime poate găsi un drum soluție mai lung decât drumul optim

■ Avantaje

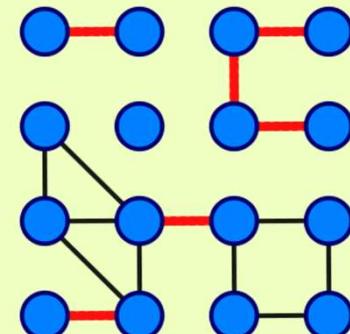
- Se soluționează problemele de completitudine ale căutării în adâncime (DFS)

■ Dezavantaje

- Cum se alege o limită d_{lim} bună?

■ Aplicații

- Determinarea “[podurilor](#)” într-un graf



SCnI în structuri arborescente – căutare în adâncime iterativă (iterative deepening depth search – IDDS)



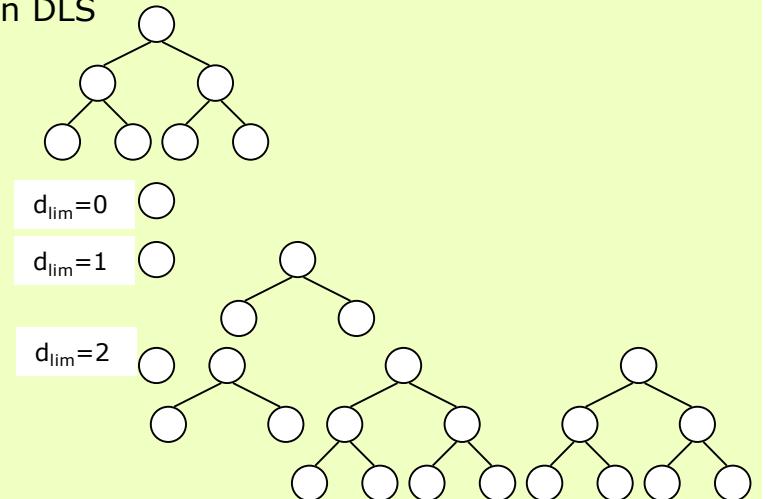
□ Aspecte teoretice

- U DLS(d_{lim}), unde $d_{lim} = 1, 2, 3, \dots, d_{max}$
- Se soluționează problema stabilirii limitei d_{lim} optime din DLS
- De obicei, se aplică acolo unde:
 - spațiul de căutare este mare și
 - se cunoaște lungimea (adâncimea) soluției

□ Exemplu

□ Algoritm

```
bool IDS(elem, list){  
    found = false;  
    dlim = 0;  
    while ((!found) && (dlim < dmax)) {  
        found = DLS(elem, list, dlim);  
        dlim++;  
    }  
    return found;  
}
```



SCnI în structuri arborescente – căutare în adâncime iterativă (iterative deepening depth search – IDDS)



□ Analiza complexității

- Complexitate temporală:
 - Nodurile situate la adâncimea d_{max} (în număr de $b^{d_{max}}$) se expandează o singură dată => $1 * b^{d_{max}}$
 - Nodurile situate la adâncimea $d_{max}-1$ (în număr de $b^{d_{max}-1}$) se expandează de 2 ori => $2 * (b^{d_{max}-1})$
 - ...
 - Nodurile situate la adâncimea 1 (în număr de b) se expandează de d_{max} ori => $d_{max} * b^1$
 - Nodurile situate la adâncimea 0 (în număr de 1 - rădăcina) se expandează de $d_{max}+1$ ori => $(d_{max}+1)*b^0$
- Complexitate spațială
 - $S(n) = b * d_{max}$
- Completitudine
 - Da
- Optimalitate
 - Da

$$T(n) = \sum_{i=0}^{d_{max}} (i+1)b^{d_{max}-i} \Rightarrow O(b^{d_{max}})$$

□ Avantaje

- Necesară memorie liniară
- Asigură atingerea nodului ţintă urmând un drum de lungime minimă
- Mai rapidă decât BFS și DFS

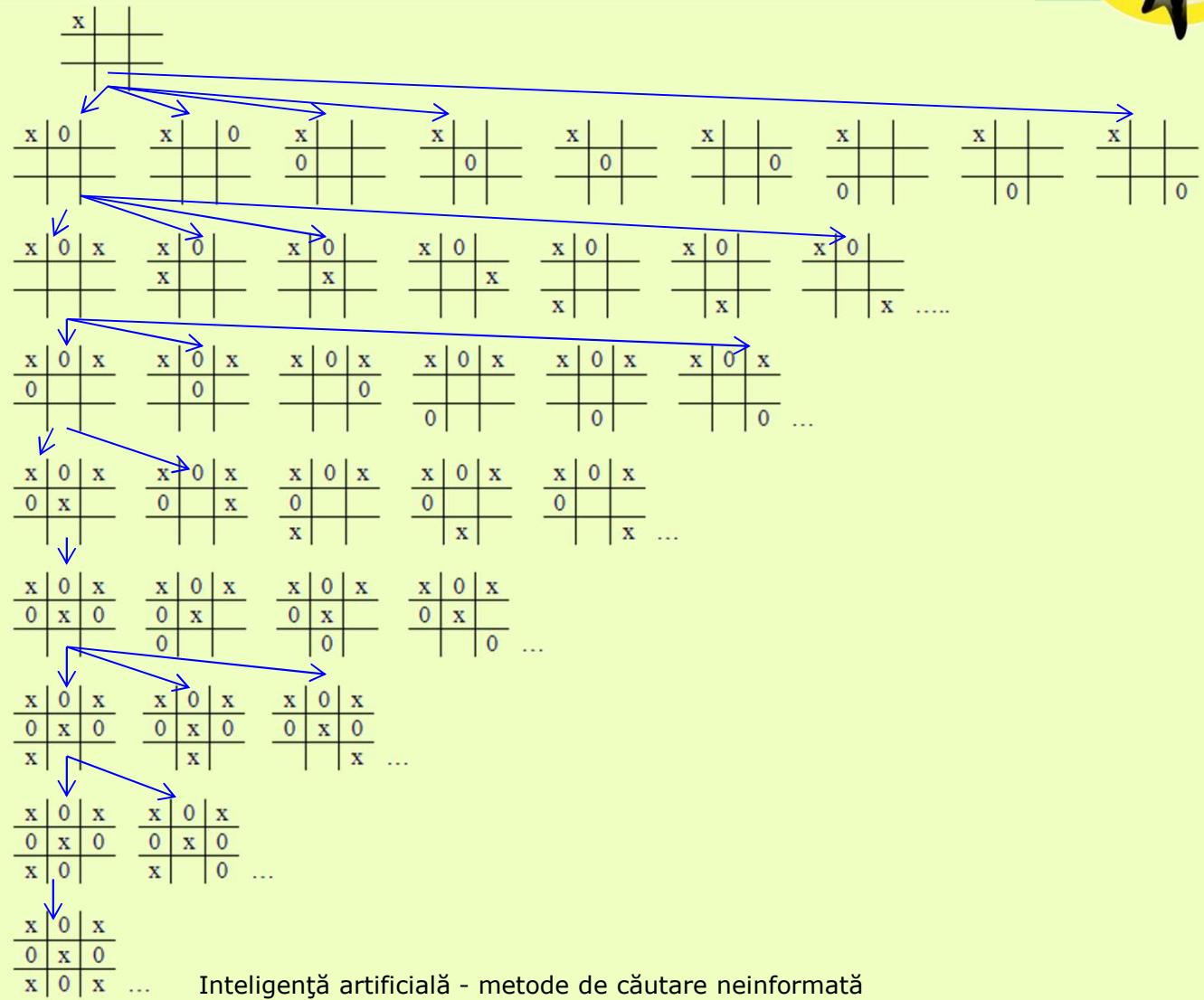
□ Dezavantaje

- Necesară cunoașterea "adâncimii" soluției

□ Aplicații

- Jocul Tic tac toe

SCnI în structuri arborescente – căutare în adâncime iterativă (iterative deepening depth search – IDDS)



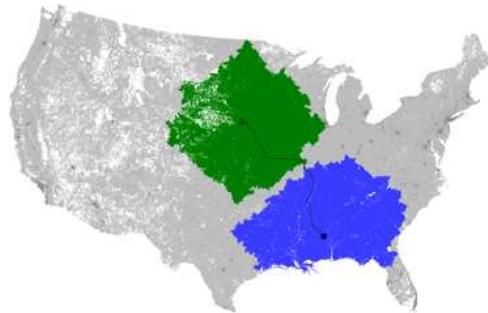
SCnI în structuri arborescente căutare bidirecțională (bi-directional search – BDS)



□ Aspecte teoretice

- 2 căutări simultane
 - Înainte (*forward*): de la rădăcină spre frunze
 - Înapoi (*backward*): de la frunze spre rădăcină
- care se opresc atunci când ajung la un nod comun
- într-o direcție pot fi folosite oricare dintre stategiile de căutare anterioare
- necesită
 - stabilirea succesorilor, respectiv a predecesorilor unui nod
 - stabilirea locului de întâlnire

□ Exemplu



□ Algoritm

- În funcție de strategia de căutare folosită

SCnI în structuri arborescente căutare bidirecțională (bi-directional search – BDS)



□ Analiza complexității

- Complexitate temporală
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d - lungimea (adâncimea) soluției
 - $O(b^{d/2}) + O(b^{d/2}) \Rightarrow O(b^{d/2})$
- Complexitate spațială
 - $S(n) = T(n)$
- Completitudine
 - da
- Optimalitate
 - da

□ Avantaje

- Complexitate spațială și temporală redusă

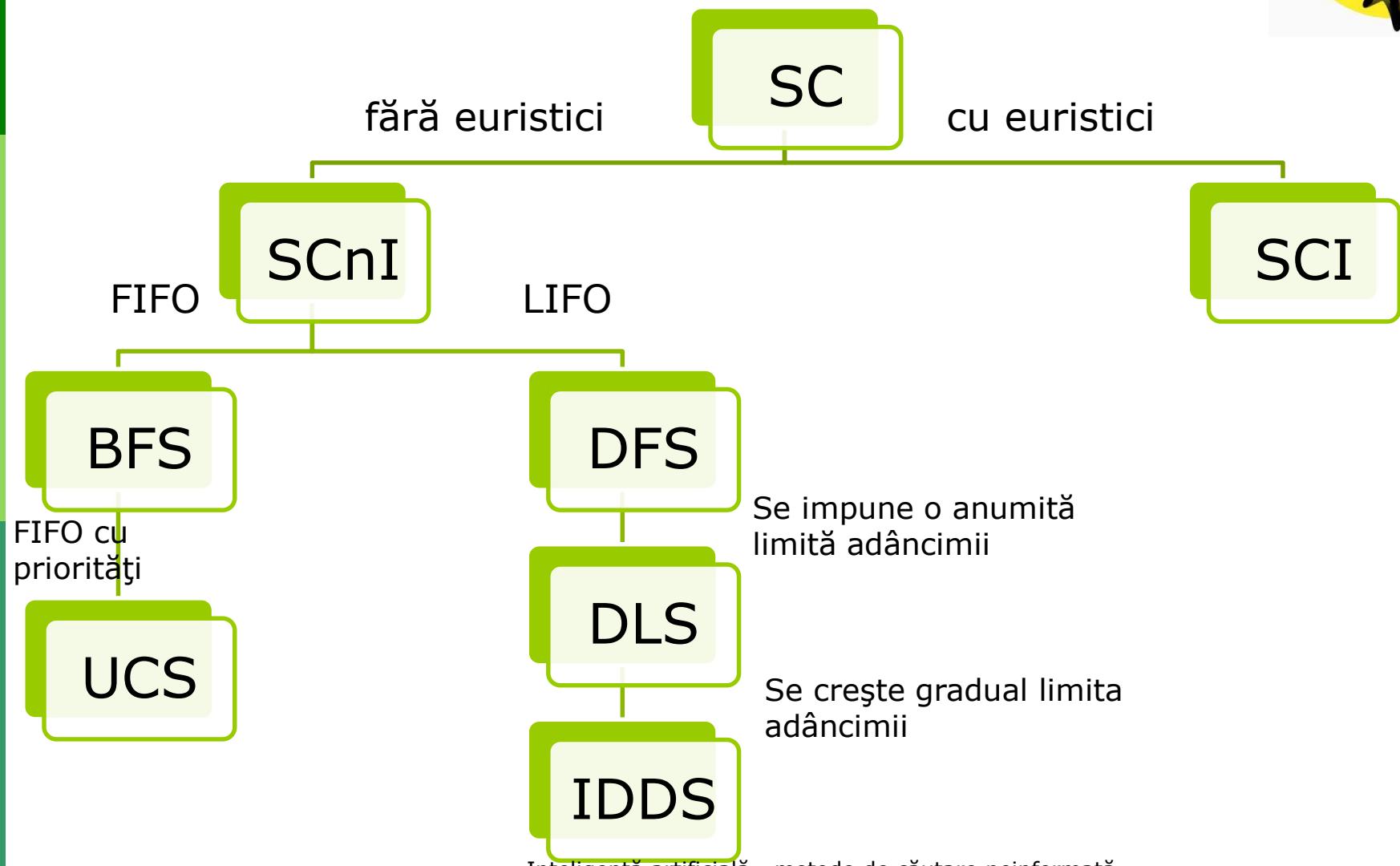
□ Dezavantaje

- Dificultăți în formularea problemei astfel încât fiecare stare să poată fi inversată
 - Parcurgere dinspre cap spre coadă
 - Parcurgere dinspre coadă spre cap
- Implementare dificilă
- Trebuie determinați succesorii și predecesorii tuturor stărilor
- Trebuie cunoscută starea finală (obiectiv)

□ Aplicații

- [Problema partiționării](#)
- Cel mai scurt drum

SCnI în structuri arborescente



SCnI în structuri arborescente



Compararea performanțelor

Metoda de căutare	Complexitate temporală	Complexitate spațială	Complexitate	Optimalitate
BFS	$O(b^d)$	$O(b^d)$	Da	Da
UCS	$O(b^d)$	$O(b^d)$	Da	Da
DFS	$O(b^{d_{\max}})$	$O(b * d_{\max})$	Nu	Nu
DLS	$O(b^{d_{\text{lim}}})$	$O(b * d_{\text{lim}})$	Da dacă $d_{\text{lim}} > d$	Nu
IDS	$O(b^d)$	$O(b * d)$	Da	Da
BDS	$O(b^{d/2})$	$O(b^{d/2})$	Da	Da

Rezolvarea problemelor prin căutare



□ Strategii de căutare (SC)

■ Topologie

□ În funcție de informația disponibilă

- SC ne-informate (oarbe)
- SC informate (euristice)

Strategii de căutare informate (SCI)



□ Caracteristici

- se bazează pe informații specifice problemei încercând să restrângă căutarea prin alegerea intelligentă a nodurilor care vor fi explorate
- ordonarea nodurilor se face cu ajutorul unei funcții (euristici) de evaluare
- sunt particulare

□ Topologie

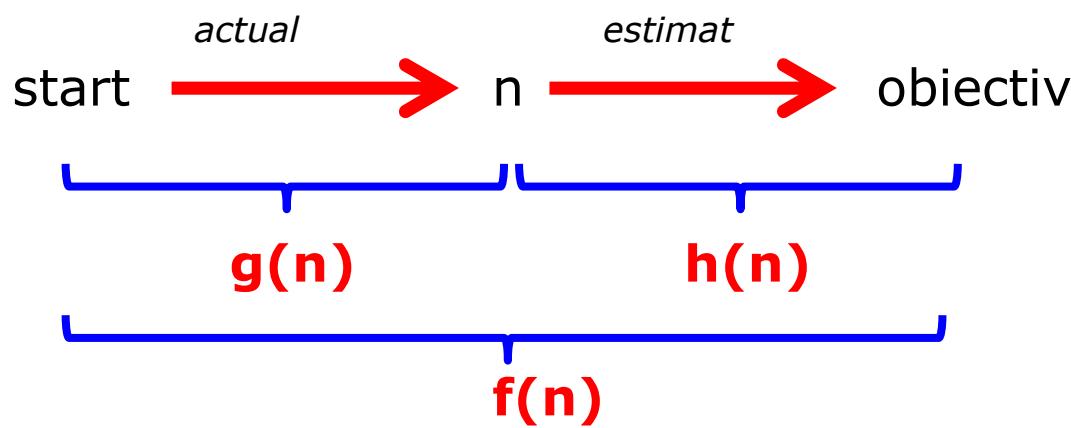
- Strategii globale
 - Best-first search
 - Greedy best-first search
 - A* + versiuni ale A*
- Strategii locale
 - Căutare tabu
 - *Hill climbing*
 - *Simulated annealing*

SC în structuri arborescente



❑ Noțiuni necesare

- $f(n)$ – funcție de evaluare pentru estimarea costului soluției prin nodul (starea) n
- $h(n)$ – funcție euristică pentru estimarea costului drumului de la nodul n la nodul obiectiv
- $g(n)$ – funcție de cost pentru estimarea costului drumului de la nodul de start până la nodul n
- $f(n) = g(n) + h(n)$



SCI – Best first search



□ Aspecte teoretice

- Best first search = mai întâi cel mai bun
- Se determină costul fiecărei stări cu ajutorul funcției de evaluare f
- Nodurile de expandant sunt reținute în structuri (cozi) ordonate
- Pentru expandare se alege starea cu cea mai bună evaluare
 - Stabilirea nodului care urmează să fie expandat
- Exemple de SC care depind de funcția de evaluare
 - Căutare de cost uniform (SCnI)
 - $f = \text{costul drumului}$
 - În SCI se folosesc funcții euristice
- Două categorii de SCI de tip best first search
 - SCI care încearcă să expandeze nodul cel mai apropiat de starea obiectiv
 - SCI care încearcă să expandeze nodul din soluția cu costul cel mai mic

□ Exemplu

- Detalii în slide-urile următoare

SCI – Best first search



□ Algoritm

```
bool BestFS(elem, list){  
    found = false;  
    visited = Φ;  
    toVisit = {start}; //FIFO sorted list (priority queue)  
    while((toVisit != Φ) && (!found)){  
        if (toVisit == Φ)  
            return false  
        node = pop(toVisit);  
        visited = visited ∪ {node};  
        if (node == elem)  
            found = true;  
        else  
            aux = Φ;  
        for all unvisited children of node do{  
            aux = aux ∪ {child};  
        }  
        toVisit = toVisit ∪ aux; //adding a node into the FIFO list based on its  
                           // evaluation (best one in the front of list)  
    } //while  
    return found;  
}
```

SCI – best first search



□ Analiza căutării

- Complexitate temporală:
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d - lungimea (adâncimea) maximă a soluției
 - $T(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
- Complexitate spațială
 - $S(n) = T(n)$
- Completitudine
 - Nu- drumuri infinite dacă euristică evaluează fiecare stare din drum ca fiind cea mai bună alegere
- Optimalitate
 - Posibil – depinde de euristică

□ Avantaje

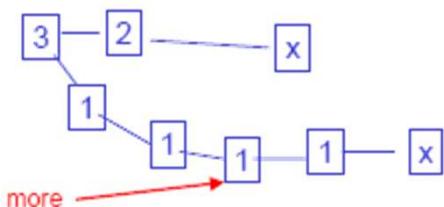
- Informațiile despre domeniul problemei ajută căutarea (SCI)
- Viteză mai mare de a ajunge la starea obiectiv

□ Dezavantaje

- Necesită evaluarea stărilor → efort computațional, dar nu numai
- Anumite path-uri pot “arăta” ca fiind bune conform funcției euristicice

□ Aplicații

- *Web crawler (automatic indexer)*
- Jocuri



SCI – Funcții euristică



- Etimologie: *heuriskein* (gr)
 - *a găsi, a descoperi*
 - *studiu metodelor și regulilor de descoperire și inventie*
- Utilitate
 - Evaluarea potențialului unei stări din spațiul de căutare
 - Estimarea costului drumului (în arborele de căutare) din starea curentă până în starea finală (cât de aproape de țintă a ajuns căutarea)
- Caracteristici
 - Depind de problema care trebuie rezolvată
 - Pentru probleme diferite trebuie proiectate sau învățate diferite euristică
 - Se evaluatează o anumită stare (nu operatorii care transformă o stare în altă stare)
 - Funcții pozitive pentru orice stare n
 - $h(n) \geq 0$ pentru orice stare n
 - $h(n) = 0$ pentru starea finală
 - $h(n) = \infty$ pentru o stare din care începe un drum mort (o stare din care nu se poate ajunge în starea finală)

SCI – Funcții euristică



□ Exemple

- Problema misionarilor și canibalilor
 - $h(n)$ – nr. persoanelor aflate pe malul inițial
- 8-puzzle
 - $h(n)$ – nr pieselor care nu se află la locul lor
 - $h(n)$ – suma distanțelor Manhattan (la care se află fiecare piesă de poziția ei finală)
- Problema comisului voiajor
 - $h(n)$ – cel mai apropiat vecin !!!
- Plata unei sume folosind un număr minim de monezi
 - $h(n)$ – alegerea celei mai valoroase monezi mai mică decât suma (rămasă) de plată

SCI - Greedy



□ Aspecte teoretice

- Funcția de evaluaare $f(n) = h(n)$
 - estimarea costului drumului de la starea curentă la starea finală – $h(n)$
 - minimizarea costului drumului care mai trebuie parcurs

□ Exemplu

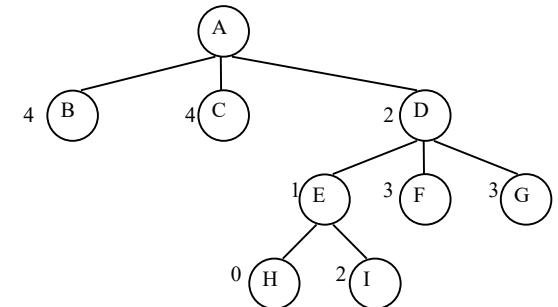
- A,D,E,H

□ Algoritm

```

bool BestFS(elem, list){
    found = false;
    visited = {};
    toVisit = {start};           //FIFO sorted list (priority queue
    while((toVisit != {}) && (!found)){
        if (toVisit == {})
            return false;
        node = pop(toVisit);
        visited = visited U {node};
        if (node == elem)
            found = true;
        else
            aux = {};
        for all unvisited children of node do{
            aux = aux U {child};
        }
        toVisit = toVisit U aux; //adding a node onto the FIFO list based on its evaluation h(n)
                               //(best one in the front of list)
    }
    return found;
}

```



Vizitate deja	De vizitat
Φ	A
A	D, B, C
A, D	E, F, G, B, C
A, D, E	H, I, F, G, B, C
A, D, E, H	Φ

SCI - Greedy



□ **Analiza căutării:**

- Complexitate temporală → DFS
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d^{max} – lungimea (adâncimea) maximă a unui arbore explorat
 - $T(n) = 1 + b + b^2 + \dots + b^{d^{max}} \Rightarrow O(b^{d^{max}})$
- Complexitate spațială → BFS
 - d – lungimea (adâncimea) soluției
 - $S(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
- Completitudine
 - Nu (există posibilitatea intrării în cicluri infinite)
- Optimalitate
 - posibil

□ **Avantaje**

- Găsirea rapidă a unei soluții (dar nu neapărat soluția optimă), mai ales pentru probleme mici

□ **Dezavantaje**

- Suma alegerilor optime de la fiecare pas nu reprezintă alegerea globală optimă
 - Ex. Problema comisului voiajor

□ **Aplicații**

- Probleme de planificare
- Probleme de sume parțiale
 - Plata unei sume folosind diferite tipuri de monezi
 - Problema rucsacului
- Puzzle-uri
- Drumul optim într-un graf

SCI – A*

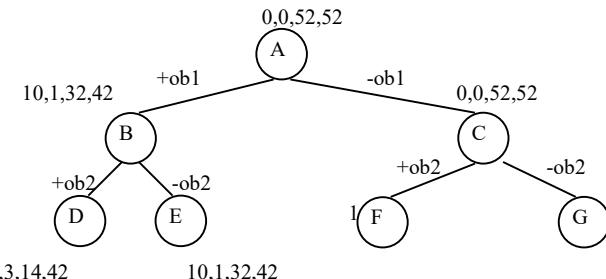


□ Aspecte teoretice

- Combinarea aspectelor pozitive ale
 - căutării de cost uniform
 - optimalitate și completitudine
 - utilizarea unei cozi ordonate
 - căutării Greedy
 - viteza mare
 - ordonare pe baza unei funcții de evaluare
- Funcția de evalaure $f(n)$
 - estimarea costului celui mai bun drum care trece prin nodul n
 - $f(n) = g(n) + h(n)$
 - $g(n)$ – funcție folosită pentru stabilirea costului drumului de la starea inițială până la starea curentă n
 - $h(n)$ – funcție euristică folosită pentru estimarea costului drumului de la starea curentă n la starea finală
- Minimizarea costului total al unui drum

□ Exemplu

- Problema rucsacului de capacitate W în care pot fi puse n obiecte (o_1, o_2, \dots, o_n) fiecare având o greutate w_i și aducând un profit p_i , $i=1,2,\dots,n$
 - Soluția: pentru un rucsac cu $W = 5 \rightarrow$ alegerea obiectelor o_1 și o_3
- $g(n) = \sum p_i$, pentru acele obiecte o_i care au fost selectate
- $h(n) = \sum p_j$, pentru acele obiecte care nu au fost selectate și $\sum w_j \leq W - \sum w_i$
- Fiecare nod din graf este un tuplu: (p, w, p^*, f) , unde:
 - p – profitul adus de obiectele deja selectate (funcția $g(n)$)
 - w – greutatea obiectelor selectate
 - p^* - profitul maxim care se poate obține pornind din starea curentă și ținând cont de locul disponibil în rucsac (funcția $h(n)$)



	o_1	o_2	o_3	o_4
p_i	10	18	32	14
w_i	1	2	4	3

SCI – A*



□ Algorithm

```
bool BestFS(elem, list){  
    found = false;  
    visited =  $\emptyset$ ;  
    toVisit = {start}; //FIFO sorted list (priority queue  
    while((toVisit !=  $\emptyset$ ) && (!found)) {  
        if (toVisit ==  $\emptyset$ )  
            return false;  
        node = pop(toVisit);  
        visited = visited U {node};  
        if (node == elem)  
            found = true;  
        else  
            aux =  $\emptyset$ ;  
        for all unvisited children of node do{  
            aux = aux U {child};  
        }  
        toVisit = toVisit U aux; //adding a node onto the FIFO list  
                           // based on its evaluation  $f(n) = g(n) + h(n)$   
                           // (best one in the front of list)  
    } //while  
    return found;  
}
```

SCI – A*



□ **Analiza căutării:**

- Complexitate temporală:
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d^{max} - lungimea (adâncimea) maximă a unui arbore explorat
 - $T(n) = 1 + b + b^2 + \dots + b^{d^{max}} \Rightarrow O(b^{d^{max}})$
- Complexitate spațială
 - d - lungimea (adâncimea) soluției
 - $T(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
- Completitudine
 - Da
- Optimalitate
 - Da

□ **Avantaje**

- Algoritmul care expandează cele mai puține noduri din arborele de căutare

□ **Dezavantaje**

- Utilizarea unei cantități mari de memorie

□ **Aplicații**

- Probleme de planificare
- Probleme de sume parțiale
 - Plata unei sume folosind diferite tipuri de monezi
 - Problema rucsacului
- Puzzle-uri
- Drumul optim într-un graf

SCI – A*



□ Variante

- iterative deepening A* (IDA*)
- memory-bounded A* (MA*)
- simplified memory bounded A* (SMA*)
- recursive best-first search (RBFS)
- dynamic A* (DA*)
- real time A*
- hierarchical A*

□ Bibliografie suplimentară

- 02/A_IDA.pdf
- 02/A_IDA_2.pdf
- 02/SMA_RTA.pdf
- 02/Recursive Best-First Search.ppt
- 02/IDS.pdf
- 02/IDA_MA.pdf
- <http://en.wikipedia.org/wiki/IDA%2A>
- <http://en.wikipedia.org/wiki/SMA%2A>

Rezolvarea problemelor prin căutare



- Tipologia strategiilor de căutare
 - În funcție de modul de **generare** a soluției
 - Căutare **constructivă**
 - Construirea progresivă a soluției
 - Ex. TSP
 - Căutare **perturbativă**
 - O soluție candidat este modificată în vederea obținerii unei noi soluții candidat
 - Ex. SAT - Propositional Satisfiability Problem
 - În funcție de modul de **traversare** a spațiului de căutare
 - Căutare **sistematică**
 - Traversarea completă a spațiului
 - Ideintificarea soluției dacă ea există → algoritmi compleți
 - Căutare **locală**
 - Traversarea spațiului de căutare dintr-un punct în alt punct vecin → algoritmi incompleți
 - O stare a spațiului poate fi vizitată de mai multe ori
 - În funcție de elementele de **certitudine** ale căutării
 - Căutare **deterministă**
 - Algoritmi de identificare exactă a soluției
 - Căutare **stocastică**
 - Algoritmi de aproximare a soluției
 - În funcție de stilul de **explorare** a spațiului de căutare
 - Căutare **secvențială**
 - Căutare **paralelă**

Rezolvarea problemelor prin căutare



Poate consta în:

- ❑ Construirea progresivă a soluției
- ❑ Identificarea soluției potențiale optime
 - Componentele problemei
 - ❑ Condiții (constrângerii) pe care trebuie să le satisfacă (parțial sau total) soluția
 - ❑ Funcție de evaluare a unei soluții potențiale → identificarea optimului
 - Spațiul de căutare
 - ❑ mulțimea tuturor soluțiilor potențiale complete
 - ❑ Stare = o soluție completă
 - ❑ Stare finală (scop) → soluția optimă
 - Exemple
 - ❑ Problema celor 8 regine,
 - Stările posibile: configurații ale tablei de sah cu câte 8 regine
 - Operatori: modificarea coloanei în care a fost plasată una din regine
 - Scopul căutării: configurația în care nu există atacuri între regine
 - Funcția de evaluare: numărul de atacuri
 - ❑ probleme de planificare,
 - ❑ proiectarea circuitelor digitale, etc



Rezolvarea problemelor prin căutare



www.shutterstock.com - 38774760

Poate consta în:

- ❑ Construirea progresivă a soluției
- ❑ Identificarea soluției potențiale optime
 - Algoritmi
 - ❑ Algoritmii discutați până acum explorau în mod **sistemtic** spațiul de căutare
 - De ex. A* → 10^{100} stări ≈ 500 variabile binare
 - ❑ Problemele reale pot avea 10 000 – 100 000 variabile → nevoia unei alte categorii de algoritmi care explorează **local** spațiul de căutare (algoritmi iterativi)
 - ❑ Ideea de bază:
 - se începe cu o stare care nu respectă anumite constrângeri pentru a fi soluție optimă și
 - se efectuează modificări pentru a elimina aceste violări (se deplasează căutarea într-o soluție vecină cu soluția curentă) astfel încât căutarea să se îndrepte spre soluția optimă
 - ❑ Iterativi → se memorează o singură stare și se încearcă îmbunătățirea ei
 - versiunea intelligentă a algoritmului de forță brută
 - ❑ Istoricul căutării nu este reținut

```
bool LS(elem, list){  
    found = false;  
    crtState = initState  
    while ((!found) && timeLimitIsNotExceeded) {  
        toVisit = neighbours(crtState)  
        if (best(toVisit) is better than crtState)  
            crtState = best(toVisit)  
        if (crtState == elem)  
            found = true;  
    } //while  
    return found;  
}
```

Rezolvarea problemelor prin căutare



Poate constă în:

- ❑ Construirea progresivă a soluției
- ❑ Identificarea soluției potențiale optime
 - Avantaje
 - ❑ Simplu de implementat
 - ❑ Necesară puțină memorie
 - ❑ Poate găsi soluții rezonabile în spații de căutare (continue) foarte mari pentru care alți algoritmi sistematici nu pot fi aplicati
 - E utilă atunci când:
 - ❑ Se pot genera soluții complete rezonabile
 - ❑ Se poate alege un bun punct de start
 - ❑ Există operatori pentru modificarea unei soluții complete
 - ❑ Există o măsură pentru a aprecia progresul (avansarea căutării)
 - ❑ Există un mod de a evalua soluția completă (în termeni de constrângeri violate)



Strategii de căutare locală

□ Tipologie

- Căutare locală simplă - se reține o singură stare vecină
 - Hill climbing → alege cel mai bun vecin
 - Simulated annealing → alege probabilistic cel mai bun vecin
 - Căutare tabu → reține lista soluțiilor recent vizitate
- Căutare locală în fascicol (*beam local search*) – se rețin mai multe stări (o populație de stări)
 - Algoritmi evolutivi
 - Optimizare bazată pe comportamentul de grup (*Particle swarm optimisation*)
 - Optimizare bazată pe furnici (*Ant colony optimisation*)

Strategii de căutare locală



□ Căutare locală simplă

■ elemente de interes special:

- Reprezentarea soluției
- Funcția de evaluare a unei potențiale soluții
- Vecinătatea unei soluții
 - Cum se definește/generează o soluție vecină
 - Cum are loc căutarea soluțiilor vecine
 - Aleator
 - Sistematic
- Criteriul de acceptare a unei noi soluții
 - Primul vecin mai bun decât soluția curentă
 - Cel mai bun vecin al soluției curente mai bun decât soluția curentă
 - Cel mai bun vecin al soluției curente mai slab decât soluția curentă
 - Un vecin aleator

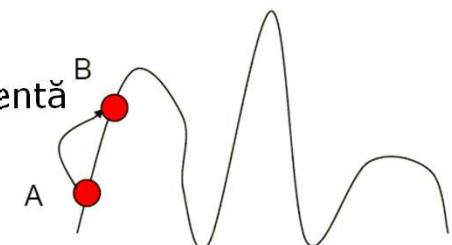
} dependente
de problemă

Strategii de căutare locală – Hill climbing (HC)



□ Aspecte teoretice

- Urcarea unui munte în condiții de ceată și amnezie a excursionistului :D
- Mișcarea continuă spre valori mai bune (mai mari → urcușul pe munte)
- Căutarea avansează în direcția îmbunătățirii valorii stărilor succesor până când se atinge un optim
- Criteriul de acceptare a unei noi soluții
 - cel mai bun vecin al soluției curente mai bun decât soluția curentă
- Îmbunătățire prin
 - Maximizarea calității unei stări → *steepest ascent HC*
 - Minimizarea costului unei stări → *gradient descent HC*
- HC ≠ *steepest ascent/gradient descent* (SA/GD)
 - HC optimizează $f(x)$ cu $x \in \mathbb{R}^n$ prin modificarea unui element al lui x
 - SA/GD optimizează $f(x)$ cu $x \in \mathbb{R}^n$ prin modificarea tuturor elementelor lui x



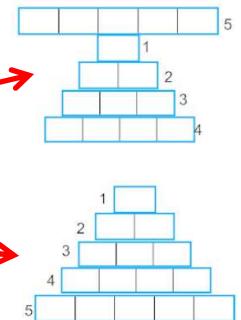
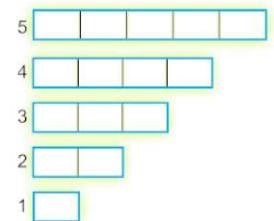
Strategii de căutare locală – Hill climbing (HC)



□ Exemplu

- Construirea unor turnuri din diferite forme geometrice
 - Se dau n piese de formă dreptunghiulară (de aceeași lățime, dar de lungimi diferite) așezate unele peste altele formând un turn (stivă). Să se re-așeze piesele astfel încât să se formeze un nou turn știind că la o mutare se poate mișca doar o piesă din vârful stivei, piesă care poate fi mutată pe una din cele 2 stive ajutătoare.

- Reprezentarea soluției
 - Stare x – vectori de n perechi de forma (i,j) , unde i reprezintă indexul piesei ($i=1,2,\dots,n$), iar j indexul stivei pe care se află piesa ($j=1,2,3$)
 - Starea inițială – vectorul corespunzător turnului inițial
 - Starea finală – vectorul corespunzător turnului final
- Funcția de evaluare a unei stări
 - f_1 = numărul pieselor corect amplasate \rightarrow maximizare
 - conform turnului final – $f_1 = n$
 - f_2 = numărul pieselor greșit amplasate \rightarrow minimizare
 - conform turnului final – $f_2 = 0$
 - $f = f_1 - f_2 \rightarrow$ maximizare
- Vecinătate
 - Mutări posibile
 - Mutarea unei piese i din vârful unei stive j_1 pe altă stivă j_2
- Criteriul de acceptare a unei noi soluții
 - Cel mai bun vecin al soluției curente mai bun decât soluția curentă



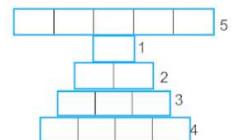
Strategii de căutare locală – Hill climbing (HC)



❑ Exemplu

■ Iterația 1

- ❑ Starea curentă $x =$ starea inițială:
 - $x = s_1 = ((5,1), (1,1), (2,1), (3,1), (4,1))$
 - Piese 1, 2 și 3 sunt corect așezate
 - Piese 4 și 5 nu sunt corect așezate
 - $f(s_1) = 3 - 2 = 1$



- ❑ $x^* = x$

- ❑ Vecinii stării curente x – un singur vecin → piesa 5 se mută pe stiva 2 ➔
 - $s_2 = ((1,1), (2,1), (3,1), (4,1), (5,2))$
 - $f(s_2) = 4 - 1 = 3 > f(x) \rightarrow x = s_2$



Strategii de căutare locală – Hill climbing (HC)



□ Exemplu

■ Iterația 2

- Starea curentă $x = ((1,1), (2,1), (3,1), (4,1), (5,2))$
 - $f(x) = 3$
- Vecinii stării curente – doi vecini:
 - piesa 1 se mută pe stiva 2 $\rightarrow s_3 = ((2,1), (3,1), (4,1), (1,2), (5,2)) \rightarrow f(s_3) = 3-2=1 < f(x)$ 
 - piesa 1 se mută pe stiva 3 $\rightarrow s_4 = ((2,1), (3,1), (4,1), (5,2), (1,3)) \rightarrow f(s_4) = 3-2=1 < f(x)$ 

■ $x^* = x = ((1,1), (2,1), (3,1), (4,1), (5,2))$

■ Dar x^* este doar optim local, nu global

Strategii de căutare locală – Hill climbing (HC)



□ Exemplu

- Construirea unor turnuri din diferite forme geometrice

□ Funcția de evaluare a unei stări

- $f_1 = \text{suma înălțimilor stivelor pe care sunt amplasate corect piese (conform turnului final) - } f_1 = 10 \rightarrow \text{maximizare}$
- $f_2 = \text{suma înălțimilor stivelor pe care sunt amplasate incorect piese (conform turnului final) - } f_2 = 0 \rightarrow \text{minimizare}$
- $f = f_1 - f_2 \rightarrow \text{maximizare}$

□ Vecinătate

- Mutări posibile
 - Mutarea unei piese i din vârful unei stive j_1 pe altă stivă j_2

Strategii de căutare locală – Hill climbing (HC)



❑ Exemplu

■ Iterația 1

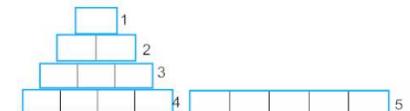
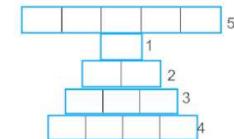
- ❑ Starea curentă $x =$ starea inițială $s_1 = ((5,1), (1,1), (2,1), (3,1), (4,1))$

- Toate piesele nu sunt corect așezate $\rightarrow f_1 = 0, f_2 = 3+2 + 1 + 0 + 4 = 10$
 - $f(s_1) = 0 - 10 = -10$

- ❑ $x^* = x$

- ❑ Vecinii stării curente x – un singur vecin \rightarrow piesa 5 se mută pe stiva 2 $\rightarrow s_2 = ((1,1), (2,1), (3,1), (4,1), (5,2))$

- $f(s_2) = 0 - (3+2+1+0) = -6 > f(x) \rightarrow x = s_2$



Strategii de căutare locală – Hill climbing (HC)



□ Exemplu

■ Iterația 2

□ Starea curentă $x = ((1,1), (2,1), (3,1), (4,1), (5,2))$

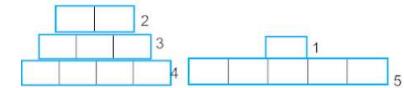
▪ $f(x) = -6$

□ Vecinii stării curente – doi vecini:

▪ piesa 1 se mută pe stiva 2 $\rightarrow s_3 = ((2,1), (3,1), (4,1), (1,2), (5,2)) \rightarrow f(s_3) = 0 - (0+2+3+0) = -5 > f(x)$



▪ piesa 1 se mută pe stiva 3 $\rightarrow s_4 = ((2,1), (3,1), (4,1), (5,2), (1,3)) \rightarrow f(s_4) = 0 - (1+2+1) = -4 > f(x)$



▪ cel mai bun vecin al lui x este $s_4 \rightarrow x = s_4$

■ Iterația 3

□ ...

■ Se evită astfel blocarea în optimele locale

Strategii de căutare locală – Hill climbing (HC)



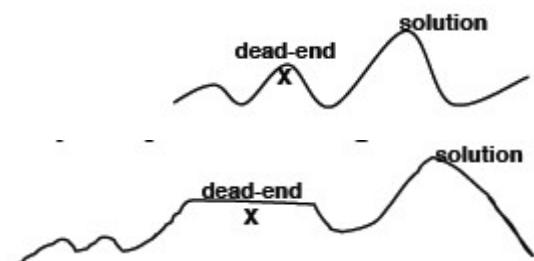
□ Algoritm

```
Bool HC(S) {  
    x = s1      //starea inițială  
    x*=x        // cea mai bună soluție găsită  
    (până la un moment dat)  
    k = 0        // numarul de iterații  
    while (not termiantion criteria) {  
        k = k + 1  
        generate all neighbours of x (N)  
        Choose the best solution s from N  
        if f(s) is better than f(x) then x = s  
        else stop  
    } //while  
    x* = x  
    return x*;  
}
```

Strategii de căutare locală – Hill climbing (HC)



- **Analiza căutării**
 - Convergența spre optimul local
- **Avantaje**
 - Simplu de implementat → se poate folosi ușor pentru a aproxima soluția unei probleme când soluția exactă este dificil sau imposibil de găsit
 - Ex. TSP cu forate multe orașe
 - Nu necesită memorie (nu se revine în starea precedentă)
- **Dezavantaje**
 - Funcția de evaluare (eurisitică) poate fi dificil de estimat
 - Dacă se execută foarte multe mutări algoritmul devine inefficient
 - Dacă se execută prea puține mutări algoritmul se poate bloca
 - Într-un optim local (nu mai poate "cobeală" din vârf)
 - Pe un platou – zonă din spațiul de căutare în care funcția de evaluare este constantă
 - Pe o creastă – saltul cu mai mulți pași ar putea ajuta căutarea
- **Aplicații**
 - Problema canibalilor
 - 8-puzzle, [15-puzzle](#)
 - TSP
 - Problema damelor



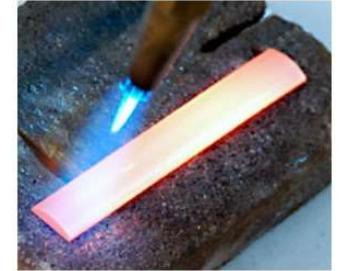
Strategii de căutare locală – Hill climbing (HC)



□ Variante

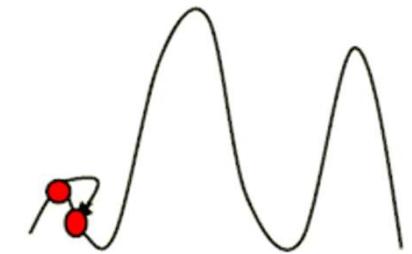
- HC stocastic
 - Alegerea aleatoare a unui succesor
- HC cu prima opțiune
 - Generarea aleatoare a successorilor până la întâlnirea unei mutări neefectuate
- HC cu repornire aleatoare → *beam local search*
 - Repornirea căutării de la o stare inițială aleatoare atunci când căutarea nu progresează

Strategii de căutare locală – Simulated Annealing

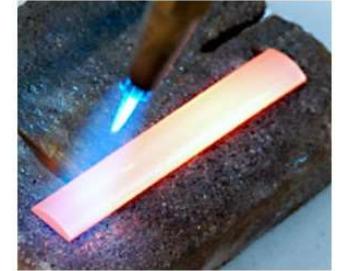


□ Aspecte teoretice

- Inspirată de modelarea proceselor fizice
 - Metropolis et al. 1953, Kirkpatrick et al. 1982;
- Succesorii stării curente sunt aleși și în mod aleator
 - Dacă un succesor este mai bun decât starea curentă
 - atunci el devine noua stare curentă
 - altfel succesorul este reținut doar cu o anumită probabilitate
- Se permit efectuarea unor mutări "slabe" cu o anumită probabilitate p
 - → evadarea din optimele locale
- Probabilitatea $p = e^{\Delta E/T}$
 - Proporțională cu valoarea diferenței (energia) ΔE
 - Modelată de un parametru de temperatură T
- Frecvența acestor mutări "slabe" și mărimea lor se reduce gradual prin scăderea temperaturii
 - $T = 0 \rightarrow$ hill climbing
 - $T \rightarrow \infty \rightarrow$ mutările "slabe" sunt tot mai mult executate
- Soluția optimă se identifică dacă temperatura se scade treptat ("slowly")
- Criteriul de acceptare a unei noi soluții
 - Un vecin aleator mai bun sau mai slab (cu probabilitatea p) decât soluția curentă



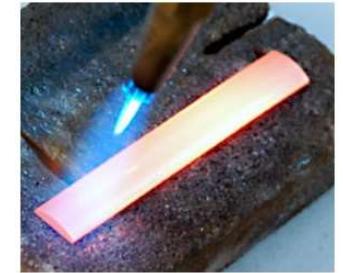
Strategii de căutare locală – Simulated Annealing



□ Exemplu – Problema celor 8 regine

- Enunț
 - Să se amplaseze pe o tablă de săh 8 regine astfel încât ele să nu se atace reciproc
- Reprezentarea soluției
 - Stare x – permutare de n elemente $x = (x_1, x_2, \dots, x_n)$, unde x_i – linia pe care este plasată regina de pe coloana j
 - Nu există atacuri pe verticală sau pe orizontală
 - Pot exista atacuri pe diagonală
 - Starea inițială – o permutare oarecare
 - Starea finală – o permutare fără atacuri de nici un fel
- Funcția de evaluare a unei stări
 - f – suma reginelor atacate de fiecare regină → minimizare
- Vecinătate
 - Mutări posibile
 - Mutarea unei regine de pe o linie pe alta (interschimbarea a 2 elemente din permutare)
- Criteriul de acceptare a unei noi soluții
 - Un vecin oarecare al soluției curente
 - mai bun decât soluția curentă
 - mai slab decât soluția curentă – cu o anumită probabilitate $P(\Delta E) = e^{-\frac{\Delta E}{T}}$ unde:
 - ΔE – diferența de energie (evaluare) între cele 2 stări (vecină și curentă)
 - T – temperatura, $T(k) = 100/k$, unde k este nr iterătiei

Strategii de căutare locală – Simulated Annealing



Exemplu – Problema celor 8 regine

Iterația 1 ($k = 1$)

■ Starea curentă x = starea inițială

$$s_1 = (8, 5, 3, 1, 6, 7, 2, 4)$$

- $f(s_1) = 1+1 = 2$

- $x^* = x$

- $T = 100/1 = 100$

■ Un vecin al stării curente $x \rightarrow$ regina de pe linia 5 se interschimbă cu regina de pe linia 7

$\Rightarrow s_2 = (8, 7, 3, 1, 6, 5, 2, 4)$

- $f(s_2) = 1+1+1=3 > f(x)$

- $\Delta E = f(s_2) - f(s_1) = 1$

- $P(\Delta E) = e^{-1/100}$

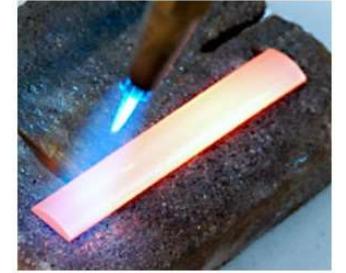
- $r = \text{random}(0,1)$

- Dacă $r < P(\Delta E) \Rightarrow x = s_2$

	a	b	c	d	e	f	g	h	
1	Q			Q					1
2									2
3			Q						3
4								Q	4
5	Q								5
6									6
7		Q						Q	7
8	Q								8
	a	b	c	d	e	f	g	h	

	a	b	c	d	e	f	g	h	
1	Q			Q					1
2									2
3			Q						3
4								Q	4
5	Q								5
6									6
7		Q						Q	7
8	Q								8
	a	b	c	d	e	f	g	h	

Strategii de căutare locală – Simulated Annealing



□ Algoritm

```
bool SA(S){  
    x = s1           //starea inițială  
    x*=x // cea mai bună soluție găsită (până la un moment dat)  
    k = 0 // numarul de iterații  
    while (not termiantion criteria){  
        k = k + 1  
        generate a neighbour s of x  
        if f(s) is better than f(x) then x = s  
        else  
            pick a random number r (in (0,1) range)  
            if r < P( $\Delta E$ ) then x = s  
    } //while  
    x* = x  
    return x*;  
}
```

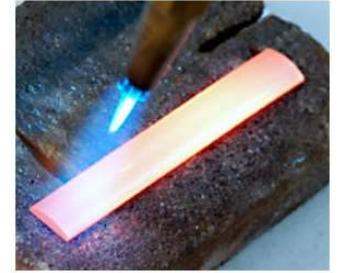
□ Criterii de oprire

- S-a ajuns la soluție
- S-a parcurs un anumit număr de iterații
- S-a ajuns la temepratura 0 (îngheț)

□ Cum se alege o probabilitate mică?

- $p = 0.1$
- p scade de-a lungul iterațiilor
- p scade de-a lungul iterațiilor și pe măsură ce "răul" $|f(s) - f(x)|$ crește
 - $p = \exp(-|f(s) - f(x)|/T)$
 - Unde T – temepratura (care crește)
 - Pentru o T mare se admite aproape orice vecin v
 - Petnru o T mică se admite doar un vecin mai bun decât s
 - Dacă "răul" e mare, atunci probabilitatea e mică

Strategii de căutare locală – Simulated Annealing



□ Analiza căutării

- Convergența (complet, optimal) lentă spre optimul global

□ Avantaje

- Algoritm fundamentat statistic → garantează găsirea soluției optime, dar necesită multe iterații
- Ușor de implementat
- În general găsește o soluție relativ bună (optim global)
- Poate rezolva probleme complexe (cu zgomot și multe constrângeri)

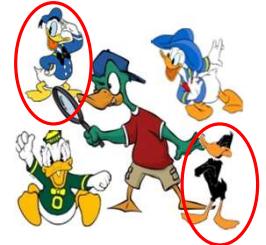
□ Dezavantaje

- Algoritm încet – convergența la soluție durează foarte mult timp
 - Compromis între calitatea soluției și timpul necesar calculării ei
- Depinde de anumiți parametri (temperatura) care trebuie reglați
- Nu se știe dacă soluția oferită este optimă (local sau global)
- Calitatea soluției găsite depinde de precizia variabilelor implicate în algoritm

□ Aplicații

- Probleme de optimizare combinatorială → problema rucsacului
- Probleme de proiectare → Proiectarea circuitelor digitale
- Probleme de planificare → Planificarea producției, planificarea meciurilor în turnurile de tenis US Open

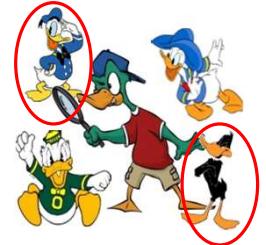
Strategii de căutare locală – Căutare tabu



□ Aspecte teoretice

- “Tabu” → interdicție socială severă cu privire la activitățile umane sacre și interzise
- Propusă în anii 1970 de către F. Glover
- Ideea de bază
 - se începe cu o stare care nu respectă anumite constrângeri pentru a fi soluție optimă și
 - se efectuează modificări pentru a elimina aceste violări (se deplasează căutarea în **cea mai bună soluție vecină** cu soluția curentă) astfel încât căutarea să se îndrepte spre soluția optimă
 - se memorează
 - **Starea curentă**
 - **Stările vizitate** până la momentul curent al căutării și **mutările efectuate** pentru a ajunge în fiecare din acele stări de-a lungul căutării (se memorează o **listă limitată de stări care nu mai trebuie revizitate**)
- Criteriul de acceptare a unei noi soluții
 - Cel mai bun vecin al soluției curente mai bun decât soluția curentă și nevizitat încă
- 2 elemente importante
 - Mutări tabu (T) – determinate de un proces non-Markov care se folosește de informațiile obținute în timpul căutării de-a lungul ultimelor generații
 - Condiții tabu – pot fi inegalități liniare sau legături logice exprimate în funcție de soluția curentă
 - Au rol în alegerea mutărilor tabu

Strategii de căutare locală – Căutare tabu



□ Exemplu

■ Enunț

- Plata sumei S folosind cât mai multe dintre cele n monezi de valori v_i (din fiecare monedă există b_i bucăți)

■ Reprezentarea soluției

- Stare x – vector de n întregi $x = (x_1, x_2, \dots, x_n)$ cu $x_i \in \{0, 1, 2, \dots, b_i\}$
- Starea inițială – aleator

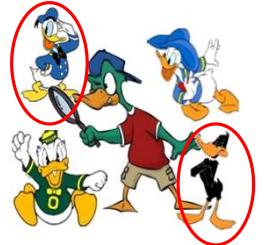
■ Funcția de evaluare a unei stări

- f_1 = Diferența între S și valoarea monezilor alese \rightarrow minimă
 - Dacă valoarea monezilor depășește $S \rightarrow$ penalizare de 500 unități
- f_2 = Numărul monezilor selectate \rightarrow maxim
- $f = f_1 - f_2 \rightarrow$ minimizare

■ Vecinătate

- Mutări posibile
 - Includerea în sumă a monezii i în j exemplare ($\text{plus}_{i,j}$)
 - Excluderea din sumă a monezii i în j exemplare ($\text{minus}_{i,j}$)
- Lista tabu reține mutările efectuate într-o iterație
 - Mutare – moneda adăugată/eliminată din sumă

Strategii de căutare locală – Căutare tabu



■ Exemplu

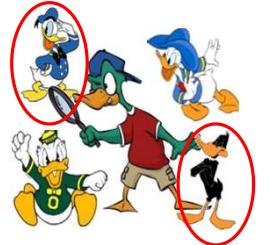
- $S = 500$, penalizare = 500, $n = 7$

$S=500$	m_1	m_2	m_3	m_4	m_5	m_6	m_7
v_i	10	50	15	20	100	35	5
b_i	5	2	6	5	5	3	10

Stare curentă	Val. f	Listă tabu	Stări vecine	Mutări	Val. f
2 0 1 0 0 2 1	384	∅	2 0 1 3 0 2 1	plus _{4,3}	321
			2 0 1 0 0 3 1	plus _{6,1}	348
			0 0 1 0 0 2 1	minus _{1,2}	406
2 0 1 3 0 2 1	321	plus _{4,3}	2 0 1 3 5 2 1	plus _{5,5}	316
			2 0 1 1 0 2 1	minus _{4,2}	363
			2 1 1 3 0 2 1	plus _{2,1}	270
2 1 1 3 0 2 1	270	plus _{4,3} plus _{2,1}	...		

- Soluția finală: 4 1 5 4 1 3 10 ($f = -28$)

Strategii de căutare locală – Căutare tabu



□ Exemplu

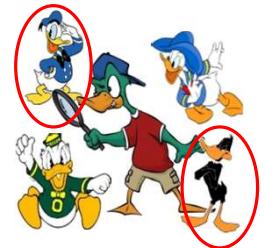
- $S = 500$, penalizare = 500, $n = 7$

$S=50$	m_1	m_2	m_3	m_4	m_5	m_6	m_7
v_i	10	50	15	20	100	35	5
b_i	5	2	6	5	4	3	10

Stare currentă	Val. f	Listă tabu	Stări vecine	Mutări	Val. f
2 0 1 0 0 2 1	384	\emptyset	1 0 1 4 0 2 1	minus _{1,1} , plus _{4,4}	311
			2 0 4 0 1 2 1	plus _{3,3} , minus _{5,1}	235
			2 0 1 0 4 2 6	plus _{5,4} , plus _{7,5}	450
2 0 4 0 1 2 1	235	plus _{3,3} , minus _{5,1}	2 0 5 0 5 2 1	plus _{3,1} , plus _{5,4}	315
			5 0 4 0 4 2 1	plus _{1,3} , plus _{5,3}	399
			2 2 4 0 5 2 1	plus _{2,2} , plus _{5,4}	739
2 0 4 0 1 2 1	235	plus _{3,3} , minus _{5,1}	...		

- Soluția finală: 4 1 5 4 1 3 10 ($f = -28$)

Strategii de căutare locală – Căutare tabu



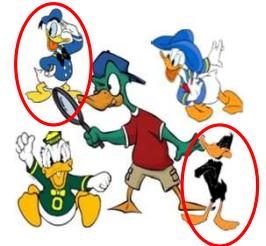
▣ Algoritm

```
bool TS(S) {
    Select x∈S //S - spațiul de căutare
    x*=x        //cea mai bună soluție (până la un mom. dat)
    k = 0         // numarul de iterații
    T = Ø          // listă de mutări tabu
    while (not termiantion criteria) {
        k = k + 1
        generate a subset of solutions in the neighborhood N-T of x
        choose the best solution s from N-T and set x=s.
        if f(x)<f(x*) then x*=x
        update T with moves of generating x
    } //while
    return x*;
}
```

■ Criterii de terminare

- ▣ Număr fix de iterații
- ▣ Număr de iterații fără îmbunătățiri
- ▣ Apropierea suficientă de soluție (dacă aceasta este cunoscută)
- ▣ Epuizarea elementelor nevizitate dintr-o vecinătate

Strategii de căutare locală – Căutare tabu



□ **Analiza căutării**

- Convergența rapidă spre optimul global

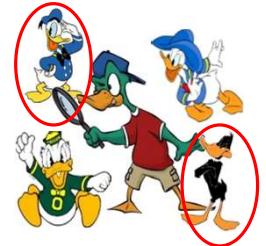
□ **Avantaje**

- Algoritm general și simplu de implementat
- Algoritm rapid (poate oferi soluția optimă globală în scurt timp)

□ **Dezavantaje**

- Stabilirea stărilor vecine în spații continue
- Număr mare de iterații
- Nu se garantează atingerea optimului global

Strategii de căutare locală – Căutare tabu



□ Aplicații

- Determinarea structurii tridimensionale a proteinelor în secvențe de aminoacizi (optimizarea unei funcții de potențial energetic cu multiple optime locale)
- Optimizarea traficului în rețele de telecomunicații
- Planificare în sisteme de producție
- Proiectarea rețelelor de telecomunicații optice
- Ghidaj automat pentru vehicule
- Probleme în grafuri (partiționări)
- Planificări în sistemele de audit
- Planificări ale task-urilor paralele efectuate de procesor (multiprocesor)
- Optimizarea structurii electromagnetice (imagistica rezonanței magnetice medicale)
- Probleme de asignare quadratică (proiectare VLSI)
- Probleme de combinatorică (ricsac, plata sumei)
- Problema tăierii unei bucăți în mai multe părți
- Controlul structurilor spațiale (NASA)
- Optimizarea proceselor cu decizii multi-stagiu
- Probleme de transport
- Management de portofoliu
- Chunking



Recapitulare

- SCI best first search
 - Nodurile mai bine evaluate (de cost mai mic) au prioritate la expandare
 - SCI de tip greedy
 - minimizarea costului de la starea curentă la starea obiectiv – $h(n)$
 - Timp de căutare < SCnI
 - Ne-completă
 - Ne-optimală
 - SCI de tip A*
 - minimizarea costului de la starea inițială la starea curentă – $g(n)$ – și a costului de la starea curentă la starea obiectiv – $h(n)$
 - Evitarea repetării stărilor
 - Fără supraestimarea lui $h(n)$
 - Timp și spațiu de căutare mare → în funcție de euristica folosită
 - Complet
 - Optimal



Recapitulare

□ SC locale

■ Algoritmi iterativi

- Lucrează cu o soluție potențială → soluția optimă
- Se pot bloca în optime locale

	Alegerea stării următoare	Criteriul de acceptare	Convergența
HC	Cel mai bun vecin	Vecinul este mai bun decât strarea curentă	Optim local sau global
SA	Un vecin oarecare	Vecinul este mai bun sau mai slab (acceptat cu probabilitatea p) decât starea curentă	Optim global (lentă)
TS	Cel mai bun vecin nevizitat încă	Vecinul este mai bun decât strarea curentă	Optim global (rapidă)

Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, **Algoritmi evolutivi, PSO, ACO**)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor – Materiale de citit și legături utile

- capitolul 14 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998*
- capitolul 7.6 din *A. A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

-
- Informațiile prezentate au fost colectate din diferite surse bibliografice, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan - www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ



Rezolvarea problemelor de căutare

Strategii de căutare informată locală

Algoritmi Evolutivi

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Sumar

- Rezolvarea problemelor prin căutare
 - Strategii de căutare informate (euristice) – SCI
 - Strategii locale
 - Algoritmi evolutivi

Materiale de citit și legături utile

- capitolul 14 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998*
- capitolul 7.6 din *A. A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Căutare locală

□ Tipologie

- Căutare locală simplă - se reține o singură stare vecină
 - Hill climbing → alege cel mai bun vecin
 - Simulated annealing → alege probabilistic cel mai bun vecin
 - Căutare tabu → reține lista soluțiilor recent vizitate
- Căutare locală în fascicol (beam local search) – se rețin mai multe stări (o populație de stări)
 - Algoritmi evolutivi
 - Optimizare bazată pe comportamentul de grup (Particle swarm optimisation)
 - Optimizare bazată pe furnici (Ant colony optimisation)

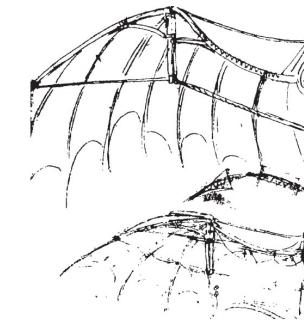
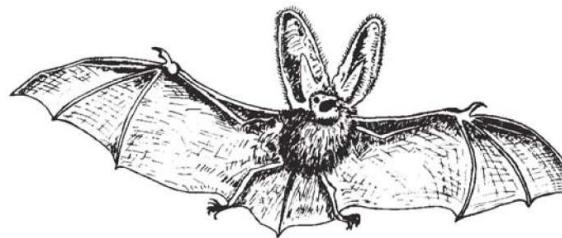
Algoritmi inspirați de natură

- Care este cea mai bună metodă de rezolvare a unei probleme?
 - Creierul uman
 - a creat roata, mașina, orașul, etc
 - Mecanismul evoluției
 - a creat creierul (mintea) umană
- Simularea naturii
 - Cu ajutorul mașinilor → rețelele neuronale artificiale simulează mintea umană
 - mașini de zbor, computere bazate pe ADN, computere cu membrane
 - Cu ajutorul algoritmilor
 - algoritmii evolutivi simulează evoluția naturii
 - algoritmii inspirați de comportamentul de grup simulează adaptarea colectivă și procesele sociale dintr-un colectiv (*Particle Swarm Optimisation*)
 - algoritmii inspirați de furnici (*Ant Colony Optimisation*)

Algoritmi evolutivi – aspecte teoretice

□ Simularea naturii

■ Zborul lileicilor

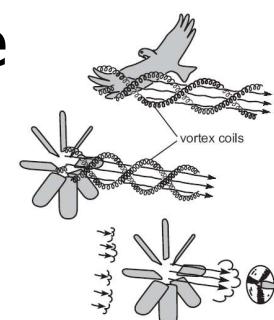


■ Leonardo da Vinci – schema unei mașini de zbor



■ Zborul păsărilor și al avioanelor

■ Zborul păsărilor și turbinele eoliene



Algoritmi evolutivi – aspecte teoretice

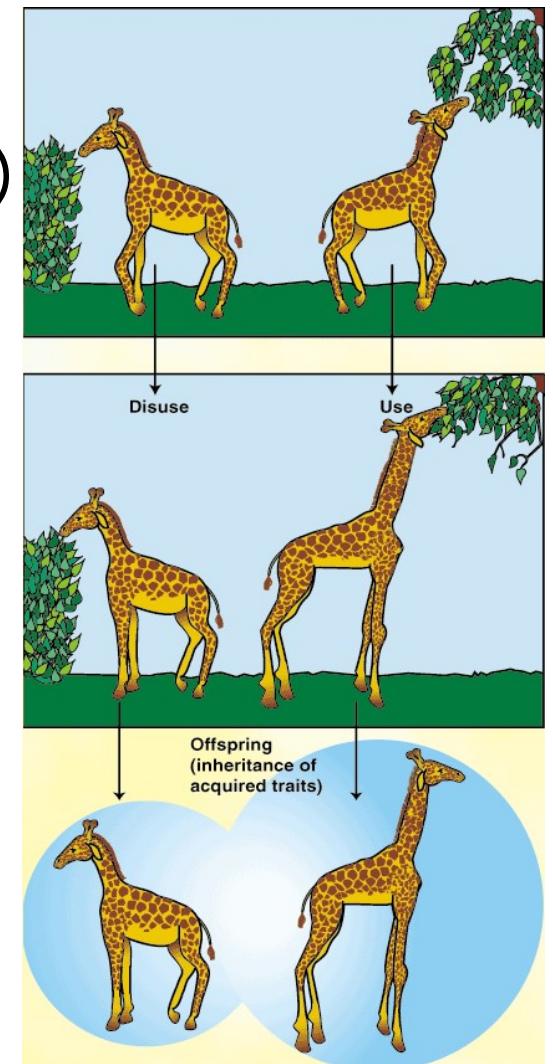
□ Care sunt caracteristicile de bază ale AE?

- Implică procese iterative și paralele
- Folosesc populații de potențiale soluții
- Se bazează pe o căutare aleatoare
- Sunt inspirați de biologie – implică mecanisme precum:
 - selecția naturală
 - reproducerea
 - recombinarea
 - mutația

Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

- ❑ Jean Baptise de Lamark (1744-1829)
 - A propus în 1809 o explicație pentru originea speciilor în cartea *Zoological Philosophy*:
 - ❑ Nevoile unui organism determină caracteristicile care evoluează
 - ❑ Caracteristicile utile dobândite în cursul vieții unui organism se pot transfera urmașilor acestuia
 - Legea utilizării și neutilizării
 - ❑ *use and disuse*



Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

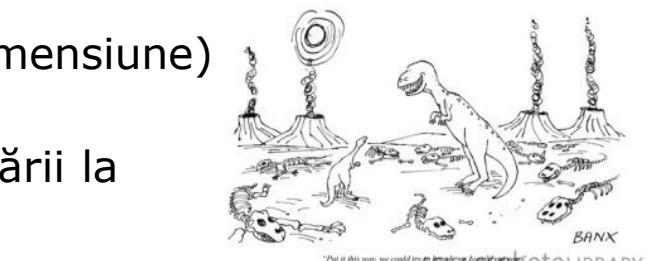
□ Charles Darwin (1807-1882)

- În cartea *Origin of Species* demostrează că toate organismele au evoluat din alte organisme pe baza:

- variației
 - supraproducția de descendenți



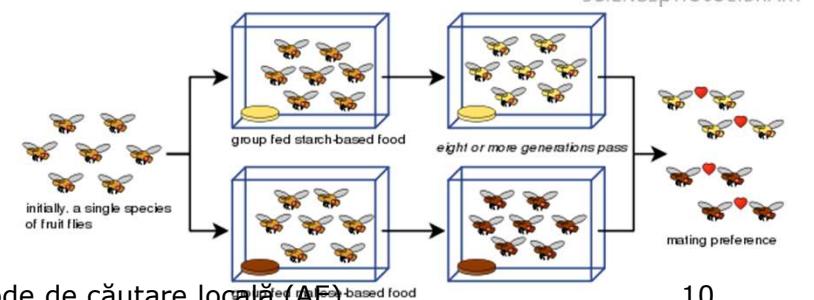
- selecției naturale
 - competiția (generații constante ca dimensiune)



- supraviețuirea pe baza calității/adaptării la mediul de viață (fitness)

- reproducerea

- apariția de specii noi



Inteligentă artificială - metode de căutare locală (AE)

Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

□ Teoria evolutivă modernă

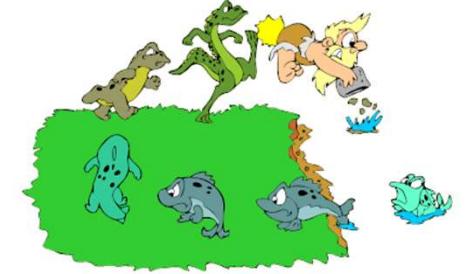
- Îmbogățește teoria Darwiniană cu mecanismul moștenirii genetice
- Variația genetică se produce prin:
 - mutație – spontană – și
 - reproducere sexuală
- L. Fogel 1962 (San Diego, CA) → programare evolutivă – PE – (*Evolutionary Programming*)
- J. Holland 1962 (Ann Arbor, MI) → algoritmi genetici – AG – (*Genetic Algorithms*)
- I. Rechenberg & H.-P. Schwefel 1965 (Berlin, Germany) → strategii evolutive – SE – (*Evolution Strategies*)
- J. Koza 1989 (Palo Alto, CA) → programare genetică – PG – (*Genetic Programming*)

Algoritmi evolutivi – aspecte teoretice

□ Metafora evolutivă

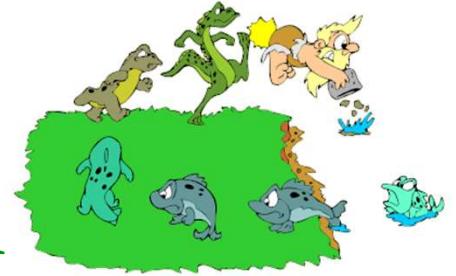
Evoluția naturală		Rezolvarea problemelor
Individ	↔	Soluție potențială
Populație	↔	Mulțime de soluții potențiale
Cromozom	↔	Codarea unei soluții potențiale
Genă	↔	Parte a codării
Fitness	↔	Calitate
Încrucișare și mutație	↔	Operatori de căutare
Mediu	↔	Problemă

Algoritmi evolutivi - algoritm



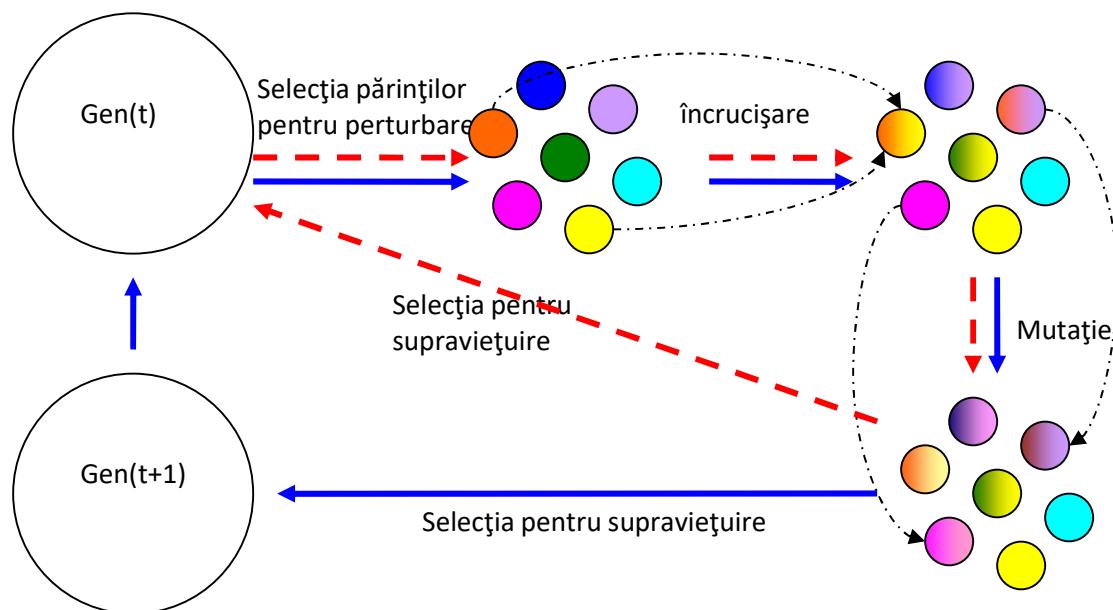
- Schema generală
- Proiectare

Algoritmi evolutivi – algoritm

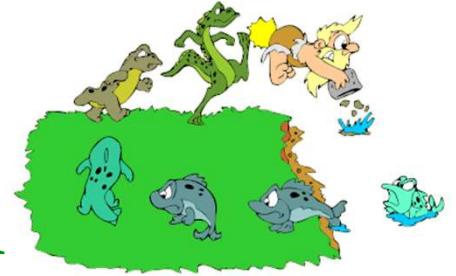


□ Schema generală a unui AE

- Generațional →
- Steady-state →



Algoritmi evolutivi – algoritm



□ Proiectare

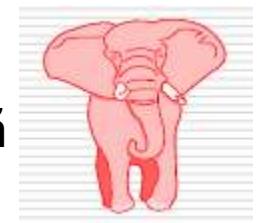
- Alegerea unei reprezentări a cromozomilor
- Alegerea unui model de populație
- Stabilirea unei funcții de evaluare
- Stabilirea operatorilor genetici
 - Selectie
 - Mutatie
 - Recombinare
- Stabilirea unui criteriu de stop

Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

□ 2 nivale de existență pentru o soluție candidat

■ Nivel exterior → fenotip

- Individ - obiectul original în contextul dat de problemă
- Aici are loc evaluarea unei potențiale soluții
- Furnică, rucsac, elefant, orașe, ...



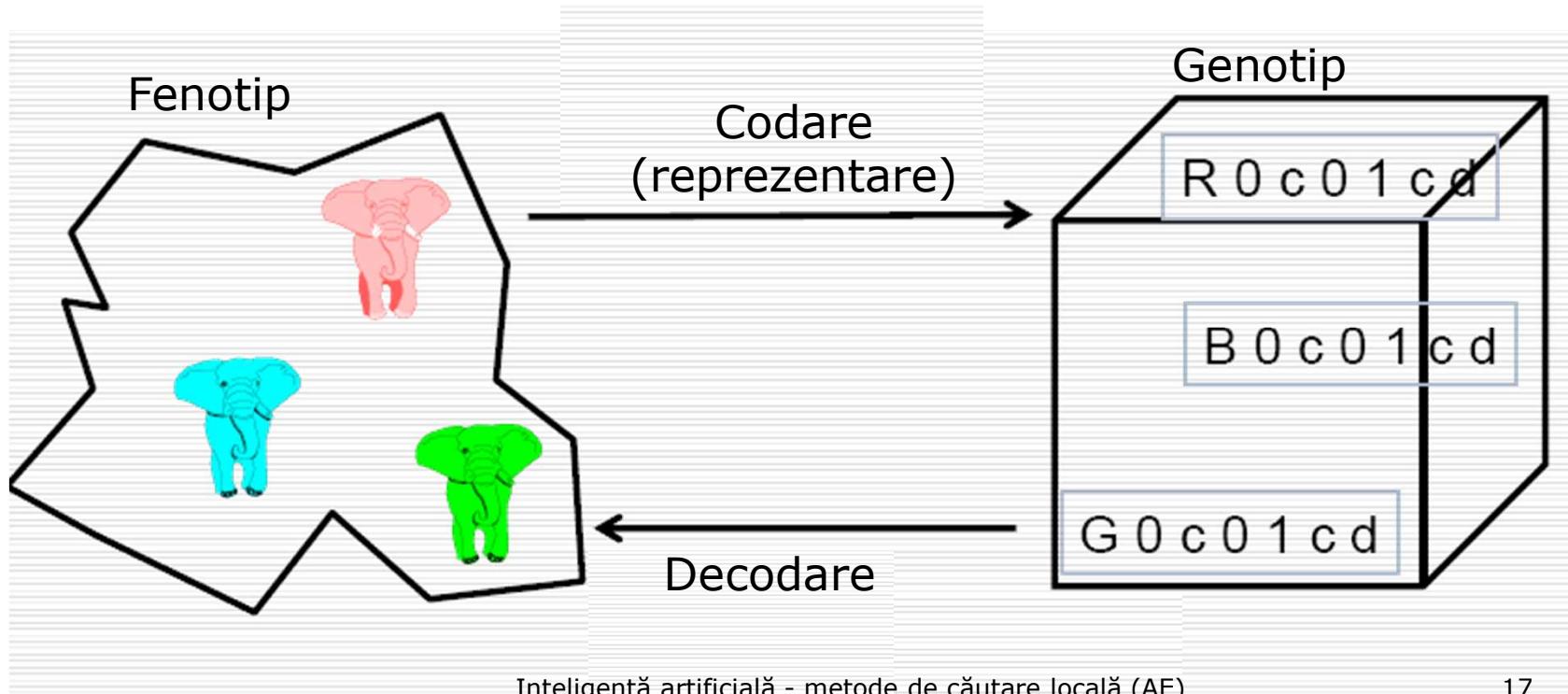
■ Nivel interior → genotip

- Cromozom – codul asociat unui obiect
 - format din gene, poziționate în locuri (fixe) – loci – și având anumite valori – alele
- Aici are loc căutarea unei noi potențiale soluții
- Vector unidimensional (numeric, boolean, string), matrice,
...



Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

- Reprezentarea trebuie să fie relevantă pentru:
 - problemă,
 - funcția de evaluare și
 - operatorii genetici



Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

Tipologia reprezentării cromozomilor

□ Liniară

■ Discretă

- Binară → problema rucsacului

- Ne-binară

- Întreagă

- Oarecare → procesarea imaginilor

- Permutări → problema comisului voiajor

- Categorială → problema colorării hărților

- Continuă (reală) → optimizări de funcții

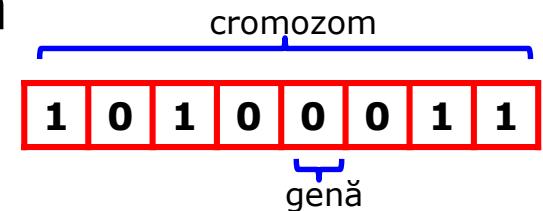
□ Arborescentă → probleme de regresie

Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

□ Reprezentare liniară discretă binară

■ Genotip

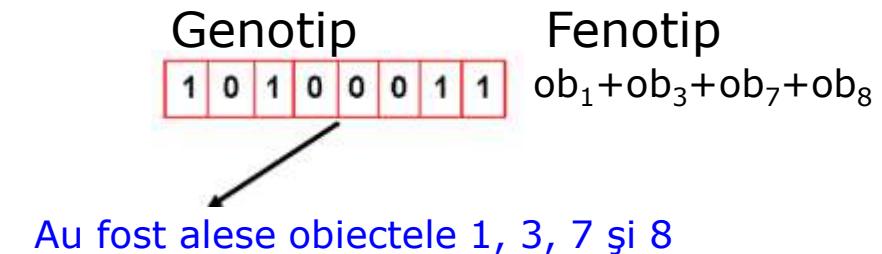
- sir de biți



Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

□ Reprezentare liniară discretă binară

- Genotip
 - sir de biți
- Fenotip
 - Elemente de tip Boolean
 - Ex. Problema rucsacului – obiectele alese pentru umplerea rucsacului



Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă binară
 - Genotip
 - sir de biți
 - Fenotip
 - Elemente de tip Boolean
 - Ex. Problema rucsacului – obiectele alese pentru umplerea rucsacului

- Numere întregi

Genotip	Fenotip $= 163$
1 0 1 0 0 0 1 1	

$$1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 =$$
$$128 + 32 + 2 + 1 = 163$$

Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă binară
 - Genotip
 - sir de biți
 - Fenotip
 - Elemente de tip Boolean
 - Ex. Problema rucsacului – obiectele alese pentru umplerea rucsacului
 - Numere întregi
 - Numere reale într-un anumit Interval (ex. [2.5, 20.5])

Genotip	Fenotip								
<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td style="padding: 2px;">1</td><td style="padding: 2px;">0</td><td style="padding: 2px;">1</td><td style="padding: 2px;">0</td><td style="padding: 2px;">0</td><td style="padding: 2px;">0</td><td style="padding: 2px;">1</td><td style="padding: 2px;">1</td></tr></table>	1	0	1	0	0	0	1	1	$= 13.9609$
1	0	1	0	0	0	1	1		

$$x = 2.5 + \frac{163}{256} (20.5 - 2.5) = 13.9609$$

Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

Transformarea valorilor reale reprezentate pe biți

- Fie $z \in [x,y] \subseteq \mathcal{R}$ reprezentat ca $\{a_1, \dots, a_L\} \in \{0,1\}^L$
- Funcția $[x,y] \rightarrow \{0,1\}^L$ trebuie să fie inversabilă (un fenotip corespunde unui genotip)
- Funcția $\Gamma: \{0,1\}^L \rightarrow [x,y]$ definește reprezentarea
$$\Gamma(a_1, \dots, a_L) = x + \frac{y-x}{2^L - 1} \cdot \left(\sum_{j=0}^{L-1} a_{L-j} \cdot 2^j \right) \in [x, y]$$
- Observații
 - Se pot reprezenta doar 2^L valori
 - L indică precizia maximă a soluției
 - Pentru o precizie cât mai bună \rightarrow cromozomi lungi \rightarrow evoluție încetinită

Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă ne-binară întreagă oarecare
 - Genotip
 - sir de numere întregi dintr-un anumit interval
 - Fenotip
 - Utilitatea numerelor în problemă
 - Ex. Problema plății unei sume folosind diferite monezi
 - Genotip → sir de nr întregi de lungime egală cu numărul de monezi diferite, fiecare număr din intervalul $[0, \text{suma/valoarea monezii curente}]$
 - Fenotip → câte monezi din fiecare tip trebuie considerate

Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă ne-binară întreagă de tip permutare
 - Genotip
 - Permutare de n numere (n – numărul de gene)
 - Fenotip
 - Utilitatea permutării în problemă
 - Ex. Problema comisului voiajor
 - Genotip → permutare de n elemente
 - Fenotip → ordinea de vizitare a orașelor, știind că fiecărui oraș îi corespunde un număr din mulțimea $\{1, 2, \dots, n\}$

Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă ne-binară categorială
 - Similară cu cea întreagă, dar în loc de numere se folosesc etichete
 - Genotip
 - sir de etichete dintr-o anumită mulțime
 - Fenotip
 - Interpretarea etichetelor
 - Ex. Problema colorării hărților
 - Genotip → sir de etichete (culori) de lungime egală cu numărul de țări, fiecare etichetă aparținând unei mulțimi de culori date
 - Fenotip → cu ce culoare trebuie hașurată fiecare hartă a unei țări

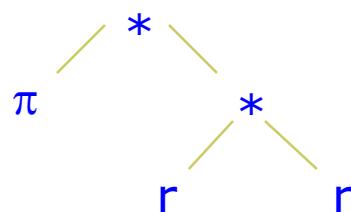
Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

- Reprezentare liniară continuă (reală)
 - Genotip
 - Sir de numere reale
 - Fenotip
 - Utilitatea numerelor în problemă
 - Ex. Problema optimizării funcțiilor $f: R^n \rightarrow R$
 - Genotip \rightarrow tuplu de numere reale $X=[x_1, x_2, \dots, x_n]$, $x_i \in R$
 - Fenotip \rightarrow valorile asociate argumentelor funcției f

Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

- Reprezentare arborescentă
 - Genotip
 - Arbori care codează S-Expresii
 - Nodurile interne ale arborelui → funcții (F)
 - Matematice
 - Operatori aritmetici
 - Operatori de tip Boolean
 - Instrucțiuni
 - Într-un limbaj de programare
 - Alt tip de instrucțiuni
 - Frunzele arborelui → terminale (T)
 - Valori reale sau Booleene, constante sau variabile
 - Subprograme
 - Fenotip
 - Interpretarea S-expresiilor
 - Ex. Calculul ariei unui cerc

$$\pi * r^2$$



Algoritmi evolutivi – algoritm Proiectare – formarea unei populații

□ Populație – concept

■ Scop

- reține o colecție de soluții candidat
 - se permit repetiții
- este folosită în întregime în procesul de selecție pentru reproducere

■ Proprietăți

- dimensiune (de obicei) fixă μ
- diversitate
 - Nr de fitness-uri/fenotipuri/genotipuri diferite

■ Observații

- Reprezintă unitatea de bază care evoluează
 - populația întreagă evoluează, nu indivizi!!!

Algoritmi evolutivi – algoritm Proiectare – formarea unei populații

- Populație – inițializare
 - Uniformă (dacă e posibil) în spațiul de căutare
 - Stringuri binare
 - generarea de 0 și 1 cu probabilitatea 0.5
 - Siruri de numere reale generate uniform (într-un anumit interval)
 - Permutări
 - generarea permutării identice și efectuarea unor schimbări

Algoritmi evolutivi – algoritm Proiectare – formarea unei populații

□ Populație – inițializare

- Uniformă (dacă e posibil) în spațiul de căutare
 - Arbori
 - Metoda *Full* – arbori compleți
 - Nodurile de la adâncimea $d < D_{\max}$ se inițializează aleator cu o funcție din setul de funcții F
 - Nodurile de la adâncimea $d = D_{\max}$ se inițializează aleator cu un terminal din setul de terminale T
 - Metoda *Grow* – arbori incompleți
 - Nodurile de la adâncimea $d < D_{\max}$ se inițializează aleator cu un element din $F \cup T$
 - Nodurile de la adâncimea $d = D_{\max}$ se inițializează aleator cu un terminal din setul de terminale T
 - Metoda *Ramped half and half*
 - $\frac{1}{2}$ din populație se creează cu metoda *Full*
 - $\frac{1}{2}$ din populație se creează cu metoda *Grow*
 - Folosind diferite adâncimi

Algoritmi evolutivi – algoritm Proiectare – formarea unei populații

- Modele de populații – algoritm evolutiv:
 - Generațional
 - În fiecare generație se crează μ descendenți
 - Fiecare individ supraviețuiește o singură generație
 - Mulțimea părinților este înlocuită în întregime cu mulțimea descendenților
 - Steady-state
 - În fiecare generație se obține un singur descendant
 - Un singur părinte (cel mai slab) este înlocuit cu descendantul obținut
- Discrepanța între generații (*Generation Gap*)
 - Proporția populației înlocuite
 - $1 = \mu/\mu$, pentru modelul generațional
 - $1/\mu$, pentru modelul steady-state

Algoritmi evolutivi – algoritm Proiectare – funcția de evaluare

- Scop
 - Reflectă condițiile la care trebuie să se adapteze populația
 - Funcție de calitate sau funcție obiectiv
 - Asociază o valoare fiecărei soluții candidat
 - Consecințe asupra selecției → cu cât sunt mai multe valori diferite, cu atât e mai bine
- Proprietăți
 - Etapa cea mai costisitoare
 - Nu se re-evaluează indivizii nemodificați
- Tipologie:
 - După nr de obiective urmărite:
 - Uni-obiectiv
 - Multi-obiectiv → fronturi Pareto
 - După direcția optimizării
 - De maximizat
 - De minimizat
 - După gradul de exactitate
 - Exactă
 - Euristică

Algoritmi evolutivi – algoritm Proiectare – funcția de evaluare

- Exemple
 - Problema rucsacului
 - reprezentare → liniară discretă binară
 - fitness → $\text{abs}(\text{greutatea rucsacului} - \text{greutatea obiectelor alese}) \rightarrow$ minimizare
 - Problema plății unei sume folosind diferite monezi
 - reprezentare → liniară discretă întreagă
 - fitness → $\text{abs}(\text{suma de plată} - \text{suma monezilor selectate}) \rightarrow$ minimizare
 - Problema comisului voiaior
 - reprezentare → liniară discretă întreagă sub formă de permutare
 - fitness → costul drumului parcurs → minimizare
 - Problema optimizării funcțiilor
 - Reprezentare → liniară continuă reală
 - fitness → valoarea funcției → minimizare/maximizare
 - Calculul ariei unui cerc
 - reprezentare → arborescentă
 - fitness → suma pătratelor erorilor (diferențelor între valoarea reală și cea calculată pe un set de exemple) → minimizare

Algoritmi evolutivi – algoritm Proiectare – selecția



- Scop:
 - acordă şanse de reproducere/supravieţuire mai mari indivizilor mai buni
 - şi indivizii mai slabii trebuie să aibă şansa să se reproducă/supravieţuiască pentru că pot conţine material genetic util
 - direcţionează populaţia spre îmbunătăţirea calităţii
- Proprietăţi
 - lucrează la nivel de populaţie
 - se bazează doar pe fitnessul indivizilor (este independentă de reprezentare)
 - ajută la evadarea din optimele locale datorită naturii sale stocastice

Algoritmi evolutivi – algoritm Proiectare – selecția



□ Tipologie

- În funcție de scop:
 - ▣ Selectia părintilor (din generația curentă) pentru reproducere
 - ▣ Selectia supraviețuitorilor (din părinti și descendenți) pentru generația următoare
- În funcție de modul de decizie al câștigătorului
 - ▣ Deterministă – cel mai bun câștigă
 - ▣ Stocastică – cel mai bun are cele mari sanse să câștige
- În funcție de mecanism
 - ▣ Selectia pentru reproducere
 - Selectie proporțională (bazată pe fitness)
 - Selectie bazată pe ranguri
 - Selectie prin turnir ----> Bazată pe o parte din populație
 - ▣ Selectia pentru supraviețuire
 - Bazată pe vîrstă
 - Bazată pe calitate (fitness)

Algoritmi evolutivi – algoritm Proiectare – selectia pt. reproducere



■ Selectie proportională (bazată pe fitness) – SP

□ Ideea de bază

- Algoritmul ruletei la nivelul întregii populații
- Estimarea numărului de copii ale unui individ

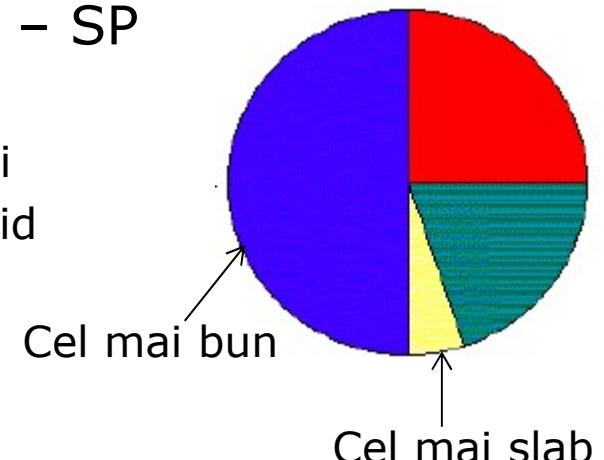
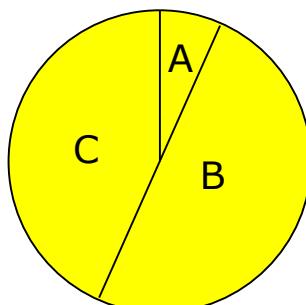
$$E(n_i) = \mu \frac{f(i)}{\langle f \rangle}, \text{ unde:}$$

- μ = dimensiunea populației,
- $f(i)$ = fitnessul individului i ,
- $\langle f \rangle$ = fitnessul mediu al populației

□ Indivizii mai buni

- au alocat mai mult spațiu în ruletă
- au șanse mai mari să fie selectați

□ Ex. O populație cu $\mu = 3$ indivizi



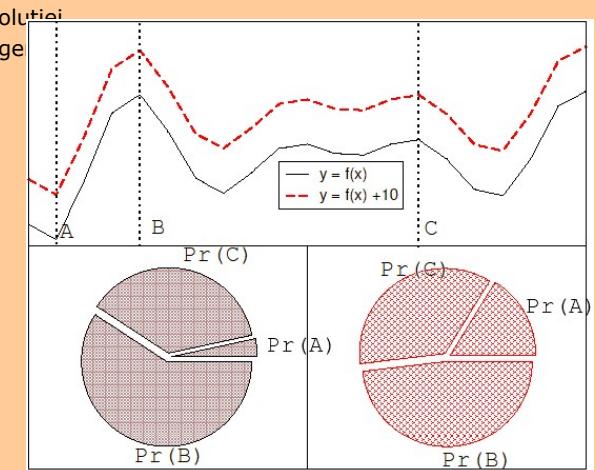
	$f(i)$	$P_{selSP}(i)$
A	1	$1/10=0.1$
B	5	$5/10=0.5$
C	4	$4/10=0.4$
Suma	10	1

Algoritmi evolutivi – algoritm Proiectare – selecția pt. reproducere



Selectie proporțională (bazată pe fitness)

- Avantaje
 - Algoritm simplu
- Dezavantaje
 - Convergența prematură
 - cromozomii foarte buni tend să domine populația
 - Presiune de selecție foarte mică atunci când fintessurile indivizilor sunt foarte apropiate (la sfârșitul rulării)
 - Susceptibilă de traspoziția funcției
 - Rezultatele reale ale unei astfel de selecții diferă de distribuția probabilistică teoretică
 - Lucrează cu întreaga populație
- Soluții
 - scalarea fitnessului
 - Windowing
 - $f'(i) = f(i) - \beta^t$, unde β este un parametru care depinde de istoria recentă a evoluției
 - ex. β este fitnessul celui mai slab individ din populația curentă (a t -a generație)
 - Scalare de tip sigma (de tip Goldberg)
 - $f'(i) = \max\{f(i) - (\langle f \rangle - c * \sigma_f), 0.0\}$, unde:
 - c este o constantă (de obicei 2)
 - $\langle f \rangle$ - fitnessul mediu al populației
 - σ_f - deviația standard a fitnessului populației
 - Scalare prin normalizare
 - Se începe cu fitnessurile absolute (inițiale)
 - Se standardizează astfel încât se aiustează fitnessurile a.î.:
 - ele să aparțină $[0,1]$
 - cel mai bun fitness să fie cel mai mic (egal cu 0)
 - suma lor să fie 1
 - alt mecanism de selecție



Algoritmi evolutivi – algoritm Proiectare – selecția pt. reproducere



□ Selecția bazată pe ranguri – SR

■ Ideea de bază

- Se ordonează întreaga populație pe baza fitnessului
 - Crește puțin complexitatea algoritmului, dar se poate neglija această creștere comparativ cu timpul necesar evaluării unui individ
- Se acordă ranguri fiecărui individ
- Se calculează probabilitățile de selecție pe baza rangurilor
 - Cel mai slab individ are rangul 1
 - Cel mai bun individ are rangul μ
- Încearcă să rezolve problemele selecției proporționale prin folosirea fitnessurilor relative (în locul celor absolute)

Algoritmi evolutivi – algoritm Proiectare – selecția pt. reproducere



□ Selecția bazată pe ranguri – SR

■ Modalități de acordare a rangurilor

□ Liniară (RL) $P_{lin_rank}(i) = \frac{2-s}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$

- s – presiunea de selecție
 - măsoară avantajele celui mai bun individ
 - $1.0 < s \leq 2.0$
 - în algoritmul genetic generațional s este numărul de copii ai unui individ
- Ex. pentru o populație cu $\mu = 3$ indivizi

	f(i)	$P_{selSP}(i)$	Rang	$P_{selRL}(i)$ pt. $s=2$	$P_{selRL}(i)$ pt. $s=1$
A	1	1/10=0.1	1	0.33	0.33
B	5	5/10=0.5	3	1.00	0.33
C	4	4/10=0.4	2	0.67	0.33
Suma	10	1			

□ Exponențială (RE) $P_{exp_rank}(i) = \frac{1-e^{-i}}{c}$

- Cel mai bun individ poate avea mai mult de 2 copii
- c – factor de normalizare
 - depinde de dimensiunea populației (μ)
 - trebuie ales a.î. suma probabilităților de selecție să fie 1

Algoritmi evolutivi – algoritm Proiectare – selecția pt. reproducere



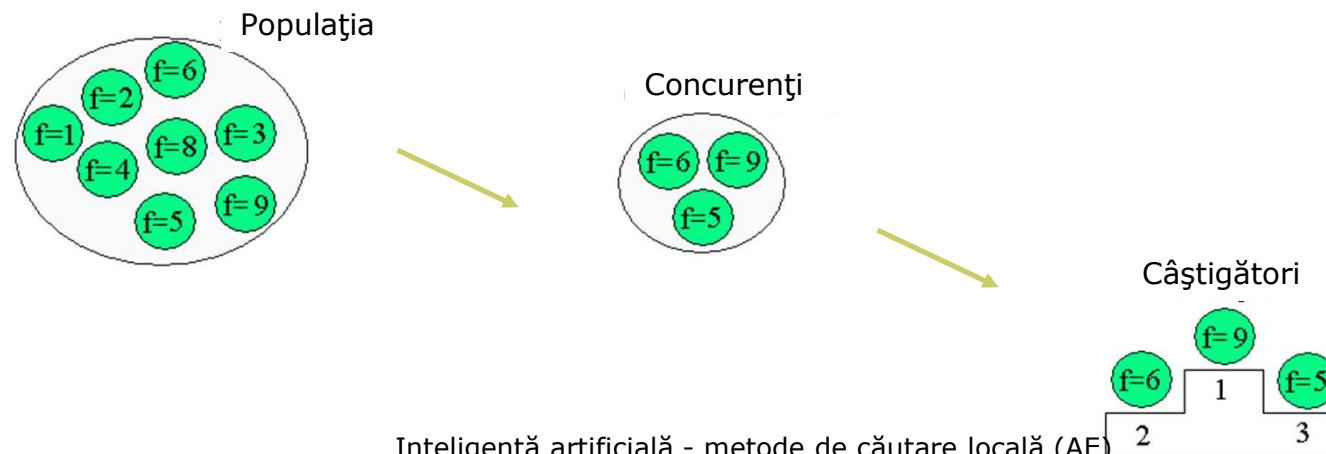
- Selecția bazată pe ranguri – SR
 - Avantaje
 - Păstrează presiunea de selecție constantă
 - Dezavantaje
 - Lucrează cu întreaga populație
 - Soluții
 - Alt mecanism de selecție

Algoritmi evolutivi – algoritm Proiectare – selecția pt. reproducere



□ Selectia prin turnir

- Ideea de bază
 - Se aleg aleator k indivizi \rightarrow eșantion de k indivizi (k – mărimea turnirului)
 - Se selectează cel mai bun individ dintre cei aleși anterior
 - Probabilitatea alegерii unui individ în eșantion depinde de
 - Rangul individualului
 - Dimensiunea eșantionului (k)
 - Cu cât k este mai mare, cu atât crește și presiunea de selecție
 - Modul în care se face alegerea – dacă se realizează cu înlocuire (model steady-state) sau nu
 - Alegerea fără înlocuire crește presiunea de selecție
 - Pt $k = 2$ timpul necesar ca cel mai bun individ să domine populația este același cu cel de la selecția pe bază de ranguri liniare cu $s = 2 * p$, p – probabilitatea alegерii celui mai bun individ din populație



Algoritmi evolutivi – algoritm Proiectare – selecția pt. reproducere



□ Selecția prin turnir

■ Avantaje

- Nu implică lucrul cu întregă populație
- Ușor de implementat
- Ușor de controlat presiunea de selecție prin intermediul parametrului k

■ Dezavantaje

- Rezultatele reale ale unei astfel de selecții diferă de distribuția probabilistică teoretică (similar selecției prin mecanismul ruletei)

Algoritmi evolutivi

Proiectare – selecția



□ Selecția pentru supraviețuire (înlocuire)

- Pe baza vârstei
 - eliminarea celor mai "bătrâni" indivizi
- Pe baza calității (fitness-ului)
 - selecției proporțională
 - selecție bazată pe ranguri
 - selecție prin turnir
 - elitism
 - Păstrarea celor mai buni indivizi de la o generație la alta (dacă descendenții sunt mai slabii ca părinții se păstrează părinții)
 - GENITOR (înlocuirea celui mai slab individ)
 - Eliminarea celor mai slabii λ indivizi

Algoritmi evolutivi - algoritm Proiectare – operatori de variație



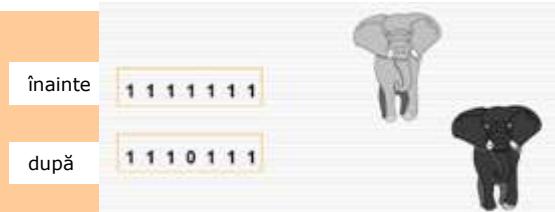
- Scop:
 - Generarea unor soluții potențiale noi
- Proprietăți
 - lucrează la nivel de individ
 - se bazează doar pe reprezentarea indivizilor (independent de fitness)
 - Aiută la explorarea și exploatarea spațiului de căutare
 - Trebuie să producă indivizi valizi
- Tipologie
 - În funcție de aritate
 - Aritate 1 → operatori de mutație
 - Aritate > 1 → operatori de recombinare/încrucișare

Algoritmi evolutivi – algoritm Proiectare – mutația



- Scop
 - Reintroducerea în populație a materialului genetic pierdut
 - Operator unar de căutare (spațiul continuu)
 - Introducerea diversității în populație (în spațiul discret – binar)

- Proprietăți
 - Acționează la nivel de genotip
 - Bazată pe elemente aleatoare
 - Responsabilă cu explorarea unor noi regiuni promițătoare ale spațiului de căutare
 - Este responsabilă de evadarea din optimele locale
 - Trebuie să producă mici schimbări stocastice ale individului
 - Mărimea mutației trebuie să fie controlabilă
 - Se produce cu o anumită probabilitate (p_m) la nivelul fiecărei gene a unui cromozom



Algoritmi evolutivi – algoritm Proiectare – mutația



□ Tipologie

- Reprezentare binară
 - Mutație tare - bit-flipping
 - Mutație slabă
- Reprezentare întreagă
 - Random resetting
 - Creep mutation
- Reprezentare permutare
 - Mutație prin inserție
 - Mutație prin interchimpare
 - Mutație prin inversare
 - Mutație prin amestec
 - Mutație k-opt
- Reprezentare reală
 - Mutație uniformă
 - Mutație neuniformă
 - Mutație Gaussiană
 - Mutație Cauchy
 - Mutație Laplace
- Reprezentare arborescentă → într-un curs viitor
 - Mutație grow
 - mutație shrink
 - Mutație switch
 - Mutație cycle
 - Mutație tip Koza
 - Mutație pentru terminalele numerice

Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. binară)



- ❑ Un cromozom $c = (g_1, g_2, \dots, g_L)$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in \{0, 1\}$, pt. $i=1, 2, \dots, L$
- ❑ Mutație tare – *bit flipping*
 - Ideea de bază
 - ❑ Schimbarea cu probabilitatea p_m (rată de mutație) a unor gene în complementul lor
 - $1 \rightarrow 0$
 - $0 \rightarrow 1$
 - ❑ Ex. Un cromozom cu $L = 8$ gene, $p_m = 0.1$



Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. binară)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in \{0,1\}$, pt. $i=1,2,\dots,L$

□ Mutație slabă

■ Ideea de bază

- Schimbarea cu probabilitatea p_m (rată de mutație) a unor gene în 0 sau 1

- $1 \rightarrow 0/1$
 - $0 \rightarrow 1/0$

- Ex. Un cromozom cu $L = 8$ gene, $p_m = 0.1$



Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. întreagă)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in \{val_1, val_2, \dots, val_k\}$, pt. $i=1,2,\dots,L$
- Mutație *random resetting*
 - Ideea de bază
 - Valoarea unei gene este schimbată (cu probabilitatea p_m) într-o altă valoare (din setul de valori posibile)



Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. întreagă)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in \{val_1, val_2, \dots, val_k\}$, pt. $i=1,2,\dots,L$
- Mutație *creep*
 - Ideea de bază
 - Valoarea unei gene este schimbată (cu probabilitatea p_m) prin adăugarea unei valori (pozitivă sau negativă)
 - valoarea → face parte dintr-o distribuție simetrică față de zero
 - modificarea produsă este fină (mică)



Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. permutare)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ cu $g_i \neq g_i'$ pentru orice $i \neq i'$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in \{\text{val}_1, \text{val}_2, \dots, \text{val}_L\}$, pt. $i = 1, 2, \dots, L$ a.î. $g'_i \neq g_i'$ pentru orice $i \neq i'$.
- Mutație prin interschimbare (*swap mutation*)
 - Ideea de bază
 - Se aleg aleator 2 gene și se interschimbă valorile lor



Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. permutare)



- Un cromozom $c=(g_1, g_2, \dots, g_L)$ cu $g_i \neq g_j$ pentru orice $i \neq j$ devine $c'=(g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in \{\text{val}_1, \text{val}_2, \dots, \text{val}_L\}$, pt. $i=1, 2, \dots, L$ a.î. $g'_i \neq g_j$ pentru orice $j \neq i$.

□ Mutăție prin inserție

■ Ideea de bază

- Se aleg 2 gene oarecare g_i și g_j cu $j > i$
- Se inserează g_j după g_i a.î. $g'_i = g_i, g'_{i+1} = g_j, g'_{k+2} = g_{k+1}, \dots$



Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. permutare)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ cu $g_i \neq g_i'$ pentru orice $i \neq i'$ devine $c' = (g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{\text{val}_1, \text{val}_2, \dots, \text{val}_L\}$, pt. $i = 1, 2, \dots, L$ a.î. $g_i' \neq g_i$ pentru orice $i \neq i'$.
- Mutăție prin inversare
 - Ideea de bază
 - Se aleg aleator 2 gene și se inversează ordinea genelor situate între ele (substringul dintre gene)



Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. permutare)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ cu $g_i \neq g_i'$ pentru orice $i \neq i'$ devine $c' = (g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{\text{val}_1, \text{val}_2, \dots, \text{val}_L\}$, pt. $i = 1, 2, \dots, L$ a.î. $g_i' \neq g_i'$ pentru orice $i \neq i'$.
- Mutație prin amestec (*scramble mutation*)
 - Ideea de bază
 - Se alege aleator un subșir (continuu sau discontinuu) de gene și se rearanjează acele gene



Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. permutare)

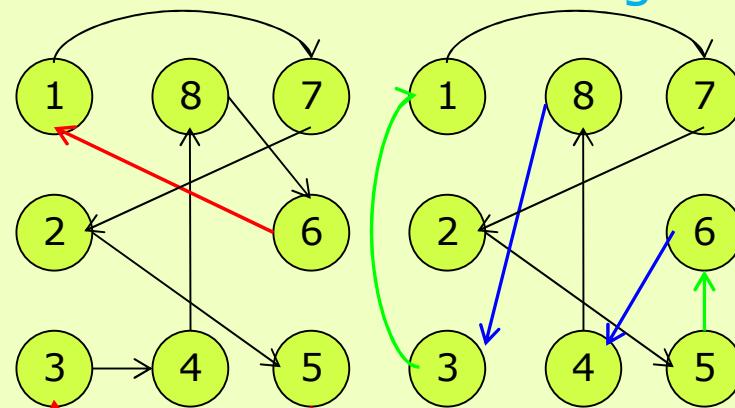
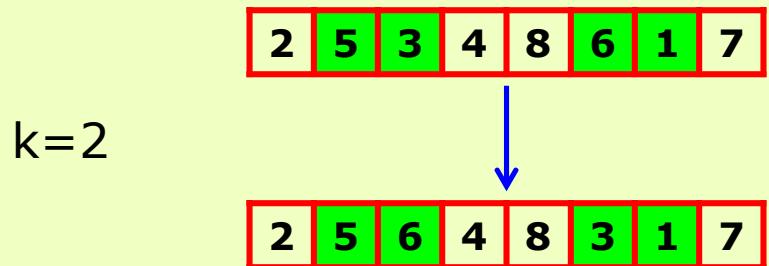


- Un cromozom $c = (g_1, g_2, \dots, g_L)$ cu $g_i \neq g_i'$ pentru orice $i \neq i'$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in \{\text{val}_1, \text{val}_2, \dots, \text{val}_L\}$, pt. $i = 1, 2, \dots, L$ a.î. $g'_i \neq g_i'$ pentru orice $i \neq i'$.

□ Mutăție k-opt

■ Ideea de bază

- Se aleg 2 substringuri disjuncte și de lungime k
- Se interchimbă 2 elemente ale acestor substringuri de gene



Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. reală)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Mutație uniformă
 - Ideea de bază
 - g'_i este schimbată cu probabilitatea p_m la o valoare aleasă aleator uniform din $[LI_i, LS_i]$

Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. reală)



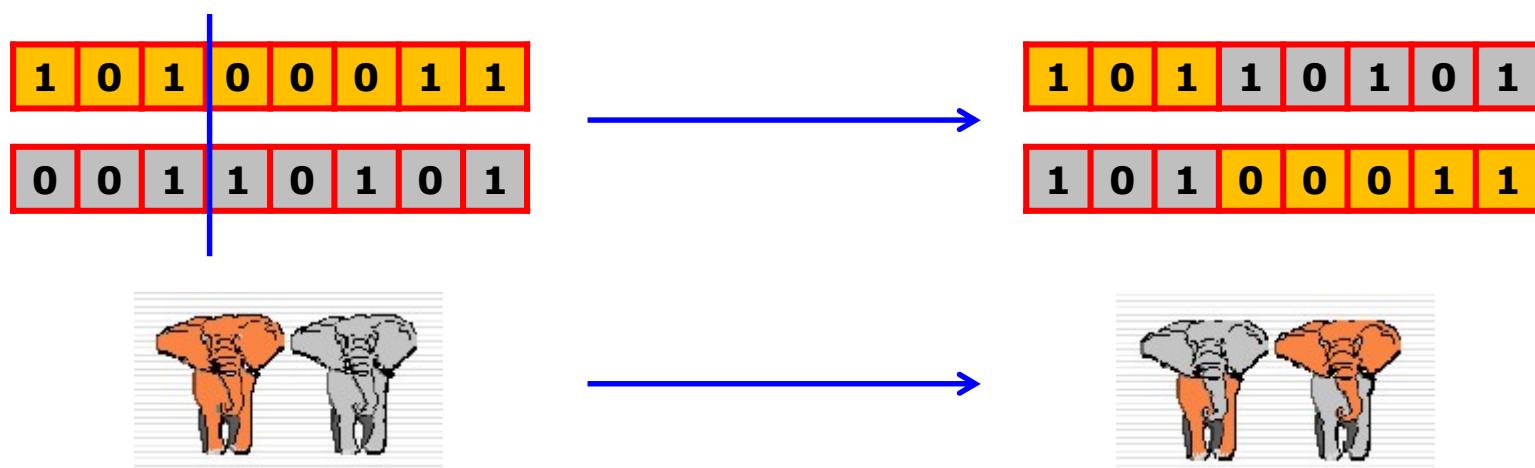
- Un cromozom $c = (g_1, g_2, \dots, g_L)$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Mutație neuniformă
 - Ideea de bază
 - Valoarea unei gene este schimbată (cu probabilitatea p_m) prin adăugarea unei valori (pozitivă sau negativă)
 - valoarea → face parte dintr-o distribuție
 - $N(\mu, \sigma)$ (Gaussiană) cu $\mu = 0$
 - Cauchy (x_0, γ)
 - Laplace (μ, b)
 - și readusă la $[LI_i, LS_i]$ (dacă este necesar) - *clamping*

Algoritmi evolutivi – algoritm Proiectare - recombinarea



- Scop
 - Amestecarea informațiilor preluate din părinți

- Proprietăți
 - Descendentul trebuie să moștenească ceva de la fiecare dintre părinți
 - Alegerea informațiilor care se amestecă este aleatoare
 - Operator de exploatare probabilistică (p_c) a spațiilor deja descoperite
 - Descendenții pot să fie mai buni, la fel de buni sau mai slabi decât părinții lor
 - Efectele sale se reduc pe măsură ce căutarea converge



Algoritmi evolutivi – algoritm Proiectare - recombinarea

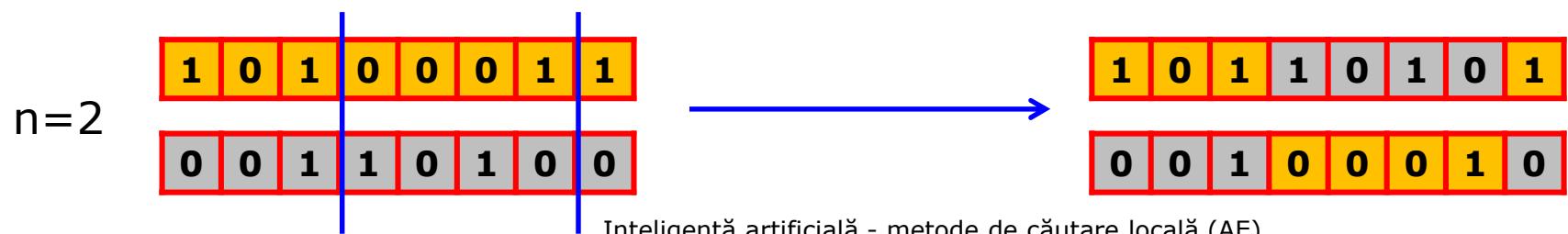


- Tipologie – în funcție de reprezentarea indivizilor
 - Reprezentare binară și întreagă
 - Cu puncte de tăietură
 - Uniformă
 - Reprezentare cu permutări
 - Încrucișare prin ordonare (versiunea 1 și versiunea 2)
 - Încrucișare transformată parțial (Partially Mapped Crossover)
 - Încrucișare ciclică
 - Încrucișare bazată pe legături (muchii)
 - Reprezentare reală
 - Discretă
 - Intermediară (aritmetică)
 - Aritmetică singulară
 - Aritmetică simplă
 - Aritmetică completă
 - Geometrică
 - Încrucișare amestecată
 - Încrucișare binară simulată
 - Reprezentare cu arbori
 - Încrucișare de sub-arbori → într-un curs viitor

Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. binară și întreagă)



- ❑ Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in \{0,1\} / \{\text{val}_1, \text{val}_2, \dots, \text{val}_k\}$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare cu n puncte de tăietură
 - Ideea de bază
 - ❑ Se aleg n puncte de tăietură ($n < L$)
 - ❑ Se taie cromozomii părinți prin aceste puncte
 - ❑ Se lipesc părțile obținute, alternând părinții



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. binară și întreagă)



■ Încrucișare cu n puncte de tăietură

■ Proprietăți

- Media valorilor codate de părinți = media valorilor codate de descendenți
 - Ex. Reprezentarea binară pe 4 biți a numerelor întregi – XO cu $n = 1$ după bitul 2
 - $p_1 = (1,0,1,0)$, $p_2 = (1,1,0,1)$
 - $d_1 = (1,0,0,1)$, $d_2 = (1,1,1,0)$
 - $\text{val}(p_1) = 10$, $\text{val}(p_2) = 13 \Rightarrow (\text{val}(p_1) + \text{val}(p_2))/2 = 23/2=11.5$
 - $\text{val}(d_1) = 9$, $\text{val}(d_2) = 14 \Rightarrow (\text{val}(d_1) + \text{val}(d_2))/2 = 23/2=11.5$
 - Ex. Reprezentare binară pe 4 biți pentru problema rucsacului de capacitate K = 10 cu 4 obiecte de greutate și valoare ((2,7), (1,8), (3,1), (2,3))
 - $p_1 = (1,0,1,0)$, $p_2 = (1,1,0,1)$
 - $d_1 = (1,0,0,1)$, $d_2 = (1,1,1,0)$
 - $\text{val}(p_1) = 8$, $\text{val}(p_2) = 18 \Rightarrow (\text{val}(p_1) + \text{val}(p_2))/2 = 26/2=13$
 - $\text{val}(d_1) = 10$, $\text{val}(d_2) = 16 \Rightarrow (\text{val}(d_1) + \text{val}(d_2))/2 = 26/2=13$
- Probabilitatea apariției unui factor de răspândire $\beta \approx 1$ este mai mare decât probabilitatea oricărui alt factor

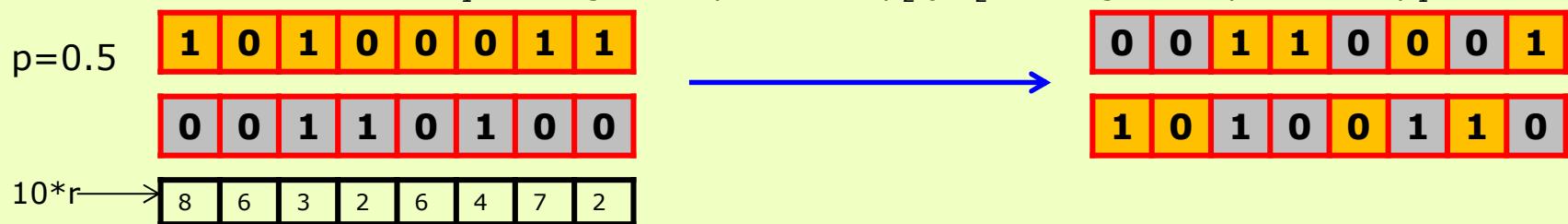
$$\beta = \left| \frac{\text{val}(d_1) - \text{val}(d_2)}{\text{val}(p_1) - \text{val}(p_2)} \right|$$

- Încrucișare prin contracție $\beta < 1$
 - Valorile descendenților se află între valorile părinților
- Încrucișare prin extensie $\beta > 1$
 - Valorile părinților se află între valorile descendenților
- Încrucișare staționară $\beta = 1$
 - Valorile descendenților coincid cu valorile părinților

Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. binară și întreagă)



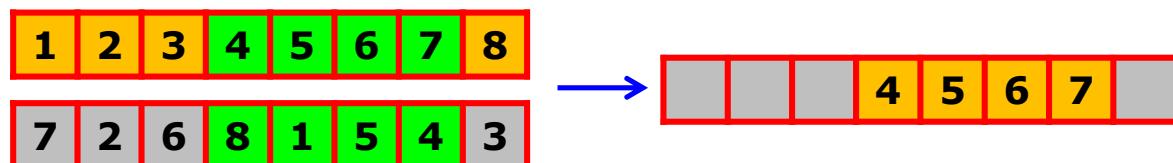
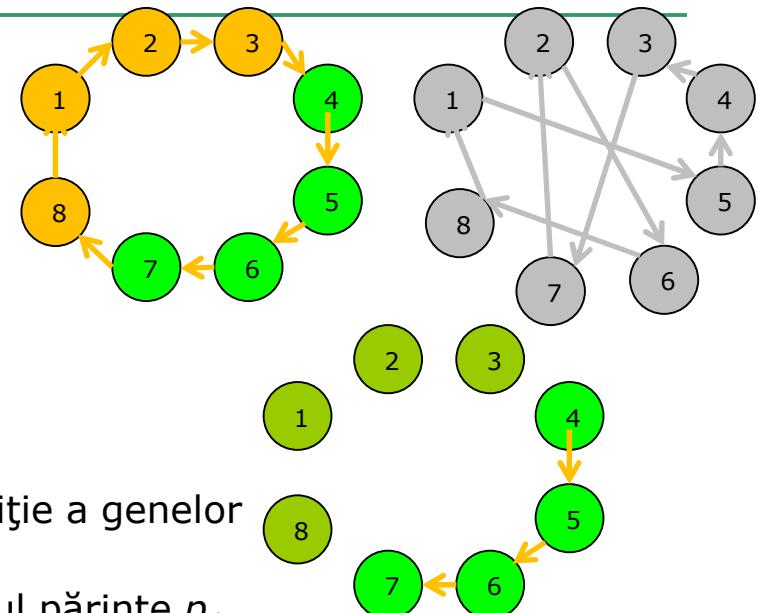
- ❑ Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in \{0,1\} / \{\text{val}_1, \text{val}_2, \dots, \text{val}_k\}$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare uniformă
 - Ideea de bază
 - ❑ Fiecare genă a unui descendant provine dintr-un părinte ales aleator și uniform:
 - Pentru fiecare genă în parte se generează un număr aleator r care respectă legea uniformă
 - Dacă numărul generat $r <$ probabilitatea p (de obicei $p=0.5$), c_1 va lua gena respectivă din p_1 și c_2 va lua gena respectivă din p_2 ,
 - Altfel c_1 va lua gena respectivă din p_2 și c_2 va lua gena respectivă din p_1



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)



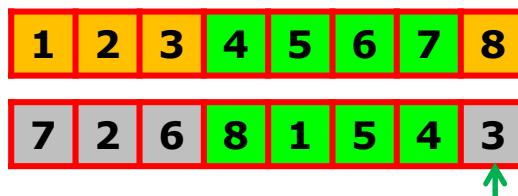
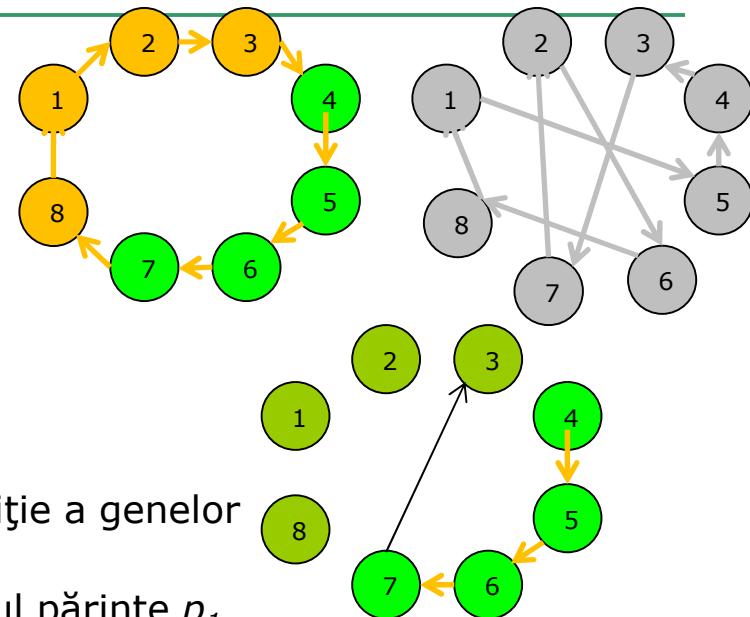
- ❑ Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obtin 2 descendenti
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare ordonată
 - Ideea de bază
 - Descendenții păstrează ordinea de apariție a genelor părintilor
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentalul d_1 (pe poziții corespondente)



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)



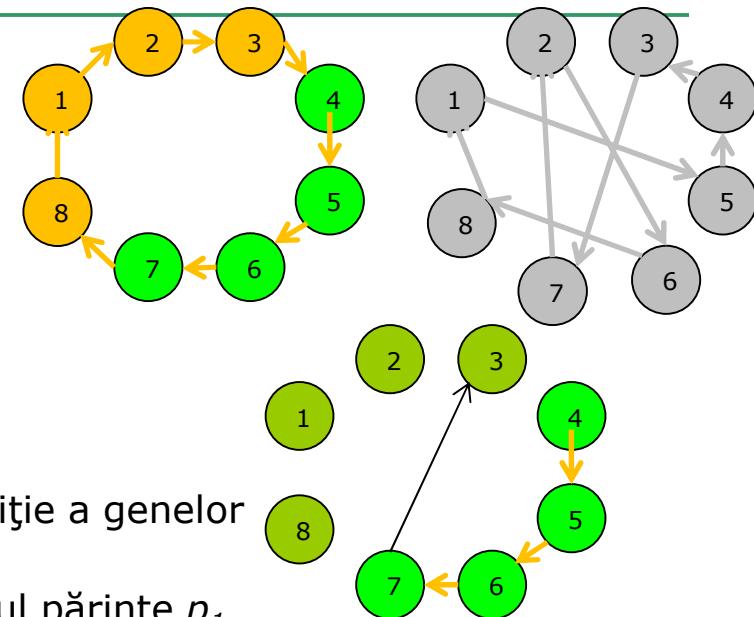
- ❑ Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obtin 2 descendenti
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare ordonată
 - Ideea de bază
 - ❑ Descendenții păstrează ordinea de apariție a genelor părintilor
 - ❑ Se alege un substring de gene din primul părinte p_1
 - ❑ Se copiază substringul din p_1 în descendental d_1 (pe poziții corespondente)
 - ❑ Se copiază genele din p_2 în descendental d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)



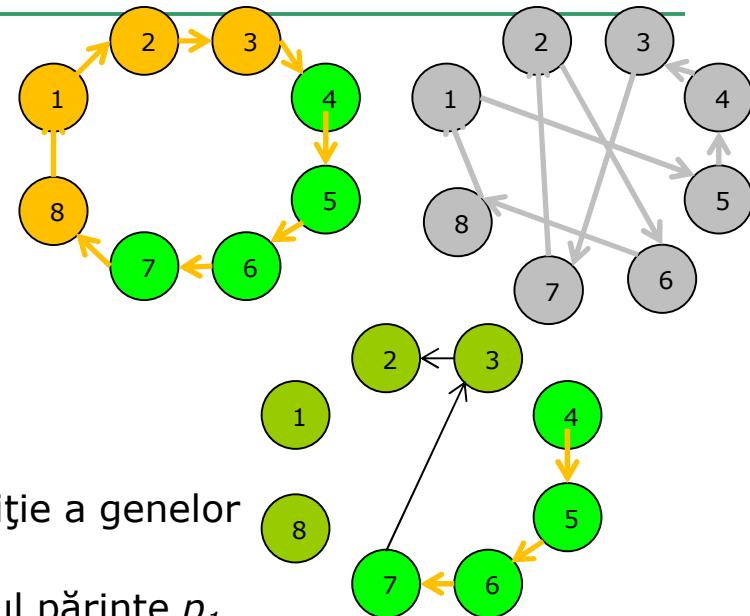
- ❑ Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obtin 2 descendenti
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare ordonată
 - Ideea de bază
 - ❑ Descendenții păstrează ordinea de apariție a genelor părintilor
 - ❑ Se alege un substring de gene din primul părinte p_1
 - ❑ Se copiază substringul din p_1 în descendental d_1 (pe poziții corespondente)
 - ❑ Se copiază genele din p_2 în descendental d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)



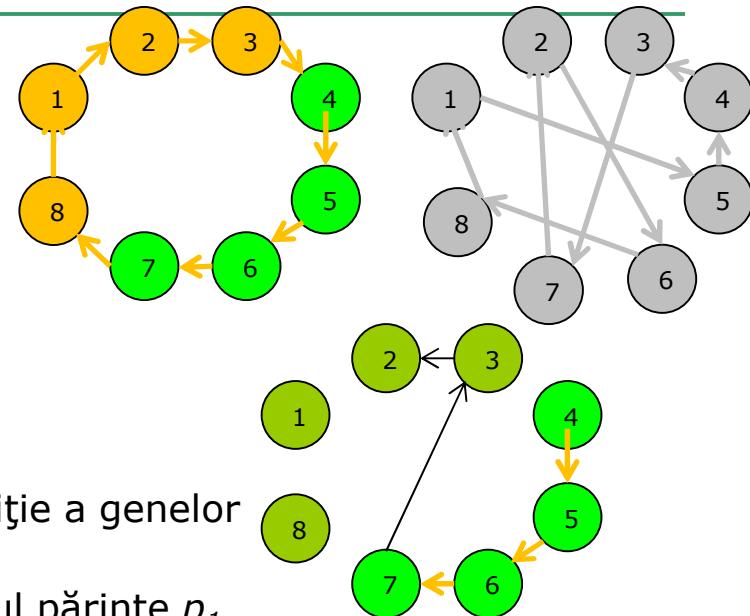
- ❑ Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obtin 2 descendenti
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare ordonată
 - Ideea de bază
 - ❑ Descendenții păstrează ordinea de apariție a genelor părintilor
 - ❑ Se alege un substring de gene din primul părinte p_1
 - ❑ Se copiază substringul din p_1 în descendentalul d_1 (pe poziții corespondente)
 - ❑ Se copiază genele din p_2 în descendentalul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)



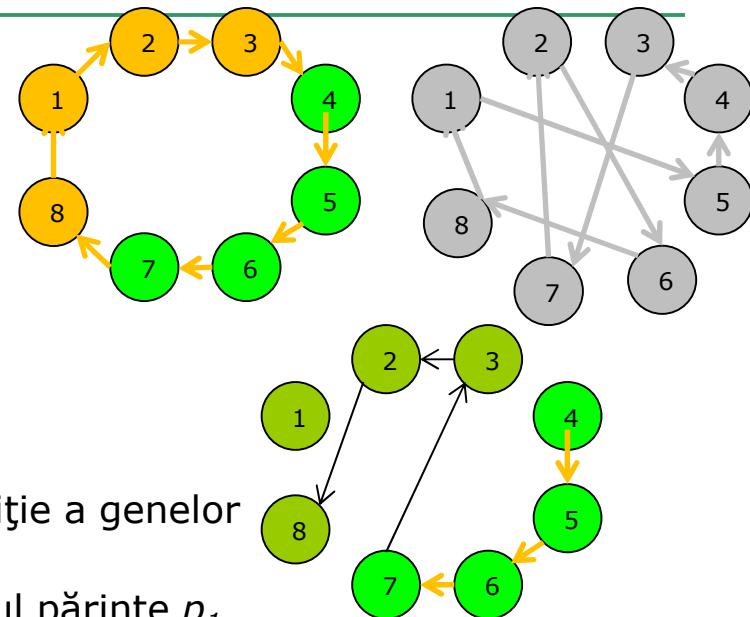
- ❑ Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obtin 2 descendenti
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare ordonată
 - Ideea de bază
 - ❑ Descendenții păstrează ordinea de apariție a genelor părintilor
 - ❑ Se alege un substring de gene din primul părinte p_1
 - ❑ Se copiază substringul din p_1 în descendental d_1 (pe poziții corespondente)
 - ❑ Se copiază genele din p_2 în descendental d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)



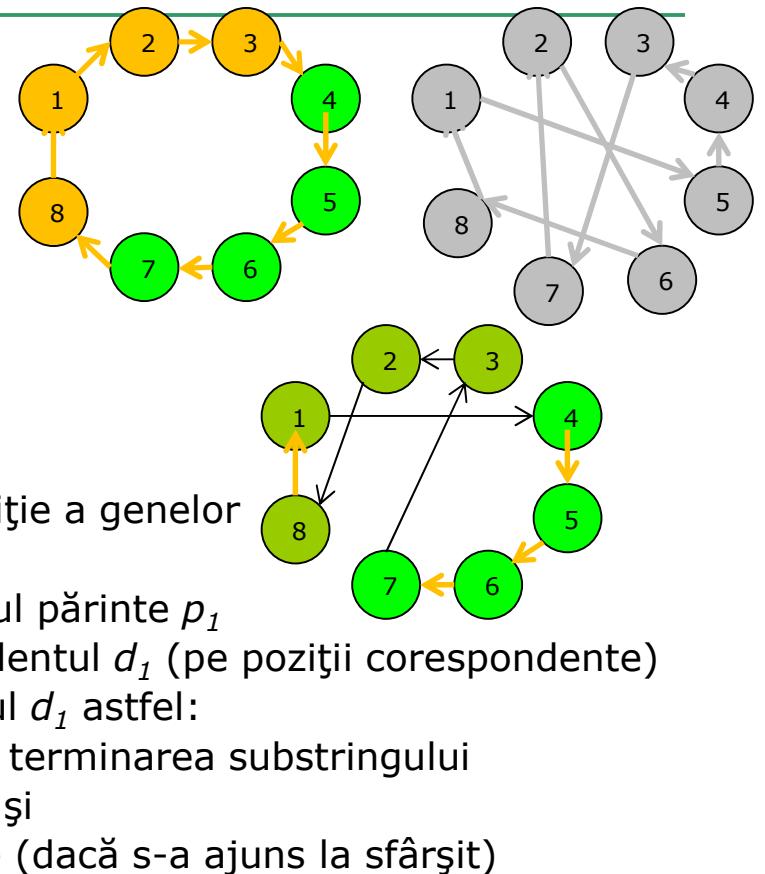
- ❑ Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obtin 2 descendenti
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare ordonată
 - Ideea de bază
 - ❑ Descendenții păstrează ordinea de apariție a genelor părintilor
 - ❑ Se alege un substring de gene din primul părinte p_1
 - ❑ Se copiază substringul din p_1 în descendental d_1 (pe poziții corespondente)
 - ❑ Se copiază genele din p_2 în descendental d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)



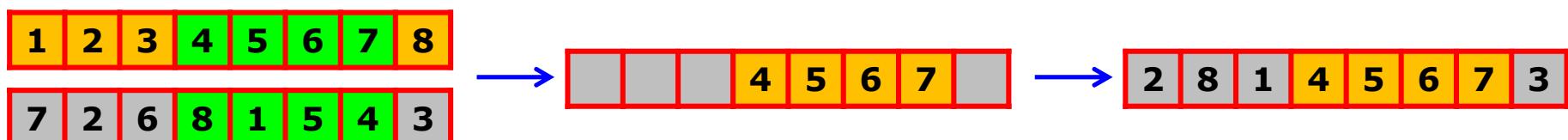
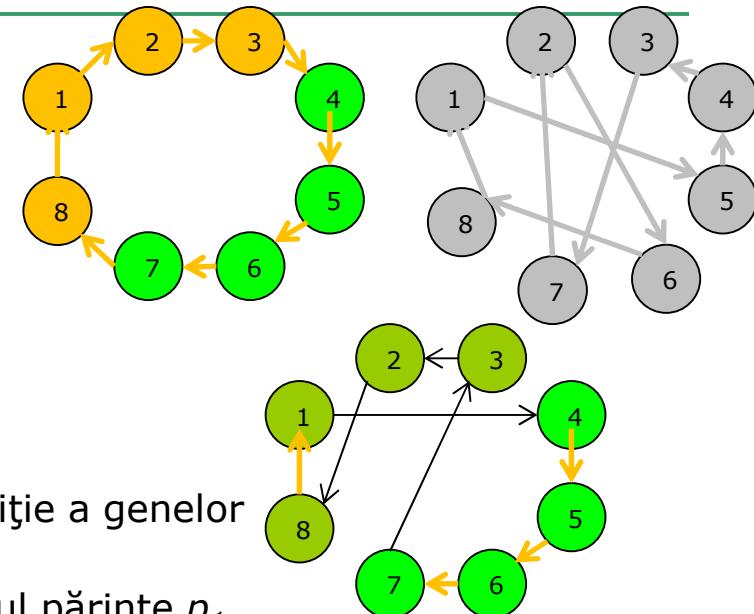
- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
 - se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
 - Încrucișare ordonată
 - Ideea de bază
 - Descendenții păstrează ordinea de apariție a genelor părintilor
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendental d_1 (pe poziții corespondente)
 - Se copiază genele din p_2 în descendental d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)



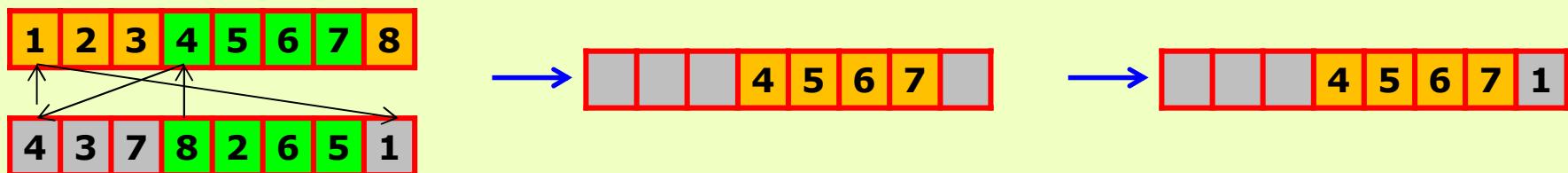
- ❑ Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obtin 2 descendenti
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare ordonată
 - Ideea de bază
 - ❑ Descendenții păstrează ordinea de apariție a genelor părintilor
 - ❑ Se alege un substring de gene din primul părinte p_1
 - ❑ Se copiază substringul din p_1 în descendentalul d_1 (pe poziții corespondente)
 - ❑ Se copiază genele din p_2 în descendentalul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)
 - ❑ Se reia procedeul pentru al doilea descendental d_2 .



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)



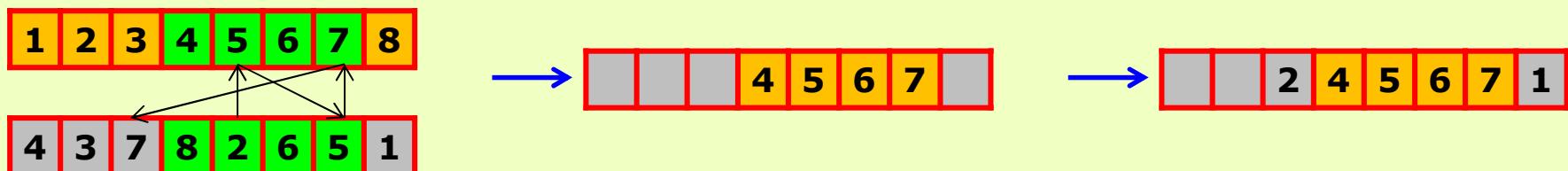
- ❑ Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obtin 2 descendenti
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare parțial transformată (*partially mapped XO*)
 - Ideea de bază
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendantul d_1 (pe poziții corespondente)
 - Se iau pe rând elementele i din substringul din p_2 care nu apar în substringul din p_1 și se determină care element j a fost copiat în locul lui din p_1
 - Se plasează i în d_1 în poziția ocupată de j în p_2 (dacă locul este liber)
 - Dacă locul ocupat de j în p_2 a fost deja completat în d_1 cu elementul k , i se pune în locul ocupat de k în p_2
 - Restul elementelor se copiază din p_2 în d_1
 - Pentru descendantul d_2 se procedează similar, dar inversând părintii



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)



- ❑ Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obtin 2 descendenti
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare parțial transformată (*partially mapped XO*)
 - Ideea de bază
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendantul d_1 (pe poziții corespondente)
 - Se iau pe rând elementele i din substringul din p_2 care nu apar în substringul din p_1 și se determină care element j a fost copiat în locul lui din p_1
 - Se plasează i în d_1 în poziția ocupată de j în p_2 (dacă locul este liber)
 - Dacă locul ocupat de j în p_2 a fost deja completat în d_1 cu elementul k , i se pune în locul ocupat de k în p_2
 - Restul elementelor se copiază din p_2 în d_1
 - Pentru descendantul d_2 se procedează similar, dar inversând părintii



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)



- ❑ Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obtin 2 descendenti
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare parțial transformată
 - Ideea de bază
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendantul d_1 (pe poziții corespondente)
 - Se iau pe rând elementele i din substringul din p_2 care nu apar în substringul din p_1 și se determină care element j a fost copiat în locul lui din p_1
 - Se plasează i în d_1 în poziția ocupată de j în p_2 (dacă locul este liber)
 - Dacă locul ocupat de j în p_2 a fost deja completat în d_1 cu elementul k , i se pune în locul ocupat de k în p_2
 - Restul elementelor se copiază din p_2 în d_1
 - Pentru descendantul d_2 se procedează similar, dar inversând părintii

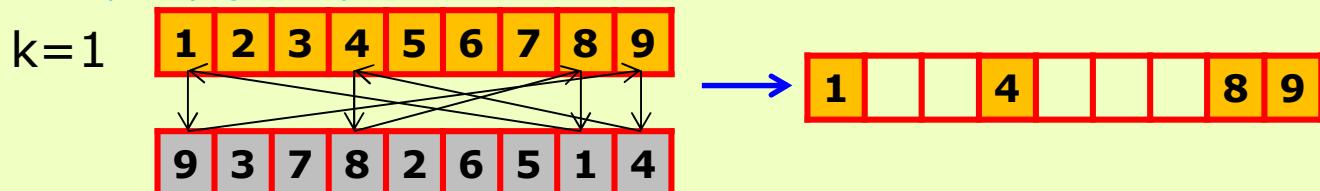


Inteligentă artificială - metode de căutare locală (AE)

Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)



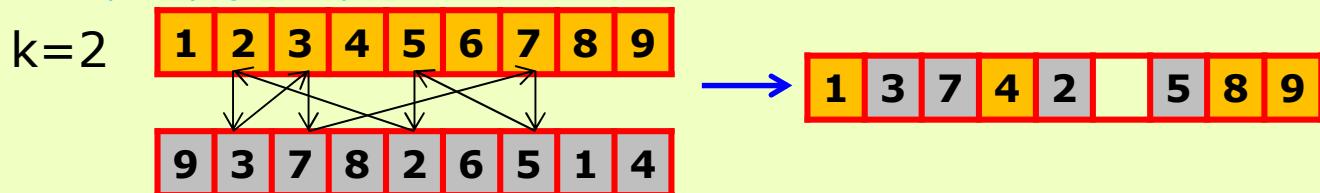
- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare ciclică
 - Ideea de bază
 1. inițial $k = 1$
 2. Se formează un ciclu
 - Se adaugă în ciclu gena de pe poziția k din p_1 (g_k^1)
 - Se consideră gena de pe poziția k din p_2 (g_k^2)
 - Se alege gena din p_1 cu valoarea egală cu g_k^2 (g_r^1) și se include în ciclu
 - Se consideră gena de pe poziția r din p_2 (g_r^2)
 - Se repetă pașii anteriori până când se ajunge la gena de pe poziția k din p_1
 3. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_1)
 4. Se incrementează k și se formează un nou ciclu dar cu genele din p_2
 5. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_2)
 6. Se repetă pașii 2-5 până când $k = L$



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)



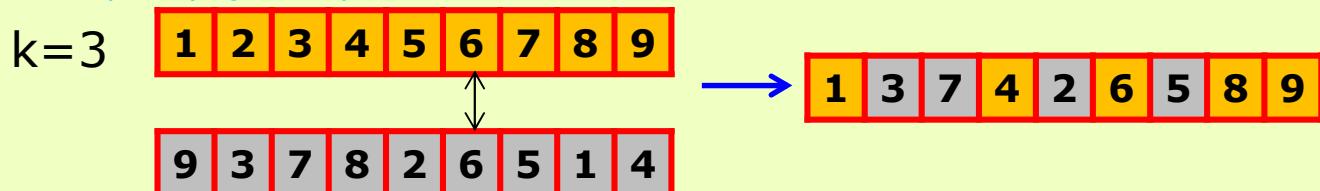
- ❑ Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare ciclică
 - Ideea de bază
 1. inițial $k = 1$
 2. Se formează un ciclu
 - Se adaugă în ciclu gena de pe poziția k din p_1 (g_k^1)
 - Se consideră gena de pe poziția k din p_2 (g_k^2)
 - Se alege gena din p_1 cu valoarea egală cu g_k^2 (g_r^1) și se include în ciclu
 - Se consideră gena de pe poziția r din p_2 (g_r^2)
 - Se repetă pașii anteriori până când se ajunge la gena de pe poziția k din p_1
 3. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_1)
 4. Se incrementează k și se formează un nou ciclu dar cu genele din p_2
 5. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_2)
 6. Se repetă pașii 2-5 până când $k = L$



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)



- ❑ Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare ciclică
 - Ideea de bază
 1. inițial $k = 1$
 2. Se formează un ciclu
 - Se adaugă în ciclu gena de pe poziția k din p_1 (g_k^1)
 - Se consideră gena de pe poziția k din p_2 (g_k^2)
 - Se alege gena din p_1 cu valoarea egală cu g_k^2 (g_r^1) și se include în ciclu
 - Se consideră gena de pe poziția r din p_2 (g_r^2)
 - Se repetă pașii anteriori până când se ajunge la gena de pe poziția k din p_1
 3. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_1)
 4. Se incrementează k și se formează un nou ciclu dar cu genele din p_2
 5. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_2)
 6. Se repetă pașii 2-5 până când $k = L$



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

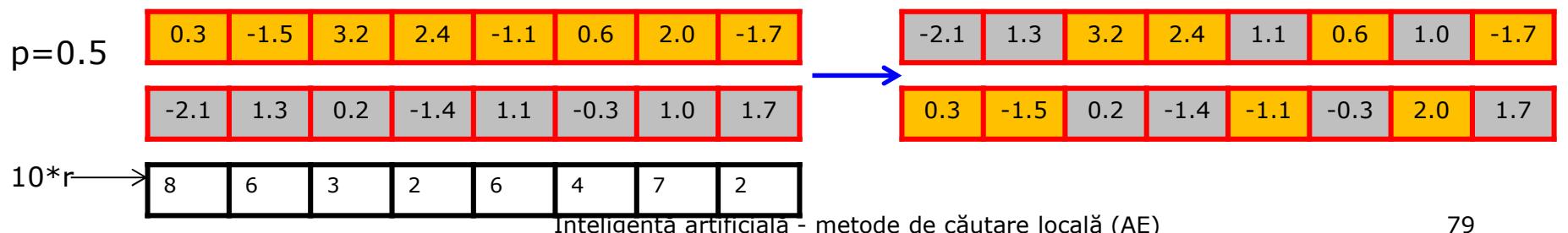


- ❑ Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obtin 2 descendenti
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare bazată pe muchii
 - A se consulta: Whitley, Darrell, Timothy Starkweather, D'Ann Fuquay (1989).
"Scheduling problems and traveling salesman: The genetic edge recombination operator". International Conference on Genetic Algorithms. pp. 133-140 [link](#)

Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)



- ❑ Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obtin 2 descendenti
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare discretă
 - Ideea de bază
 - ❑ Fiecare genă a unui descendant este luată (cu aceeași probabilitate, $p = 0.5$) dintr-unul din părinti
 - ❑ Similar încrucișării uniforme de la reprezentarea binară/întreagă
 - ❑ Nu se modifică valorile efective ale genelor (nu se creează informație nouă)



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)



- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenti
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [Ll_i, Ls_i]$, pt. $i=1,2,\dots,L$
- Încrucișare intermediară (aritmetică)
 - Ideea de bază
 - Se creează copii aflați (ca valoare) între părinti → încrucișare aritmetică
 - $z_i = \alpha x_i + (1 - \alpha) y_i$ unde $\alpha : 0 \leq \alpha \leq 1$.
 - Parametrul α poate fi:
 - Constant → încrucișare aritmetică uniformă
 - Variabil → ex. dependent de vârstă populației
 - Aleator pt fiecare încrucișare produsă
 - Apar noi valori ale genelor
 - Tipologie
 - Încrucișare aritmetică singulară
 - Încrucișare aritmetică simplă
 - Încrucișare aritmetică completă

Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)



- ❑ Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare intermediară (aritmetică) singulară
 - Se alege câte o genă (de același index k) din cei doi părinți și se combină
 - $g_k' = \alpha g_k^1 + (1-\alpha)g_k^2$
 - $g_k'' = (1-\alpha)g_k^1 + \alpha g_k^2$
 - Restul genelor rămân neschimilate
 - $g_i' = g_i^1$
 - $g_i'' = g_i^2$, pentru $i = 1,2,\dots,L$ și $i \neq k$

$$[LI, LS] = [-2.5, +3]$$

$k=3$

$\alpha = 0.6$

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7



$$0.6 * 3.2 + (1-0.6) * 0.2 = 2.0$$

$$(1-0.6) * 3.2 + 0.6 * 0.2 = 1.4$$

0.3	-1.5	2.0	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	1.4	-1.4	1.1	-0.3	1.0	1.7

Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)



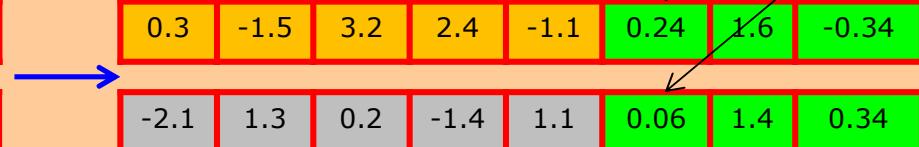
- ❑ Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare intermediară (aritmetică) simplă
 - Se alege o poziție k și se combină toate genele de după acea poziție
 - ❑ $g_i' = \alpha g_i^1 + (1-\alpha)g_i^2$
 - ❑ $g_i'' = (1-\alpha)g_i^1 + \alpha g_i^2$, pentru $i=k, k+1, \dots, L$
 - Genele de pe poziții $< k$ rămân neschimbate
 - ❑ $g_i' = g_i^1$
 - ❑ $g_i'' = g_i^2$, pentru $i = 1,2,\dots,k-1$

$$[LI, LS] = [-2.5, +3]$$

$$k=6$$

$$\alpha = 0.6$$

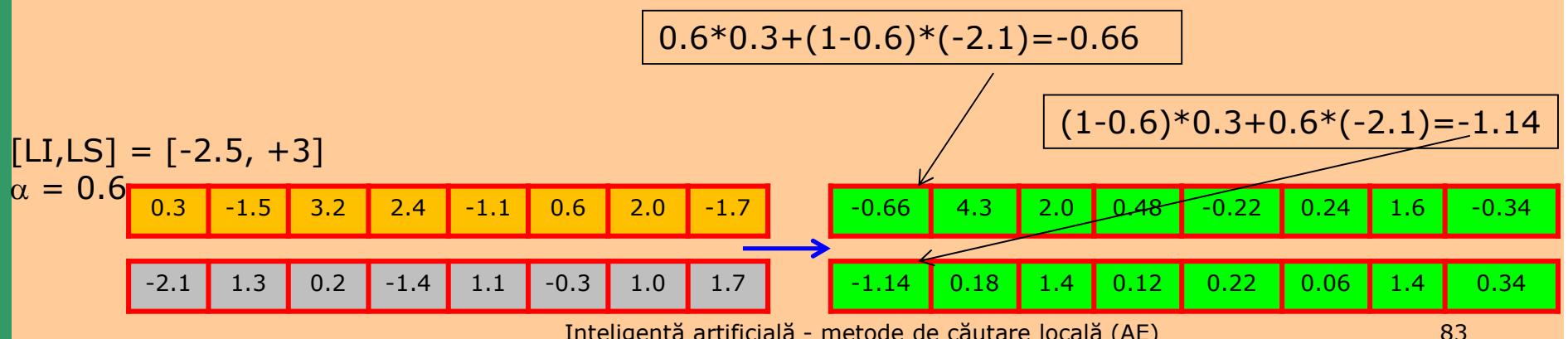
0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)



- ❑ Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare intermediară (aritmetică) completă
 - Toate genele (de pe poziții corespunzătoare) se combină
 - $g_i' = \alpha g_i^1 + (1-\alpha)g_i^2$
 - $g_i'' = (1-\alpha)g_i^1 + \alpha g_i^2$, pentru $i=1,2,\dots,L$



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)



- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenti
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare geometrică
 - Ideea de bază
 - Fiecare genă a unui descendant reprezintă produsul genelor părintilor, fiecare cu un anumit exponent ω , respectiv $1-\omega$ (unde ω număr real pozitiv subunitar)
 - $g_i' = (g_i^1)^\omega (g_i^2)^{1-\omega}$
 - $g_i'' = (g_i^1)^{1-\omega} (g_i^2)^\omega$

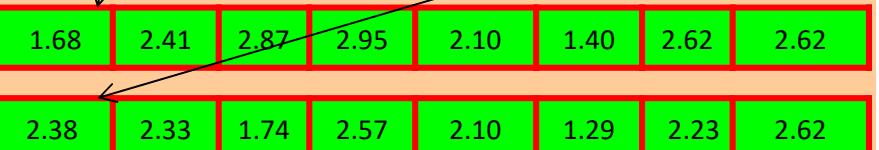
$$[LI, LS] = [-2.5, +3]$$

$$\omega = 0.7$$



$$0.3^{0.7} + 2.1^{1-0.7} = 1.68$$

$$0.3^{1-0.7} + 2.1^{0.7} = 2.38$$



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)



- ❑ Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obține 1 descendant
 - $c_1 = (g_1', g_2', \dots, g_L')$,
 - unde $g_i^1, g_i^2, g_i' \in [L_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare amestecată (*blend crossover – BLX*)
 - Ideea de bază
 - Se generează un singur descendant
 - Genele g_i' ale descendantului sunt alese aleator în intervalul $[Min_i - I*a, Max_i + I*a]$, unde:
 - $Min_i = \min\{g_i^1, g_i^2\}$, $Max_i = \max\{g_i^1, g_i^2\}$
 - $I = Max - Min$, a – parametru din $[0,1]$

$[L_i, LS_i] = [-2.5, +3]$

$a = 0.7$

0.3	1.5	3.2	2.4	1.1	0.6	2.0	1.7
2.1	1.3	0.2	1.4	1.1	0.3	1.0	1.7



1.25	1.45	-1.11	2.37	1.10	0.11	0.70	1.70
------	------	-------	------	------	------	------	------

Min	0.3	1.3	0.2	1.4	1.1	0.3	1.0	1.7
Max	2.1	1.5	3.2	2.4	1.1	0.6	2.0	1.7
I	0.8	0.2	3.0	1.0	0	0.3	1.0	0.0

Min-Ia	-0.26	1.16	-1.90	0.70	1.10	0.09	0.30	1.70
Max+Ia	2.66	1.50	3.20	2.40	1.10	0.60	2.00	1.70

Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)



- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare binară simulată
 - Ideea de bază
 - Fiecare genă a unui descendent reprezintă o combinație a genelor părinților
$$d_1 = \frac{p_1 + p_2}{2} - \beta \frac{p_2 - p_1}{2}, \quad d_2 = \frac{p_1 + p_2}{2} + \beta \frac{p_2 - p_1}{2}$$
 - a.î. să se respecte cele 2 proprietăți de la încrucișarea cu n puncte de tăietură (pt. reprezentarea binară)
 - media valorilor codate în părinți = media valorilor codate în descendenți
 - probabilitatea apariției unui factor de răspândire $\beta \approx 1$ este mai mare decât a oricărui alt factor

Algoritmi evolutivi – algoritm Proiectare – recombinarea



□ Recombinarea multiplă

- Bazată pe frecvența valorilor din părinți (încrucișare uniformă generală)
- Bazată pe segmentare și recombinare (încrucișare generală cu puncte de tăietură diagonală)
- Bazată pe operații numerice specifice valorilor reale (încrucișare bazată pe centrul de masă, încrucișare generală aritmetică)

Algoritmi evolutivi – algoritm Proiectare – mutație sau recombinare?

- Dezbateri aprinse
 - Întrebări:
 - care operator este mai bun?
 - care operator este necesar?,
 - care operator este mai important?
 - Răspunsuri:
 - Depinde de problemă, dar
 - În general, este bine să fie folosiți ambii operatori
 - Fiecare având alt rol
 - Sunt posibili AE doar cu mutație, dar nu sunt posibili AE doar cu încrucișare
- Aspecte ale căutării:
 - Explorare → descoperirea regiunilor promițătoare ale spațiului de căutare (acumulând informație utilă despre problemă)
 - Exploatare → optimizarea într-o regiune promițătoare (folosind informația existentă)
 - Trebuie să existe cooperare și competiție între aceste 2 aspecte
- Încrucișarea
 - Operator exploataativ, realizând un mare salt într-o regiune undeva între regiunile asociate părinților
 - Efectele exploataitive se reduc pe măsură ce AE converge
 - Operator binar (n-ar) care poate combina informația din 2 (sau mai mulți) părinți
 - Operator care nu schimbă frecvența valorilor din cromozomi la nivelul întregii populații
- Mutația
 - Operator explorativ, realizând mici diversiuni aleatoare, rămânând în regiunea apropiată părintelui
 - Evadarea din optimele locale
 - Operator care poate introduce informație genetică nouă
 - Operator care schimbă frecvența valorilor din cromozomi la nivelul întregii populații

Algoritmi evolutivi – algoritm Proiectare – criteriu de oprire



- Stabilirea unui criteriu de stop
 - S-a identificat soluția optimă
 - S-au epuizat resursele fizice
 - S-a efectuat un anumit număr de evaluaări ale funcăiei de fitness
 - S-au epuizat resursele utilizatorului (timp, răbdare)
 - S-au “născut” câteva generaăii fără îmbunătăăiri

Algoritmi evolutivi - algoritm



□ Evaluarea performanțelor unui AE

- După mai multe rulări se calculează:

- Măsuri statistice

- media soluțiilor,
 - mediana soluțiilor,
 - cea mai bună soluție,
 - cea mai slabă soluție,
 - deviația standard – pentru comparabilitate

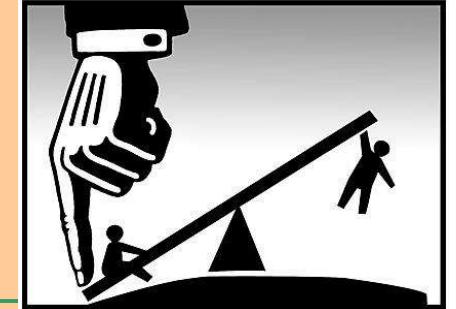
- Calculate pentru un număr suficient de mare de rulări independente

Algoritmi evolutivi



- Analiza complexității
 - Partea cea mai costisitoare → calculul fitnessului

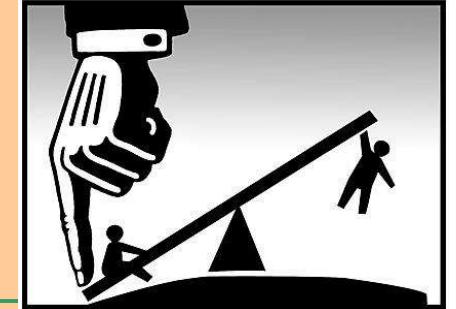
Algoritmi evolutivi



□ Avantaje

- Schema AE universală pentru toate problemele
 - se modifică doar
 - reprezentarea
 - funcția de fitness
- AE sunt capabili să producă rezultate mai bune decât metodele convenționale de optimizare pentru că:
 - nu necesită liniarizare
 - nu implică anumite presupuneri (continuitate, derivabilitate, etc. a funcției obiectiv)
 - nu ignoră anumite potențiale soluții
- AE sunt capabili să exploreze mai multe potențiale soluții decât poate explora omul

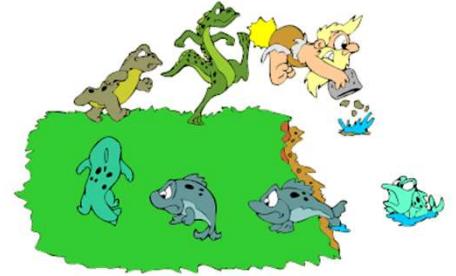
Algoritmi evolutivi



□ Dezavantaje

- Timp de rulare îndelungat

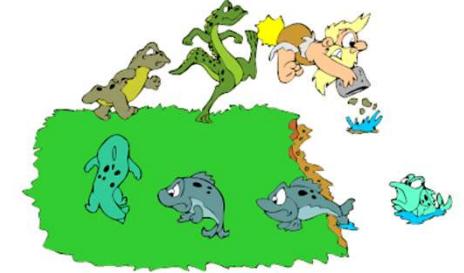
Algoritmi evolutivi



□ Aplicații

- Proiectări vehicule
 - Componența materialelor
 - Forma vehiculelor
- Proiectări inginerești
 - Optimizarea structurală și organizatorică a construcțiilor (clădiri, roboți, sateliți, turbine)
- Robotică
 - Optimizarea proiectării, funcționării componentelor
- Evoluare de hardware
 - Optimizarea de circuite digitale
- Optimizarea telecomunicațiilor
- Generarea de glume și jocuri de cuvinte
- Invenții biomimetice (inspirate de arhitecturi naturale)
- Rutări pentru trafic și transporturi
- Jocuri de calculator
- Criptări
- Profilul expresiv al genelor
- Analiza chimică a cinceticii
- Strategii financiare și marketing

Algoritmi evolutivi



□ Tipuri de algoritmi evolutivi

- Strategii evolutive
- Programare evolutivă
- Algoritmi genetici
- Programare genetică

Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor – Materiale de citit și legături utile

- capitolul 16 din C. Groșan, A. Abraham, *Intelligent Systems: A Modern Approach*, Springer, 2011
- James Kennedy, Russel Eberhart, *Particle Swarm Optimisation*, Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942–1948, 1995
(04_ACO_PSO/PSO_00.pdf)
- Marco Dorigo, Christian Blum, *Ant colony optimization theory: A survey*, Theoretical Computer Science 344 (2005) 243 – 27 (04_ACO_PSO/Dorigo05_ACO.pdf)

-
- Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan - www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ



Rezolvarea problemelor de căutare

Strategii de căutare informată
algoritmi evolutivi

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Materiale de citit și legături utile

- capitolul 14 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998*
- capitolul 7.6 din *A. A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

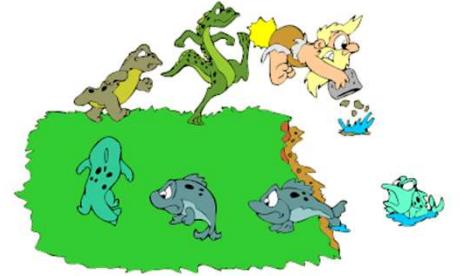
Căutare locală



□ Tipologie

- Căutare locală simplă - se reține o singură stare vecină
 - Hill climbing → alege cel mai bun vecin
 - Simulated annealing → alege probabilistic cel mai bun vecin
 - Căutare tabu → reține lista soluțiilor recent vizitate
- Căutare locală în fascicol (beam local search) – se rețin mai multe stări (o populație de stări)
 - Algoritmi evolutivi
 - Optimizare bazată pe comportamentul de grup (Particle swarm optimisation)
 - Optimizare bazată pe furnici (Ant colony optimisation)

Algoritmi evolutivi



- Tipuri de algoritmi evolutivi
 - Algoritmi genetici
 - Strategii evolutive
 - Programare evolutivă
 - Programare genetică

Algoritmi genetici



- Aspecte teoretice
- Algoritm
 - Schema generală a unui AGS
 - Reprezentare și operatori
- Exemplu
- Proprietăți
- Aplicații



Algoritmi genetici – aspecte teoretice

- Propuși
 - J. Holland → AG simpli (AGS)
- Căutare
 - **Concurențială**, ghidată de calitatea **absolută** a indivizilor
- Operatori de căutare
 - Selecția
 - Încrucișarea **SI** mutația
- Elemente speciale
 - Accent deosebit pe încrucișare



Algoritmi genetici – schema generală

Algoritm generațional

```

Initializare P(0)
Evaluare(P(0))
g = 0;
while (not condiție_stop) do
    repeat
        Selectarea a 2 părinți  $p_1$  și  $p_2$  din  $P(g)$ 
        Încrucișare( $p_1$ ,  $p_2$ ) →  $o_1$  și  $o_2$ 
        Mutatie( $o_1$ ) →  $o_1^*$ 
        Mutatie( $o_2$ ) →  $o_2^*$ 
        Evaluare( $o_1^*$ )
        Evaluare( $o_2^*$ )
        Adăugare  $o_1^*$  și  $o_2^*$  în  $P(g+1)$ 
    until  $P(g+1)$  este plină
    g++
endWhile

```

Algoritm steady-state

```

Initializare P
Evaluare(P)

while (not condiție_stop) do
    For i = 1 to |P|
        Selectarea a 2 părinți  $p_1$  și  $p_2$  din  $P(g)$ 
        Încrucișare( $p_1$ ,  $p_2$ ) →  $o_1$  și  $o_2$ 
        Mutatie( $o_1$ ) →  $o_1^*$ 
        Mutatie( $o_2$ ) →  $o_2^*$ 
        Evaluare( $o_1^*$ )
        Evaluare( $o_2^*$ )
        B = Best( $o_1^*$ ,  $o_2^*$ )
        W = Worst( $o_1^*$ ,  $o_2^*$ )
        Dacă B e mai bun ca W, W ← B
    EndFor
endWhile

```



Algoritmi genetici – schema generală

Algoritm generațional

1. Generarea aleatoare a unei populații (generația 0) cu n cromozomi
2. Evaluarea tuturor cromozomilor
3. Crearea unei noi populații (generații) prin repetarea următorilor 4 pași
 - Selecția, bazată pe fitness, a 2 părinți
 - Încrucișarea părinților pentru obținerea unui descendant cu o anumită probabilitate; dacă încrucișarea nu are loc, descendantul va fi:
 - Unul dintre părinți
 - Cel mai bun dintre părinți
 - Mutarea cu o anumită probabilitate a fiecărui element al descendantului
 - Acceptarea descendantului și plasarea lui în noua populație (generație)
4. Înlocuirea vechii populații cu noua populație (schimbul de generații)
5. Testarea condițiilor de terminare a căutării; dacă ele sunt satisfăcute, se returnează cea mai bună soluție din populația (generația) curentă
6. Ciclarea algoritmului – întoarcerea la pasul 2



Algoritmi genetici – schema generală

Algoritm steady-state

1. Generarea aleatoare a unei populații cu n cromozomi
2. Evaluarea tuturor cromozomilor
3. Crearea unei noi populații prin repetarea următorilor 4 pași
 - Selecția, bazată pe fitness, a 2 părinți
 - Încrucișarea părinților pentru obținerea unui descendant cu o anumită probabilitate; dacă încrucișarea nu are loc, descendantul va fi:
 - Unul dintre părinți
 - Cel mai bun dintre părinți
 - Mutarea cu o anumită probabilitate a fiecărui element al descendantului
 - Alegerea celui mai bun descendant B
 - Dacă B este mai bun decât cel mai slab individ al populației W, atunci B îl înlocuiește pe W
4. Testarea condițiilor de terminare a căutării; dacă ele sunt satisfăcute, se returnează cea mai bună soluție din populația (generația) curentă
5. Ciclarea algoritmului – întoarcerea la pasul 2

Algoritmi genetici – reprezentare și operatori



- Reprezentare
 - Stringuri binare (initial), de numere întregi, de numere reale, de alte elemente
- Populația
 - μ părinți, μ descendenți
- Selecția pentru recombinare
 - Proporțională cu fitness-ul
- Recombinarea
 - Cu n puncte de tăietură sau uniformă cu o probabilitate p_c fixată ce acționează la nivel de cromozom
- Mutăția
 - *Bitwise bit-flipping* cu o probabilitate p_m fixată pentru fiecare genă (bit)
- Selecția pentru supraviețuire
 - Toți descendenții înlocuiesc părinții



Algoritmi genetici – exemplu

- Să se determine valoarea maximă a funcției
 $f: \{0, 1, \dots, 31\} \rightarrow \mathbb{Z}, f(x) = x^2$
- Configurarea AG
 - Stringuri binare de lungime 5, ex. $c = (10101) \rightarrow x = 21$
 - O populație cu $\mu = 4$ cromozomi
 - Selecție proporțională prin ruletă
 - Încrucișare cu 1 punct de tăietură
 - Mutăție tare
- Evaluare → optimizare prin maximizare



Algoritmi genetici – exemplu

Initializare

No cromo-zom	Cromo zom
1	01101
2	11000
3	01000
4	10011
sumă	



Algoritmi genetici – exemplu

Evaluare

No cromo-zom	Cromozom	Valoarea x	Fitness $f(x^2)$
1	01101	13	169
2	11000	24	576
3	01000	8	64
4	10011	19	361
sumă			1170



Algoritmi genetici – exemplu

Selectie

No cromozom	Cromozom	Valoare x	Fitness f(x ²)	P _{selSP} (i)	$\Sigma P_{selSP}(i)$
1	01101	13	169	169/117 0=0.14	0.14
2	11000	24	576	576/117 0=0.49	0.63
3	01000	8	64	64/1170 =0.06	0.69
4	10011	19	361	361/117 0=0.31	1.00
sumă			1170		



Algoritmi genetici – exemplu

Selectie

No cromozom	Cromozom	Valoarea x	Fitness	$P_{selSP}(i)$	$\sum P_{selSP}(i)$	$r_1=0.5$	$r_2=0.8$
1	01101	13	169	169/117 0=0.14	0.14		
2	11000	24	576	576/117 0=0.49	0.63	X	
3	01000	8	64	64/1170 =0.06	0.69		
4	10011	19	361	361/117 0=0.31	1.00		X
sumă			1170				

$$p_1=c_2 = (11000) \text{ și } p_2 = c_4 = (10011)$$



Algoritmi genetici – exemplu

Încrucișare

No cromozom	Cromo zomi părinti	Cromo zomi fii	Valoarea x (pt. fii)	Fitness (pt. fii)
2	11 000	11 011	27	729
4	0011	10000	16	256

Mutăție

No cromozom	Cromo zomi fii	Cromo zomi fii*	Valoarea x (pt. fii*)	Fitness (pt. fii*)
o1	11 011	10011	19	361
o2	00000	10010	18	324



Algoritmi genetici – exemplu

Adăugarea în următoarea generație

No cromo-zom	Cromo zom
1	10011
2	10010
3	
4	



Algoritmi genetici – exemplu

Selectie

No cromozom	Cromozom	Valoare x	Fitness f(x ²)	P _{selSP} (i)	$\Sigma P_{selSP}(i)$	r ₁ =0.1	r ₂ =0.7
1	01101	13	169	169/117 0=0.14	0.14	x	
2	11000	24	576	576/117 0=0.49	0.63		
3	01000	8	64	64/1170 =0.06	0.69		
4	10011	19	361	361/117 0=0.31	1.00		x
sumă			1170				



Algoritmi genetici – exemplu

Selectie

No cromozom	Cromozom	Valoarea x	Fitness f(x ²)	P _{selSP} (i)	$\Sigma P_{selSP}(i)$	r ₁ =0.5	r ₂ =0.8
1	01101	13	169	169/117 0=0.14	0.14	x	
2	11000	24	576	576/117 0=0.49	0.63		
3	01000	8	64	64/1170 =0.06	0.69		
4	10011	19	361	361/117 0=0.31	1.00		x
sumă			1170				

$$p_1=c_1 = (01101) \text{ și } p_2 = c_4 = (10011)$$



Algoritmi genetici – exemplu

Încrucișare

No cromozom	Cromo zomi părinti	Cromo zomi fii	Valoarea x (pt. fii)	Fitness (pt. fii)
1	01 101	01 011	11	121
4	100 11	101 01	21	441

Mutatie

No cromozom	Cromo zomi fii	Cromo zomi fii*	Valoarea x (pt. fii*)	Fitness (pt. fii*)
o1	01 011	000 11	3	9
o2	101 01	101 11	23	529



Algoritmi genetici – exemplu

Adăugarea în următoarea generație

No cromo-zom	Cromo zom
1	10011
2	10010
3	00011
4	10111

Algoritmi genetici – proprietăți



- Cromozomi liniari de aceeași dimensiune
- Evidențiază avantajele combinării informațiilor de la părinți buni prin încrucișare
- Numeroase variante
- Numeroși operatori (selecție, încrucișare, mutație)
- Nu sunt foarte rapizi
- Euristici bune pentru probleme de combinatorică

Algoritmi genetici – aplicații



- Probleme de combinatorică
- Optimizări în proiectarea compoziției materialelor și a formei aerodinamice a vehiculelor (auto, aeriene, navale, trenuri)
- Optimizări în proiectarea structurală și funcțională a clădirilor (locuințe, fabrici, etc)
- Optimizări în robotică
- Optimizări în proiectarea circuitelor digitale

Strategii evolutive



- ❑ Aspecte teoretice
- ❑ Algoritm
 - Schema generală
 - Reprezentare și operatori
- ❑ Exemplu
- ❑ Proprietăți
- ❑ Aplicații



Strategii evolutive – aspecte teoretice

- Propuse
 - în anii '60-'70 în Germania de către Bienert, Rechenberg și Schwefel
- Căutare
 - **Concurențială**, ghidată de calitatea **absolută** a indivizilor
- Operatori de căutare
 - Selecția
 - Încrucișarea **SI** mutația
- Elemente speciale
 - Auto-adaptarea parmetrilor (în special a parametrilor mutației)



Strategii evolutive – schema generală

Inițializare $P(0)$

Evaluare($P(0)$)

$g = 0;$

while (*not* condiție_stop) do

repeat

Selectarea a 2 părinți p_1 și p_2 din $P(g)$

Încrucișare(p_1, p_2) $\rightarrow o_1$

Mutație(o_1 , param) $\rightarrow o_1^*, \text{param}'$

Evaluare(o_1^*)

Adăugare o_1^* în $P(g+1)$

until $P(g+1)$ este plină

$g++$

endWhile

Strategii evolutive – reprezentare și operatori



- Reprezentare
 - Reală
 - Codează și rata de mutație
- Populația
 - μ părinți, λ descendenți
- Selecția pentru recombinare
 - Uniformă aleatoare
- Recombinarea
 - Discretă sau intermediară
- Mutația
 - Perturbare Gaussiană
 - Auto-adaptare a pasului de mutație
- Selecția pentru supraviețuire
 - (μ, λ) sau $(\mu + \lambda)$

Strategii evolutive – reprezentare și operatori



- Pp. că dorim minimizarea funcției $f: R^n \rightarrow R$

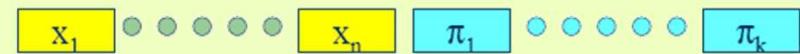
□ Reprezentare

- 3 părți:

- Variabile obiect: x_1, x_2, \dots, x_n cu $x_i \in R \rightarrow$ reprezentare reală
- Parametri posibili ai SE:
 - Pași de mutație: $\sigma_1, \dots, \sigma_{n(\sigma)}$
 - Unghiuri de rotație $\omega_1, \dots, \omega_{n(\alpha)}$

- Completă

- $n(\sigma)=n, n(\alpha) = n(n-1)/2$ – nr de perechi $(i,j), i, j = 1, 2, \dots, n$



Vector of real-valued fields

Real-valued evolutionary parameters



Strategii evolutive – reprezentare și operatori



□ Selecția părintilor (pentru reproducere)

- Uniformă aleatoare
- Fiecare individ are aceeași probabilitate de a fi selectat

Strategii evolutive – reprezentare și operatori



□ Reproducerea

- Combină doi sau mai mulți părinți
- Crează un singur descendant

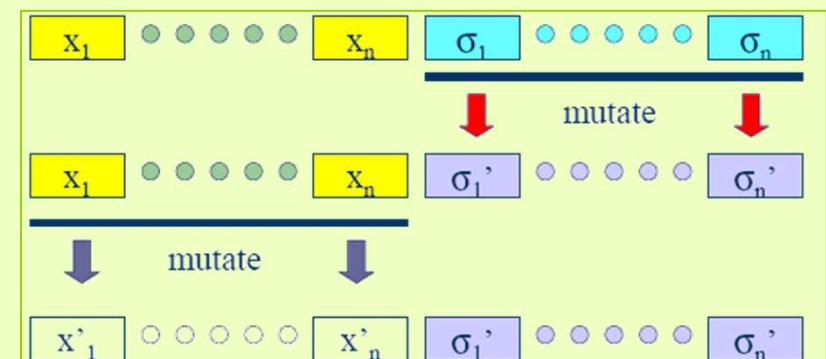
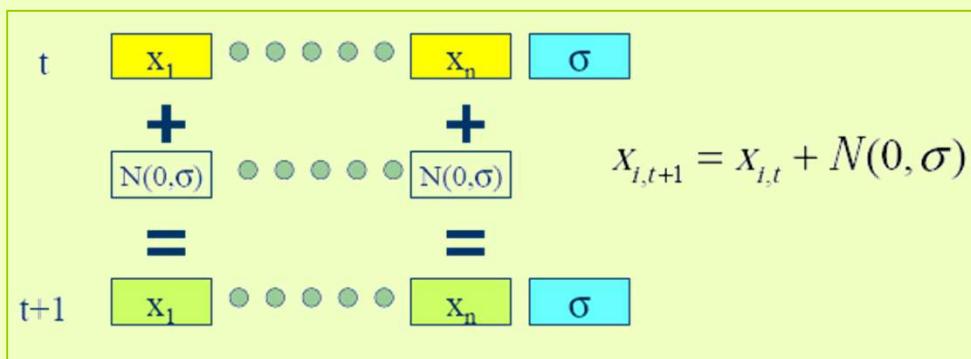
	2 părinți	Câte 2 părinți pentru fiecare element x_i al unui cromozom
$z_i = (x_i + y_i) / 2$	Intermediară locală	Intermediară globală
z_i este fie x_i , fie y_i (alegerea fiind aleatoare)	Discretă locală	Discretă globală

Strategii evolutive – reprezentare și operatori



■ Mutația

- Parametrii σ se coevoluează cu soluția x
- Mutație Gaussiană
 - σ este evoluat în σ'
 - $x_i' = x_i + N(0, \sigma')$
- Nu este necesar ca parametrii σ să evolueze (să se modifice) cu aceeași frecvență ca soluția (x)
- Noua pereche (x', σ') se evaluatează de 2 ori:
 - x' este bună dacă evaluarea $f(x')$ este bună
 - σ' este bun dacă x' este bună



Strategii evolutive – reprezentare și operatori



- Cum este evoluat pasul de mutație din σ în σ' ? → diferite metode

■ Regula succesului 1/5

- Se determină procentajul p_s al mutațiilor “folositoare” (care au îmbunătățit potențiala soluție) din ultimele k iterații
- Se modifică σ după fiecare k iterații astfel:
 - $\sigma = \sigma / c$, dacă $p_s > 1/5$
 - $\sigma = \sigma * c$, dacă $p_s < 1/5$
 - $\sigma = \sigma$, dacă $p_s = 1/5$,
- unde $0.8 \leq c \leq 1$

■ Regula auto-adaptării

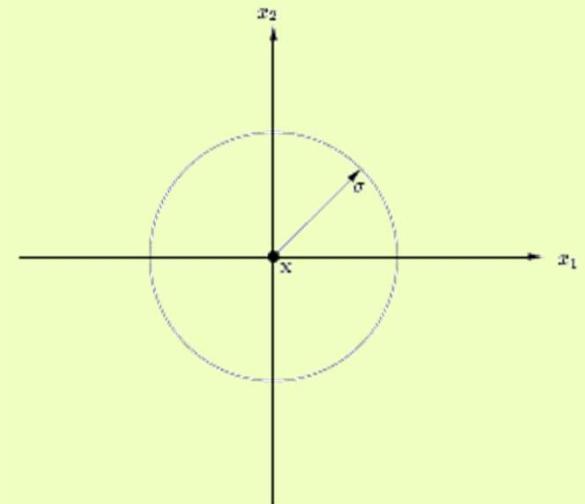
- Mutație necorelată cu un singur parametru σ
- Mutație necorelată cu n parametri σ
- Mutație corelată

Strategii evolutive – reprezentare și operatori



□ Mutație necorelată cu un singur parametru σ

- Cromozomi de forma: $(x_1, x_2, \dots, x_n, \sigma)$
- Mutație
 - $\sigma' = \sigma * \exp(\tau * N(0, 1))$
 - $x_i' = x_i + \sigma' * N(0, 1)$
 - Unde τ - rata de învățare
 - de obicei $\tau = 1/(n^{1/2})$
 - Dacă $\sigma' < \varepsilon_0 \rightarrow \sigma' = \varepsilon_0$



Strategii evolutive – reprezentare și operatori



□ Mutăție necorelată cu n parametri σ

- Cromozomi de forma: $(x_1, x_2, \dots, x_n, \sigma_1, \sigma_2, \dots, \sigma_n)$

- Mutăție

- $\square \sigma'_i = \sigma_i * \exp(\tau' * N(0,1) + \tau * N_i(0,1))$

- $\square x'_i = x_i + \sigma'_i * N_i(0,1)$

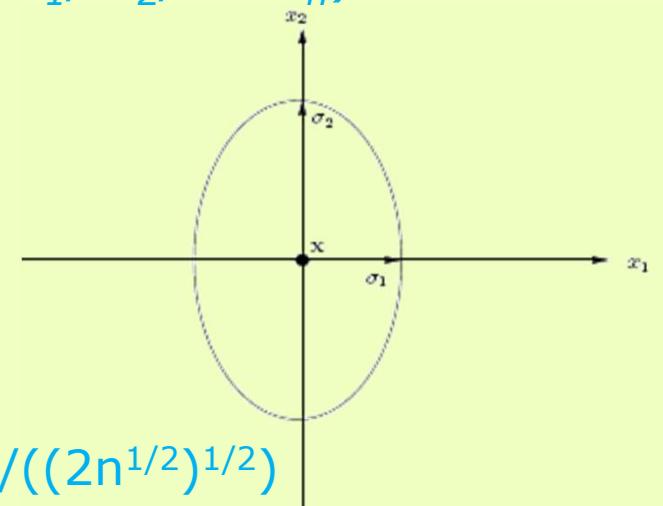
- \square unde:

- τ' - rata globală de învățare

- τ - rata individuală de învățare

- de obicei $\tau' = 1/((2n)^{1/2})$ și $\tau = 1/((2n^{1/2})^{1/2})$

- \square dacă $\sigma' < \varepsilon_0 \rightarrow \sigma' = \varepsilon_0$



Strategii evolutive – reprezentare și operatori



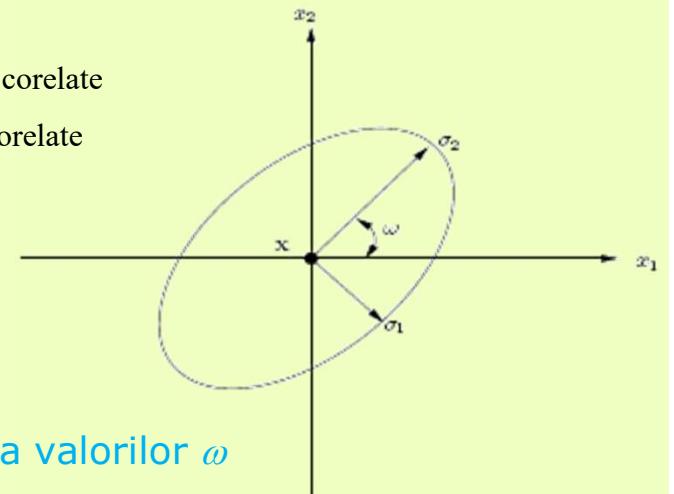
□ Mutăție corelată cu $n+k$ parametri

- Cromozomi de forma: $(x_1, x_2, \dots, x_n, \sigma_1, \sigma_2, \dots, \sigma_n, \omega_1, \omega_2, \dots, \omega_k)$,
- unde $k=n(n-1)/2$
- Matricea de covariație \mathbf{C} este definită prin:

$$c_{ij} = \begin{cases} \sigma_i^2, & \text{dacă } i = j \\ 0, & \text{dacă } i \text{ și } j \text{ nu sunt corelate} \\ \frac{1}{2}(\sigma_i^2 - \sigma_j^2) * \tan(2\omega_{ij}), & \text{dacă } i \text{ și } j \text{ sunt corelate} \end{cases}$$

■ Mutăție

- $\sigma'_i = \sigma_i * \exp(\tau' * N(0, 1) + \tau * N_i(0, 1))$
- $\omega'_{ij} = \omega_{ij} + \beta * N(0, 1)$
- $\mathbf{x}' = \mathbf{x} + \mathbf{N}(\mathbf{0}, \mathbf{C}')$
- unde:
 - $\mathbf{x} = (x_1, x_2, \dots, x_n)$
 - \mathbf{C}' – matricea de covariație \mathbf{C} după mutarea valorilor ω
 - τ' - rata globală de învățare
 - τ - rata intelligentă de învățare
 - de obicei $\tau' = 1/((2n)^{1/2})$ și $\tau = 1/((2n^{1/2})^{1/2})$ și $\beta \approx 5^\circ$
- dacă $\sigma' < \varepsilon_0 \rightarrow \sigma' = \varepsilon_0$
- dacă $|\omega'_{ij}| > \pi \rightarrow \omega'_{ij} = \omega'_{ij} - 2\pi \operatorname{sign}(\omega'_{ij})$



Strategii evolutive – reprezentare și operatori



□ Selecția de supraviețuire

- Aplicată după crearea a λ descendenți din μ părinți prin recombinare și mutație
- Alegerea celor mai buni μ indivizi din
 - Mulțimea copiilor – SE (μ, λ)
 - Selecție “uitucă”
 - Are performanțe mai bune
 - Mulțimea părinților și copiilor – SE($\mu + \lambda$)
 - Selecție elitistă
- De obicei, $\lambda = 7 * \mu$ (→ presiune de selecție mare)



Strategii evolutive – proprietăți

□ Caracteristici

- cromozomi liniari de aceeași dimensiune
- oferă viteză de lucru
- lucrează cu vectori de numere reale
- se bazează pe o teorie matematică fundamentată
- evoluează și parametrii algoritmului în sine (auto-adaptează parametrii mutației)

□ SE inițiale

- SE($\mu+\lambda$), cu $\mu=1$, $\lambda=1$
- Căutare locală de tip Hill Climbing
- Dar, cromozomul codează și:
 - rata de mutație
 - strategie de modificare pentru deviația standard a distribuției mutației



Strategii evolutive - aplicații

- Probleme de optimizare numerică
- Optimizarea formei lentilelor necesare refracției luminii
- Distribuția lichidului într-o rețea sangvină
- Curba Brachystochrone
- Rezolvarea cubului Rubik

Programare evolutivă



- Aspecte teoretice
- Algoritm
 - Schema generală
 - Reprezentare și operatori
- Proprietăți
- Aplicații

Programare evolutivă – aspecte teoretice



- Propusă
 - în SUA în anii 1960 de către D. Fogel
- Căutare
 - **Concurențială**, ghidată de calitatea **relativă** a indivizilor → selecția de supraviețuire
- Operatori de căutare
 - Selecția
 - **DOAR** mutația
- Elemente speciale
 - AE fără recombinare
 - Auto-adaptarea parametrilor (similar SE)

Programare evolutivă – schema generală



Initializare $P(0)$

Evaluare($P(0)$)

$g = 0;$

while (not condiție_stop) do

 Pentru fiecare cromozom c_i din $P(g)$

 Mutație(c_i , param) $\rightarrow o_i$, param'

 Evaluare(o_i)

 Alegerea probabilistică a μ cromozomi dintre c_1, \dots, c_μ , o_1, \dots, o_μ și
 adăugarea lor în $P(g+1)$

$g++$

endWhile

Programare evolutivă – reprezentare și operatori



- Reprezentare
 - Reală
 - Codează și parametrii mutației (pasul de mutație)
- Populația
 - μ părinți, $\lambda = \mu$ descendenți
- Selecția pentru mutație
 - Deterministă
- Mutația
 - Perturbare Gaussiană
 - Auto-adaptare a parametrilor
- Selecția pentru supraviețuire
 - $(\mu+\mu)$ probabilistică

Programare evolutivă – reprezentare și operatori



- Pp că dorim optimizarea funcției $f: R^n \rightarrow R$
- Reprezentarea cromozomilor
 - 2 părți:
 - Variabile obiect: x_1, x_2, \dots, x_n
 - Pași de mutație: $\sigma_1, \dots, \sigma_n$
 - Completă
 - $(x_1, \dots, x_n, \sigma_1, \dots, \sigma_n)$

Programare evolutivă – reprezentare și operatori



- Selecția părintilor (pentru mutație)
 - Fiecare părinte produce prin mutație un descendant → selecție
 - deterministă
 - ne-bazată pe calitatea (fitnessul) indivizilor

Programare evolutivă – reprezentare și operatori



□ Mutația

- Singurul operator care introduce variație în PE
- Cromozomul $(x_1, \dots, x_n, \sigma_1, \dots, \sigma_n)$
- Modificări de tip Gaussian
 - $\sigma'_i = \sigma_i * (1 + \alpha * N(0, 1))$
 - $x'_i = x_i + \sigma'_i * N(0, 1)$
 - $\alpha \approx 0.2$ – rata de învățare
- Limitări
 - dacă $\sigma' < \varepsilon_0 \rightarrow \sigma' = \varepsilon_0$

Programare evolutivă – reprezentare și operatori



■ Selectia de supraviețuire

- Populația (la momentul t) are μ părinti care produc μ descendeți
- Campionat “fiecare cu fiecare” (round-robin)
 - Fiecare soluție s_i , $i = 1, 2, \dots, \mu^2$ din cei μ părinti și μ descendeți este comparată cu alte q soluții (diferite de s) alese aleator (din aceeași mulțime a părintilor și urmașilor)
 - Pentru fiecare soluție s_i se stabilește de câte ori a câștigat un meci jucat

$$p_i = \sum_{l=1}^q p_{il},$$
$$p_{il} = \begin{cases} 1, & f(s_i) \text{ e mai bun ca } f(s_l) \\ 0, & \text{altfel} \end{cases}$$

- Se aleg cele mai bune μ soluții (cu cele mai multe jocuri câștigate - p_i)
- Parametrul q reglează presiunea de selecție
 - de obicei $q = 10$
- → procesul de căutare este ghidat de calitatea *relativă* a indivizilor



Programare evolutivă – proprietăți

- Cromozomi liniari de aceeași dimensiune
- Algoritmi evolutivi fără recombinare
- Auto-adaptare a parametrilor (similar SE)
- Cadru foarte permisiv: orice reprezentare și mutație poate funcționa bine
 - Mutație uniformă
 - Mutație Cauchy
 - Mutație Lévy



Programare evolutivă – aplicații

- Învățare automată cu mașini cu stări finite
- Optimizare numerică
- Distribuția și planificarea traficului în rețele
- Proiectarea farmaceutică
- Epidemiologie
- Detectiona cancerului
- Planificare militară
- Procesarea semnalelor

Programare genetică



- Aspecte teoretice
- Algoritm
 - Schema generală
 - Reprezentare și operatori
- Exemplu
- Aplicații

Programare genetică – aspecte teoretice



- Propusă
 - În SUA în anii 1990 de către J. Koza
 - Evoluarea de programe → evaluarea unui individ implică execuția programului codat în cromozom
- Căutare
 - **Concurențială**, ghidată de calitatea **absolută** a indivizilor
- Operatori de căutare
 - Selecția
 - Recombinarea **SAU** mutația
- Elemente speciale
 - Cromozomi ne-liniari (arbori sau grafe) și de dimensiuni diferite
 - Pot folosi mutația (dar nu e neapărat necesar)

Programare genetică – schema generală



Inițializare $P(0)$

Evaluare($P(0)$)

$g = 0;$

while (*not* condiție_stop) do
 repeat

 Selectarea a 2 părinți p_1 și p_2 din $P(g)$

 Încrucișare(p_1 , p_2) $\rightarrow o_1$ și o_2

 Mutație(o_1) $\rightarrow o_1^*$

 Mutație(o_2) $\rightarrow o_2^*$

 Evaluare(o_1^*)

 Evaluare(o_2^*)

 Adăugare o_1^* și o_2^* în $P(g+1)$

until $P(g+1)$ este plină

$g++$

endWhile

Programarea genetică – reprezentare și operatori



- Reprezentare
 - Structuri arborescente de dimensiune variabilă
- Populația
 - μ părinti, μ descendenți
- Selectia pentru recombinare
 - Proporțională cu fitness-ul
- Recombinarea
 - Schimbul de sub-arbori
- Mutatia
 - Schimbări aleatoare în arbore
- Selectia pentru supraviețuire
 - Schema generațională - toți descendenții înlocuiesc părinții
 - Schema steady-state – cu elitism

Programarea genetică – reprezentare și operatori



□ Reprezentare

- Potențialele soluții sub forma unor arbori → implicații:

- Indivizi adaptivi

- Dimensiunea cromozomilor nu este prefixată
 - Dimensiunea cromozomilor depinde de adâncimea și factorul de ramificare al arborilor

- Gramatici specifice domeniului problemei de rezolvat

- Necesitatea definirii exacte a unei gramatici reprezentative pentru problema abordată
 - Gramatica trebuie să permită reprezentarea oricărei soluții posibile/potențiale

Programarea genetică – reprezentare și operatori



□ Reprezentare

■ Gramatica conține:

- Setul de terminale specifică toate variabilele și constantele problemei
- Setul de funcții conține toți operatorii care pot fi aplicați terminalelor:
 - Operatori aritmetici (+,-,*,/,sin, cos, log, ...)
 - Operatori de tip Boolean (and, or, not, ...)
 - Operatori de tip instrucțiune (if-then, for, while, set,...)
- Regulile care asigură obținerea unor soluții potențiale valide

■ De ex. arbori care codifică

- Formule logice
- Formule aritmetice
- Programe

Programarea genetică – reprezentare și operatori

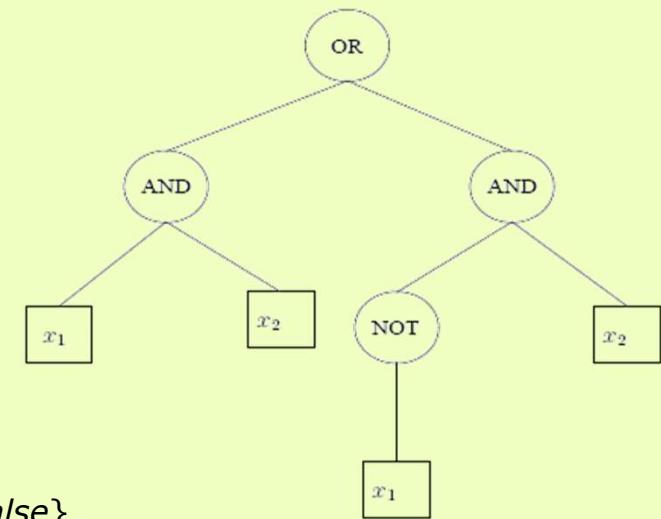


□ Reprezentare

■ exemplu de evoluare a unei expresii logice

- Problemă: să se determine expresia logică identificată prin datele:

	x1	x2	Output
	0	0	0
	0	1	1
	1	0	1
	1	1	0



- Setul de funcții $F = \{AND, OR, NOT\}$
- Setul de terminale $T = \{x_1, x_2\}$, cu $x_1, x_2 \in \{True, False\}$
- Soluție: $(x_1 \text{ AND } \text{NOT } x_2) \text{ OR } (\text{NOT } x_1 \text{ AND } x_2)$

Programarea genetică – reprezentare și operatori



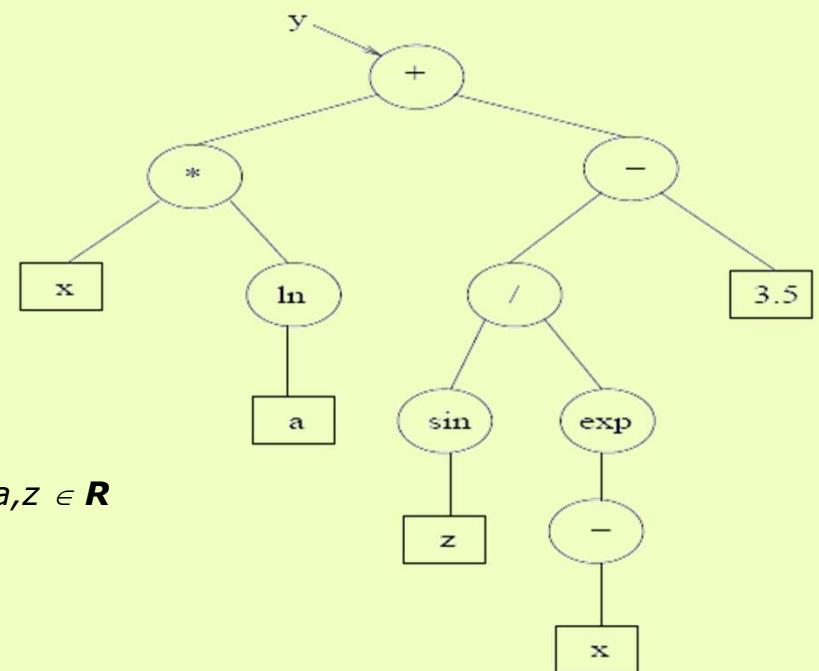
□ Reprezentare

■ exemplu de evoluare a unei expresii aritmetice

- Problemă: să se determine expresia aritmetică identificată prin datele:

x	a	z	Output
1.5	2	0.7	0.52690
0.8	0.25	2	-2.48536
2	1	0.3	-1.21638

- Setul de funcții $F = \{+, -, *, /, \sin, \exp, \ln\}$
- Setul de terminale $T = \{x, a, z, 3.14\}$, cu $x, a, z \in \mathbb{R}$
- Soluție: $y = x * \ln(a) + \sin(z) / \exp(-x) - 3.4$



Programarea genetică – reprezentare și operatori



■ Inițializarea cromozomilor

■ Aleatoare, respectând

- O limită a adâncimii maxime
- Semantica dată de gramatică

■ Problema “bloat” – supravițuirea arborilor foarte mari

■ Metode

□ Metoda *Full* – arbori compleți

- Nodurile de la adâncimea $d < D_{\max}$ se inițializează aleator cu o funcție din setul de funcții F
- Nodurile de la adâncimea $d = D_{\max}$ se inițializează aleator cu un terminal din setul de terminale T

□ Metoda *Grow* – arbori incompleți

- Nodurile de la adâncimea $d < D_{\max}$ se inițializează aleator cu un element din $F \cup T$
- Nodurile de la adâncimea $d = D_{\max}$ se inițializează aleator cu un terminal din setul de terminale T

□ Metoda *Ramped half and half*

- $\frac{1}{2}$ din populație se creează cu metoda *Full*
- $\frac{1}{2}$ din populație se creează cu metoda *Grow*
- Folosind diferite adâncimi

Programarea genetică – reprezentare și operatori



□ Evaluarea cromozomilor

- Necesitatea datelor de antrenament (cazuri de testare)
- Calculul diferenței între ce trebuie obținut și ceea ce se obține de fapt
 - Expresii de tip Boolean → numărul ieșirilor corect prezise
 - Expresii aritmetice → media pătratelor diferențelor între ieșirile corecte și ieșirile prezise
 - Programe → numărul datelor de test corect procesate
- Criteriul de optim → minimizare
- Evaluarea poate penaliza:
 - Soluțiile invalide
 - Dimensiunea (prea mare a) arborilor

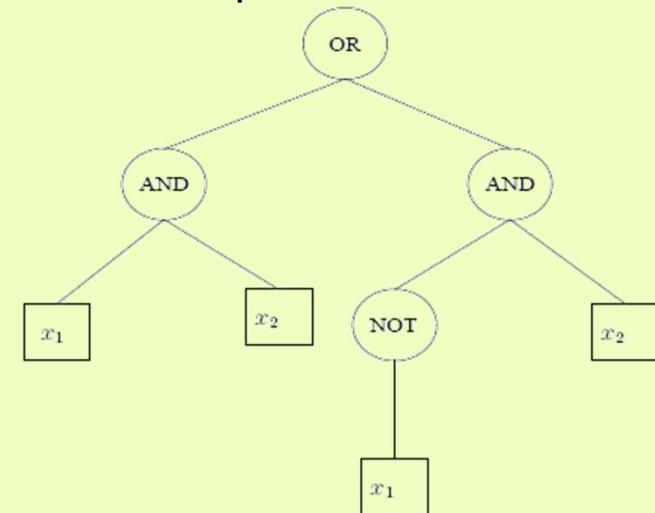
Programarea genetică – reprezentare și operatori



□ Evaluarea cromozomilor – exemplu

- Problemă: să se determine expresia logică identificată prin datele:

- $C = (x_1 \text{ AND } x_2) \text{ OR } (\text{NOT } x_1 \text{ AND } x_2)$



	x1	x2	Output real	Output calculat	Eroare = output real – output calculat
	0	0	0	0	0
	0	1	1	1	0
	1	0	1	0	1
	1	1	0	1	1
suma					2

Programare genetică – reprezentare și operatori



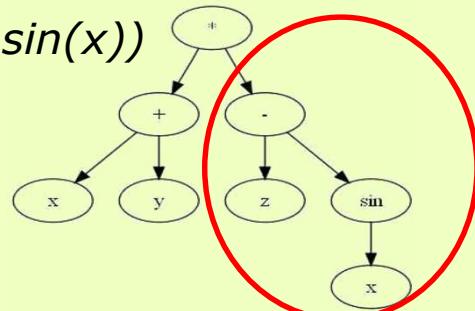
- Selecția pentru reproducere
 - Bazată pe fitness
 - Selecție proporțională (bazată pe fitness)
 - Selecție bazată pe ranguri
 - Selecție prin turnir
 - În populații foarte mari
 - Se acordă ranguri indivizilor (pe bază de fitness) și se stabilesc mai multe grupe
 - Grupa 1: cei mai buni $x\%$ din populație
 - Grupa 2: restul de $(100-x)\%$ din populație
 - Alegerea va fi făcută din:
 - grupa 1 în 80% din cazuri
 - grupa 2 în 20% din cazuri
 - Ex.
 - $\mu = 1000, x = 32\%$
 - $\mu = 2000, x = 16\%$
 - $\mu = 4000, x = 8\%$
 - $\mu = 8000, x = 4\%$

Programare genetică – reprezentare și operatori

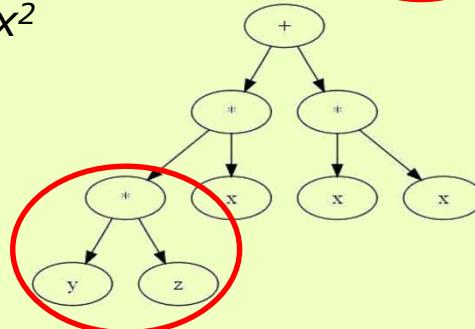


- Recombinarea (încrucișarea)
 - Cu punct de tăietură

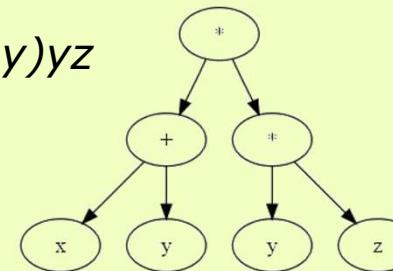
$$p_1 = (x+y)*(z-\sin(x))$$



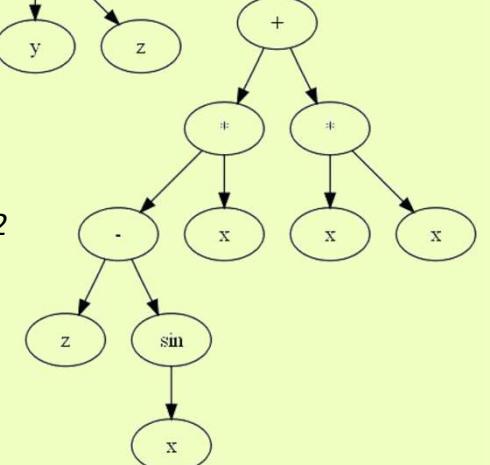
$$p_2 = xyz + x^2$$



$$f_1 = (x+y)yz$$



$$f_2 = (z-\sin(x))x + x^2$$



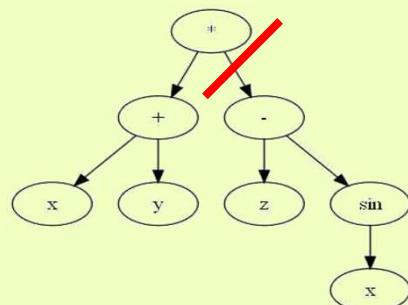
Programare genetică – reprezentare și operatori



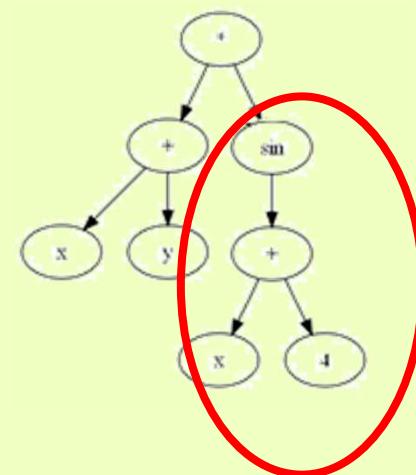
□ Mutăție

- Mutăție de tip Koza → Înlocuirea unui nod (intern sau frunză) cu un nou sub-arbore

$$p = (x+y)*(z - \sin(x))$$



$$f = (x+y)*\sin(x+4)$$





Programarea genetică – proprietăți

- ❑ Folosirea cromozomilor ne-liniari
- ❑ Necesită lucrul cu populații foarte numeroase
 - Algoritmi înceții
- ❑ Comparație AG și PG
 - Forma cromozomilor
 - ❑ AG – cromozomi liniari
 - ❑ PG – cromozomi ne-liniari
 - Dimensiunea cromozomilor
 - ❑ AG – fixă
 - ❑ PG – variabilă (în adâncime sau lățime)
 - Schema de creare a descendenților
 - ❑ AG – încrucișare și mutație
 - ❑ PG – încrucișare sau mutație



Programare genetică – aplicații

□ Învățare automată

■ probleme de regresie

- Predicții de curs valutar
- Previziunea vremii

■ probleme de clasificare (învățare supervizată)

- Proiectarea circuitelor digitale
- Recunoașterea imaginilor
- Diagnosticare medicală

■ probleme de clusterizare (învățare nesupervizată)

- Analiza secvențelor de ADN
- Cercetări și studii de piață (segmentarea pieței)
- Analiza rețelelor sociale
- Analiza rezultatelor căutărilor în Internet



Programare genetică – variante

- ❑ Linear Genetic Programming
- ❑ Gene Expression Programming
- ❑ Cartesian Genetic Programming
- ❑ Grammatical Evolution
- ❑ Multi Expression Programming
- ❑ Traceless Genetic Programming

- ❑ Mai multe detalii despre GP și variantele sale în cursurile dedicate învățării automate



Recapitulare

Generational GA	Steady – state GA
<pre> Initialization(pop) Evaluation(pop) g = 0; While (not stop_condition) do Repeat p1=Selection(pop) p2=Selection(pop) Crossover(p1,p2) =>o1 and o2 Mutation(o1) => o1* Mutation(o2) => o2* Evaluation(o1*) Evaluation(o2*) Add o1* and o2* into popAux Until popAux is full. pop ← popAux EndWhile </pre>	<pre> Initialization(pop) Evaluation(pop) While (not stop_condition) do p1=Selection(pop) p2=Selection(pop) Crossover(p1,p2) =>o1 and o2 Mutation(o1) => o1* Mutation(o2) => o2* Evaluation(o1*) Evaluation(o2*) Best(o1*,o2*) replaces Worst(pop) EndWhile </pre>
<pre> SE Initialization(pop) Evaluation(pop) g = 0; While (not stop_condition) do Repeat p1=Selection(pop) p2=Selection(pop) Crossover(p1,p2) =>o1 Mutation(o1,param) => o1*, param* Evaluation(o1*) Add o1* into popAux Until popAux contains λ cromozoms pop ← Bestμ(popAux) //SE(μ,λ) pop ← Bestμ(popUpopAux) //SE(μ+λ) EndWhile </pre>	<pre> PE Initialization(pop) Evaluation(pop) g = 0; While (not stop_condition) do For all cromozoms c from pop Mutation(c,param) => o1*, param* Evaluation(o1*) Add o1* into popAux pop ← RoundRobin(popAux) EndWhile </pre>



Recapitulare

- Reprezentare și fitness
 - Dependente de problemă
- Operatori de căutare
 - Selectia pentru reproducere și pentru supraviețuire
 - Dependentă de fitness
 - Independentă de reprezentare
 - Încrucișarea și mutația
 - Dependente de reprezentare
 - Independente de fitness
 - Probabilitatea de încrucișare
 - Acționează la nivel de cromozom
 - Probabilitatea de mutație
 - Acționează la nivel de genă

-
- ❑ Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan - www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ



Rezolvarea problemelor de căutare

Strategii de căutare informată
algoritmi inspirați de natură

Laura Dioșan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Materiale de citit și legături utile

- ❑ capitolul 16 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- ❑ *James Kennedy, Russel Eberhart, Particle Swarm Optimisation, Proceedings of IEEE International Conference on Neural Networks. IV.* pp. 1942–1948, 1995
(05_ACO_PSO/PSO_00.pdf)
- ❑ *Marco Dorigo, Christian Blum, Ant colony optimization theory: A survey, Theoretical Computer Science 344 (2005) 243 – 27* (05_ACO_PSO/Dorigo05_ACO.pdf)

Căutare locală

□ Tipologie

- Căutare locală simplă - se reține o singură stare vecină
 - Căutare tabu → reține lista soluțiilor recent vizitate
 - Hill climbing → alege cel mai bun vecin
 - Simulated annealing → alege probabilistic cel mai bun vecin
- Căutare locală în fascicol (beam local search) – se rețin mai multe stări (o populație de stări)
 - Algoritmi evolutivi
 - Optimizare bazată pe comportamentul de grup (Particle swarm optimisation)
 - Optimizare bazată pe furnici (Ant colony optimisation)

Algoritmi inspirați de natură

- Care este cea mai bună metodă de rezolvare a unei probleme?
 - Creierul uman
 - A creat roata, mașina, orașul, etc
 - Mecanismul evoluției
 - A creata creierul (mintea) umană
- Simularea naturii
 - Cu ajutorul mașinilor → rețelele neuronale artificiale simulează mintea umană
 - Mașini de zbor, computere bazate pe ADN, computere cu membrane
 - Cu ajutorul algoritmilor
 - algoritmii evolutivi simulează evoluția naturii
 - algoritmii inspirați de comportamentul de grup simulează adaptarea colectivă și procesele sociale dintr-un colectiv
 - Particle Swarm Optimisation (PSO)
 - <http://www.youtube.com/watch?feature=endscreen&v=JhZKc1Mgub8&NR=1>
 - <http://www.youtube.com/watch?v=ulucJnxT7B4&feature=related>
 - <https://www.youtube.com/watch?v=TWqx57CR69c>
 - Ant Colony Optimisation (ACO)
 - http://www.youtube.com/watch?v=jrW_TTxP1ow

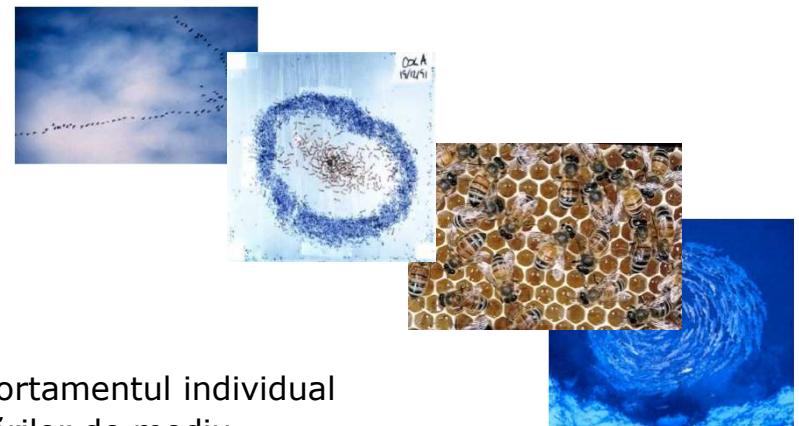
Algoritmi inspirați de natură

■ Inteligența de grup (colectivă)

- O populație de indivizi care interacționează în scopul atingerii unor obiective prin adaptarea colectivă la un mediu global sau local

- Metaforă computațională inspirată de:

- zborul păsărilor în formă de V
 - furnicile aflate în căutarea hranei
 - roiuurile de albine care își construiesc cuibul
 - bancurile de pești



- deoarece

- controlul este distribuit între mai mulți indivizi
 - comunicarea între indivizi se realizează local
 - comportamentul sistemului transcede din comportamentul individual
 - sistemul este robust și se poate adapta schimbărilor de mediu

- Insecte sociale (2% din totalul insectelor):

- Furnici
 - 50% din insectele sociale
 - 1 furnică are aprox. 1 mg → Greutatea totală a furnicilor ≈ greutatea totală a oamenilor
 - Trăiesc de peste 100 milioane de ani (oamenii trăiesc de aprox. 50 000 de ani)
 - Termite
 - Albine

Algoritmi inspirați de natură

□ Grup (roi - Swarm)

- O colecție aparent dezorganizată de indivizi care se mișcă tinzând să se grupeze, dar fiecare individ pare să se miște într-o direcție oarecare
- În interiorul colecției apar anumite procese sociale
- Colecția este capabilă să efectueze sarcini complexe
 - fără nici o ghidare sau control extern
 - fără nici o coordonare centrală
- Colecția poate atinge performanțe care nu pot fi atinse de indivizi în izolare

□ Adaptare colectivă → auto-organizare

- Mulțimea mecanismelor dinamice care generează un comportament global ca rezultat al interacțiunii componentelor individuale
- Regulile care specifică interacțiunea sunt executate doar pe baza unor informații locale, fără referințe globale
- Comportamentul global este o proprietate emergentă a sistemului (și nu una impusă din exterior)

PSO

- ❑ Aspecte teoretice
- ❑ Algoritm
- ❑ Exemplu
- ❑ Proprietăți
- ❑ Aplicații

PSO – aspecte teoretice

- Propusă
 - de Kennedy și Eberhart în 1995 <http://www.particleswarm.info/>
 - Inspirată de comportamentul social al stolurilor de păsări și al bancurilor de pești
- Căutare
 - **Cooperativă**, ghidată de calitatea **relativă** a individelor
- Operatori de căutare
 - Un fel de mutație

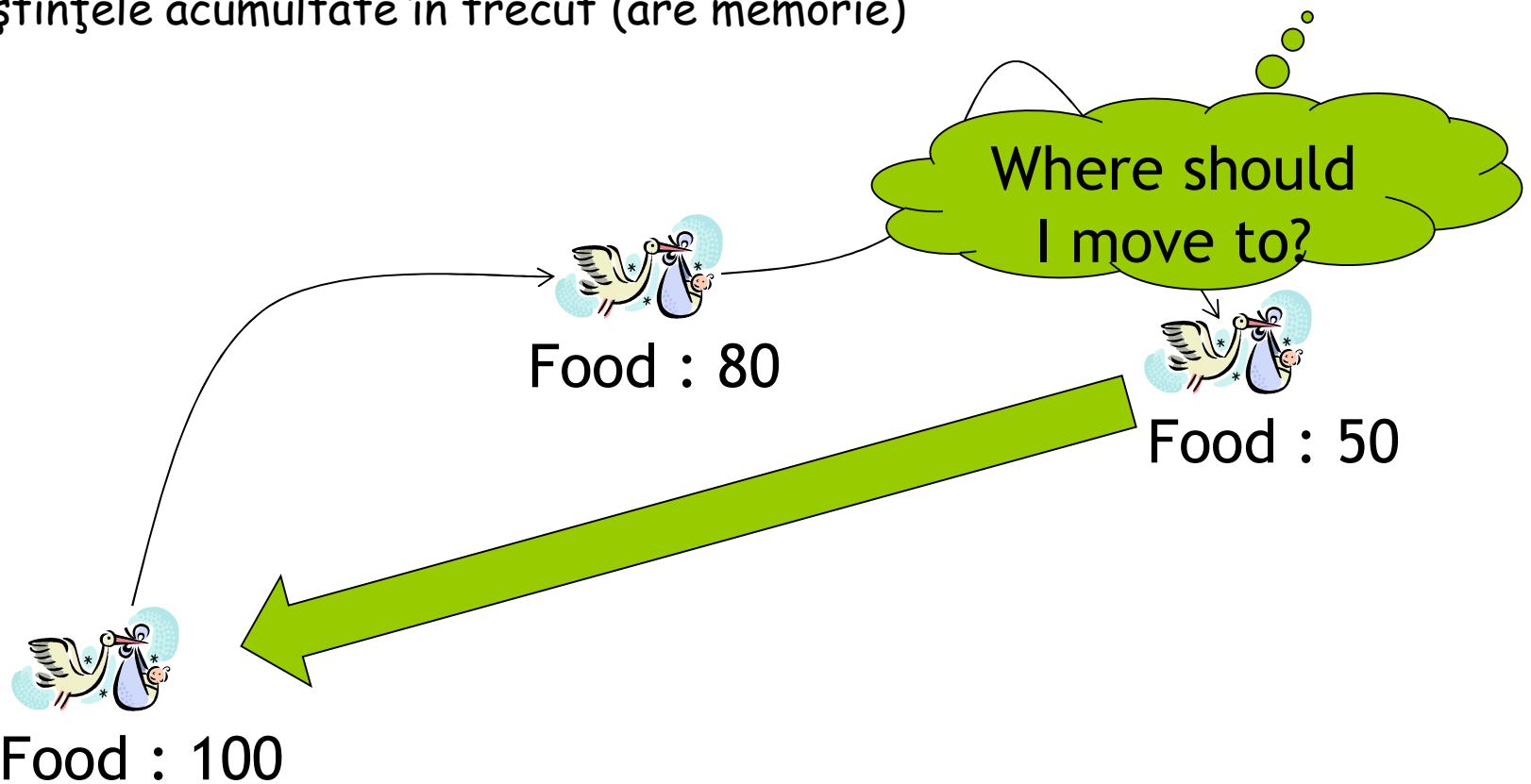
PSO – aspecte teoretice

■ Elemente speciale

- Metodă de optimizare bazată pe:
 - populații (\approx AG) de particule (\approx cromozomi) care caută soluția optimă
 - cooperare ($\hat{\in}$ loc de competiție ca $\hat{\in}$ cazul AG)
- Fiecare particulă:
 - Se mișcă (deplasează $\hat{\in}$ spațiul de căutare) și are o viteză (viteză \approx mutare pt că timpul este discret)
 - Reține locul (poziția) unde a obținut cele mai bune rezultate
 - Are asociată o vecinătate de particule
- Particulele cooperează
 - Schimbă informații (legate de descoperirile făcute $\hat{\in}$ locurile deja vizitate) între ele
 - Fiecare particulă știe fitnessul vecinilor ei a.î. poate folosi poziția celui mai bun vecin pentru a-și ajusta propria viteză

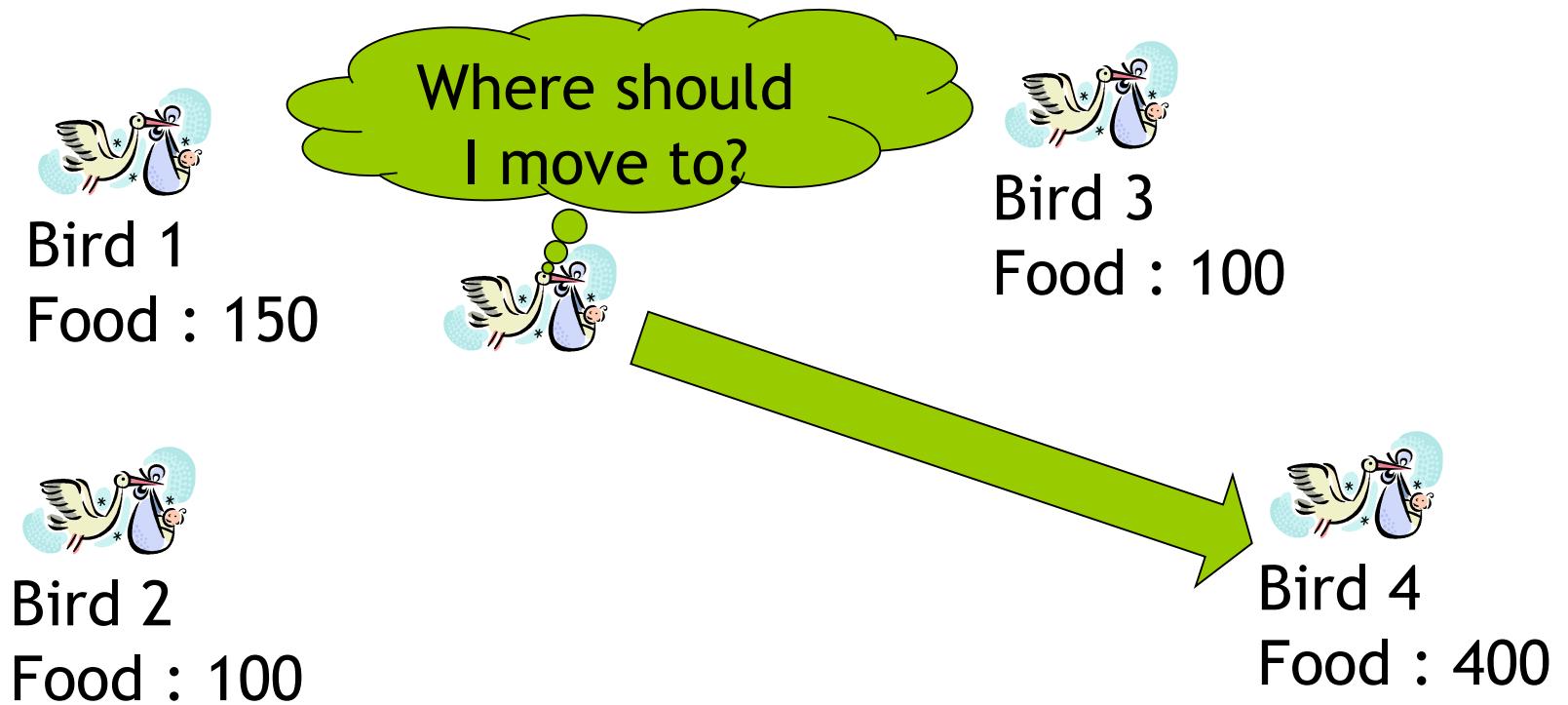
PSO – aspecte teoretice

Ideea de bază: comportament cognitiv → un individ își amintește cunoștințele acumulate în trecut (are memorie)



PSO – aspecte teoretice

Ideea de bază: comportament social → un individ se bazează și pe cunoștințele celorlalți membri ai grupului



PSO – algoritm

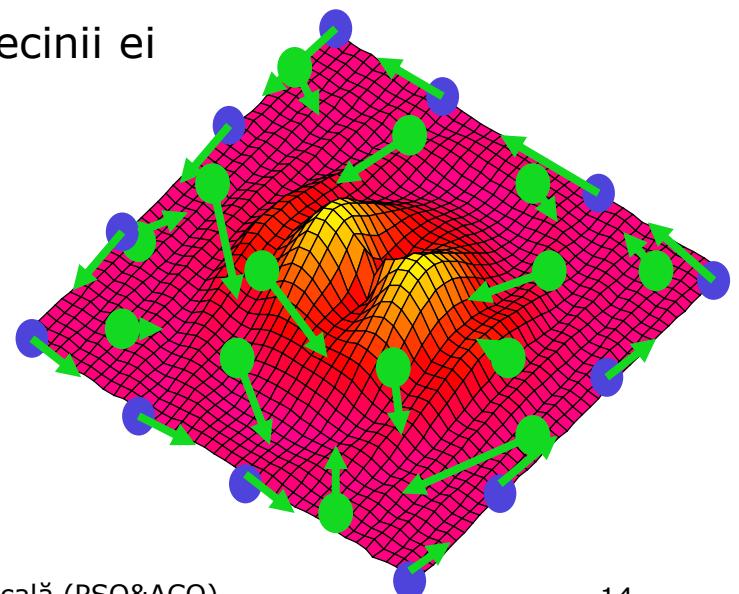
□ Schema generală

1. Crearea populației inițiale de particule
 - Poziții aleatoare
 - Viteze nule/aleatoare
2. Evaluarea particulelor
3. Pentru fiecare particulă
 - Actualizarea memoriei
 - Stabilirea celei mai bune particule din swarm (g_{Best}) / dintre particulele vecine (I_{Best})
 - Stabilirea celei mai bune poziții (cu cel mai bun fitness) în care a ajuns până atunci – p_{Best}
 - Modificarea vitezei
 - Modificarea poziției
4. Dacă nu se îndeplinesc condițiile de oprire, se revine la pasul 2, altfel STOP

PSO – algoritm

1. Crearea populației inițiale de particule

- Fiecare particulă are asociată
 - o poziție – potențială soluție a problemei
 - o viteză – modifică o poziție în altă poziție
 - o funcție de calitate (fitness)
- Fiecare particulă trebuie să poată:
 - interacționa (schimba informații) cu vecinii ei
 - memora o poziție precedentă
 - utilizează informațiile pentru a lua decizii
- Inițializarea particulelor
 - poziții aleatoare
 - viteze nule/aleatoare



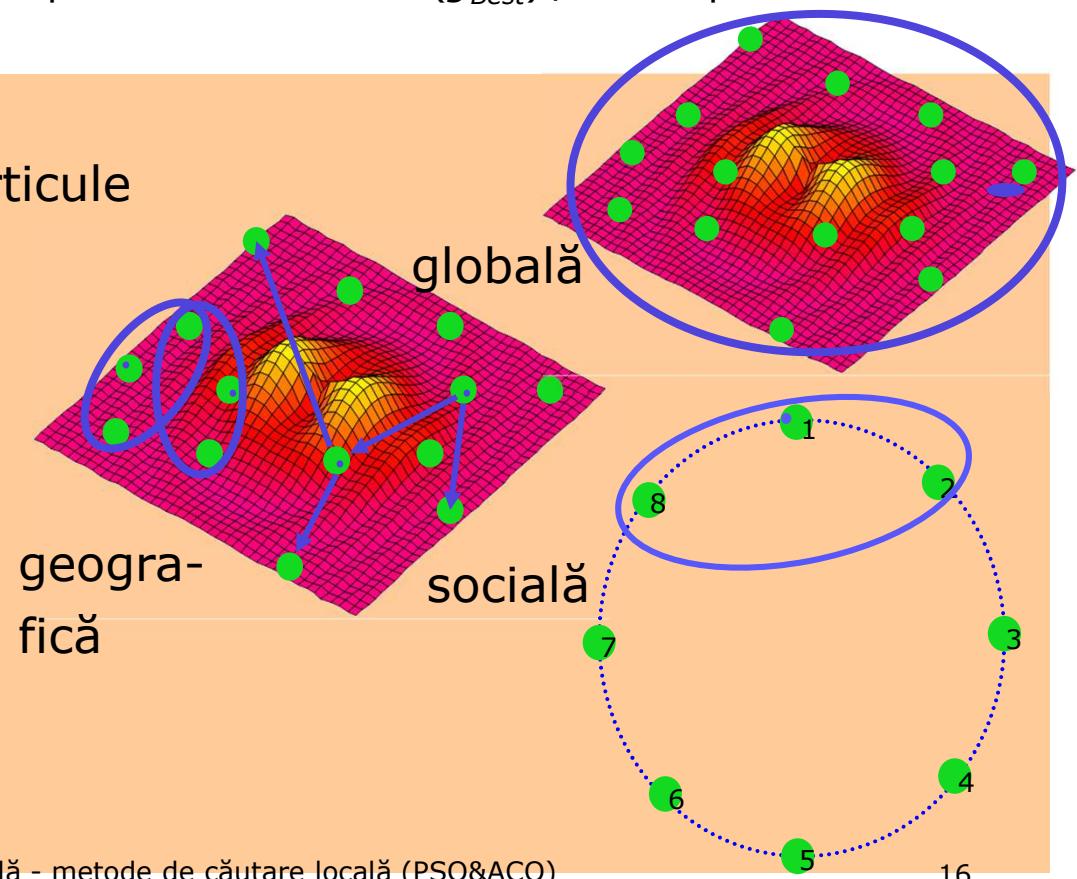
PSO – algoritm

2. Evaluarea particulelor
 - dependentă de problemă

PSO – algoritm

3. Pentru fiecare particulă x

- Actualizarea memoriei
 - Stabilirea celei mai bune particule din swarm (g_{Best}) / dintre particulele vecine (l_{Best})
- Vecinătate a unei particule
 - Întinderea vecinătății
 - Globală
 - Locală
 - Tipul vecinătății
 - Geografică
 - Socială
 - Circulară

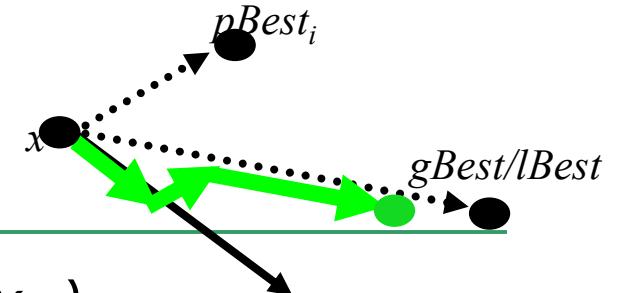


PSO – algoritm

3. Pentru fiecare particulă x

- Actualizarea memoriei
 - Stabilirea celei mai bune particule din swarm (g_{Best}) / dintre particulele vecine (l_{Best})
 - Stabilirea celei mai bune poziții (cu cel mai bun fitness) în care a ajuns până atunci – p_{Best}

PSO – algoritm



3. Pentru fiecare particulă $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{iD})$

■ Modificarea vitezei \mathbf{v} și a poziției \mathbf{x} (pe fiecare dimensiune)

- $v_{id} = w * v_{id} + c_1 * rand() * (p_{Best\ d} - x_{id}) + c_2 * rand() * (g_{Best\ d} - x_{id})$
- $X_{id} = X_{id} + v_{id}$
- unde:
 - $i=1, N$ (N – nr total de particule); $d = 1, D$
 - w – factor de inerție (Shi, Eberhart)
 - $w * v_{id}$ – termen inerțial → forțează particula să se deplaseze în aceeași direcție ca și până acum (tendință curajoasă – *audacious*)
 - balansează căutarea între explorare globală (w mare) și locală (w mic).
 - poate fi constantă sau descrescătoare (pe măsura „îmbătrânirii” grupului)
 - c_1 - factor de învățare cognitiv
 - $c_1 * rand() * (p_{Best\ d} - x_{id})$ – termen cognitiv → forțează particula să se deplaseze spre cea mai bună poziție atinsă până atunci (tendință de conservare)
 - c_2 - factor de învățare social
 - $c_2 * rand() * (g_{Best\ d} - x_{id})$ – termen social → forțează particula să se deplaseze spre cea mai bună poziție a vecinilor; spirit de turmă, de urmăritor
 - Cei doi factori c_1 și c_2 pot fi egali sau diferiți ($c_1 > c_2$ și $c_1 + c_2 < 4$ – Carlisle, 2001)
- Fiecare componentă a vectorului vitezelor este restricționată la un interval: $[-v_{max}, v_{max}]$ pentru a asigura păstrarea particulelor în spațiul de căutare.

PSO – proprietăți

□ Principii în PSO:

- proximitate – grupul trebuie să efectueze calcule în spațiu și timp
- calitate – grupul trebuie să fie capabil să răspundă la factorii calitativi ai mediului
- stabilitate – grupul nu trebuie să își schimbe comportamentul la fiecare sesizare a mediului
- adaptabilitate – grupul trebuie să fie capabil să își schimbe comportamentul atunci când costul schimbării nu este prohibit.

□ Diferențe față de EC:

- nu există un operator de recombinare directă – schimbul de informație are loc în funcție de experiența particulei și în funcție de cea a celui mai bun vecin și nu în funcție de părinții selectați pe baza fitness-ului.
- Update poziție ~ similar cu mutația
- Nu se folosește selecția – supraviețuirea nu este legată de fitness.

□ Versiuni ale algoritmului de tip PSO

- PSO binar discret
- PSO cu mai mulți termeni de învățare socială
- PSO cu particule eterogene
- PSO ierarhic

PSO – proprietăți

□ PSO discret (binar)

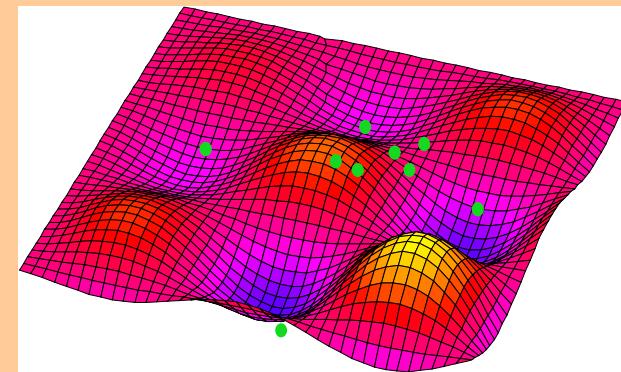
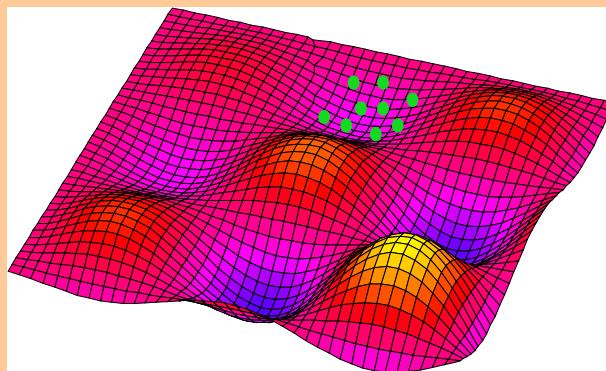
- Versiune a PSO pentru spațiu de căutare discret
- Poziția unei particule
 - Potențială soluție a problemei → string binar
 - Se modifică în funcție de viteza particulei
- Viteza unei particule
 - element din spațiu continuu
 - se modifică conform principiilor de la PSO standard
 - se interpretează ca probabilitatea de modificare a bit-ului corespunzator din poziția particulei

$$x_{ij} = \begin{cases} 1, & \text{dacă } \tau < s(v_{ij}) \\ 0, & \text{altfel} \end{cases}, \text{ unde } s(v_{ij}) = \frac{1}{1 + e^{-v_{ij}}}$$

PSO – proprietăți

□ Pericole

- Particulele tind să se grupeze în același loc
 - Converg prea repede și nu reușesc să evadzeze dintr-un optim local
 - Soluția:
 - Reinițializarea unor particule



- Deplasarea particulelor spre regiuni nefezabile

PSO – proprietăți

- Analiza algoritmilor de tip PSO
 - Comportamentul dinamic al grupului poate fi analizat cu ajutorul a 2 indici
 - Indicele de dispersie
 - Măsoară gradul de împrăștiere a particulelor în jurul celei mai bune particule din grup
 - Media distanțelor absolute (pe fiecare dimensiune) între fiecare particulă și particula cea mai bună
 - Explică gradul de acoperire (întins sau restrâns) a spațiului de căutare
 - Indicele vitezei
 - Măsoară viteza de mișcare a grupului într-o iterație
 - Media vitezelor absolute
 - Explică cum (agresiv sau lent) se mișcă grupul

PSO – aplicații

- ❑ Controlul și proiectarea antenelor
- ❑ Aplicații biologice, medicale, farmaceutice
 - Analiza tremurului în boala Parkinson
 - Clasificare cancerului
 - Predicția structurii proteinelor
- ❑ Comunicare în rețele
- ❑ Optimizare combinatorială
- ❑ Optimizări financiare
- ❑ Analiza imaginilor și analiza video
- ❑ Robotică
- ❑ Planificare
- ❑ Securitatea rețelelor, detecția intrușilor, criptografie, criptanaliză
- ❑ Procesarea semnalelor

ACO

- ❑ Aspecte teoretice
- ❑ Algoritm
- ❑ Exemplu
- ❑ Proprietăți
- ❑ Aplicații

ACO – aspecte teoretice

- Propusă
 - de Colorni și Dorigo în 1991 inițial pentru rezolvarea problemelor de optimizare discretă – gen TSP – (ca o contrapartidă pentru AG) –
<http://iridia.ulb.ac.be/~mdorigo/ACO/about.html>
 - inspirată de comportamentul social al furnicilor în căutarea unui drum între cuib și o sursă de hrana
 - De ce furnici?
 - Munca în colonie (de la câteva furnici până la milioane de furnici)
 - Diviziunea muncii
 - Au comportament social complex
- Căutare
 - **Cooperativă**, ghidată de calitatea **relativă** a indivizilor
- Operatori de căutare
 - Constructivi, adăugând elemente în soluție

ACO – aspecte teoretice

□ Elemente speciale

- Problema de optimizare trebuie transformată într-o problemă de identificare a drumului optim într-un graf orientat
- Furnicile construiesc soluția plimbându-se prin graf și depunând pe muchii feromoni
- Metodă de optimizare bazată pe:
 - Colonii (\approx AG) de furnici (în loc de cromozomi) care caută soluția optimă
 - cooperare (în loc de competiție ca în cazul AG)
- Fiecare furnică:
 - Se mișcă (deplasează în spațiul de căutare) și depune o cantitate de feromon pe drumul parcurs
 - Reține drumul parcurs
 - Alege drumul pe care să-l urmeze în funcție de
 - Feromonul existent pe drum
 - Informația euristică asociată acelui drum
 - Cooperă cu celelalte furnici prin urma de feromon corespunzătoare unui drum care
 - depinde de calitatea soluției și
 - se evaporă cu trecerea timpului

ACO – aspecte teoretice

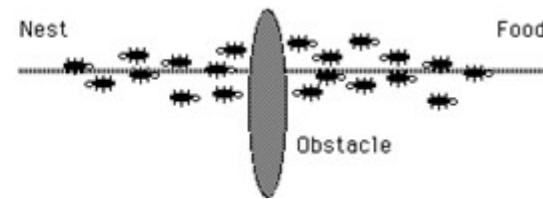
- Furnici naturale
 - O colonie de furnici pleacă în căutarea hranei



ACO – aspecte teoretice

□ Furnici naturale

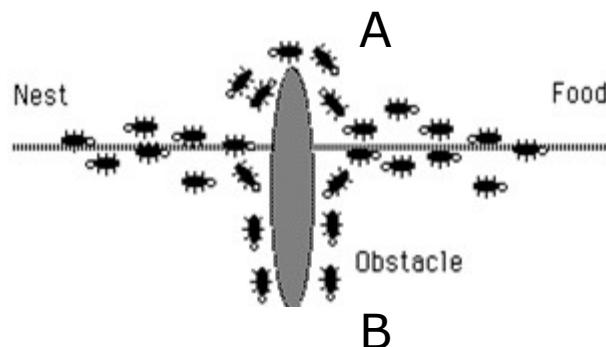
- O colonie de furnici pleacă în căutarea hranei
- La un moment dat, în drumul lor apare un obstacol



ACO – aspecte teoretice

■ Furnici naturale

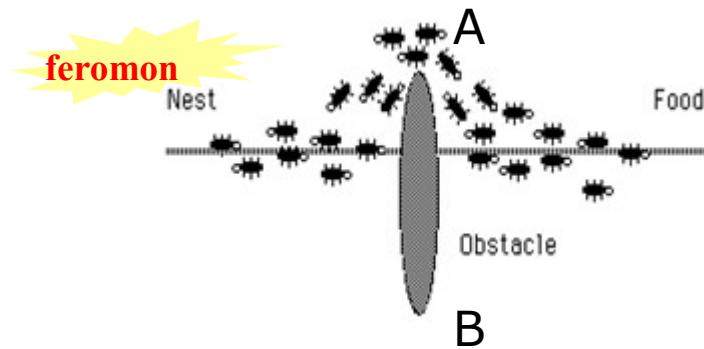
- O colonie de furnici pleacă în căutarea hranei
- La un moment dat, în drumul lor apare un obstacol
- Furnicile vor ocoli obstacolul fie pe ruta A, fie pe ruta B



ACO – aspecte teoretice

■ Furnici naturale

- O colonie de furnici pleacă în căutarea hranei
- La un moment dat, în drumul lor apare un obstacol
- Furnicile vor ocoli obstacolul fie pe ruta A, fie pe ruta B
- Pentru că ruta A este mai scurtă, furnicile de pe acest drum vor face mai multe ture, deci vor lăsa mai mult feromon
- Concentrația de feromon va crește mai accelerat pe ruta A decât pe ruta B a.î. furniciile de pe ruta B vor alege (pe bază de miros) ruta A
- Pentru că pe ruta B nu vor mai merge furnici și pentru că feromonii sunt volatili, urma furnicilor de pe ruta B va dispărea
- Deci, furnicile se vor plimba doar pe cel mai scurt drum (ruta A)



ACO – aspecte teoretice

- Furnicile artificiale seamănă cu furnicile reale
 - navighează de la cuib spre sursa de hrana
 - descoperă drumul mai scurt pe baza urmei de feromon
 - fiecare frunică execută mișcări aleatoare
 - fiecare frunică depozitează feromon pe drumul parcurs
 - fiecare frunică detectează drumul urmat de "furnica șefă", înclinând să-l urmeze
 - creșterea cantității de feromon de pe un drum îi crește acestuia probabilitatea de a fi urmat de tot mai multe furnici
- dar au anumite îmbunătățiri:
 - au memorie
 - pentru a reține acțiunile efectuate → au stare proprie (cu istoricul acțiunilor efectuate)
 - se pot întoarce la cuib (și pe baza urmei de feromon)
 - nu sunt complet oarbe – pot aprecia calitatea spațiului vecin
 - execută mișcări într-un timp discret
 - depun feromoni și în funcție de calitatea soluției identificate

ACO – aspecte teoretice

- Urma de feromon are rolul
 - unei memorii colective dinamice distribuită (în colonie)
 - unui depozit cu cele mai recente experiențe de căutare a hranei ale furnicilor din colonie

- Furnicile pot comunica indirect și se pot influența reciproc
 - prin modificarea și mirosirea acestui depozit chimic
 - în vederea identificării celui mai scurt drum de la cuib până la hrană

ACO – algoritm

- Cât timp nu s-a ajuns la nr maxim de iterații
 - 1. Inițializare
 - 2. Cât timp nu s-a parcurs numărul necesar de pași pentru identificarea soluției
 - Pentru fiecare furnică din colonie
 - Se mărește soluția parțială cu un element (furnica execută o mutare)
 - Se modifică local urma de feromon corespunzător ultimului element adăugat în soluție
 - 3. Se modifică urma de feromon de pe drumurile parcuse de
 - Toate furnicile/cea mai bună furnică
 - 4. Se returnează soluția găsită de cea mai bună furnică

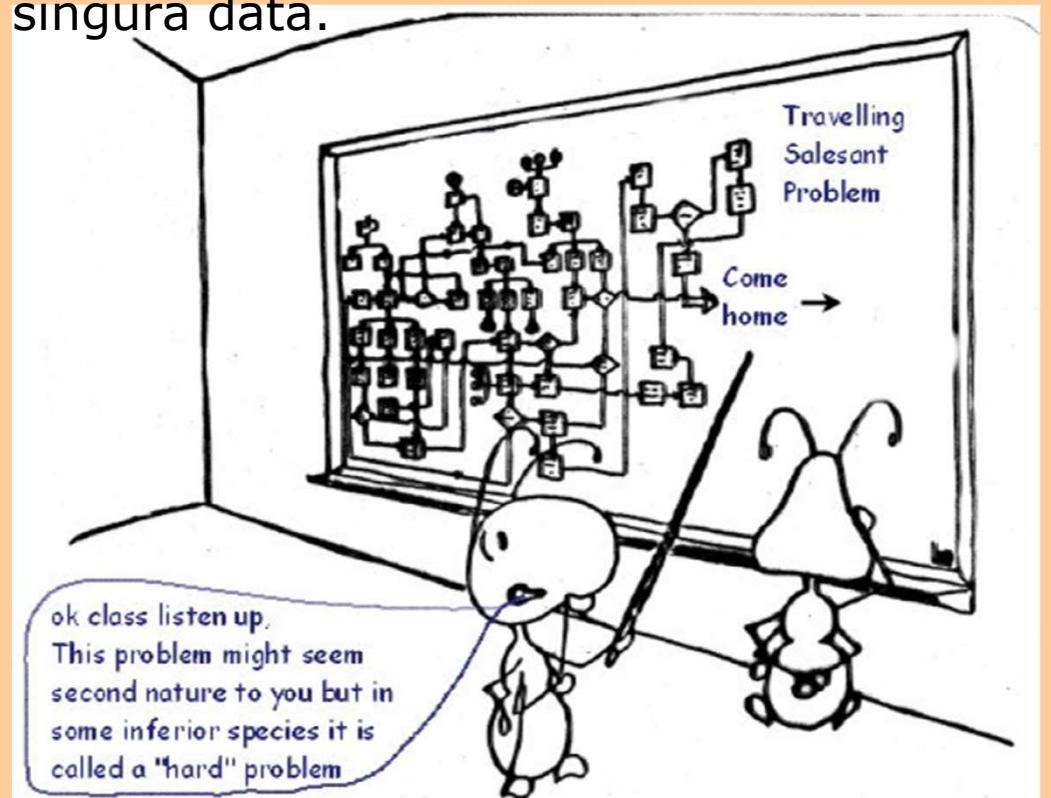
ACO – algoritm

- 3 versiuni principale în funcție de:
 - Regulile de tranziție de la o stare la alta (regulile de deplasare a furnicilor)
 - Momentul la care furnicile depun feromon:
 - pe parcursul construcției soluției
 - la sfârșitul creării unei soluții
 - Furnica deponentă de feromon
 - Toate furnicile
 - Doar cea mai bună furnică
- Versiuni:
 - Ant system (AS)
 - **Toate** furnicile depun feromon **după** construirea unei soluții **complete** (modificare globală colectivă)
 - MaxMin Ant System (MMAS) ≈ AS, dar
 - doar **cea mai bună** frunică depune feromon **după** construirea unei soluții **complete** (modificare globală a leader-ului)
 - feromonul depus este **limitat** la un interval dat
 - Ant Colony System (ACO) ≈ AS, dar
 - **toate** furnicile depun feromon **la fiecare pas** în construcția soluției (modificare locală colectivă)
 - doar **cea mai bună** furnică depune feromon după construirea unei soluții complete (modificare globală a leader-ului)

ACO – exemplu

■ Problema comisului voiajor

- *Travelling salesman problem - TSP*
- să se găsească un drum care să treacă prin n orașe (inclusiv între primul și ultimul) astfel încât costul să fie minim și fiecare oraș să fie vizitat o singură dată.



ACO – exemplu

1. Inițializare:

- $t := 0$ (timpul)
- pentru fiecare muchie (i,j) se inițializează
 - $\tau_{ij}^{(t)} = c$ (intensitatea urmei de feromon pe muchia (i,j) la momentul t)
 - $\Delta\tau_{ij} = 0$ (cantitatea de feromon lăsată pe muchia (i,j) de către toate furnicile)
- se plasează aleator m furnici în cele n noduri-oraș ($m \leq n$)
- fiecare furnică își modifică memoria (lista cu orașele vizitate)
 - adaugă în listă orașul din care pleacă în căutare

ACO – exemplu pentru TSP

2. Cât timp nu s-a parcurs numărul necesar de pași pentru construcția soluției (nr de pași = n)

■ Pentru fiecare furnică din colonie

- Se mărește soluția parțială cu un element (furnica execută o mutare)
 - fiecare furnică k (aflată în orașul i) alege următorul oraș pe care îl vizitează (j) astfel:

$$j = \begin{cases} \arg \max_{l \in permis_k} \{[\tau_{il}]^\alpha [\eta_{il}]^\beta\}, & \text{daca } q \leq q_0 \\ J, & \text{altfel} \end{cases}$$

▪ unde:

- q – număr aleator uniform distribuit în $[0,1]$
- q_0 – parametru, $0 \leq q_0 \leq 1$ ($q_0 = 0 \rightarrow$ AS/MMAS, altfel ACO)
- J este un oraș selectat cu probabilitatea

Regula aleatoare proporțională

Regula pseudo-aleatoare proporțională

$$p_{ij}^k(t) = \begin{cases} \frac{[\tau_{ij}^{(t)}]^\alpha [\eta_{ij}]^\beta}{\sum_{s=permis_k(t)} [\tau_{is}^{(t)}]^\alpha [\eta_{is}]^\beta}, & j - permis \\ 0, & altfel \end{cases}$$

unde:

- p_{ij}^k – probabilitatea de tranziție a furnicii k situată în orașul i spre orașul j
- $\eta_{ij} = \frac{1}{d_{ij}}$ – vizibilitatea din orașul i spre orașul j (atractivitatea alegerii muchiei (i,j))
- $permis_k$ – orașele pe care le mai poate vizita a k -a furnică la momentul t
- α – controlează importanța urmei (câte furnici au mai trecut pe muchia respectivă)
- β – controlează importanța vizibilității (cât de aproape se află următorul oraș)

ACO – exemplu pentru TSP

2. Cât timp nu s-a parcurs numărul necesar de pași pentru construcția soluției (nr de pași = n)

- Pentru fiecare furnică din colonie
 - Se mărește soluția parțială cu un element (furnica execută o mutare)
 - Se modifică local urma de feromon lăsată de fiecare furnică pe ultimul element adăugat în soluție

$$\tau_{ij}^{(t+1)} = (1 - \varphi)\tau_{ij}^{(t)} + \varphi * \tau_0$$

- unde:
 - φ – coeficient de degradare a feromonului; $\varphi \in [0, 1]$; pentru $\varphi = 0 \rightarrow$ AS/MMAS, altfel ACO
 - τ_0 – valoarea inițială a feromonului
 - (i, j) – ultima muchie parcursă de furnică

ACO – exemplu pentru TSP

3. Se modifică urma de feromon de pe
 - drumurile parcuse de toate furnicile (AS)
 - cel mai bun drum (ACO)
 - cel mai bun drum parcurs de cea mai bună furnică (MMAS)

ACO – exemplu pentru TSP

3. Se modifică urma de feromon de pe

- **drumurile parcuse de toate furnicile (AS)**

- Pentru fiecare muchie

- Se calculează cantitatea unitară de feromoni lăsată de a k -a furnică pe muchia (ij)

$$\Delta \tau_{ij}^k = \begin{cases} \frac{Q}{L_k} & - \text{dacă a } k\text{-a furnică a folosit muchia } (i,j) \\ 0 & \end{cases}$$

- Q – cantitatea de feromon lăsată de o furnică.
 - L_k – lungimea (costul) turului efectuat de a k -a furnică
- Se calculează cantitatea totală de feromoni de pe muchia (ij) $\Delta \tau_{ij} = \sum_{k=1}^m \Delta \tau_{ij}^k$
- Se calculează intensitatea urmei de feromoni ca sumă între evaporarea feromonilor vechi și feromonul nou lăsat $\tau_{ij}^{(t+n)} = (1-\rho) * \tau_{ij}^{(t)} + \Delta \tau_{ij}$
 - unde ρ ($0 < \rho < 1$) – coeficientul de evaporare a urmei de feromon între 2 tururi complete

ACO – exemplu pentru TSP

3. Se modifică urma de feromon de pe

- **cel mai bun drum** (ACO/MMAS)
- Pentru fiecare muchie a celui mai bun drum
 - Se calculează cantitatea unitară de feromoni lăsată de cea mai bună furnică pe muchia (ij)
 - L_{best} – lungimea (costul) celui mai bun drum
 - din iterația curentă
 - din toate iterațiile executate până atunci

$$\Delta \tau_{ij} = \frac{1}{L_{best}}$$

- Se calculează intensitatea urmei de feromoni ca sumă între evaporarea feromonilor vechi și feromonul nou lăsat

$$\tau_{ij}^{(t+n)} = [(1 - \rho) * \tau_{ij}^{(t)} + \rho * \Delta \tau_{ij}^{best}]_{\tau_{min}}^{\tau_{max}}$$

- unde ρ ($0 < \rho < 1$) – coeficientul de evaporare a urmei de feromon între 2 tururi complete
- τ_{min} și τ_{max} – limitele (inferioară și superioară) feromonului;
 - pentru $\tau_{min} = -\infty$ și $\tau_{max} = +\infty \rightarrow$ ACO, altfel MMAS

ACO – proprietăți

□ Proprietăți

- Algoritm iterativ
- Algoritm care construiește progresiv soluția pe baza
 - Informațiilor euristice
 - Urmei de feromon
- Algoritm stocastic

□ Avantaje

- Rulare neîntreruptă și adaptabilă schimbării în timp real a datelor de intrare
 - Ex. Pt TSP graful se poate modifica dinamic
- Feedback-ul pozitiv ajută la descoperirea rapidă a soluției
- Calculul distribuit evită convergența prematură
- Euristică greedy ajută la găsirea unei soluții acceptabile încă din primele stadii ale căutării
- Interacțiunea colectivă a indivizilor

□ Dezavantaje

- Converge încet față de alte căutări euristice
- Funcționează relativ slab pentru instanțe cu mai mult de 75 de orașe ale TSP
- În AS nu există un proces central care să ghideze căutarea spre soluțiile bune

ACO – aplicații

- Probleme de identificare a drumului optim în grafe
 - Ex. Traveling Salesman Problem
- Probleme de atribuiri quadratice
- Probleme de optimizări în rețele
- Probleme de transport



Recapitulare

□ PSO

- Algoritm de căutare locală în fascicol
- Potențialele soluții → particule caracterizate prin:
 - poziție în spațiul de căutare
 - Viteză
- Căutare cooperativă și perturbativă bazată pe
 - Poziția celei mai bune particule din grup
 - Cea mai bună poziție a particulei de până atunci (particula are memorie)

□ ACO

- Algoritm de căutare locală în fascicol
- Potențialele soluții → furnici caracterizate prin:
 - Memorie – rețin pașii făcuți în construirea soluției
 - Miros – iau decizii pe baza feromonului depus de celelalte furnici (comportament social, colectiv, colaborativ)
- Căutare cooperativă și constructivă

Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor – Materiale de citit și legături utile

- capitolul II.5 din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 6 din *H.F. Pop, G. Șerban, Inteligență artificială, Cluj Napoca, 2004*
- documentele din directorul *06_adversial_minimax*

-
- Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan - www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ



Rezolvarea problemelor de căutare

Strategii de căutare adversială

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Materiale de citit și legături utile

- capitolul II.5 din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 6 din *H.F. Pop, G. Șerban, Inteligență artificială, Cluj Napoca, 2004*
- documentele din directorul *06_adversial_minimax*

Conținut

Jocuri

- ❑ Un pic de istorie
- ❑ Câteva repere teoretice
- ❑ Jocurile și căutarea
 - Definiții, componente, clasificare
 - Strategii și algoritmi de căutare



Jocuri – un pic de istorie

- Provocări ale inteligenței
 - Dame (Mesopotamia, cc. 3000 îHr.)
 - Go (China, sec. VI î.Hr.)
 - Sah (India, sec. VI d.Hr.)
- Tradițional, jocurile erau văzute formal, ca o extensie a algoritmilor de căutare
 - Căutarea clasică: un singur agent, încearcă fără piedici să își atingă obiectivul
 - Jocurile: căutare în prezența unui adversar (agent ostil) + aduce incertitudine
- Jocurile și IA
 - Proiectarea și testarea algoritmilor
 - Unul dintre cele mai vechi domenii ale IA

Jocuri – un pic de istorie

□ Jucarea jocurilor

■ De către oameni

- Activitate intelligentă
- Compararea aptitudinilor

■ De către calculatoare

- Mediu propice pentru dezvoltarea tehniciilor de IA
 - Joc = problemă
 - structurată (succes sau eșec)
 - rezolvabilă prin căutare (simplă sau euristică)
 - Limite

Jocuri – câteva repere teoretice

- Probleme dificile cu o structură inițială minimă de cunoștiințe
- Stări și acțiuni ușor de reprezentat
- Puține cunoștiințe necesare despre mediu
- Jocurile sunt un caz particular al problemelor de căutare
- Deseori au spații de căutare foarte mari
 - Complexitatea jocurilor → incertitudine
 - datorată insuficienței de timp pentru a calcula consecințele tuturor mutărilor și
 - nu lipsei de informație
- Jocurile sunt interesante ☺

Jocurile și căutarea

- Formalizare
- Reprezentarea jocurilor
- Tipuri de jocuri
- Spațiul de căutare
 - Reprezentare
 - Metode de explorare

Jocurile și căutarea – formalizare

- Rezolvarea unei probleme de căutare
 - Etapa x: definirea spațiului de căutare
 - Spații liniare
 - Spații arborescente
 - Arbori
 - Grafe
 - Etapa x + 1: alegerea unei strategii de căutare
 - Care mutare/strategie?
 - Care determină câștigarea jocului
 - Care poate fi determinată cât mai repede (complexitate temporală redusă)
 - Care poate fi determinată cu efort fizic cât mai mic (complexitate spațială redusă)
- Problema:
 - cum se poate determina cea mai bună mutare următoare într-un timp cât mai scurt?
- O soluție:
 - căutarea între mutările posibile și consecințele lor

Jocurile și căutarea – formalizare

- Căutarea adversarială este folosită în jocurile în care unii jucători încearcă să-și maximizeze scorul, dar se confruntă cu unul sau mai mulți adversari
- Reprezentarea jocurilor ca probleme de căutare
 - **Stări**
 - configurațiile (tablei) de joc + jucătorul care trebuie să mute
 - totalitatea stărilor posibile → arborele de căutare
 - **Operatori (acțiuni, funcție successor)**
 - mutările permise
 - **Starea inițială**
 - configurația inițială a jocului
 - **Starea scop (finală)**
 - configurația (câștigătoare) care termină jocul
 - **Funcție de utilitate (funcție scor)**
 - asociază o valoare numerică unei stări
- Dificultăți
 - Jocurile pot fi probleme de căutare foarte dificile
 - totuși usor de formalizat
 - Găsirea soluției optime poate fi nefezabilă
 - o soluție care învinge adversarul este acceptabilă
 - o soluție „inacceptabilă” nu numai că induce costuri mai mari, dar provoacă înfrângerea

Jocurile și căutarea – formalizare

□ Definiții cheie

■ Situație conflictuală

- Situație în care acționează mai multe părți care au scopuri contrare
- Consecința acțiunii unei părți depinde de reacția celorlalte părți

■ Joc

- Modelare simplificată a unei situații conflictuale
- Înșiruire de decizii (acțiuni, mutări) luate de părți cu interese contrastante

■ Mutare

- Funcție H : $\{\text{pozițiile jocului}\} \rightarrow \{\text{pozițiile jocului}\}$
- Dacă p – o poziție în joc, $H(p)$ – o nouă poziție

■ Reguli

- Sistem de condiții privind mutările posibile

■ Strategie

- Ansamblu de reguli care definesc mutările libere în funcție de situația concretă ivită

Jocurile și căutarea – reprezentare

❑ Componente

- Jucători
- Mutări (strategii)
- Criterii de câștig

❑ Forme de reprezentare pentru un joc

- Forma strategică → matrice
 - Jucătorii
 - Strategiile
 - Câștigurile asociate fiecărui jucător și fiecărei strategii
- Forma extinsă → arbore
 - Nivel → jucător
 - Nod → alegere (mutare)

Jocurile și căutarea – reprezentare



□ Forma strategică → matrice

- Ex. Dilema prizonierului: doi prizonieri sunt chestionați de poliție. Poliția știe ceva despre ceea ce au făcut ei, dar nu are toate informațiile. Pentru a afla, cei doi prizonieri sunt înciși în două celule și sunt interogați. Prizonierii au 2 opțiuni:
 - Pot spune toată povestea (pot să se trădeze unul pe altul)
 - Pot să nu spună nimic (pot să coopereze)

Nici un prizonier nu știe ce va răspunde celălalt. În funcție de răspunsurile lor, pedepsele sunt următoarele:

- Dacă amândoi tac (cooperează), sentința este ușoară (1 an fiecare)
- Dacă unul vorbește (trădează) și unul tace (cooperează), trădătorul este eliberat, iar părătul primește 10 ani
- Dacă amândoi vorbesc (se trădează reciproc), sentința este de 5 ani fiecare

Ce vor face cei doi prizonieri?

J1 \ J2	Cooperare	Trădare
Cooperare	(1,1)	(10,0)
Trădare	(0,10)	(5,5)

Jocurile și căutarea – reprezentare



□ Forma strategică → matrice

- Ex. Vânătoarea de cerbi: doi indivizi merg la vânătoare. Fiecare poate alege să vâneze:
 - un cerb sau
 - un iepure,
- fără însă să ştie ce a ales celălalt individ. Vânarea cerbilor, mai profitoare ($\text{câştig}=4$), se poate face doar în doi, iepurilor, mai puțin valoroasă ($\text{câştig}=1$), se poate face individual.
- Ce vor alege să vâneze cei doi indivizi?

J1 \ J2	Cerb	Iepure
Cerb	(4,4)	(1,3)
Iepure	(3,1)	(3,3)

Jocurile și căutarea – reprezentare

□ Forma strategică → matrice

■ Ex. economic

- Reclama făcută de 2 firme pe aceeași piață
- Înțelegerile la nivel de cartel pentru stabilirea prețurilor

■ Ex. sportiv

- Doi cicliști aflați în fața plutonului poartă consecutiv trena (coopereză) pentru a nu fi ajunși din urmă. De multe ori, doar unul duce trena (coopereză), iar la linia de sosire este trădat de adversar.



■ Ex. sociologic



- Cand cunoaștem o nouă persoană, tindem să fim foarte atenți pentru a avea o poziție de siguranță (competiție). Ambele persoane pot semnală dorința de a se muta de la pozițiile defensive către
 - interacțiune și
 - recunoașterea unei intenții comune.

Jocurile și căutarea - tipologie

□ Numărul jucătorilor

- 1
- 2
- Mai mulți

□ Comunicarea între jucători

- Cooperative
- Ne-cooperative

□ Strategiile de joc urmate

- Simetrice
- Asimetrice

Jocurile și căutarea – tipologie

□ Câștigurile jucătorilor

■ Sumă zero

- câștigul unui jucător = pierderea celorlați jucători → un jucător trebuie să câștige sau jocul se sfârșește cu remiză

■ Sumă diferită de zero

- câștigul unui jucător ≠ pierderea celorlați jucători

□ Natura mutărilor efectuate

■ Cu mutări libere

- mutarea este aleasă conștinet dintr-o mulțime de acțiuni posibile → jocuri deterministe

■ Cu mutări întâmplătoare

- factor aleator (zar, cărți de joc, monede) → jocuri non-deterministe

Jocurile și căutarea – tipologie

□ Informațiile despre joc

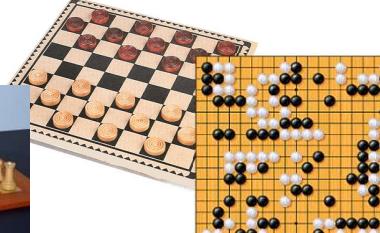
■ Cu informație perfectă

- un jucător, înainte de a executa o mutare, cunoaște rezultatele tuturor mutărilor precedente (ale lui și ale adversarilor);
- de obicei, jocul e cu mutări secvențiale

■ Cu informație imperfectă

- un jucător nu cunoaște toate efectele mutărilor precedente

Jocurile și căutarea – tipologie

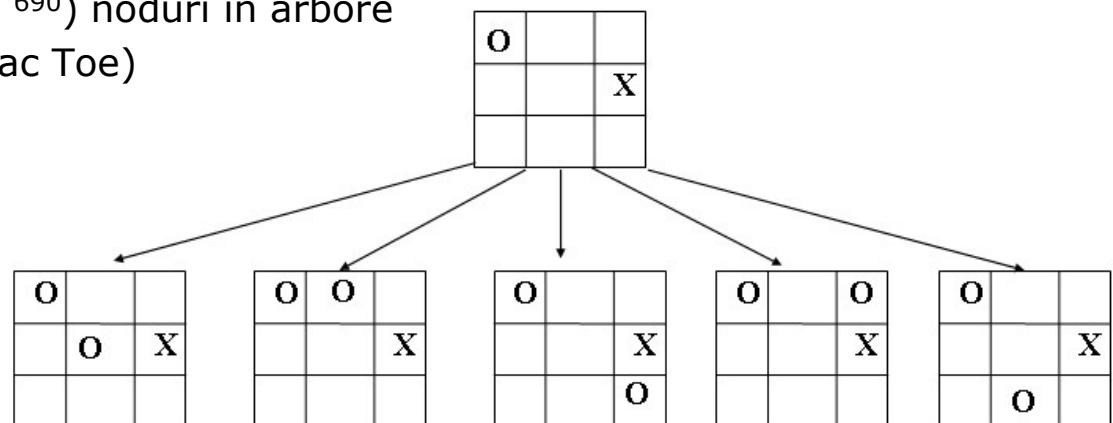
	Deterministic	Non-deterministic (aleator)
Informație perfectă	  	 
Informație imperfectă	Vaporașe/Avioane/ Submarine	 

Jocurile și căutarea – spațiul de căutare



□ Reprezentare

- Spațiu liniar
- Spațiu arborescent → Arborele jocului
 - Identificarea strategiei de câștig → explorarea întregului arbore
 - foarte mare pt anumite jocuri
 - Șah („drosofila IA”)
 - factor de ramificare ≈ 35
 - ≈ 50 de mutări pe jucător
 - $\approx 35^{100}$ (10^{154}) noduri
 - 10^{40} noduri distincte (dimensiunea grafului de căutare)
 - Go
 - ≈ 200 mutări/stare, 300 niveluri
 - 200^{300} (10^{690}) noduri în arbore
 - exemplu – XO (Tic Tac Toe)



Jocurile și căutarea – spațiul de căutare



□ Strategia de joc (a unui jucător)

■ Definire

- ansamblul mutărilor unui jucător care
- ține cont de:
 - regulile jocului
 - starea curentă a jocului
- ≠ mutare

■ Tipologie

- Scop
 - Strategii pas cu pas
 - În fiecare etapă a jocului se identifică mutarea cea mai bună
 - Strategii complete
 - Se identifică o succesiune de mutări

Jocurile și căutarea – spațiul de căutare



□ Strategia de joc

■ Pas cu pas

- Ex.: XO, Dame, Şah
- Algoritmi – pot lucra cu structuri:

- Liniare
 - Strategia simetriei
 - Strategia perechilor
 - Strategia parității
 - Programare dinamică
 - Alte strategii
- Arborescente
 - Arbori AndOr
 - MiniMax (cu tăieturi Alpha-Beta)

■ Completă

- Ex.: ieșirea dintr-un labirint, deplasarea pe o hartă între 2 locații date
- Algoritmi pot să identifice
 - Un drum optim de la o locație la alta
 - O succesiune de acțiuni care să deplaseze jucătorul de la o locație la alta

Jocurile și căutarea – spațiul de căutare



□ Strategia de joc

■ Pas cu pas

- Ex.: XO, Dame, Şah
- Algoritmi – pot lucra cu structuri:

▪ Liniare

- Strategia simetriei
- Strategia perechilor
- Strategia parității
- Programare dinamică
- Alte strategii

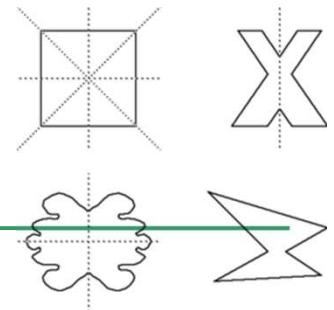
▪ Arborescente

- Arbori AndOr
- MiniMax (cu tăieturi Alpha-Beta)

■ Completă

- Ex.: ieșirea dintr-un labirint, deplasarea pe o hartă între 2 locații date
- Algoritmi pot să identifice
 - Un drum optim de la o locație la alta
 - O succesiune de acțiuni care să deplaseze jucătorul de la o locație la alta

Jocurile și căutarea – spațiul de căutare



Strategii de joc → Strategia ...

- Exemplu: jocul *hașurează căsuțe*

- Se dă:

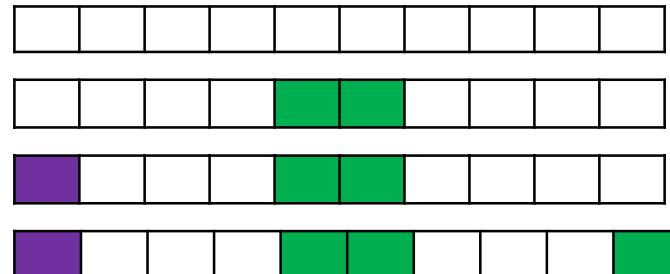
- O bandă de hârtie este împărțită în n căsuțe. Alternativ doi jucători A și B hașurează câte maxim k căsuțe adiacente, nehașurate (n și k au aceeași paritate). Cel care nu mai poate mută pierde. Inițial mută jucătorul A.

- Se cere:

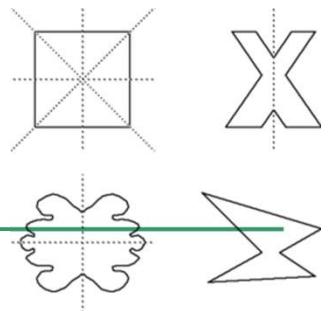
- Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.

- Caz concret:

- $n = 10, k = 4$

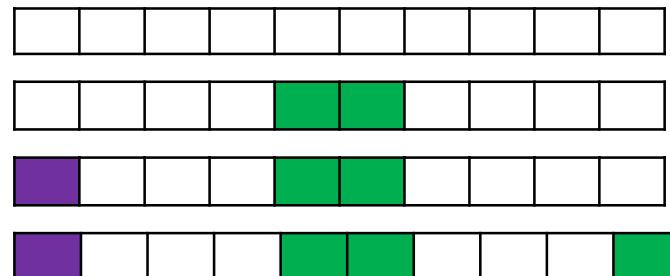


Jocurile și căutarea – spațiul de căutare

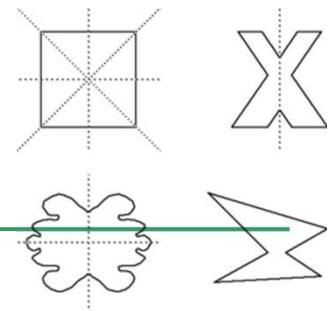


Strategii de joc → Strategia simetriei

- Exemplu: jocul *hașurează căsuțe*
 - Se dă:
 - O bandă de hârtie este împărțită în n căsuțe. Alternativ doi jucători A și B hașurează câte maxim k căsuțe adiacente, nehașurate (n și k au aceeași paritate). Cel care nu mai poate mută pierde. Inițial mută jucătorul A.
 - Se cere:
 - Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.
 - Caz concret:
 - $n = 10, k = 4$
 - Soluție
 - Dacă $n -$ nr. par
 - Prima mutare → jucătorul A hașurează un nr. par de căsuțe din mijlocul benzii
 - La următoarele mutări jucătorul A îl imită pe jucătorul B simetric față de mijlocul benzii
 - Dacă $n -$ nr. impar
 - Prima mutare → jucătorul A hașurează un nr. impar de căsuțe din mijlocul benzii
 - La următoarele mutări jucătorul A îl imită pe jucătorul B simetric față de mijlocul benzii



Jocurile și căutarea – spațiul de căutare



Strategii de joc → Strategia simetriei

□ Aspecte teoretice

- Jucătorul B imită mutările jucătorului A pe baza unei (unui) axe (centru) de simetrie
- Dacă A mai poate muta, atunci și B mai poate muta
 - Jocul se termină când A nu mai poate muta
- E posibil ca prima mutare să nu aibă simetric

Jocurile și căutarea – spațiul de căutare



Strategii de joc → Strategia ...

❑ Exemplu - Jocul *pătratelor alunecătoare*

■ Se dă:

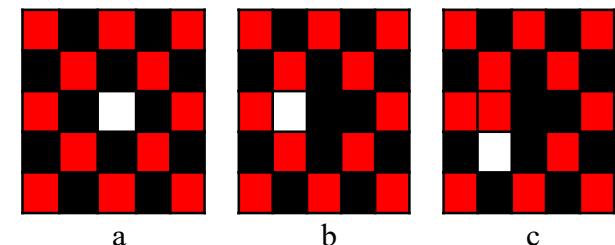
- ❑ În fiecare căsuță a unei table pătratice $n*n$ (n – nr. impar) se află un pătrat roșu sau negru astfel încât tabla se asemănă cu o tablă de șah. Căsuța din mijlocul tablei este goală (nu conține un pătrat). Alternativ, doi jucători A (cu pătratele roșii) și B (cu pătratele negre) mută câte unul din pătratele lor în căsuța liberă de pe tablă. Pătratul mutat trebuie să se afle inițial într-o căsuță vecină (pe orizontală sau verticală) cu căsuța liberă. Jucătorul care nu mai poate muta nici un pătrat de-al lui pierde jocul. Jucătorul B mută primul.

■ Se cere:

- ❑ Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.

■ Caz concret:

- ❑ $n = 5$ (jocul propus inițial de către G.W.Lewthwaite)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → Strategia perechilor

□ Exemplu - Jocul *pătratelor alunecătoare*

■ Se dă:

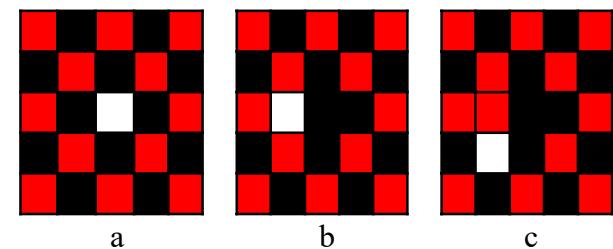
- În fiecare căsuță a unei table pătratice $n*n$ (n – nr. impar) se află un pătrat roșu sau negru astfel încât tabla se asemănă cu o tablă de șah. Căsuța din mijlocul tablei este goală (nu conține un pătrat). Alternativ, doi jucători A (cu pătratele roșii) și B (cu pătratele negre) mută câte unul din pătratele lor în căsuță liberă de pe tablă. Pătratul mutat trebuie să se afle inițial într-o căsuță vecină (pe orizontală sau verticală) cu căsuța liberă. Jucătorul care nu mai poate muta nici un pătrat de-al lui pierde jocul. Jucătorul B mută primul.

■ Se cere:

- Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.

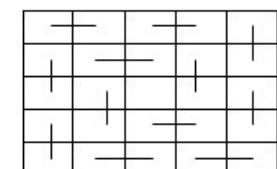
■ Caz concret:

- $n = 5$ (jocul propus inițial de către G.W.Lewthwaite)



■ Soluție

- Se împarte tabla în *Domino-uri*, căsuța din mijlocul tablei (inițial goală) nefacând parte din nici un *Domino*.
- După fiecare mutare a jucătorului B, căsuța liberă se află acoperită de același *Domino* care acoperă și o piesă de culoare roșie. Această piesă o va muta A, având în acest fel întotdeauna ceva de mutat.
- acoperirea cu *Domino-uri* în spirală, plecându-se din colțul stânga sus, spre dreapta



Jocurile și căutarea – spațiul de căutare

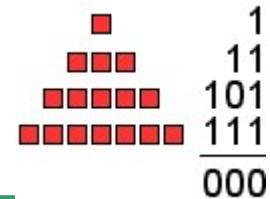


Strategii de joc → Strategia perechilor

❑ Aspecte teoretice

- Generalizare a strategiei simetriei
- Gruparea mutărilor în perechi:
 - (mutare jucător A, mutare jucător B)
- Dacă A mai poate muta, atunci și B mai poate muta
 - Jocul se termină când A nu mai poate muta
- E posibil ca prima mutare să nu poată fi împerecheată

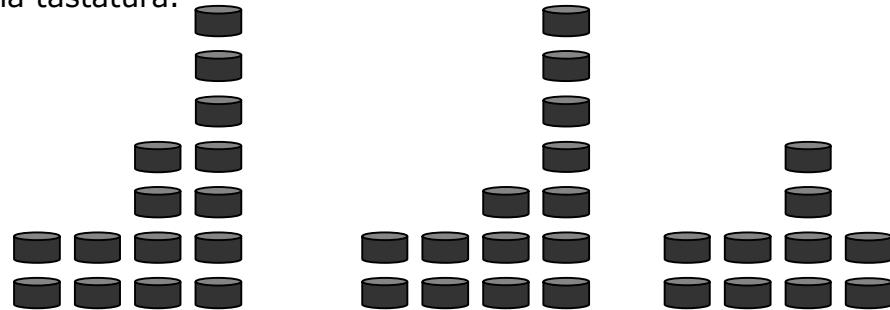
Jocurile și căutarea – spațiul de căutare



Strategii de joc → Strategia parității

□ Exemplu - Jocul *NIM*

- Se dau:
 - N stive, fiecare conținând p_i obiecte. 2 jucători, A și B, extrag, alternativ, dintr-o singură stivă, oricâte obiecte. Jucătorul care execută ultima extragere câștigă jocul. Jucătorul A mută primul.
- Se cere:
 - Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.
- Exemplu:
 - 4 stive cu 2, 2, 4 și, respectiv, 7 obiecte



Jocurile și căutarea – spațiul de căutare

1
11
101
111
000

Strategii de joc → Strategia parității

□ Exemplu - Jocul *NIM*

■ Se dau:

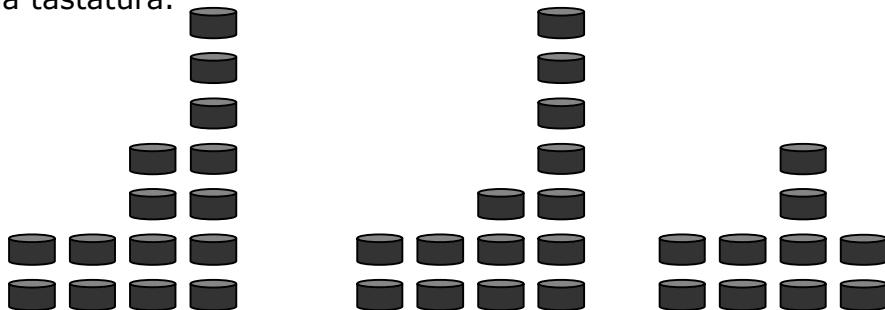
- N stive, fiecare conținând p_i obiecte. 2 jucători, A și B, extrag, alternativ, dintr-o singură stivă, oricâte obiecte. Jucătorul care execută ultima extragere câștigă jocul. Jucătorul A mută primul.

■ Se cere:

- Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.

■ Exemplu:

- 4 stive cu 2, 2, 4 și, respectiv, 7 obiecte



■ Soluție

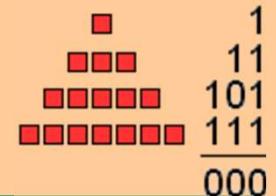
- Nr de obiecte din fiecare stivă se reprezintă binar (ca sumă a puterilor lui 2)
 - fiecare număr natural admite o descompunere unică ca sumă de puteri distincte ale lui 2
- Stare pară – toate puterile lui 2 din reprezentarea jocului apar de un număr par de ori (în toate stivele)
- Stare impară – orice stare care nu este pară
 - T1 → dintr-o stare impară se ajunge într-o stare pară printr-o mutare convenabilă
 - T2 → dintr-o stare pară se ajunge într-o stare impară prin orice fel de mutare
- Extragerea de pe stiva care conține cel mai semnificativ bit pe poziția k
 - K poziția celui mai semnificativ bit în suma NIM a tuturor stivelor

Strategii de joc → Strategia parității

□ Aspecte teoretice

- 2 tipuri de poziții în joc
 - Poziție pară (singulară)
 - Poziție impară (nesingulară)
- Teoreme
 - T1: poziția impară, printr-o mutare **convenabilă**, → poziție pară (jucătorul A)
 - T2: poziția pară, prin **orice** mutare, → poziție impară (jucătorul B)
- Câștigător
 - → poziția finală = poziție pară

Jocurile și căutarea – spațiul de căutare



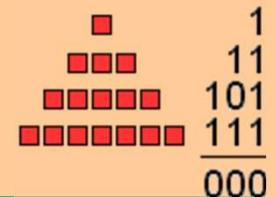
Strategii de joc → Strategia parității

□ Exemplu - Jocul *NIM*

■ Demonstrarea celor 2 teoreme T1 și T2

- $S_1: 2 = 2^1, S_2: 3 = 2^1 + 2^0, S_3: 4 = 2^2, S_4: 5 = 2^2 + 2^0$
- $2^0 - 2, 2^1 - 2, 2^2 - 2 \rightarrow$ stare pară
- Extragerea dintr-o stivă cu număr par de obiecte
 - A unui număr par de obiecte (ex. 2 obiecte din S_3) = >
 - $S_1: 2 = 2^1, S_2: 3 = 2^1 + 2^0, S_3: 4-2 = 2^1, S_4: 5 = 2^2 + 2^0$
 - $2^0 - 2, 2^1 - 3, 2^2 - 1 \rightarrow$ stare impară
 - A unui număr impar de obiecte (ex. 1 obiect din S_1)
 - $S_1: 2-1 = 2^0, S_2: 3 = 2^1 + 2^0, S_3: 4 = 2^2, S_4: 5 = 2^2 + 2^0$
 - $2^0 - 3, 2^1 - 1, 2^2 - 2 \rightarrow$ stare impară
- Extragerea dintr-o stivă cu număr impar de obiecte
 - A unui număr par de obiecte (ex. 2 obiecte din S_4) = >
 - $S_1: 2 = 2^1, S_2: 3 = 2^1 + 2^0, S_3: 4 = 2^2, S_4: 5-2 = 2^1 + 2^0$
 - $2^0 - 2, 2^1 - 3, 2^2 - 1 \rightarrow$ stare impară
 - A unui număr impar de obiecte (ex. 1 obiect din S_2)
 - $S_1: 2 = 2^1, S_2: 3-1 = 2^1, S_3: 4 = 2^2, S_4: 5 = 2^2 + 2^0$
 - $2^0 - 1, 2^1 - 2, 2^2 - 2 \rightarrow$ stare impară

Jocurile și căutarea – spațiul de căutare



Strategii de joc → Strategia parității

□ Exemplu - Jocul NIM

■ algoritm

□ Pp. următorul joc:

- $S_1: 3 = 2^1 + 2^0 \rightarrow 011, S_2: 4 = 2^2 \rightarrow 100, S_3: 5 = 2^2 + 2^0 \rightarrow 101$

□ Pași:

1. Se calculează suma Nim (S_{Nim}) a tuturor stivelor

- $S_{\text{Nim}} = S_1 \text{ XOR } S_2 \text{ XOR } S_3 = 3 \text{ XOR } 4 \text{ XOR } 5 = 011 \text{ XOR } 100 \text{ XOR } 101 = 010 = 2$

2. Se identifică poziția k a celui mai reprezentativ 1 în suma Nim

- $S_{\text{Nim}} = 2 = 010_{(2)} \rightarrow$ poziția a 2-a ($k = 2$)

3. Se caută stiva cu cel mai reprezentativ bit pe poziția k (stiva care descrește dacă se face XOR între ea și suma Nim)

- $S_1: 3 = 010, S_2: 4 = 100, S_3: 5 = 101 \Rightarrow S_1$ sau

- $S_1 \text{ XOR } S_{\text{Nim}} = 3 \text{ XOR } 2 = 011 \text{ XOR } 010 = 001 = 1$

- $S_2 \text{ XOR } S_{\text{Nim}} = 4 \text{ XOR } 2 = 100 \text{ XOR } 010 = 110 = 6$

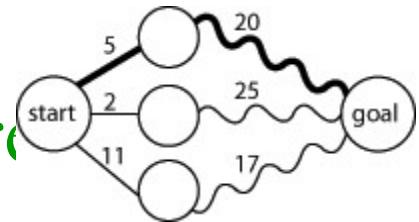
- $S_3 \text{ XOR } S_{\text{Nim}} = 5 \text{ XOR } 2 = 101 \text{ XOR } 010 = 111 = 7 \rightarrow S_1$

4. Se extrage din această stivă un număr de obiecte a.î. restul stivelor să formeze o stare pară; nr de obiecte care rămân pe stivă este dat de XOR-ul între stiva respectivă și suma Nim

- $S_1 \text{ XOR } S_{\text{Nim}} = 3 \text{ XOR } 2 = 011 \text{ XOR } 010 = 001 = 1 \rightarrow$ se extrag 2 ($=3 - 1$) obiecte de pe stiva 1

5. Se reia pasul 1

Jocurile și căutarea – spațiul de căutare



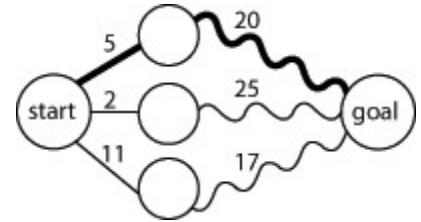
Strategii de joc → Programare dinamică (PD)

□ Aspecte teoretice

- Pași în rezolvarea unei probleme cu PD:
 - Descompunerea problemei în sub-probleme
 - Rezolvarea sub-problemelor
 - Combinarea sub-soluțiilor pentru obținerea soluției finale

- Pași în rezolvarea unui joc cu PD:
 - Descompunerea jocului în sub-jocuri
 - Găsirea unei strategii perfecte de câștig pentru fiecare sub-joc
 - Combinarea sub-strategiilor pentru obținerea strategiei finale

Jocurile și căutarea–spațiul de căutare



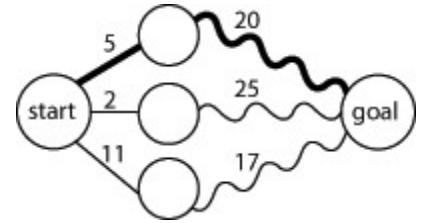
Strategii de joc → Programare dinamică (PD)

❑ Aspecte teoretice

■ Cum se rezolvă un joc cu programare dinamică?

- ❑ descompunerea jocului în sub-jocuri J_h (sub-jocuri de pe nivelul cel mai de jos) și
 - etichetarea fiecărui joc J_h cu T (*true*) sau F (*false*) în funcție de posibilitatea jucătorului (care trebuie să mute din acea poziție) de a avea strategie sigură de câștig
- ❑ un sub-joc J_k de pe un nivel interior ($k < h$) va fi etichetat cu:
 - T, dacă există cel puțin un sub-joc J_i , $k < i$, etichetat cu F, în care poate fi transformat jocul J_k
 - F, dacă toate sub-jocurile J_i , $k < i$, în care se poate ajunge printr-o mutare din jocul J_k sunt etichetate cu T

Jocurile și căutarea–spațiul de căutare



Strategii de joc → Programare dinamică (PD)

□ Exemplu - Jocul *șirului de căsuțe*

■ Se dă:

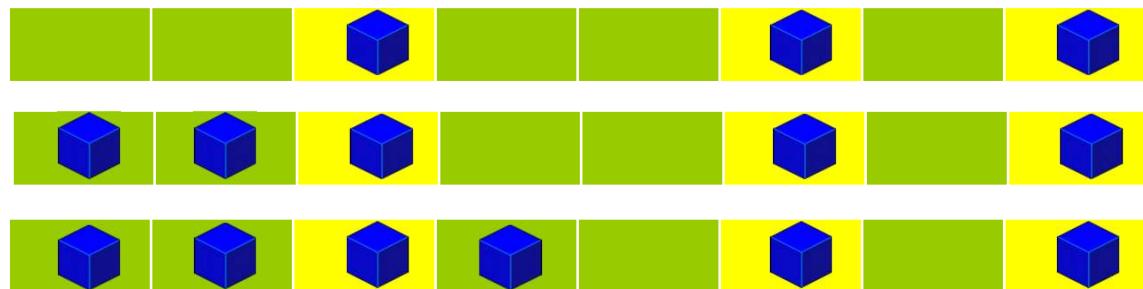
- În cele n căsuțe ale unui sir se află amplasate cuburi (maxim un cub în fiecare căsuță). Scopul jocului este umplerea tuturor căsuțelor cu cuburi. Alternativ, doi jucători A și B umplu cu cuburi (complet sau parțial) cel mai din stânga sir de căsuțe libere. Jucătorul care nu mai poate muta pierde. Jucătorul A mută primul.

■ Se cere:

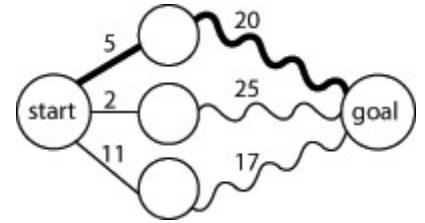
- Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.

■ Exemplu

- $n = 8$



Jocurile și căutarea–spațiul de căutare



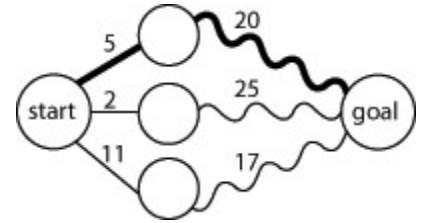
Strategii de joc → Programare dinamică (PD)

- Exemplu - Jocul *șirului de căsuțe*

- Soluție:



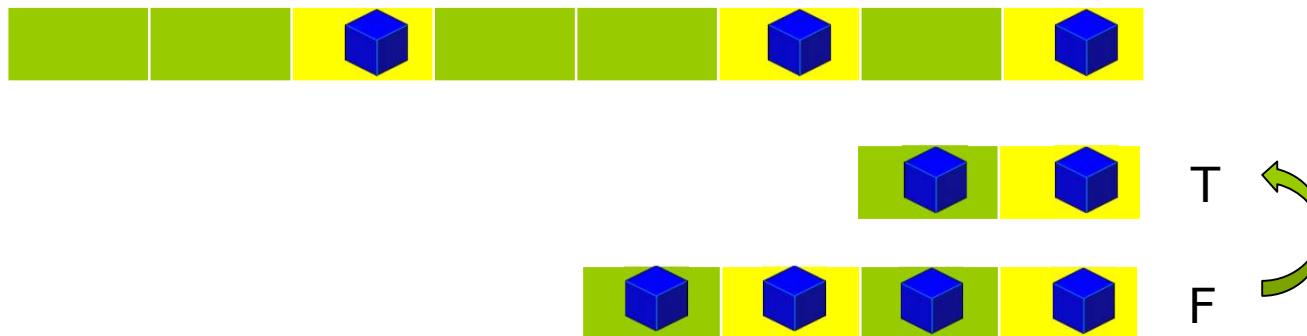
Jocurile și căutarea–spațiul de căutare



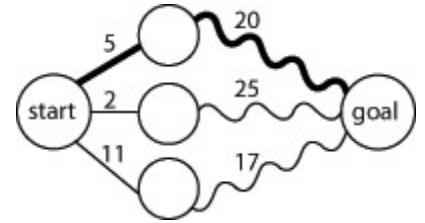
Strategii de joc → Programare dinamică (PD)

- Exemplu - Jocul *șirului de căsuțe*

- Soluție:



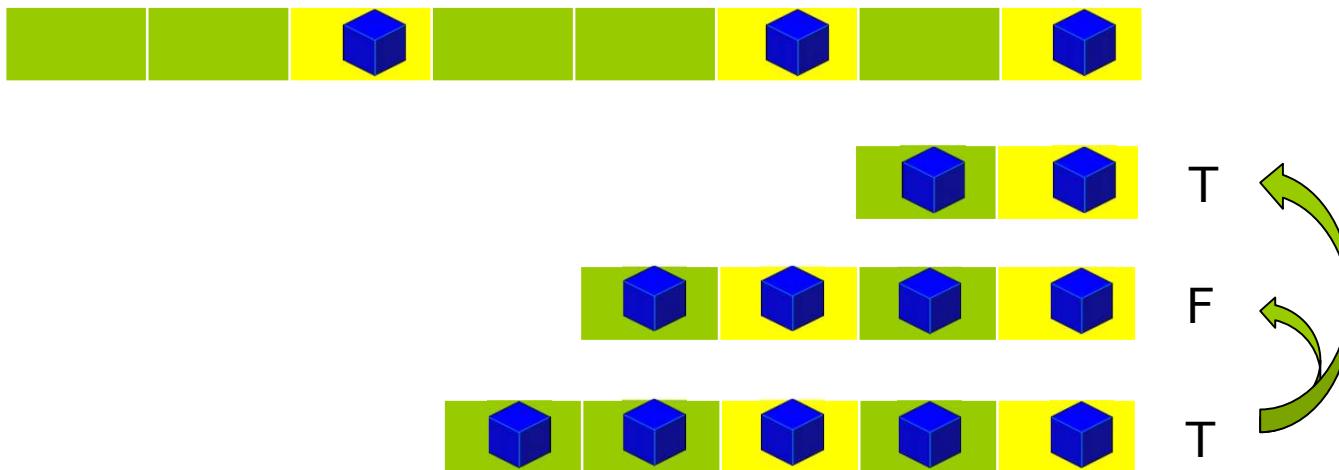
Jocurile și căutarea–spațiul de căutare



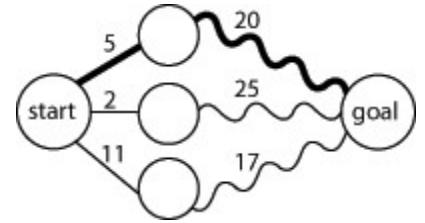
Strategii de joc → Programare dinamică (PD)

❑ Exemplu - Jocul *șirului de căsuțe*

- Soluție:



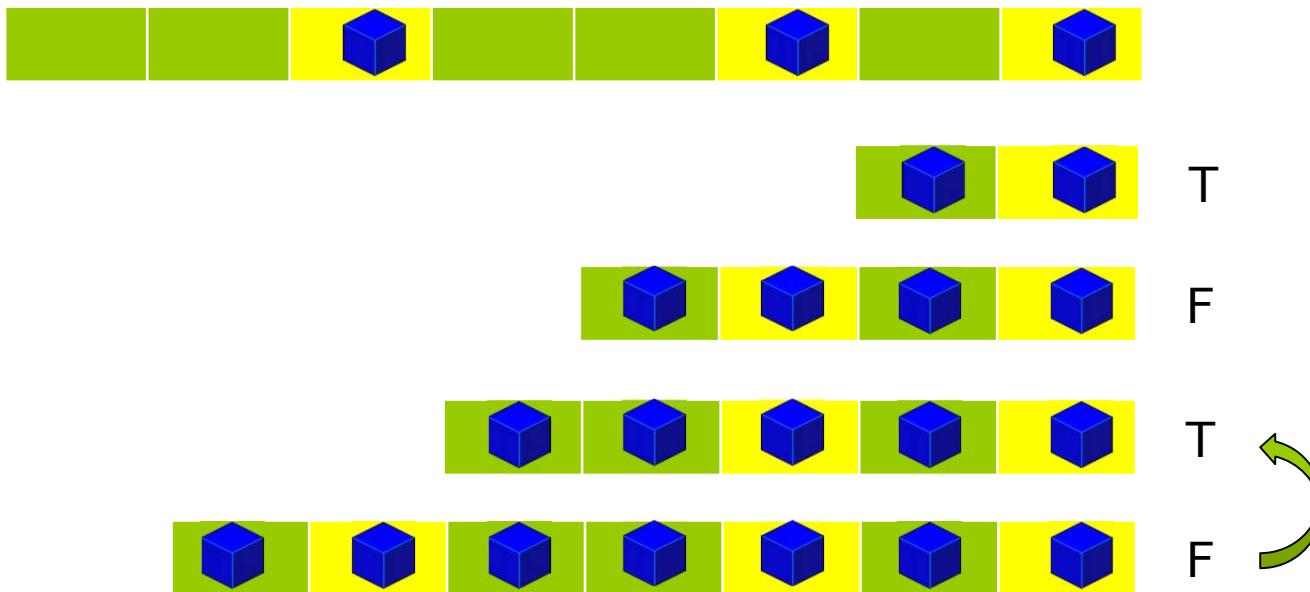
Jocurile și căutarea–spațiul de căutare



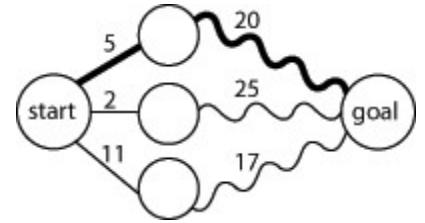
Strategii de joc → Programare dinamică (PD)

❑ Exemplu - Jocul *șirului de căsuțe*

- Soluție:



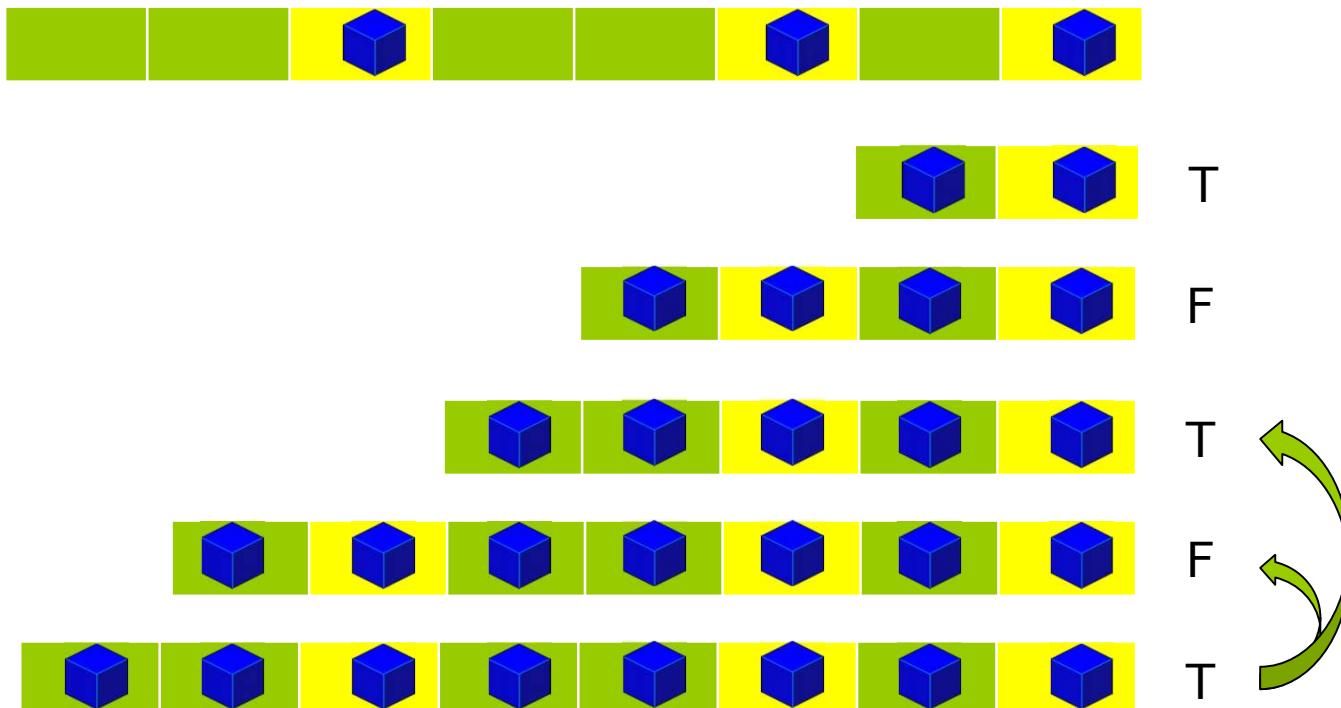
Jocurile și căutarea–spațiul de căutare



Strategii de joc → Programare dinamică (PD)

❑ Exemplu - Jocul *șirului de căsuțe*

- Soluție:



Jocurile și căutarea – spațiul de căutare



□ Strategia de joc

■ Pas cu pas

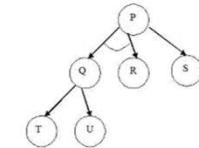
- Ex.: XO, Dame, Şah
- Algoritmi – pot lucra cu structuri:

- Liniare
 - Strategia simetriei
 - Strategia perechilor
 - Strategia parității
 - Programare dinamică
 - Alte strategii
- Arborescente
 - **Arbori AndOr**
 - MiniMax (cu tăieturi Alpha-Beta)

■ Completă

- Ex.: ieșirea dintr-un labirint, deplasarea pe o hartă între 2 locații date
- Algoritmi pot să identifice
 - Un drum optim de la o locație la alta
 - O succesiune de acțiuni care să deplaseze jucătorul de la o locație la alta

Jocurile și căutarea–spațiul de căutare

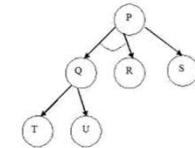


Strategii de joc - Arbori And-Or (AAO)

□ Aspecte teoretice

- **Reprezentarea spațiului de căutare** cu ajutorul AAO pentru un joc cu 2 jucători:
 - Partea (nodul) OR
 - → selectarea unei mutări pentru jucătorul curent A
 - → există o posibilitate (mutare) pentru jucătorul A să ajungă într-un nod AND
 - Partea (nodul) AND
 - → considerarea tuturor mutărilor posibile ale adversarului (jucătorul B)
 - → prin orice mutare jucătorul B ajunge într-un nod OR
 - Astfel:
 - Rădăcina → problema jucătorului câștigător (care începe jocul din starea inițială)
 - Mutările posibile ale primului jucător (A) se reunesc prin disjuncție (OR)
 - Mutările posibile ale celui de-al doilea jucător (B) se reunesc prin conjuncție (AND)
 - Jocul poate fi câștigat dacă începe cu un nod OR
- **Dificultăți:**
 - Arbori foarte foarte mari

Jocurile și căutarea–spațiul de căutare



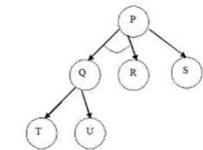
Strategii de joc - Arbori And-Or (AAO)

□ Aspecte teoretice

■ Pași în rezolvarea unui joc cu AAO:

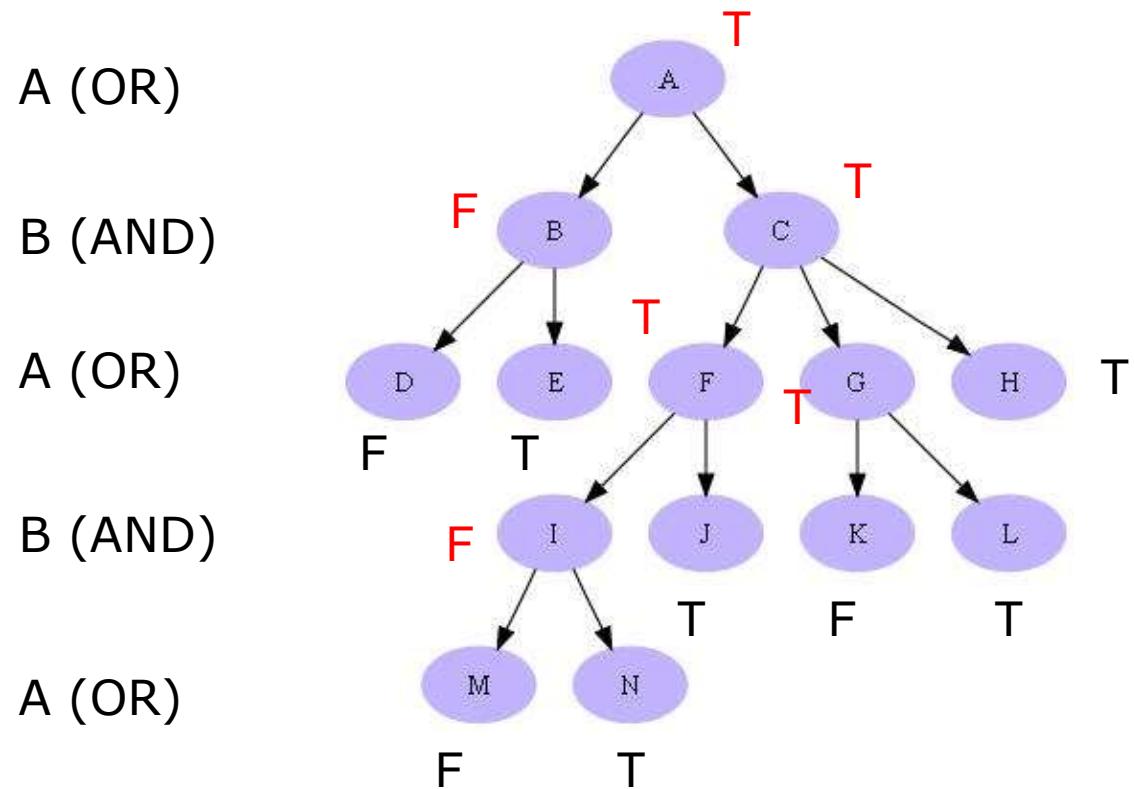
- Descompunerea jocului în sub-jocuri
 - noduri pe mai multe nivele,
 - nodurile de pe un nivel corespunzând mutărilor posibile ale unui jucător
- Găsirea unei strategii perfecte de câștig pentru fiecare sub-joc (nod)
 - Etichetarea nodurilor cu T sau F în funcție de posibilitatea jucătorului A de a avea o strategie sigură de câștig pentru acel sub-joc
 - Reguli de etichetare:
 - Frunzele se etichetează cu T sau F în funcție de configurația jocului
 - Nodurile interne se etichetează:
 - Pentru jucătorul A, cu T dacă **cel puțin un nod** fiu a fost etichetat cu T – regula OR
 - Pentru jucătorul B, cu T dacă **toate nodurile** fiu au fost etichetat cu T – regula AND
 - Combinarea sub-strategiilor pentru obținerea strategiei finale

Jocurile și căutarea–spațiul de căutare

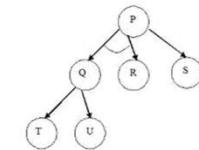


Strategii de joc - Arbori And-Or (AAO)

□ Exemplu



Jocurile și căutarea–spațiul de căutare

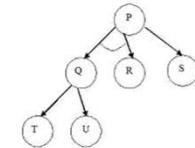


Strategii de joc - Arbori And-Or (AAO)

□ Algoritm

```
bool backAndOr(Node N, int level){  
    //level = 0 for the node in the top of the tree.  
    if (N is a terminal) {  
        if (the first player won)  
            return true;  
        else  
            return false;  
    }  
    else{  
        if (level % 2){      //B is about to move; AND  
            result = true;  
            for each child Ni of N  
                result = result && backAndOr(Ni, level+1);  
        }  
        else{              // A is about to move; OR  
            result = false;  
            for each child Ni of N  
                result = result || backAndOr(Ni, level+1);  
        }  
        return result;  
    }  
}
```

Jocurile și căutarea–spațiul de căutare



Strategii de joc - Arbori And-Or (AAO)

□ Avantaje

- Pot fi aplicăți pentru rezolvarea oricărui joc

□ Dezavantaje

- Necesită multă memorie
- Necesită mult timp de calcul

□ →algoritmul mini-max

Jocurile și căutarea – spațiul de căutare



□ Strategia de joc

■ Pas cu pas

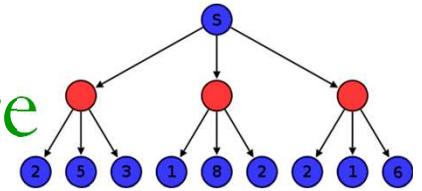
- Ex.: XO, Dame, Şah
- Algoritmi – pot lucra cu structuri:

- Liniare
 - Strategia simetriei
 - Strategia perechilor
 - Strategia parității
 - Programare dinamică
 - Alte strategii
- Arborescente
 - Arbori AndOr
 - **Minimax** (cu tăieturi Alpha-Beta)

■ Completă

- Ex.: ieșirea dintr-un labirint, deplasarea pe o hartă între 2 locații date
- Algoritmi pot să identifice
 - Un drum optim de la o locație la alta
 - O succesiune de acțiuni care să deplaseze jucătorul de la o locație la alta

Jocurile și căutarea – spațiul de căutare

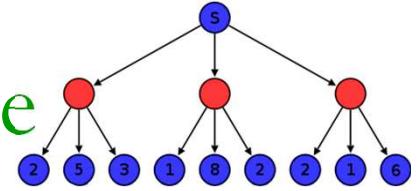


Strategii de joc → MiniMax

□ Aspecte teoretice

- Propus de John von Neuman în 1944
- Ideea de bază: maximizarea poziției unui jucător în timp ce poziția adversarului este minimizată
- Arborele de căutare constă în alternarea nivelelor pe care un jucător încearcă să-și maximizeze câștigul cu nivelele pe care adversarul minimizează câștigul (primului jucător)
 - Primul jucător va încerca să-și maximizeze câștigul (jucătorul MAX → mută primul)
 - Al doilea jucător va încerca să minimizeze câștigul primului jucător (jucătorul MIN → adversarul)
- Identificarea celor mai bune mutări
 - Se construiește arborele tuturor mutărilor posibile (fiecare mutare se aplică unei stări a jocului)
 - Se evaluatează frunzele (care jucător câștigă)
 - Se propagă (în sus în arbore) câștigurile
 - Explorarea arborelui este de tip *Depth first search*
- Generarea tuturor mutărilor
 - Posibilă doar în jocuri simple (ex. Tic-Tac-Toe)
 - Imposibilă (cu resurse limitate) pentru jocuri complexe
 - reținând minimele în MIN
 - reținând maximele în MAX

Jocurile și căutarea – spațiul de căutare



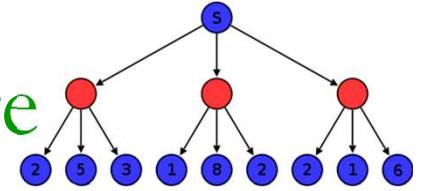
Strategii de joc → MiniMax

▫ Algoritm

- Se construiește arborele corespunzător tuturor mutărilor
- Se evaluatează frunzele
- Cât timp se mai poate alege un nod cu toți descendenții evaluați
 - Se alege un nod
 - Dacă nodul este de pe nivel Min, el va fi evaluat la cea mai mică valoare a unui descendental
 - Dacă nodul este de pe nivel Max, el va fi evaluat la cea mai mare valoare a unui descendental
- Se returnează valoarea nodului rădăcină (nod de pe nivel Max)

```
int backMiniMax(Node N, int level){  
    //level = 0 for the node in the top of the tree.  
    if (level == MaxLevel)  
        return the quality of N computed with a heuristic;  
    else if (level < MaxLevel){  
        if (level % 2){  
            //B is about to move; minimize  
            result = MaxInt;  
            for each child Ni of N  
                result = minim(result, backMiniMax(Ni, level+1));  
        }  
        else{  
            // A is about to move; Maximize  
            result = -MaxInt;  
            for each child Ni of N  
                result = maxim(result, backMiniMax(Ni, level+1));  
        }  
        return result;  
    }  
}
```

Jocurile și căutarea – spațiul de căutare



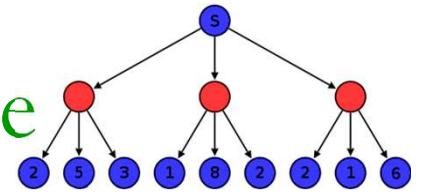
Strategii de joc → MiniMax

❑ Exemplu – jocul *Tic-Tac-Toe*

■ Funcția de evaluare a unui nod:

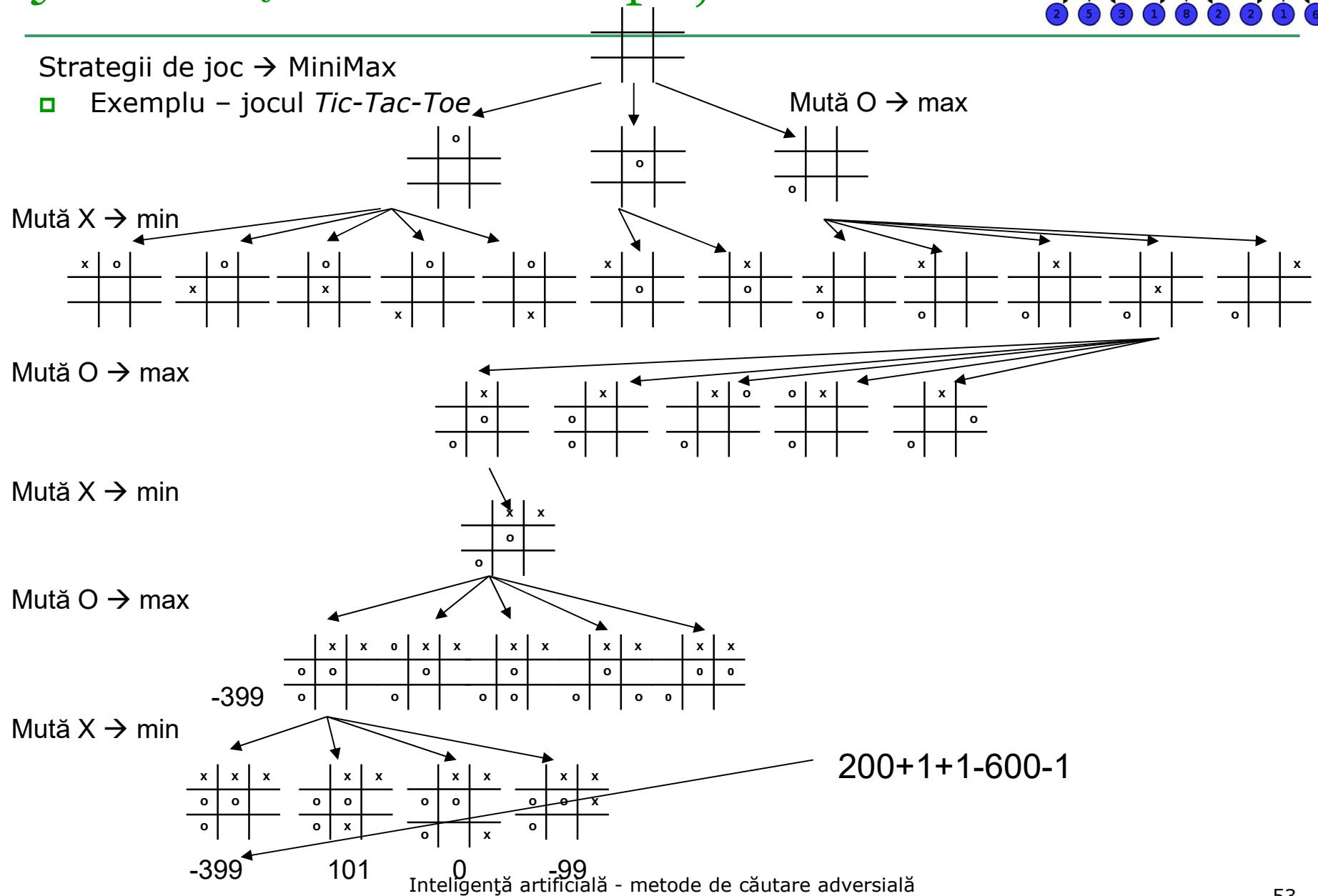
- ❑ Dacă există o linie aproape completă (cu 2 semne la fel) → 200 puncte
- ❑ Dacă există 2 linii aproape complete (cu 2 semne la fel) → 300 puncte
- ❑ O linie completă → 600 puncte
- ❑ Fiecare linie potențială → 1 punct
- ❑ Punctele jucătorului care urmează la mutare se adună
- ❑ Punctele celuilalt jucător se scad

Jocurile și căutarea – spațiul de căutare

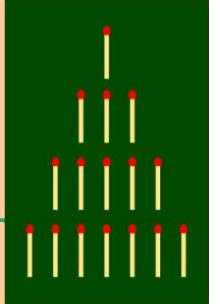


Strategii de joc → MiniMax

- Exemplu – jocul *Tic-Tac-Toe*



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax

❑ Exemplu – *Jocul NIM*

■ Se dă:

- ❑ N stive conțin fiecare p_i obiecte. 2 jucători A și B extrag, alternativ, dintr-o singură stivă oricâte obiecte. Jucătorul care execută ultima extragere câștigă jocul. Jucătorul A mută primul.

■ Se cere:

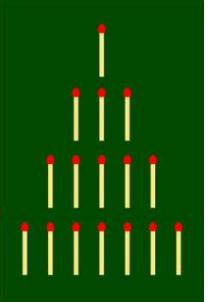
- ❑ Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.

■ Exemplu:

- ❑ 3 stive cu 1, 1 și, respectiv, 2 obiecte

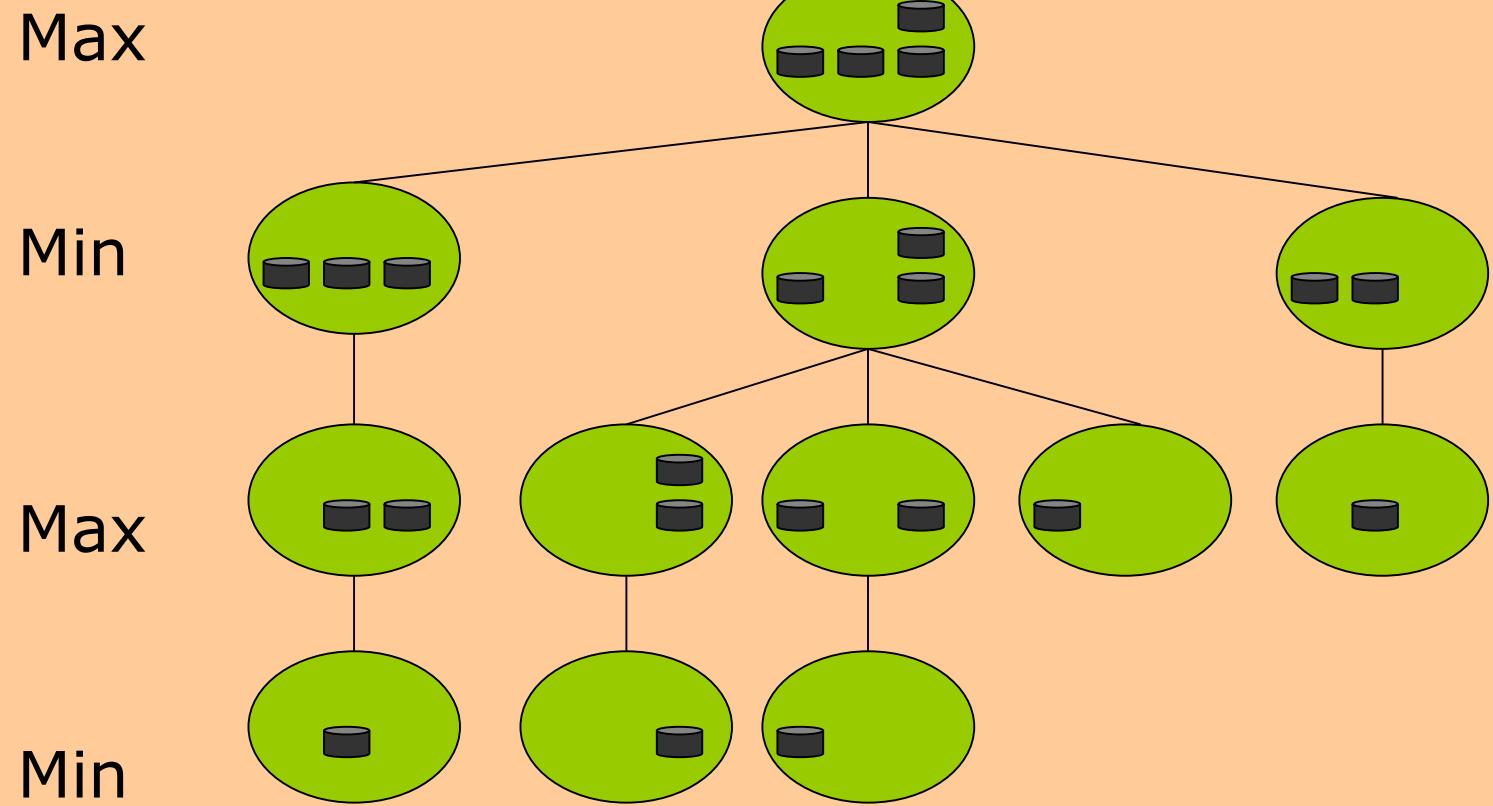


Jocurile și căutarea – spațiul de căutare

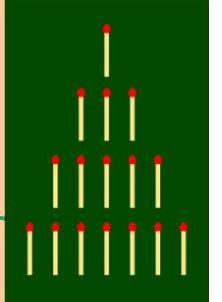


Strategii de joc → MiniMax

- ❑ Exemplu – jocul NIM



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax

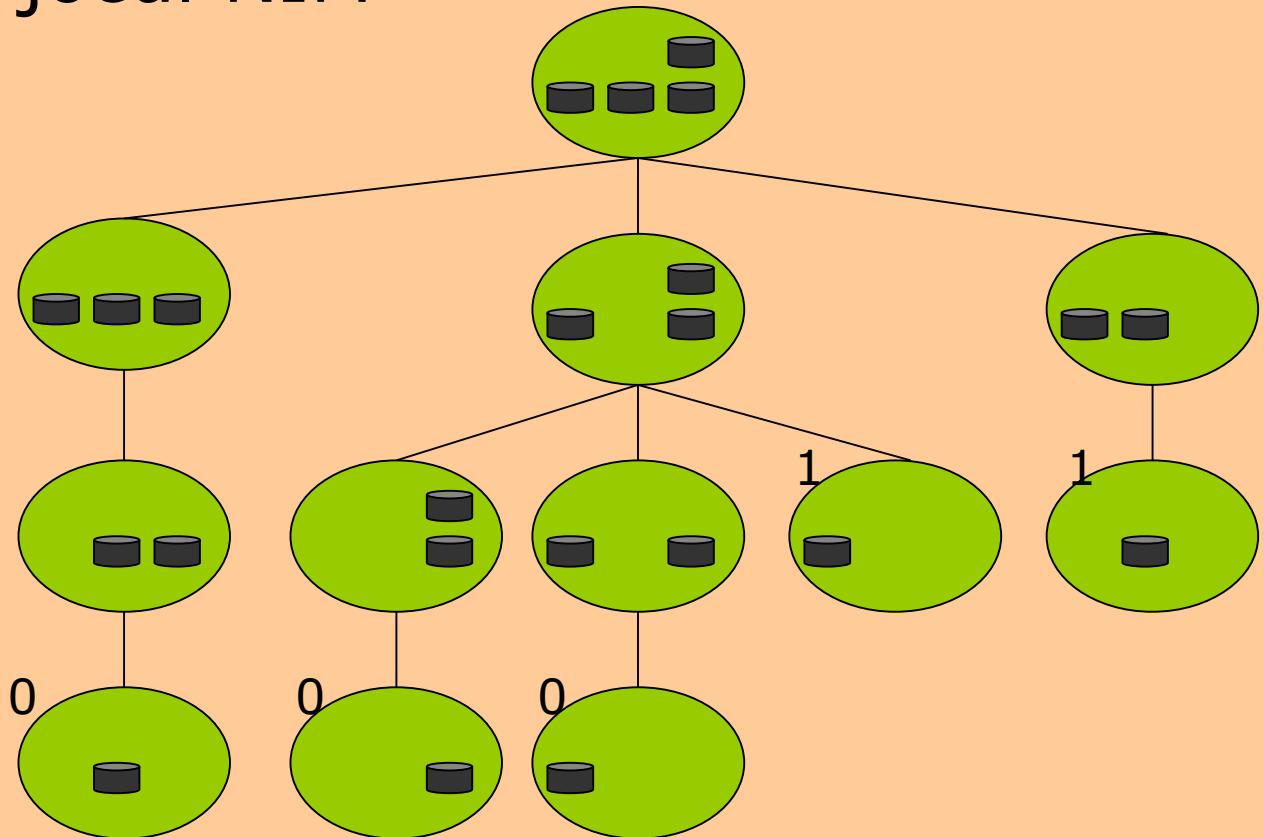
- ❑ Exemplu – jocul NIM

Max

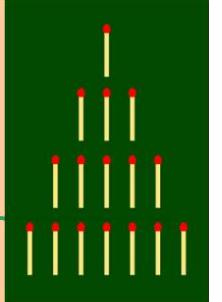
Min

Max

Min



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax

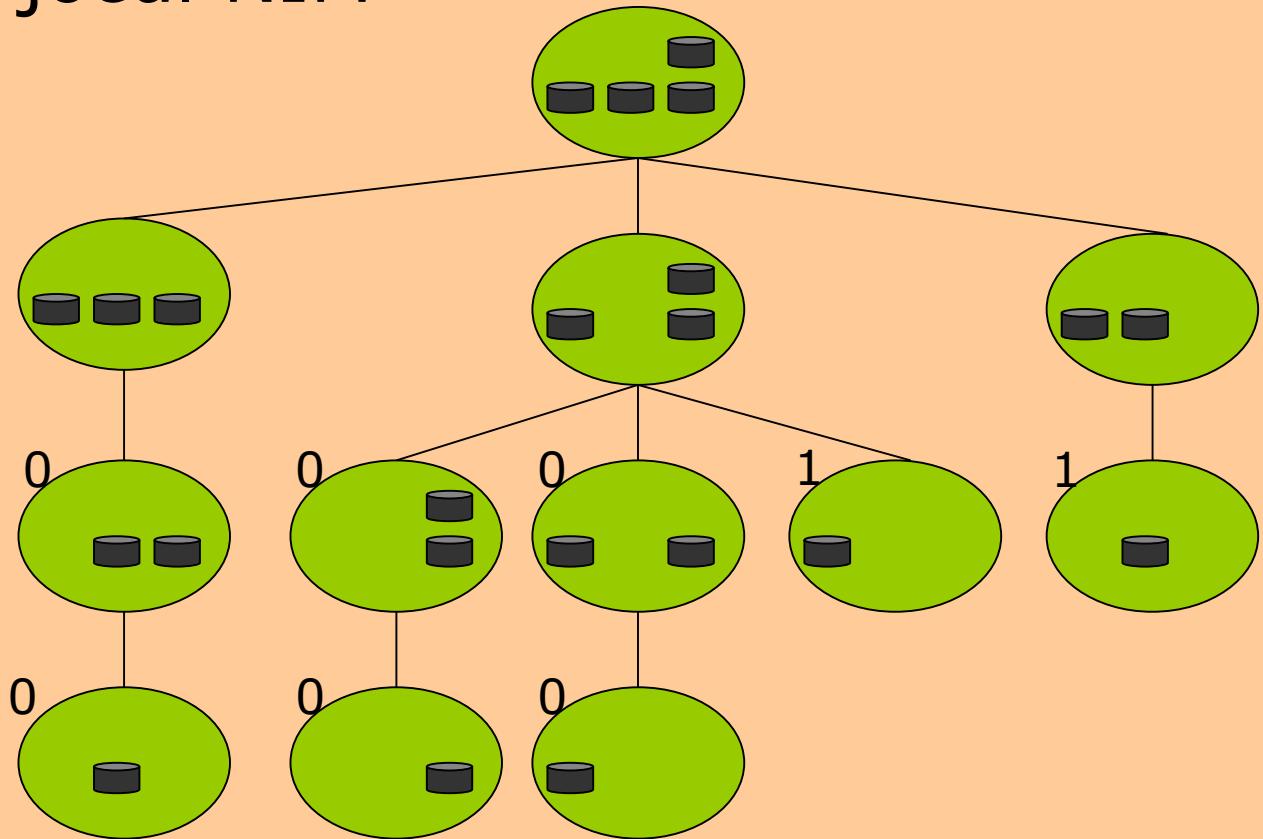
- ❑ Exemplu – jocul NIM

Max

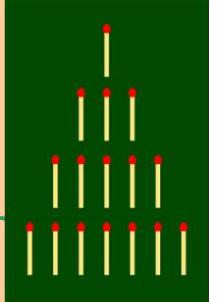
Min

Max

Min

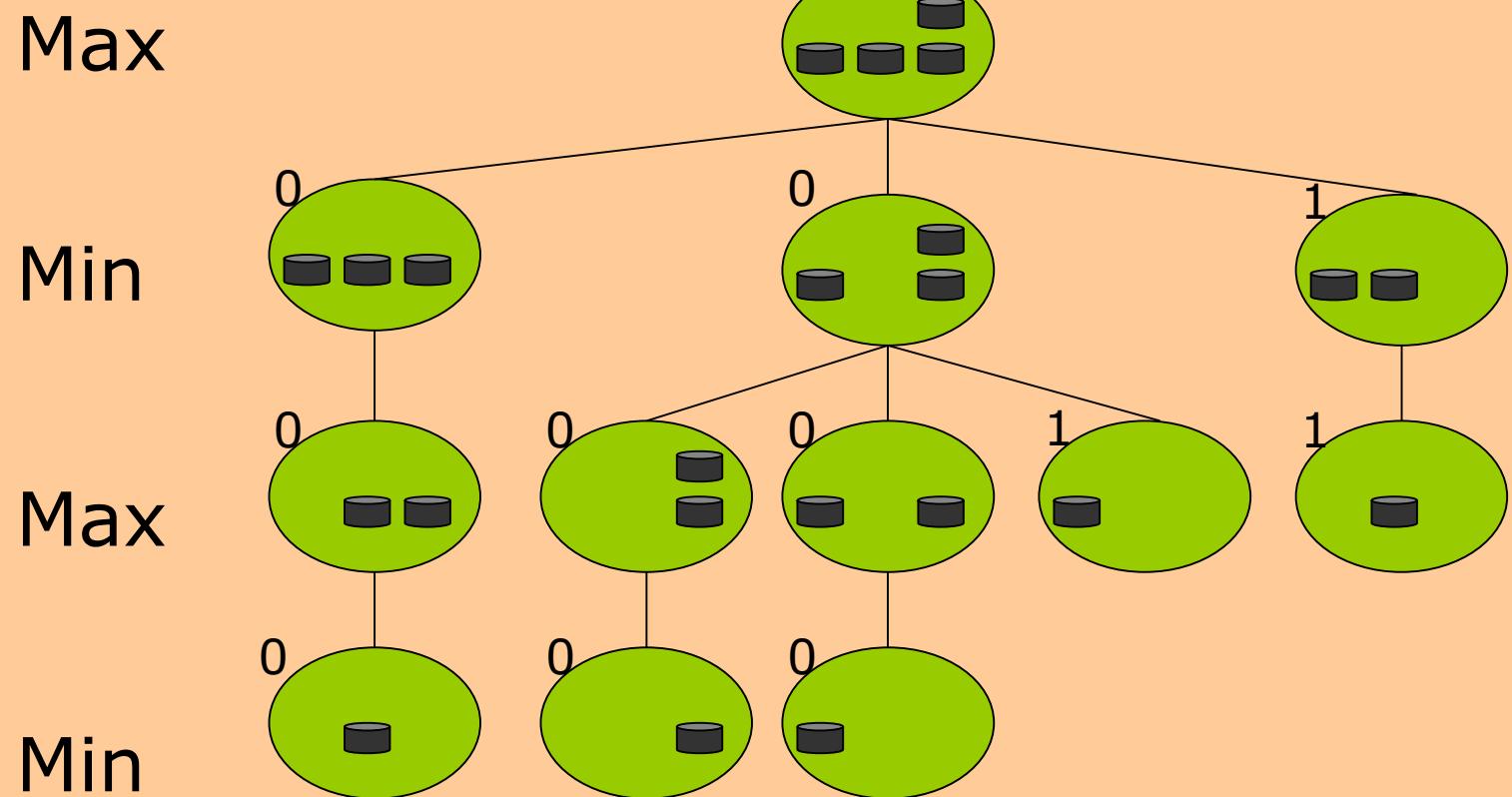


Jocurile și căutarea – spațiul de căutare

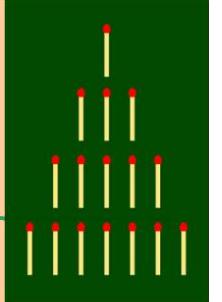


Strategii de joc → MiniMax

- ❑ Exemplu – jocul NIM



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax

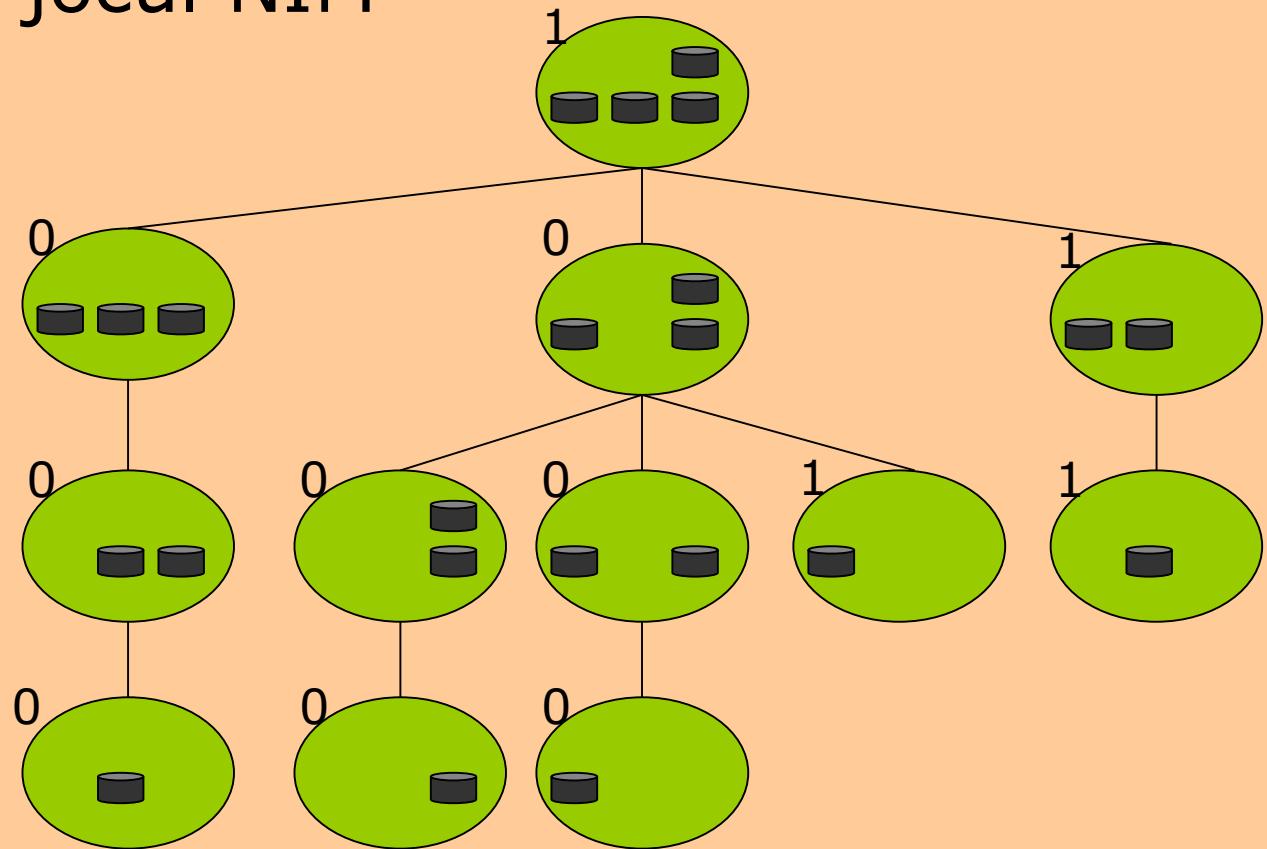
❑ Exemplu – jocul NIM

Max

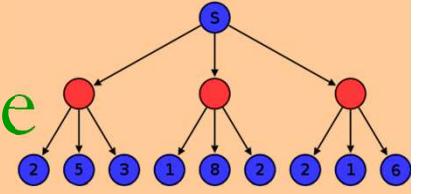
Min

Max

Min



Jocurile și căutarea – spațiul de căutare

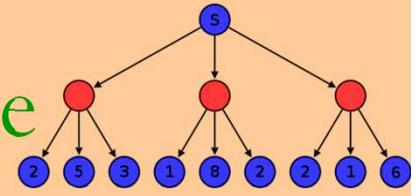


Strategii de joc → MiniMax

□ Dificultăți

- Nu explorează întregul arbore
 - Căutare depth-first
- La începutul jocului se fixează:
 - O adâncime maximă
 - Care este adâncimea optimă?
 - Adâncime mare → căutare lungă
 - Adâncime mică → anumite drumuri pot fi ratate (sacrificii timpurii pentru câștiguri târzii)
 - O funcție de evaluare
 - Fiecare nod (stare) trebuie evaluat(ă)
 - Cum? ➔ folosind euristică

Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax

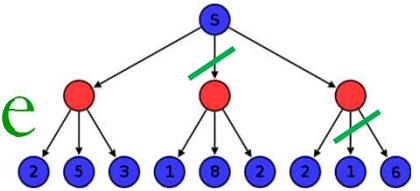
□ Dezavantaje

- Doar 100 noduri sunt explorate într-o secundă
- Sah – timp mutare = 150s → 150 000 de poziții
→ doar 3 - 4 mutări în avans

□ Soluția

- Evitarea anumitor noduri prin rețezarea unor ramuri (redundante) → rețezare (reducere)
alpha-beta → MiniMax cu rețezare α - β

Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

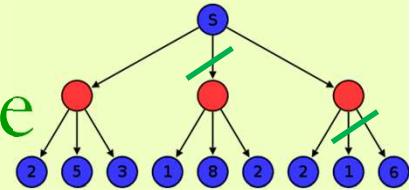
❑ Minimax

- Creaază întreg arborele
- Se propagă valorile de jos în sus

❑ MiniMax cu retezare α - β

- Crearea și propagarea în același timp
- Dacă se știe că un drum este rău, nu se mai consumă energie ca să se afle cât de rău este acel drum
 - Se pot elimina anumite evaluări inutile
 - Se pot elimina anumite expandări ale nodurilor

Jocurile și căutarea – spațiul de căutare

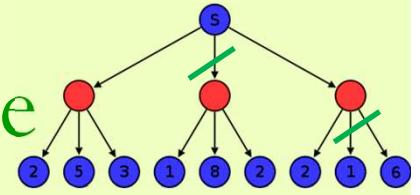


Strategii de joc → MiniMax cu retezare $\alpha\text{-}\beta$

■ Apecte teoretice

- Descoperit de McCarthy în 1956, dar publicat pentru prima dată într-un raport tehnic de la MIT
- Ideea de bază
 - Valoarea minimax a rădăcinii poate fi determinată fără examinarea tuturor nodurilor de pe frontieră căutării
 - Similar algoritmului *branch and bound*
- Denumirea $\alpha\text{-}\beta$
 - α - valoarea celei mai bune alegeri (celei mai mari) efectuate de-a lungul drumului urmat de MAX
 - dacă o valoare unui nod este mai slabă decât α atunci MAX o va evita și va reteza sub-arborele cu rădăcina în acel nod
 - β - similar lui α , dar pentru jucătorul MIN

Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

□ Aspecte teoretice

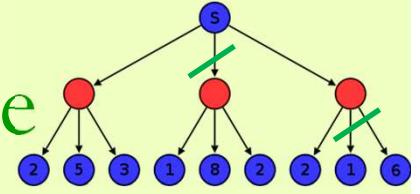
■ α

- valoarea cele mai bune (adică cea mai mare) alegeri găsită până la momentul curent în orice punct de-a lungul unui drum pentru MAX
- scorul minim pe care jucătorul MAX îl poate obține în mod garantat
- dacă un nod v este mai slab (mai mic) decât α , MAX îl va evita prin eliminarea acelei ramuri →
 - Dacă s-a ajuns cu căutarea într-un nod MIN a cărui valoare $\leq \alpha$ atunci descendenții acelui nod nu mai merită explorați pentru că oricum MAX îi va ignora

■ β

- cea mai mică valoare găsită la orice punct de-a lungul unui drum pentru MIN
- scorul minim pe care jucătorul MAX speră să-l obțină
- dacă un nod v este mai bun (mai mare) decât β , MIN îl va evita prin eliminarea acelei ramuri →
 - Dacă s-a ajuns cu căutarea într-un nod MAX a cărui valoare $\geq \beta$ atunci descendenții acelui nod nu mai merită explorați pentru că oricum MIN îi va ignora

Jocurile și căutarea – spațiul de căutare

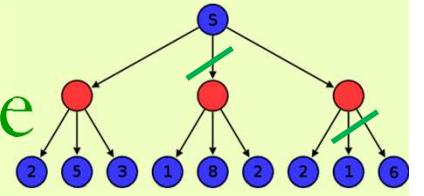


Strategii de joc → MiniMax cu retezare α - β

□ Aspecte teoretice

- Valoarea α a unui nod
 - Inițial, scorul acelui nod (dacă este frunză) sau $-\infty$
 - Apoi,
 - Nod de tip MAX = cel mai mare scor al descendenților
 - Nod de tip MIN = valoarea α a predecesorului
- Valoarea β a unui nod
 - Inițial, scorul acelui nod (daca este frunză) sau $+\infty$
 - Apoi,
 - Nod de tip MIN = cel mai mic scor al descendenților
 - Nod de tip MAX = valoarea β a predecesorului
- Scorul unui nod
 - Nod de tip MAX valoarea α finală
 - Nod de tip MIN valoarea β finală

Jocurile și căutarea – spațiul de căutare

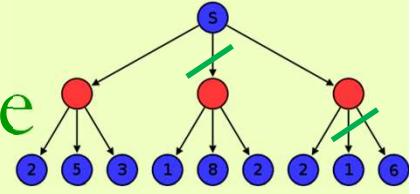


Strategii de joc → MiniMax cu retezare α - β

□ Algoritm

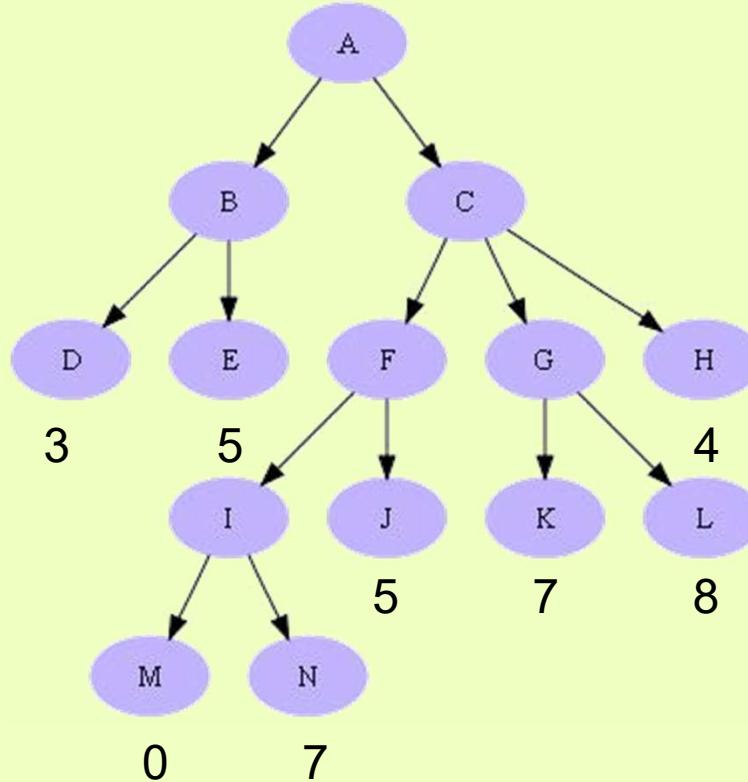
```
int backMiniMaxAB(Node N, int A, int B){  
    Set Alpha value of N to A and Beta value of N to B;  
    if N is a leaf  
        return the estimated score of this leaf  
    else{  
        if N is a Min node  
            for each child Ni of N {  
                Val = backMiniMaxAB(Ni, Alpha of N, Beta of N);  
                Beta value of N = minim(Beta value of N, Val);  
                if Beta value of N  $\leq$  Alpha value of N then exit loop;  
            }  
            return Beta value of N;  
        else // N is a Max node  
            for each child Ni of N do {  
                Val = MINIMAX-AB(Ni, Alpha of N, Beta of N);  
                Alpha value of N = Max{Alpha value of N, Val};  
                if Alpha value of N  $\geq$  Beta value of N then exit loop;  
            }  
            Return Alpha value of N;  
    }  
}
```

Jocurile și căutarea – spațiul de căutare

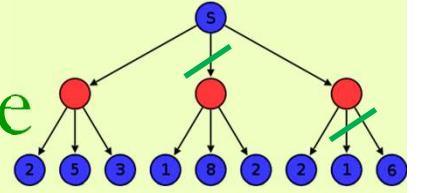


Strategii de joc → MiniMax cu retezare α-β

□ Exemplu



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

□ Exemplu

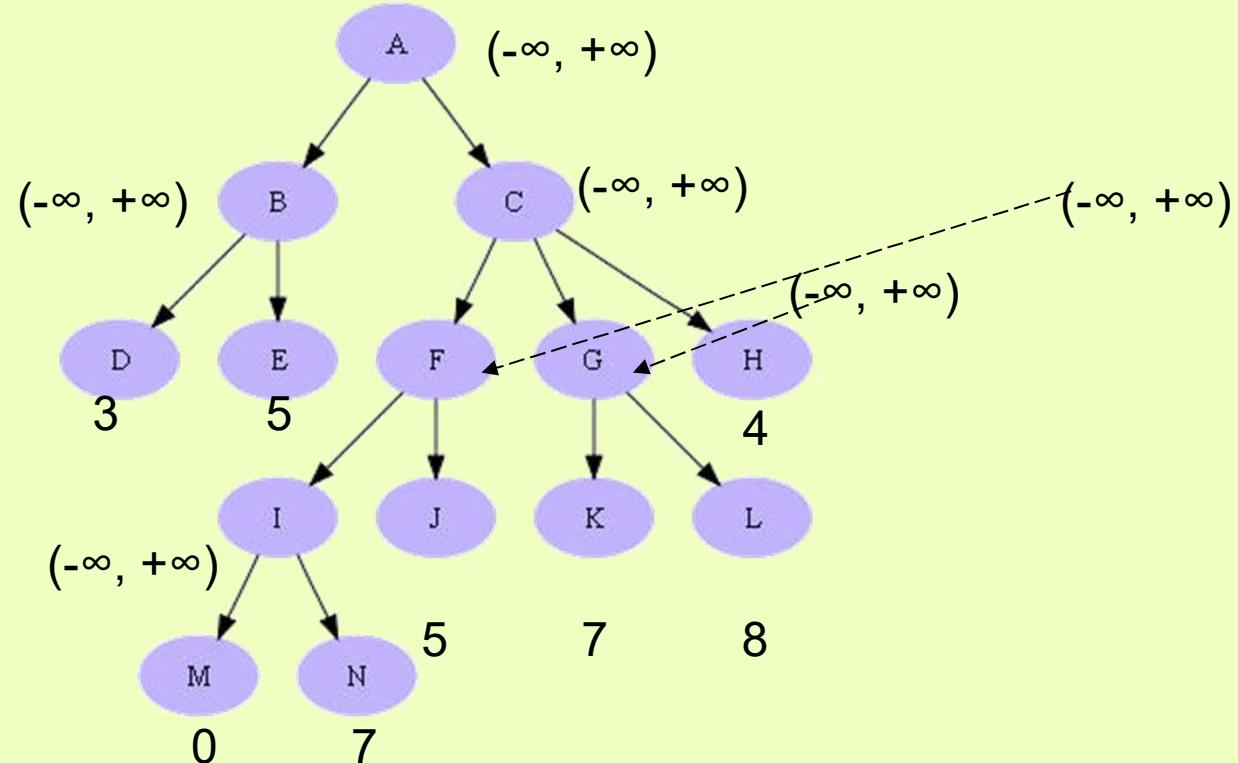
MAX

MIN

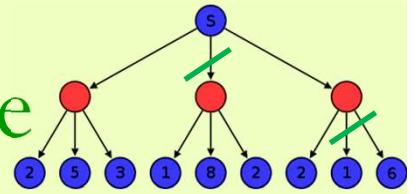
MAX

MIN

MAX



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

□ Exemplu

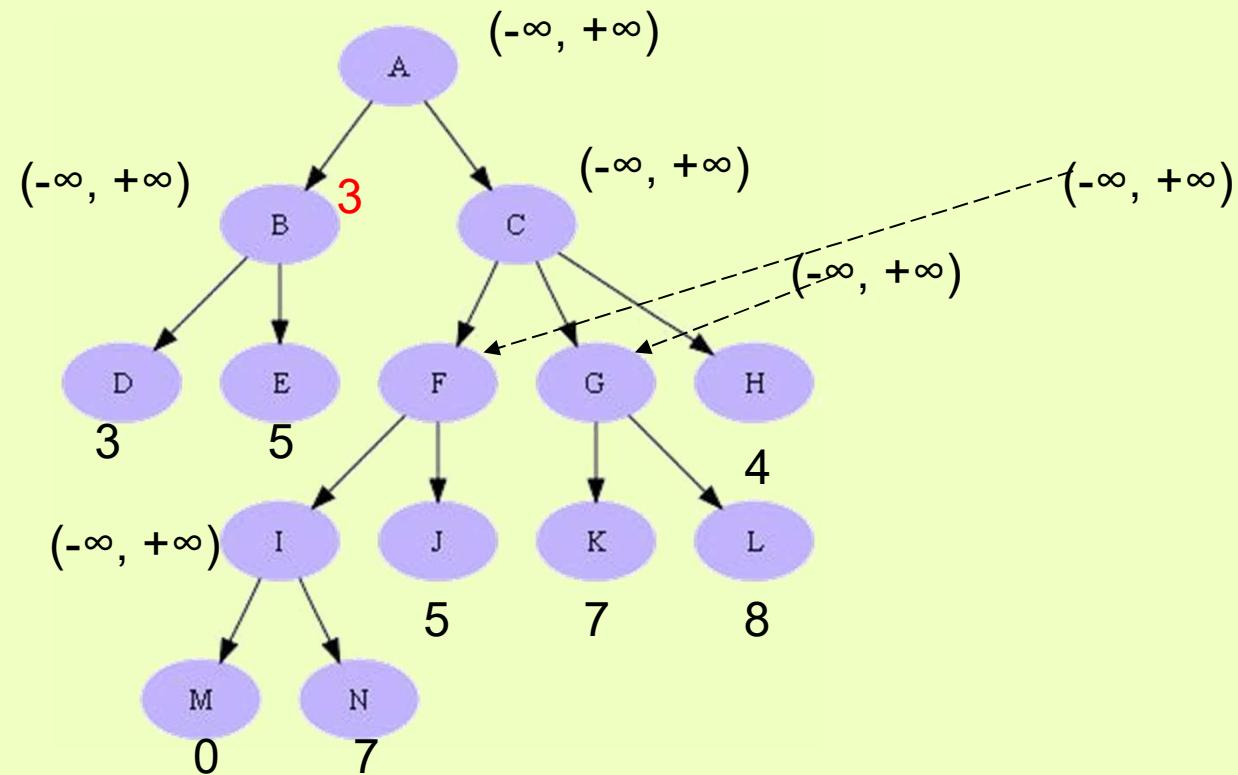
MAX(α)

MIN(β)

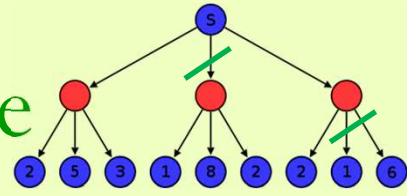
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

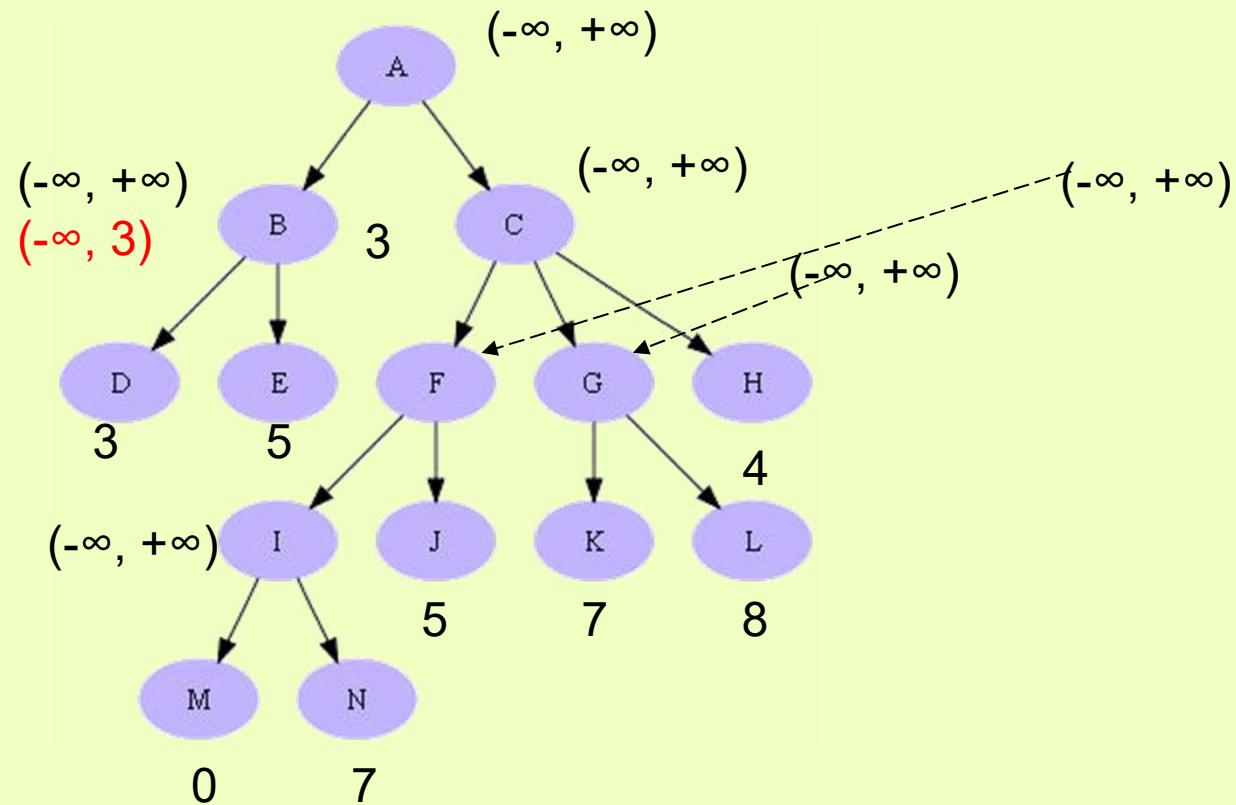
MAX(α)

MIN(β)

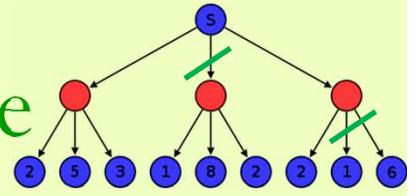
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

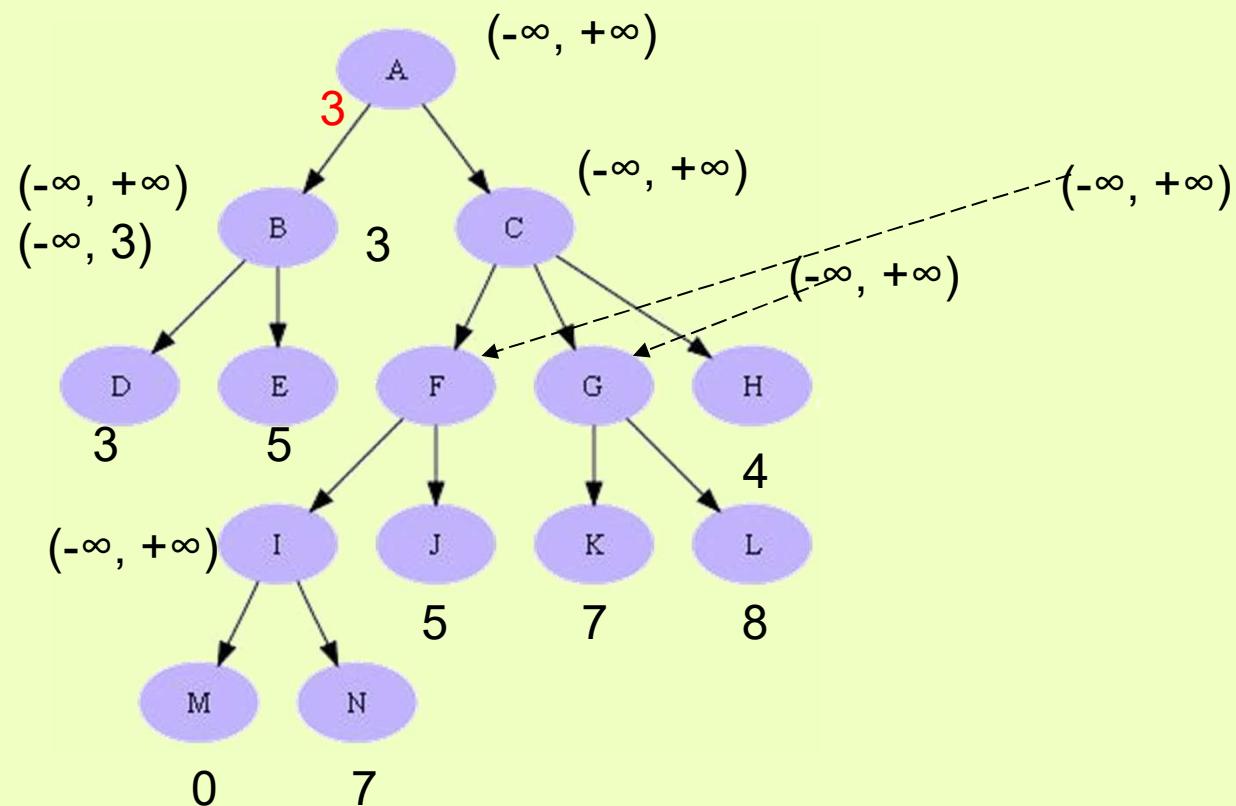
MAX(α)

MIN(β)

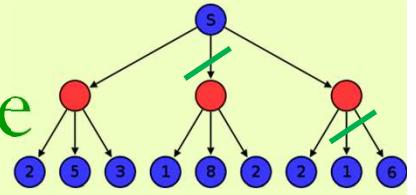
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

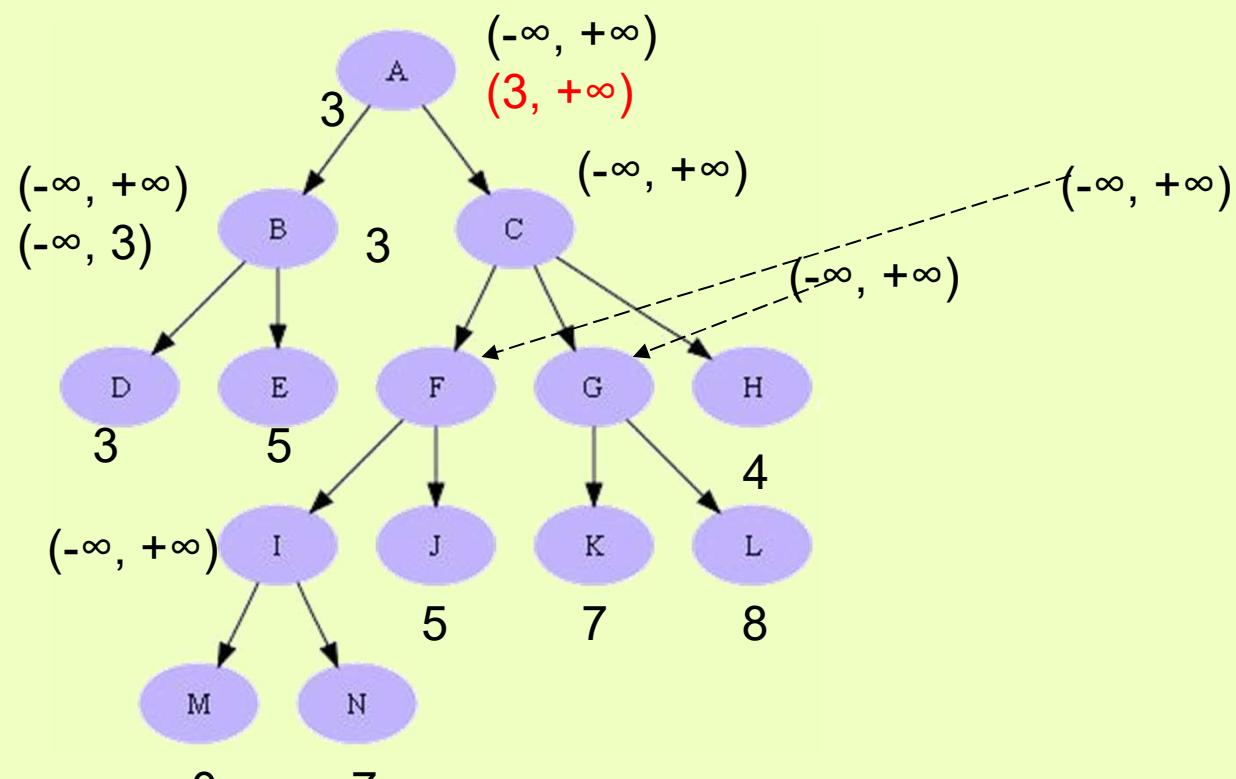
MAX(α)

MIN(β)

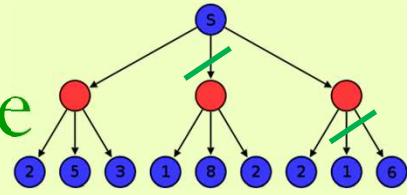
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

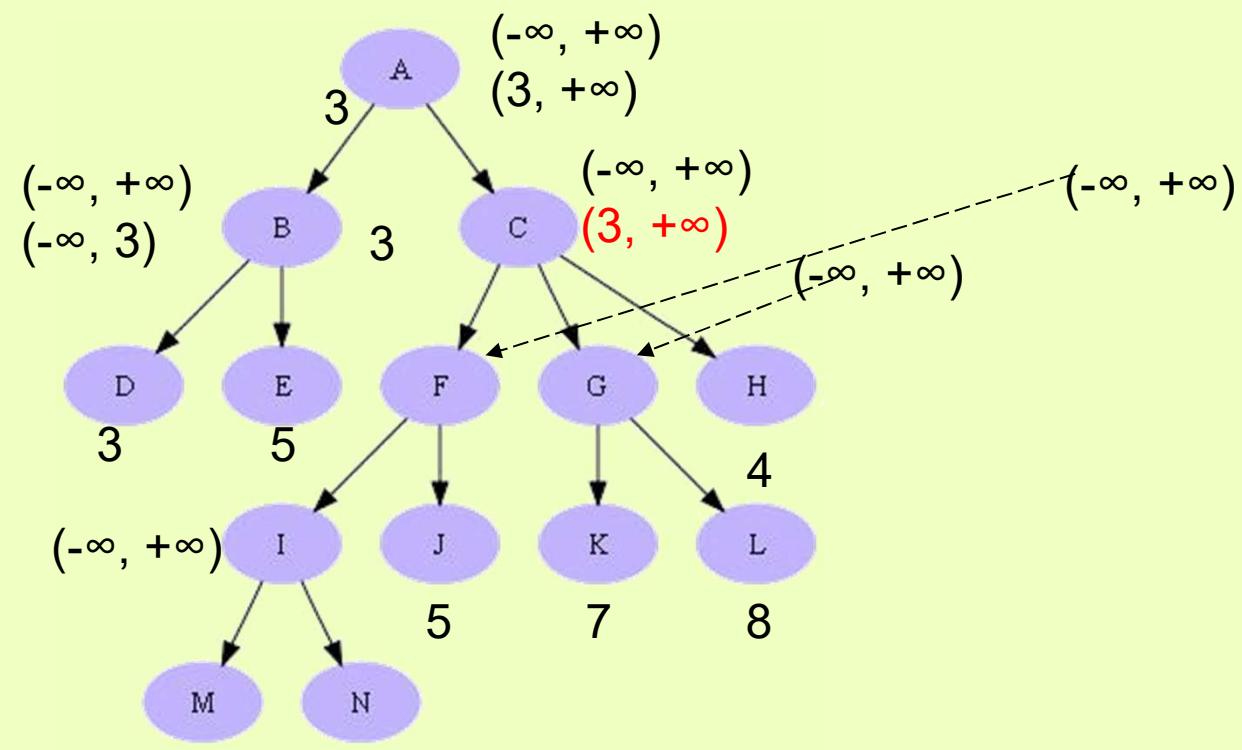
MAX(α)

MIN(β)

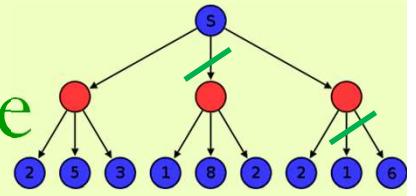
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

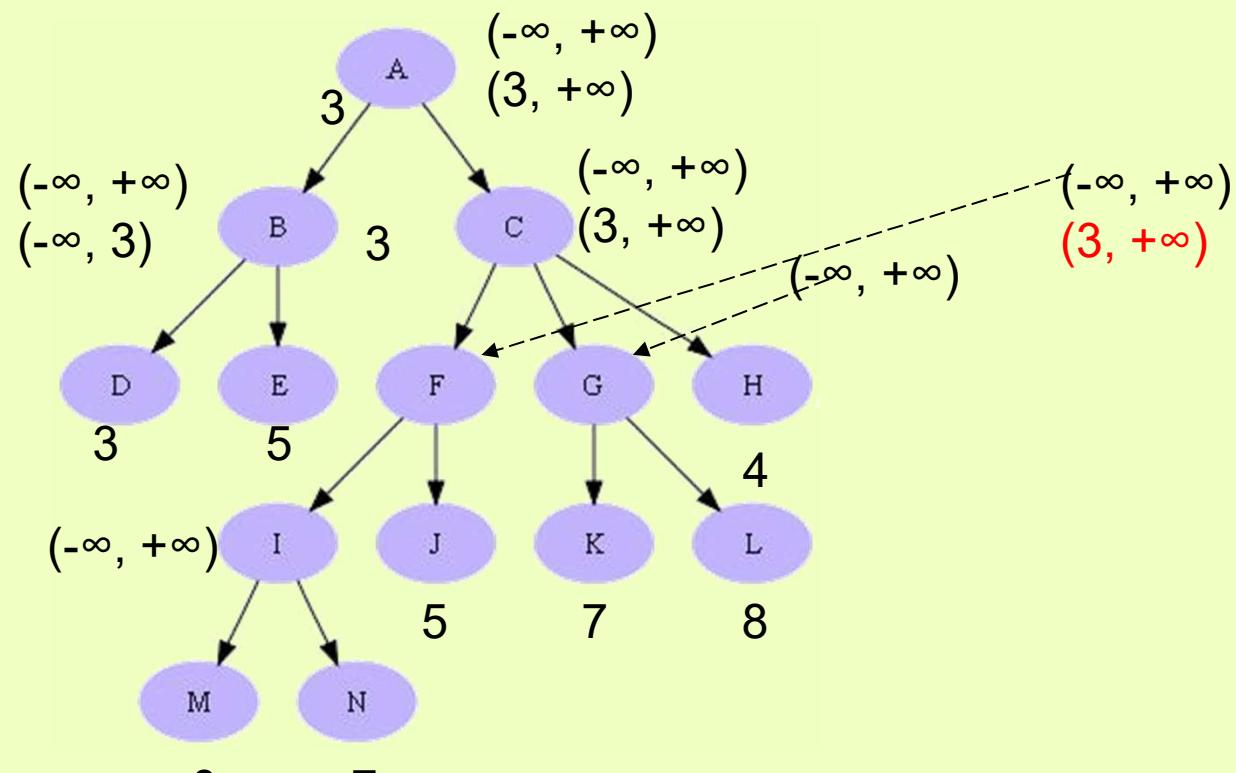
MAX(α)

MIN(β)

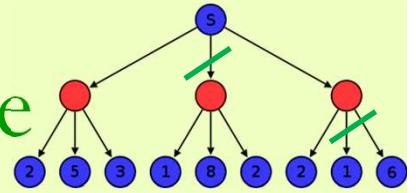
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

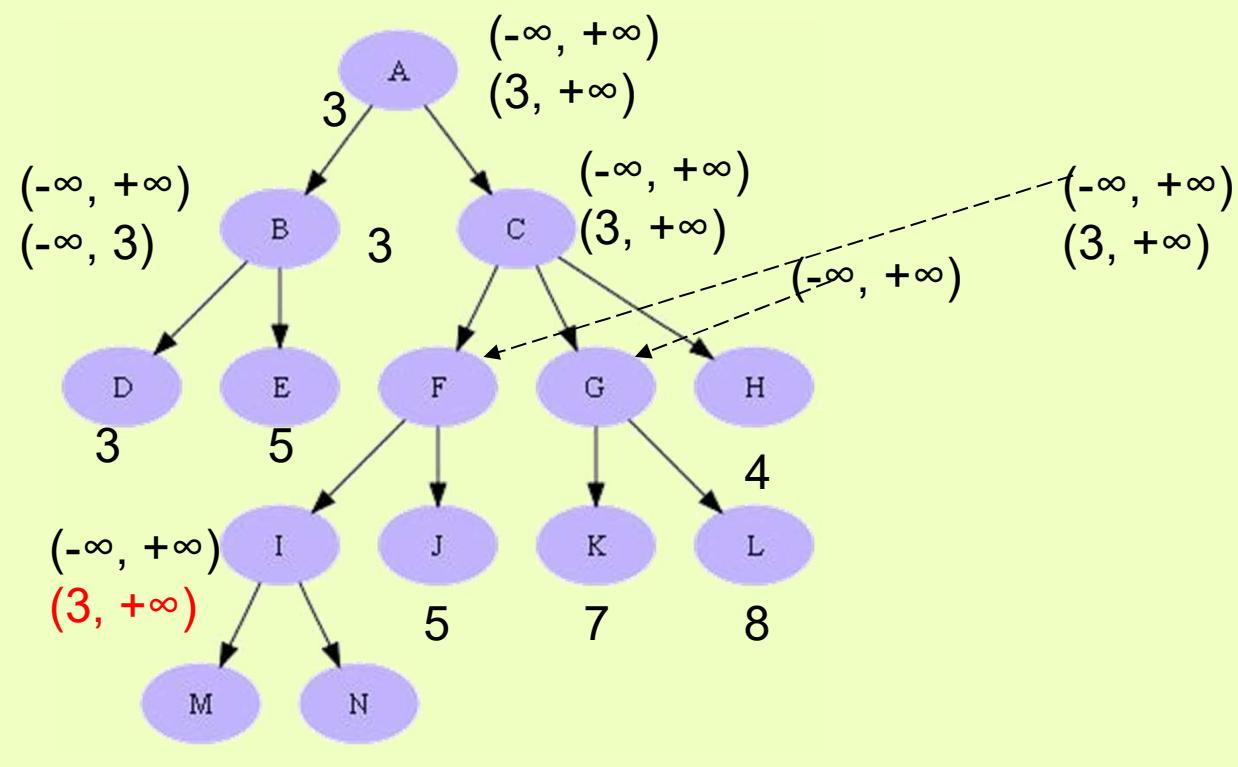
MAX(α)

MIN(β)

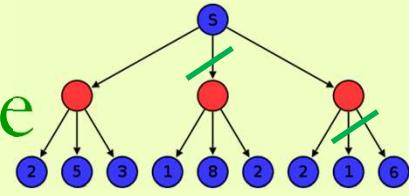
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

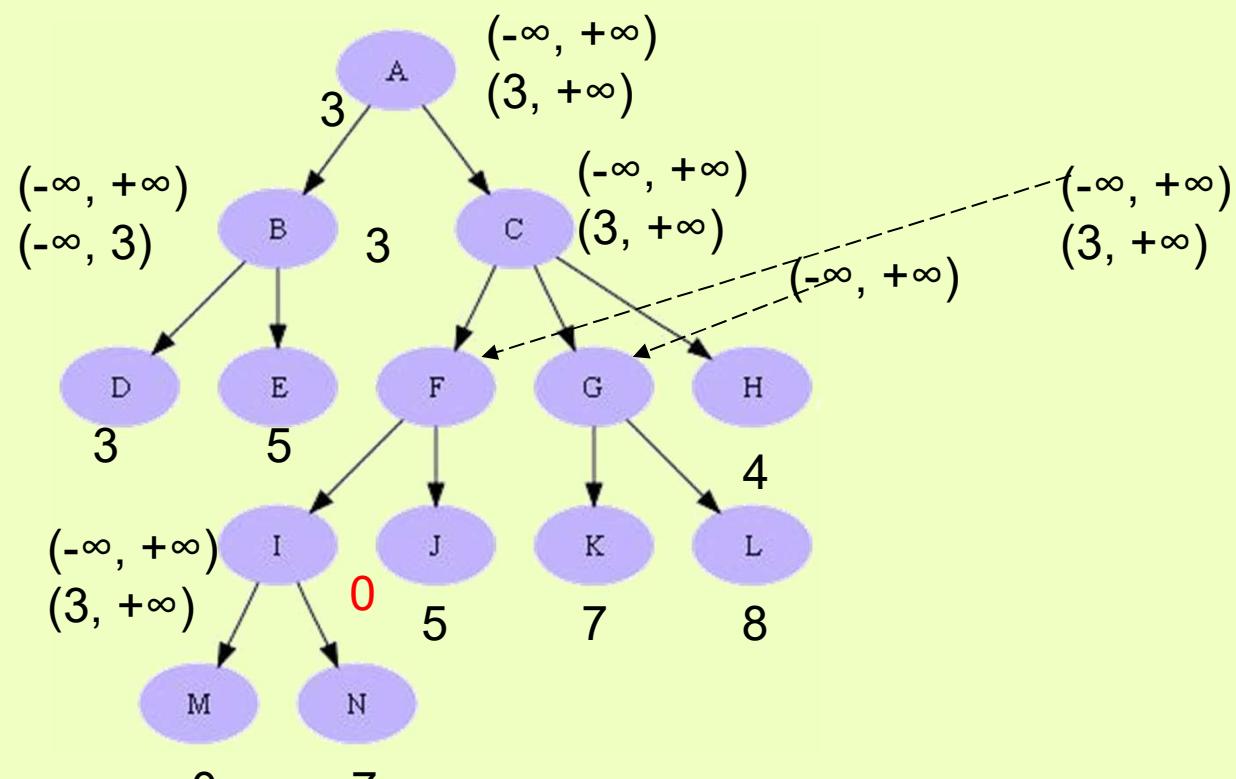
MAX(α)

MIN(β)

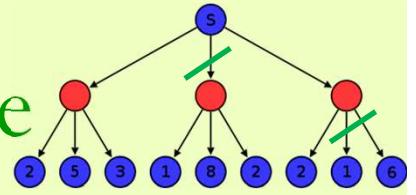
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

□ Exemplu

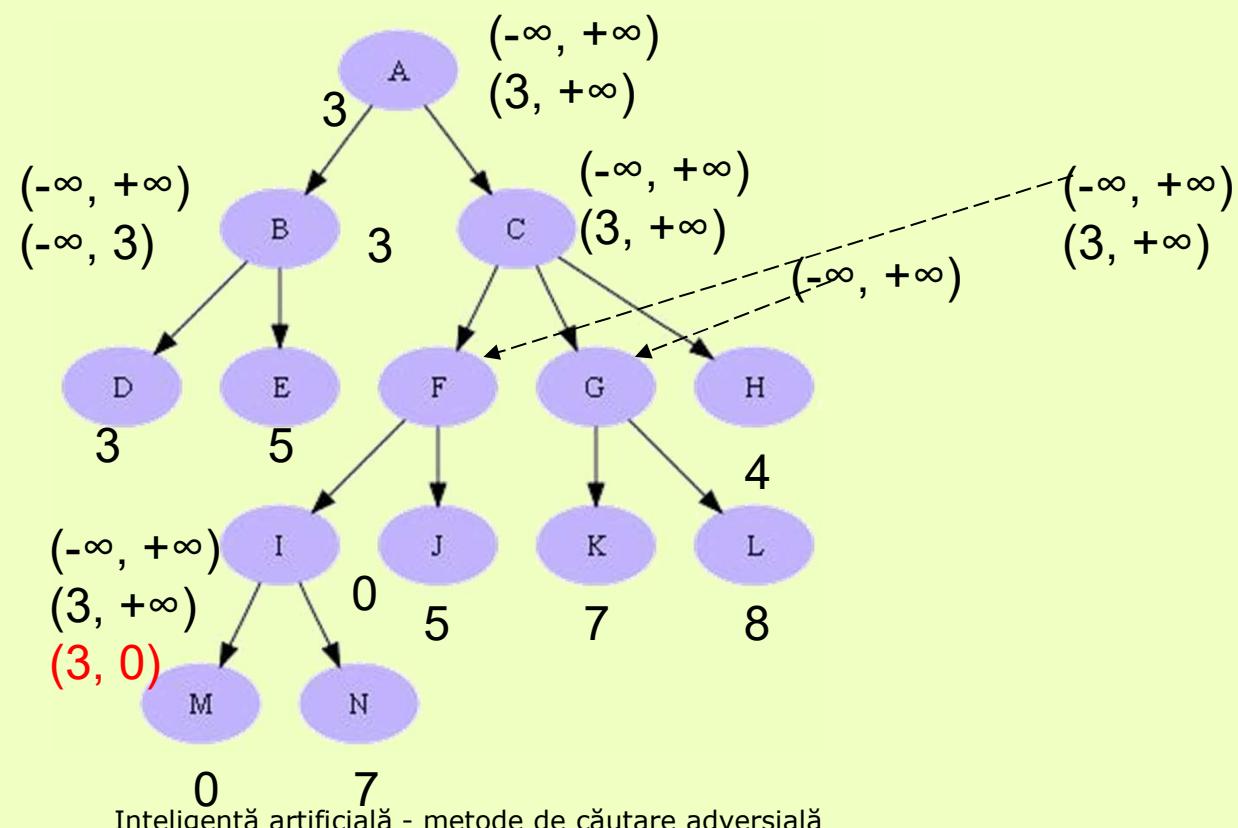
MAX(α)

MIN(β)

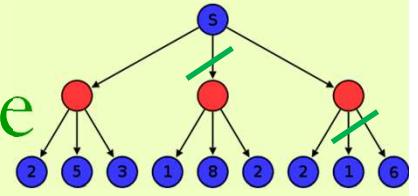
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

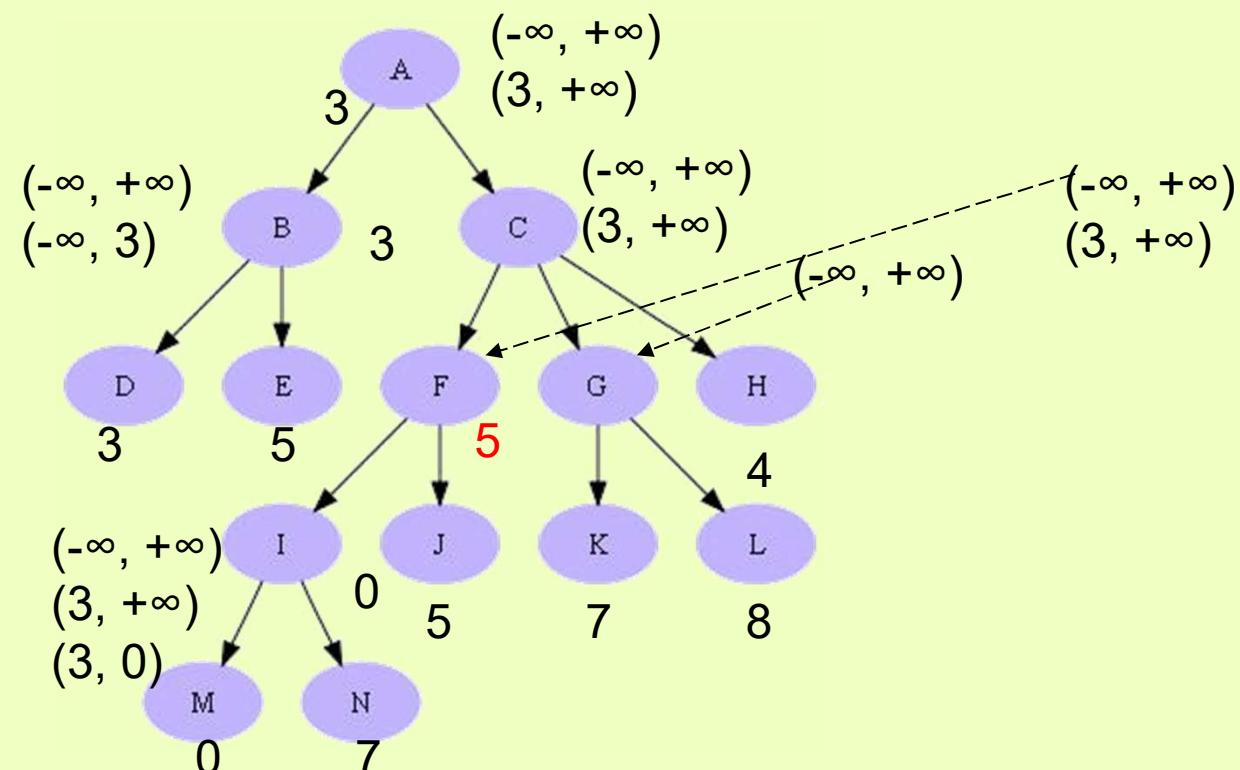
MAX(α)

MIN(β)

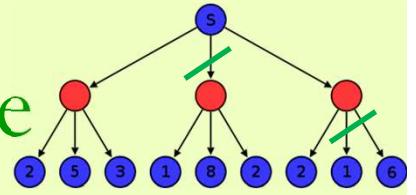
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

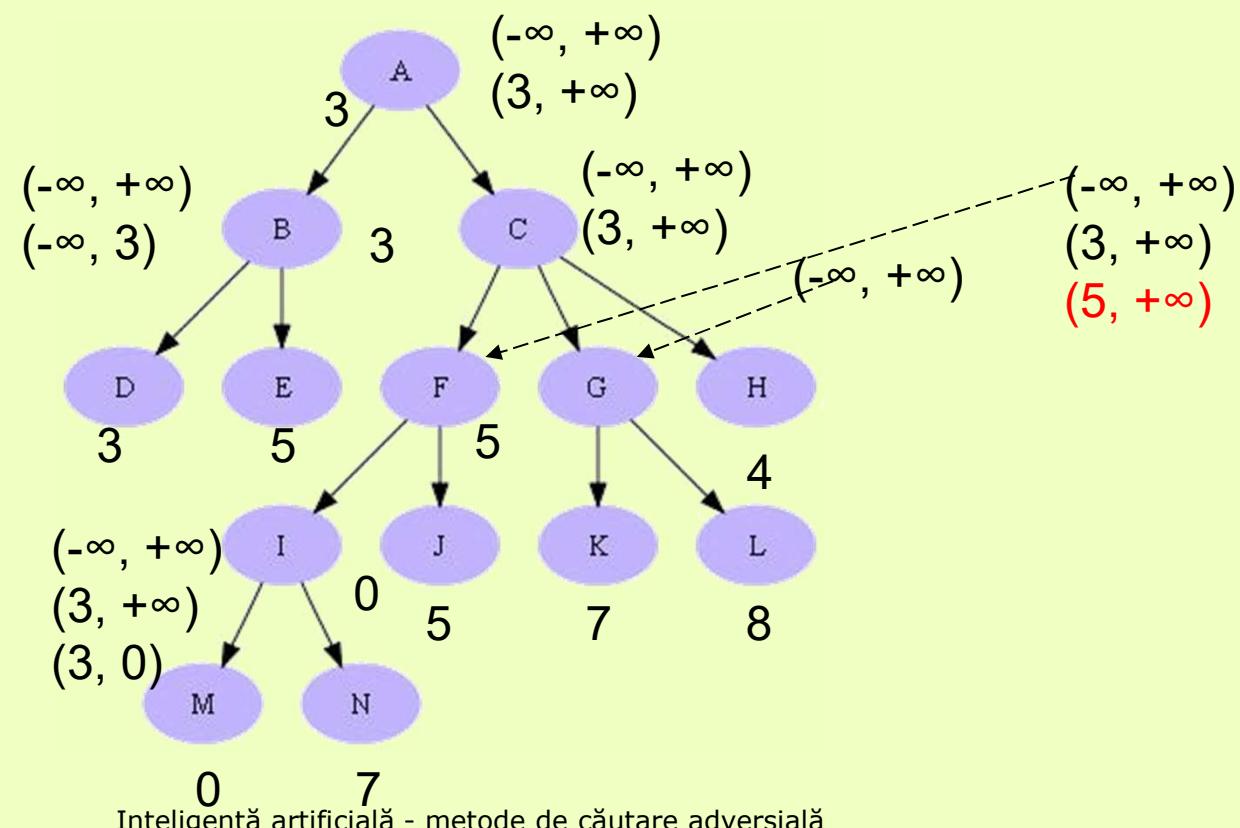
MAX(α)

MIN(β)

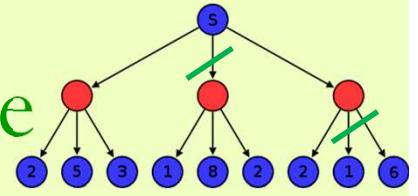
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

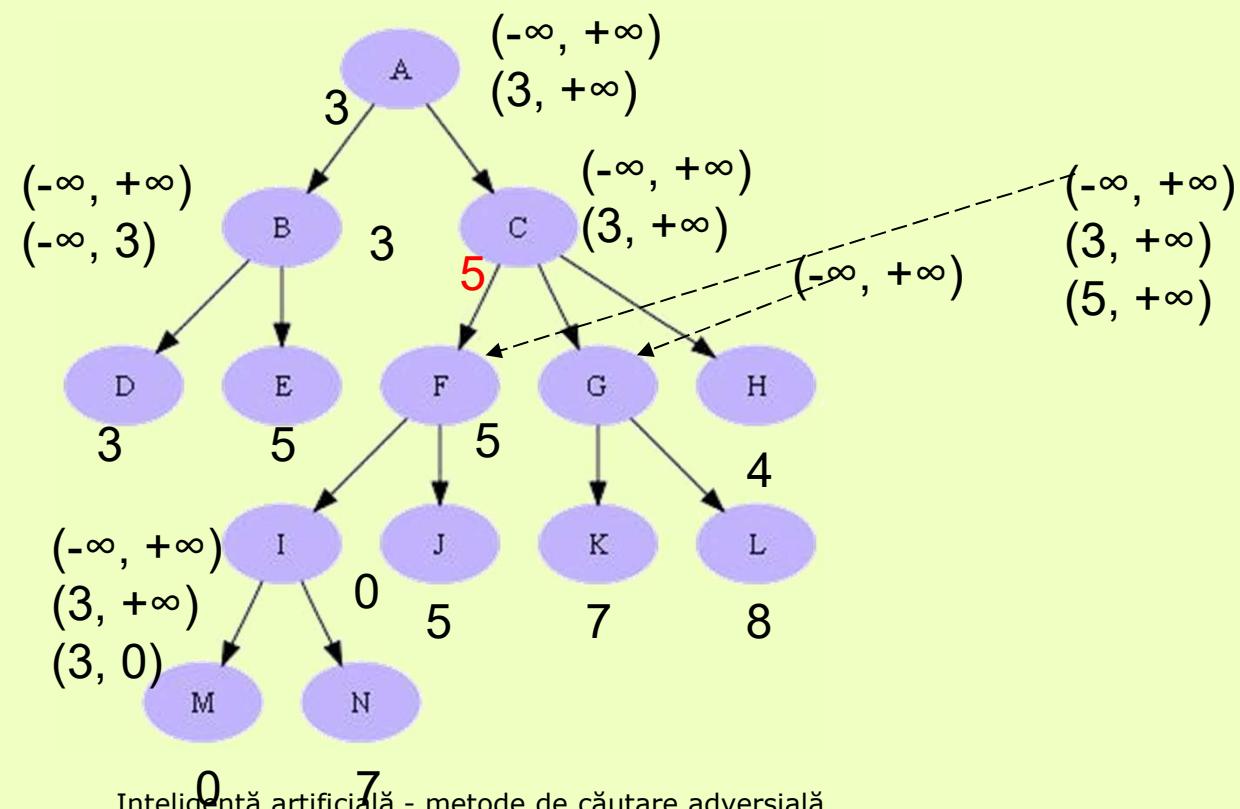
MAX(α)

MIN(β)

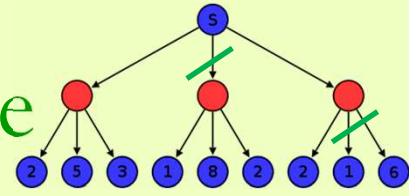
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

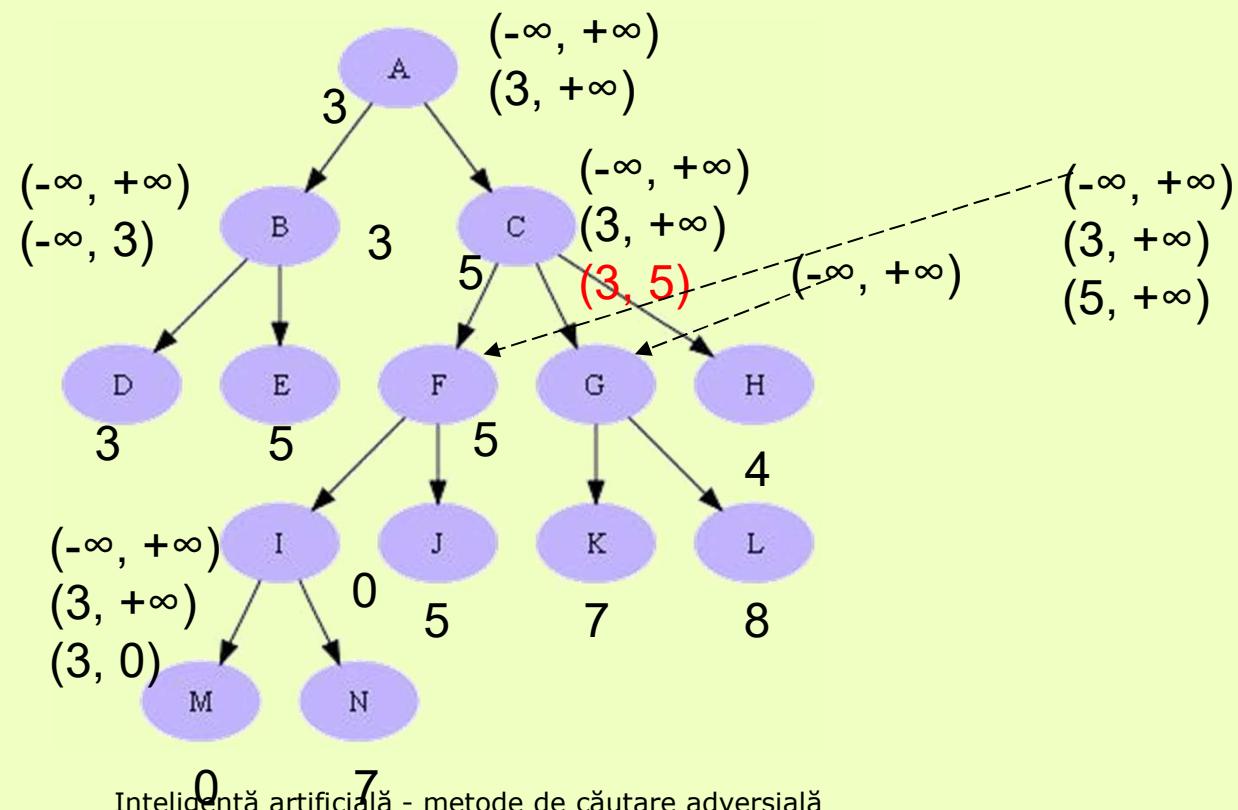
MAX(α)

MIN(β)

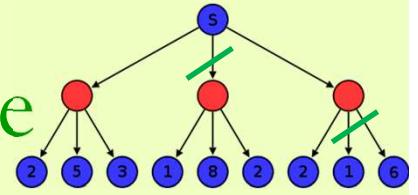
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

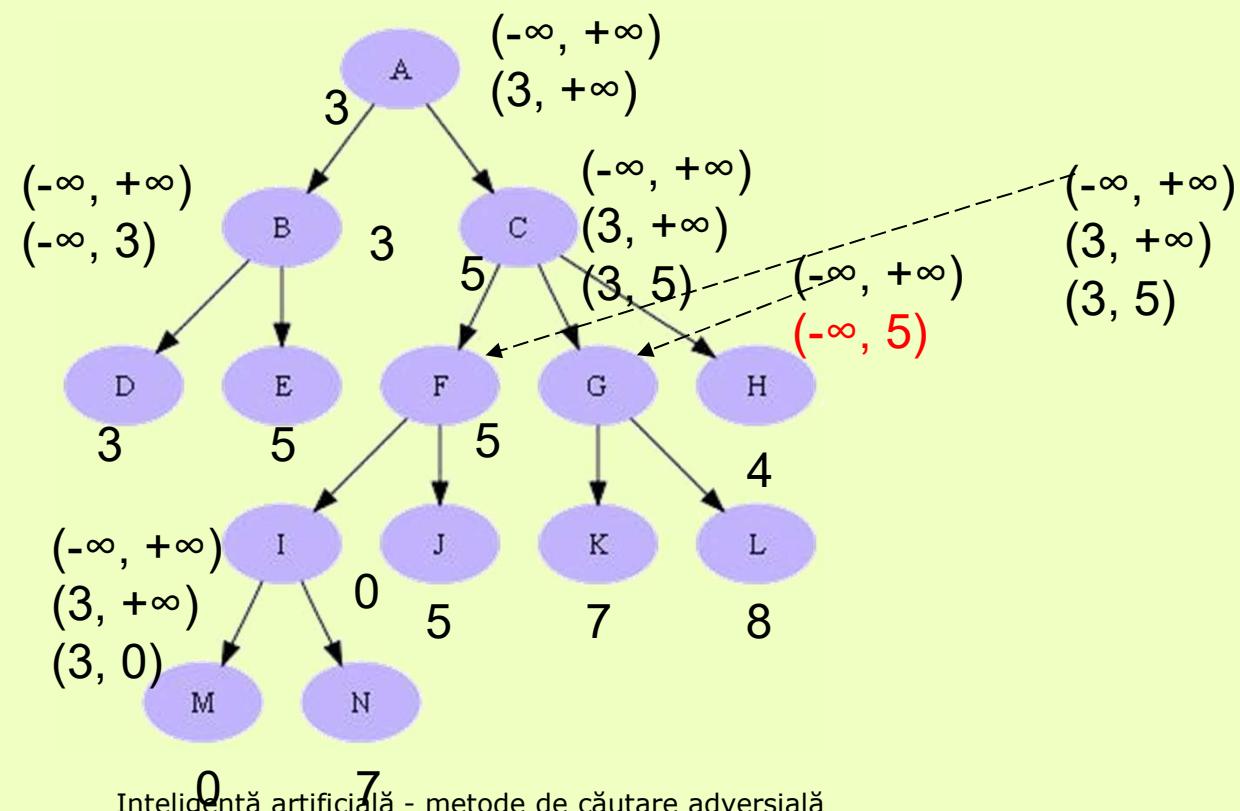
MAX(α)

MIN(β)

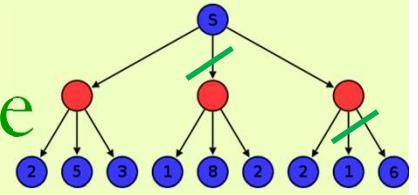
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

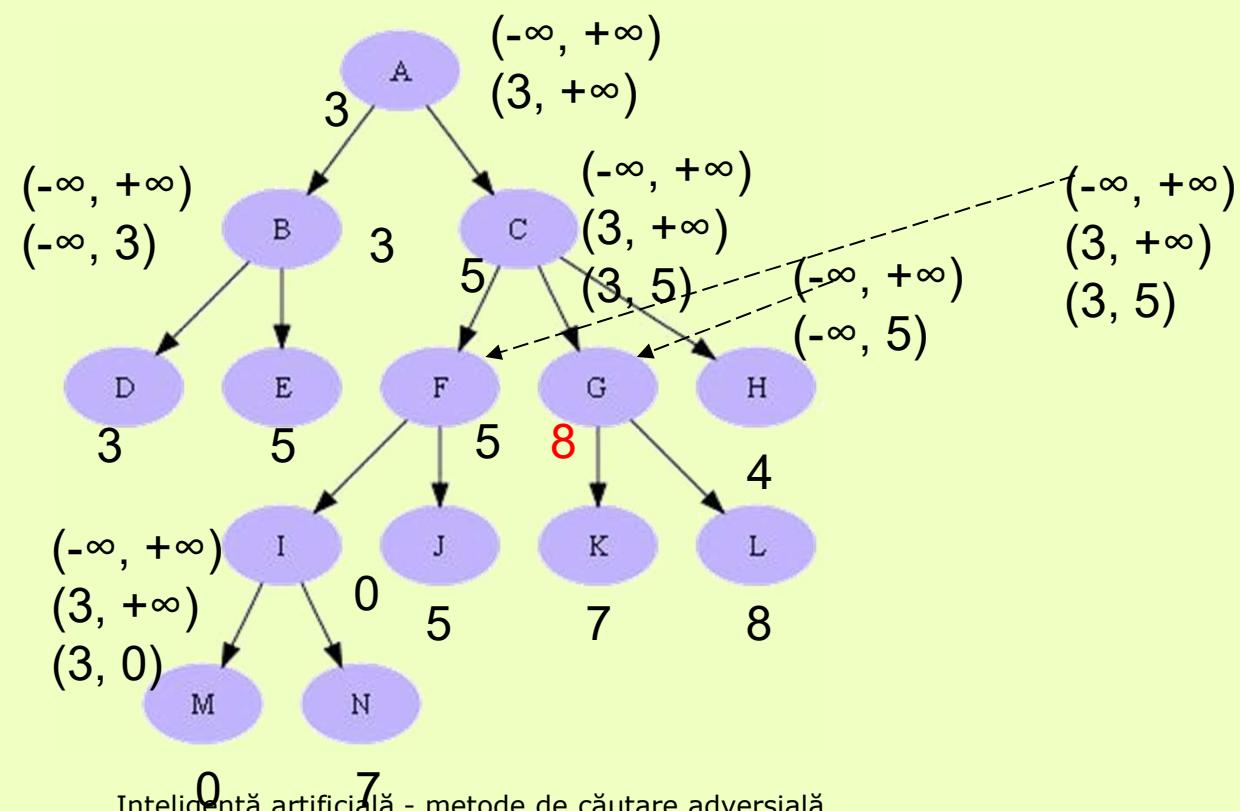
MAX(α)

MIN(β)

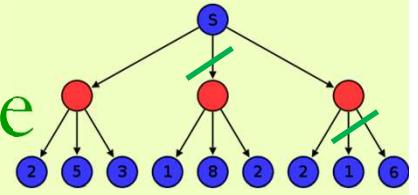
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

□ Exemplu

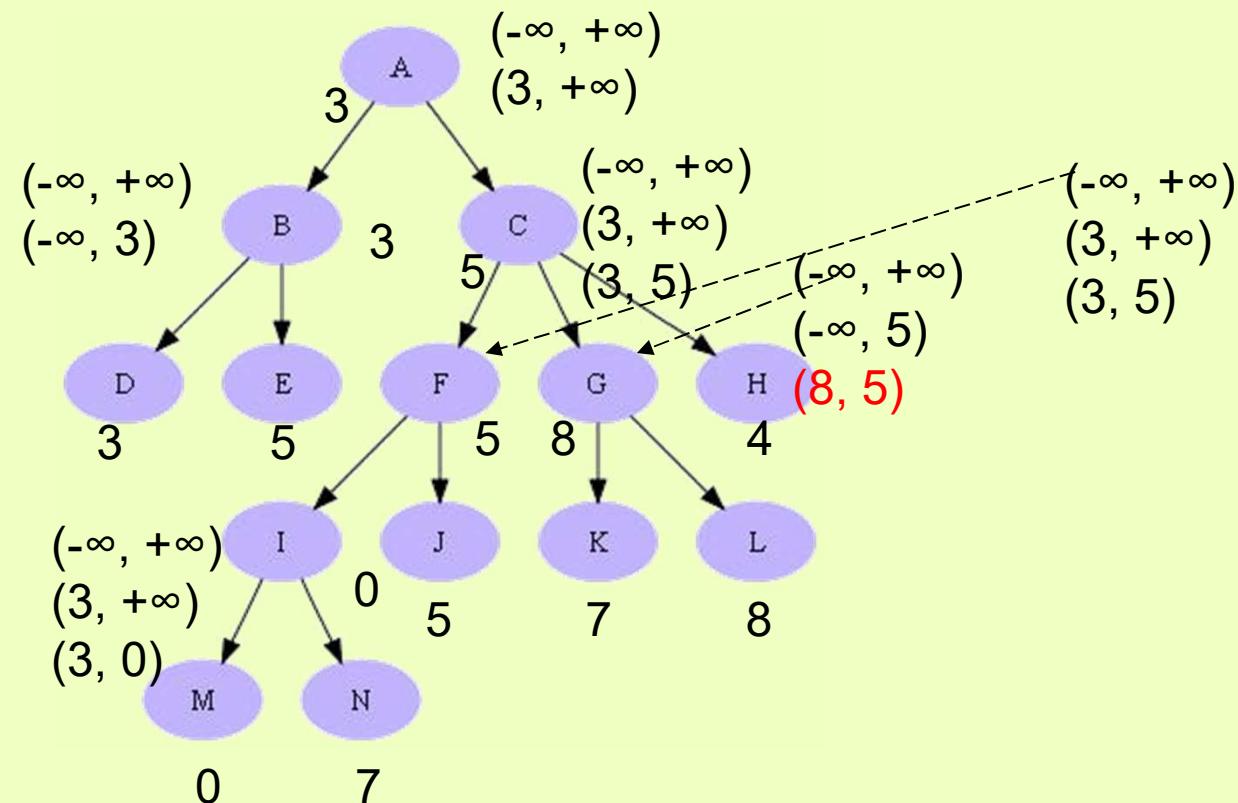
MAX(α)

MIN(β)

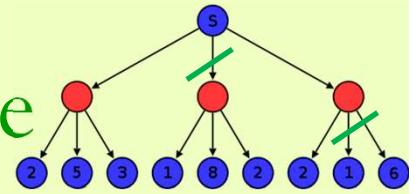
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

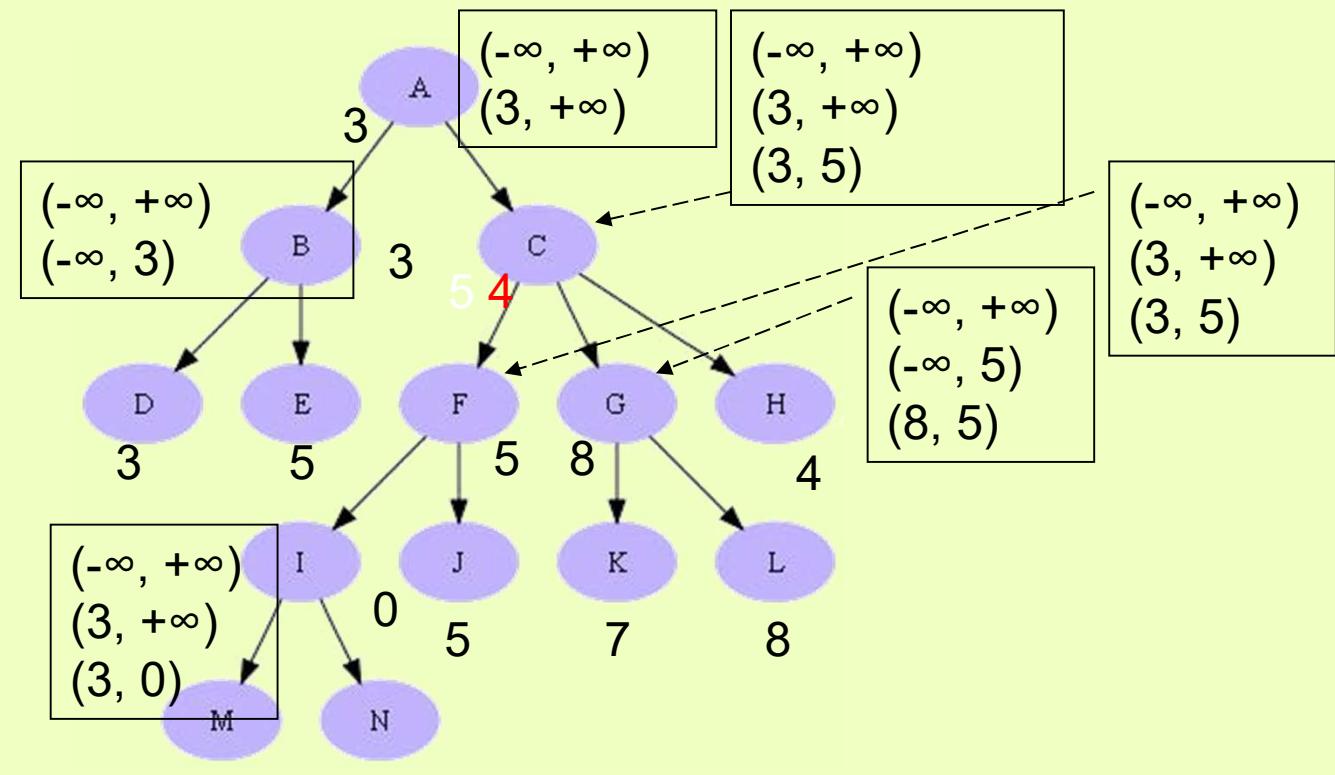
MAX(α)

MIN(β)

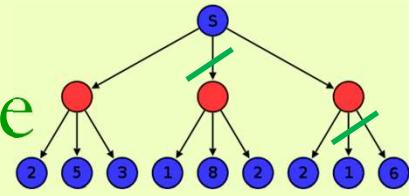
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

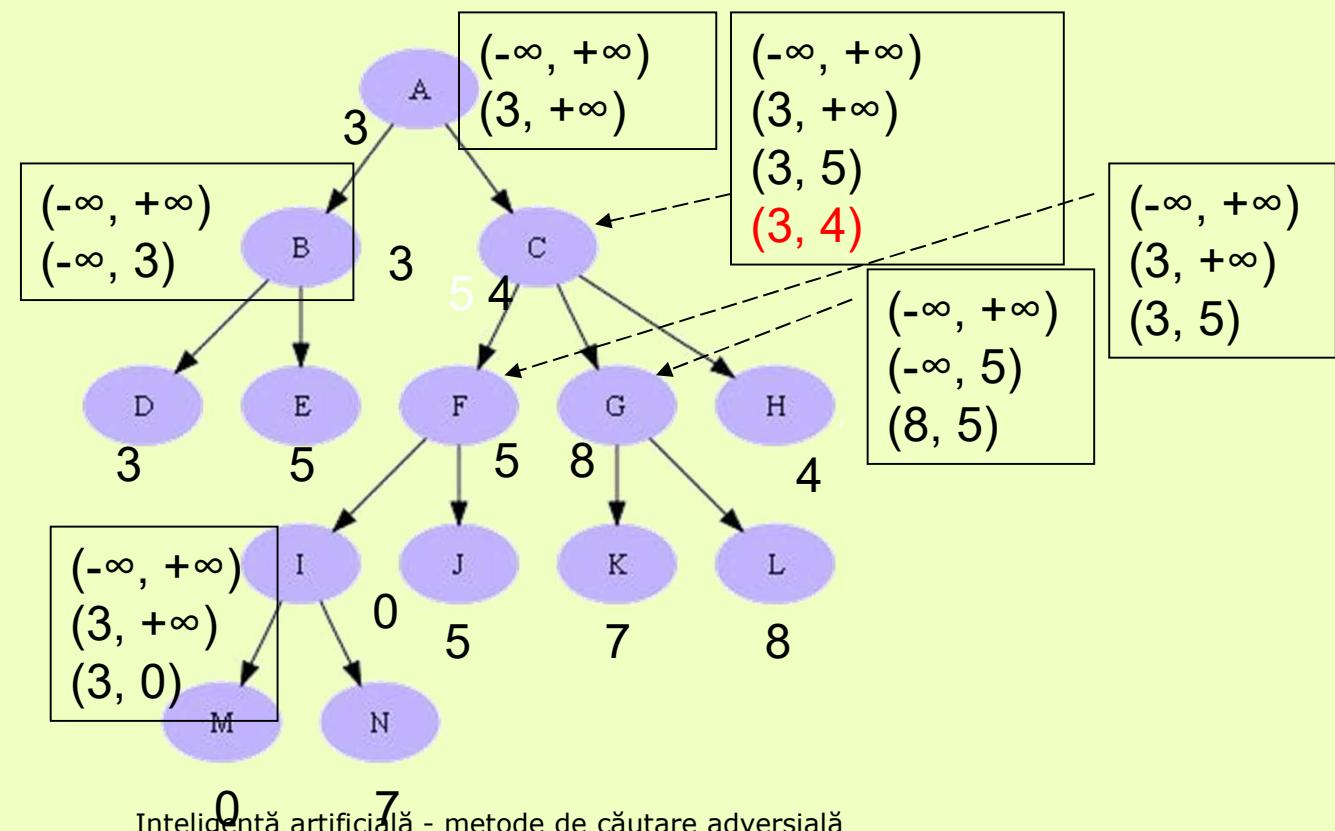
MAX(α)

MIN(β)

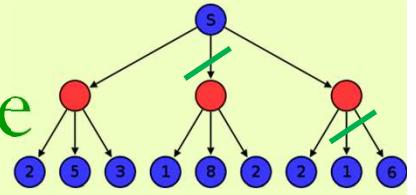
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare α - β

- Exemplu

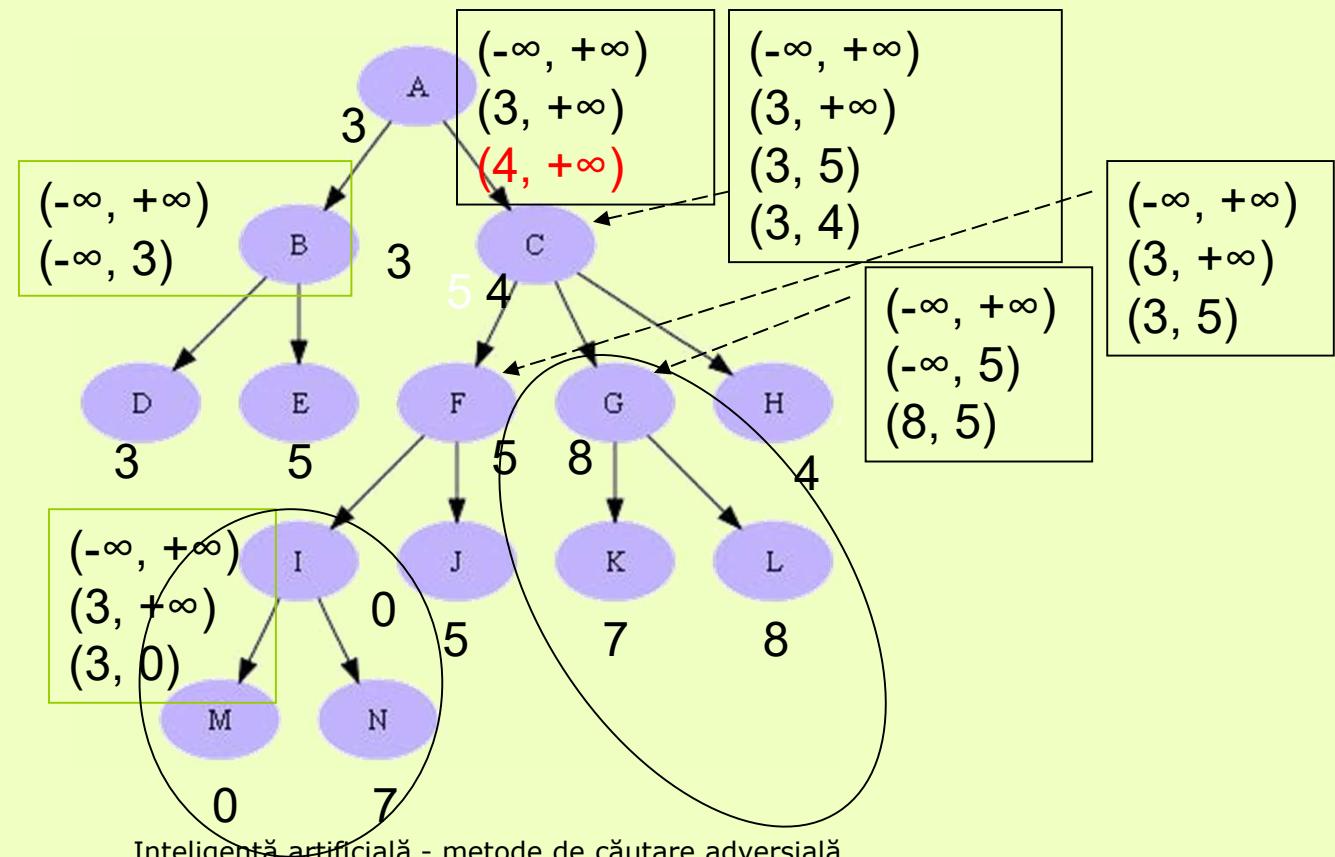
MAX(α)

MIN(β)

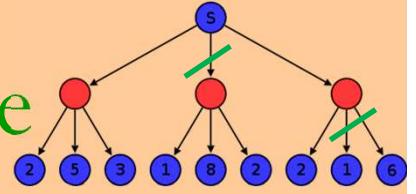
MAX(α)

MIN(β)

MAX(α)



Jocurile și căutarea – spațiul de căutare

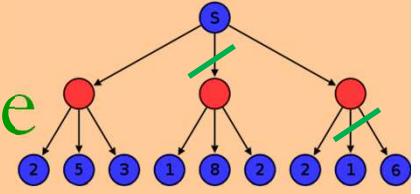


Strategii de joc → MiniMax cu retezare α - β

□ Proprietăți

- Reducerea nu afectează rezultatul final
- O bună ordonare a mutărilor îmbunătățește algoritmul de reducere
- Dacă succesorii sunt puși perfect în ordine (cei mai buni se află primii), atunci complexitatea temporara ar fi = $O(b^{d/2})$, în loc de $O(b^d)$ cat are minimax
- Se poate transforma un program de la nivelul începător la nivelul expert

Jocurile și căutarea – spațiul de căutare

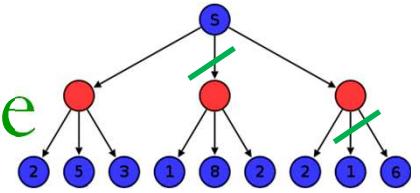


Strategii de joc → MiniMax cu retezare α - β

□ MinMax vs. MinMax cu retezare α - β

	MinMax	MinMax α-β
Complexitate temporală	$O(b^m)$	$O(b^{m/2})$ cu ordonare perfectă a nodurilor
Complexitate spațială	$O(b^m)$	$O(2b^{d/2})$ – cel mai bun caz (când unui nod Max îi este generat ca prim copil cel cu valoarea cea mai mare și când lui Min îi este generat ca prim copil cel cu valoarea cea mai mică) $O(b^d)$ – cel mai rău caz (fără retezare)
Compleitudine	Da	Da
Optimalitate	Da	Da

Jocurile și căutarea – spațiul de căutare



- Arbori AndOR, MiniMax, MiniMax cu retezare $\alpha\text{-}\beta$
 - Complexitatea mărită → necesitatea unor tehnici eficiente de căutare în rezolvarea jocurilor
 - Jocul de șah
 - $b \sim 35$
 - $d \sim 100$
 - $b^d \sim 35^{100} \sim 10^{154}$ noduri
 - Jocul Tic-Tac-Toe
 - ~ 5 mutări legale dintr-un total de 9 mutări
 - $5^9 = 1\ 953\ 125$
 - $9! = 362\ 880$ (dacă computerul mută primul)
 - $8! = 40\ 320$ (dacă computerul mută al doilea)
 - Jocul Go
 - $b > 361$ (pentru o tablă de 19x19)

Jocurile și căutarea – spațiul de căutare



□ În practică

- Jocul de dame → Chinok
 - Chinok îl învinge pe Marion Tinsley în 1994 (după 40 de ani de confruntări)
 - Se folosește o bază cu finaluri de joc calculate pentru toate pozițiile unui joc perfect jucat cu cel mult 8 piese (444 bilioane de poziții posibile)
- Jocul de șah → Deep Blue
 - Deep Blue l-a învins pe Garry Kasparov în 1997
 - Se verifică 200 mil poziții/sec
 - Factorul de ramificare se reduce la 6 cu rețeza α - β (față de 35-40)
- Jocul Othello
 - Jucătorii umani refuză să joace cu computerele (pt că sunt prea bune) – Moor (1980), Logistello (1997)
- Jocul Go
 - Jucătorii umani refuză să joace cu computerele (pt că sunt prea slabe)
 - Factorul de ramificare > 300 → necesitatea unor algoritmi bazați pe pattern-uri
- Jocul de table
 - BKG (logica fuzzy), TD-Gammon (rețele neuronale artificiale)
 - Computerul l-a învins pe campionul lumii, dar pt că a fost norocos

Jocurile și căutarea – spațiul de căutare



□ Strategia de joc

■ Pas cu pas

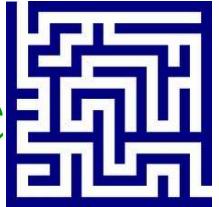
- Ex.: XO, Dame, Şah
- Algoritmi – pot lucra cu structuri:

- Liniare
 - Strategia simetriei
 - Strategia perechilor
 - Strategia parității
 - Programare dinamică
 - Alte strategii
- Arborescente
 - Arbori AndOr
 - MiniMax (cu tăieturi Alpha-Beta)

■ Completă

- Ex.: ieșirea dintr-un labirint, deplasarea pe o hartă între 2 locații date
- Algoritmi pot să identifice
 - **Un drum optim de la o locație la alta (pathfinding)**
 - O succesiune de acțiuni care să deplaseze jucătorul de la o locație la alta

Jocurile și căutarea – spațiul de căutare



Strategii de joc → Pathfinding

❑ Aspecte teoretice

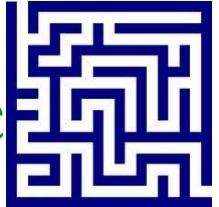
■ Problema

- ❑ Identificarea pe o hartă (posibil cu obstacole) a unui drum optim de la o locație la alta
- ❑ Unde?
 - pe o hartă
 - harta ca o matrice
 - harta ca un graf (simplu, *mesh*, *quad-tree*)
 - într-un mediu (cunoscut/necunoscut, static/dinamic)
- ❑ De către cine?
 - un agent
 - 2 agenți
 - mai mulți agenți

■ Soluția

- ❑ Utilizarea unei metode de căutare (optimizare)

Jocurile și căutarea – spațiul de căutare



Strategii de joc → Pathfinding

❑ Aspecte teoretice

■ Soluția

❑ Cum?

- Utilizarea unei metode de căutare (optimizare)

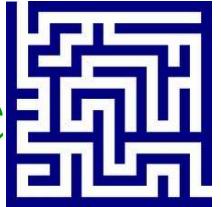
❑ Care soluție este cea mai bună?

- viteza de calcul,
- lungimea drumului,
- calitatea drumului,
- informația disponibilă

❑ Dificultăți

- soluție oferită în timp real
- resurse limitate
- spațiu de căutare imens
- modelarea situațiilor reale implică incertitudini și elemente (teren, unități, agenți) eterogene

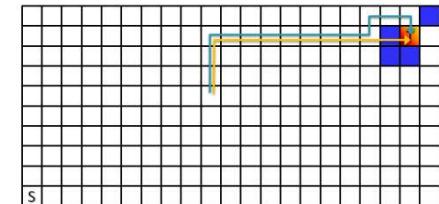
Jocurile și căutarea – spațiul de căutare



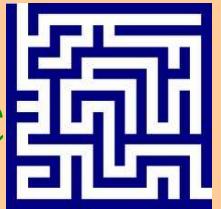
Strategii de joc → Pathfinding

- Algoritmi de căutare pentru hărți reprezentate ca “matrici de dale” → labirinturi

- De ce?
 - Simplitate
 - Caracteristici
 - Spațiu de căutare discret
 - Dalele sunt pătrate
 - Dalelele pot fi traversabile sau blocate
 - Mișcări în linie dreaptă de pe o dală pe alta (orizontală, verticală, diagonală)
 - Metode
 - A*
 - Algoritmi de tip *Best-first search*
 - Reprezintă un standard în industria jocurilor
 - Euristici → Manhattan (gen. Minkowski)
 - Estimarea distanței de la punctul curent la destinație
 - Se ignoră obstacolele



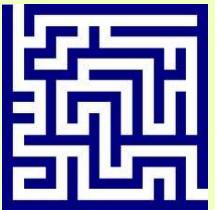
Jocurile și căutarea – spațiul de căutare



Strategii de joc → Pathfinding

- Algoritmi de căutare pentru hărți reprezentate ca un graf
 - De ce?
 - Reprezentarea matriceală → zone mari din hartă care au același cost
 - Caracteristici
 - Spațiu de căutare discret
 - “Dalele” pot avea orice formă
 - Dalelele pot fi traversabile sau blocate
 - Mișcări oarecare de pe o dală pe alta
 - Metode
 - Deplasare pe muchii
 - Deplasare pe noduri
 - Puncte
 - Marginile poligonului
 - Mijlocul poligonului
 - Ex. Dijkstra, Floyd-Warshall, Kruskal, etc

Jocurile și căutarea – spațiul de căutare

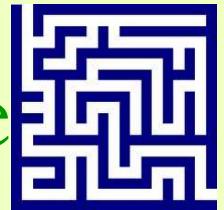


Strategii de joc → Pathfinding

□ Îmbunătățiri ale algoritmilor de căutare

- Euristici mai bune
- Abstractizări ierarhice
- Abordări descentralizate

Jocurile și căutarea – spațiul de căutare



Strategii de joc → Pathfinding

■ Îmbunătățiri ale algoritmilor de căutare

■ Euristici mai bune

□ Euristici fără memorie

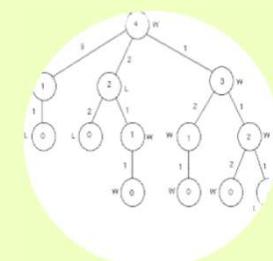
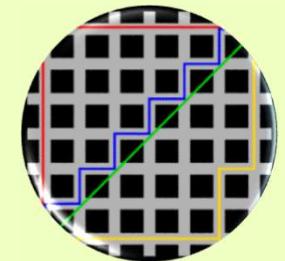
- Ex. Manhattan
- Foarte rapid de calculat
- Nu necesită memorie suplimentară
- Calitate bună uneori

□ Euristici bazate pe memorie

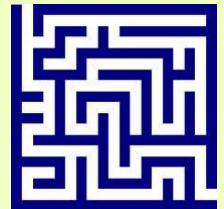
- Ex. euristici bazate pe repere
- Destul de rapide de calculat
- Necesită memorie
- Calitate bună (identificarea drumurilor moarte)

□ Informație perfectă

- Foarte costisitoare la calculare și stocare
- Foarte rapidă la utilizat (după ce au fost calculate)
- Ne-practicabile



Jocurile și căutarea – spațiul de căutare



Strategii de joc → Pathfinding

□ Îmbunătățiri ale algoritmilor de căutare

■ Abstractizări ierarhice

□ Grafuri în care

- sunt adnotate și fluxurile de mișcare (deplasări în ambele sensuri)
- sunt asociate relații de dominantă între muchii
- Sunt identificate nivele (ex. cameră, casă, oraș, țară)

□ Abordări descentralizate

□ Ideea de bază:

- Descompunerea problemei în sub-probleme care
 - pot fi rezolvate repede
 - pot fi sub-optimale
 - pot fi incomplete

□ Scop

- toate elementele (identificare colaborativă) să ajungă la destinație

□ Dificultate

- se mărește spațiul de căutare: $O(n) \rightarrow O(n^m)$
- factorul de ramificare explodează (de la 4 sau 8 la 5^m sau 9^m)

Jocurile și căutarea – spațiul de căutare



Strategii de joc → Planning

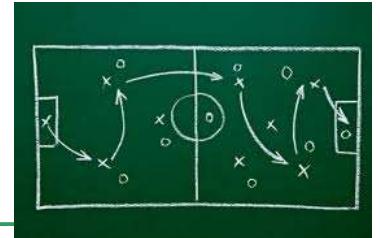
❑ Aspecte teoretice

■ Problema

- ❑ Planificarea (ordonarea) unor acțiuni
 - mutări, deplasări, alegeri, răsuciri, etc
- ❑ Se dau
 - O stare inițială



Jocurile și căutarea – spațiul de căutare



Strategii de joc → Planning

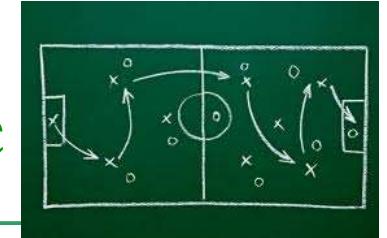
❑ Aspecte teoretice

■ Problema

- ❑ Planificarea (ordonarea) unor acțiuni
 - mutări, deplasări, alegeri, răsuciri, etc
- ❑ Se dau
 - O stare inițială
 - O stare obiectiv (finală)



Jocurile și căutarea – spațiul de căutare



Strategii de joc → Planning

❑ Aspecte teoretice

■ Problema

- ❑ Planificarea (ordonarea) unor acțiuni
 - mutări, deplasări, alegeri, răsuciri, etc
- ❑ Se dau
 - O stare inițială
 - O stare obiectiv (finală)
 - Un set de acțiuni posibile



Jocurile și căutarea – spațiul de căutare



Strategii de joc → Planning

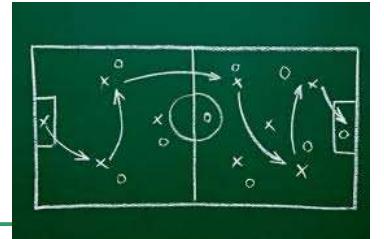
❑ Aspecte teoretice

■ Problema

- ❑ Planificarea (ordonarea) unor acțiuni
 - mutări, deplasări, alegeri, răsuciri, etc
- ❑ Se dau
 - O stare inițială
 - O stare obiectiv (finală)
 - Un set de acțiuni posibile
- ❑ Se cere
 - Să se identifice o secvență de acțiuni care transformă starea inițială în starea finală



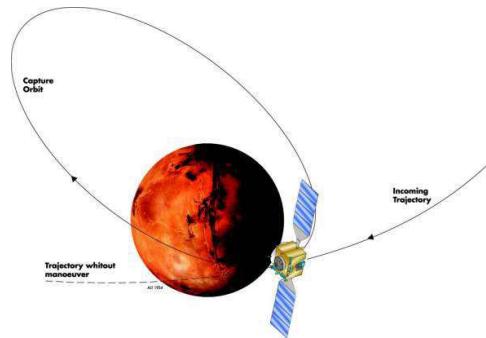
Jocurile și căutarea – spațiul de căutare



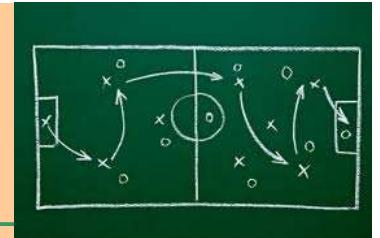
Strategii de joc → Planning

□ De ce?

- jocuri cu NPC (*First-person shooter, Role-playing, Real-time strategy*)
- Probleme reale de planificare
 - Telescop în spațiu
 - Roboței
 - UAV-uri



Jocurile și căutarea – spațiul de căutare



Strategii de joc → Planning

□ Cum?

- Algoritmi
 - Pentru alegerea unei acțiuni (shooting)
 - Pentru evaluarea mediului
- Abordări
 - STRIPS → <http://www.dis.uniroma1.it/~degiacom/didattica/dottorato-stavros-vassos/>
 - HTN
- Simularea comportamentului
 - Mașini cu stări finite (FSM)
 - Arbori de comportament (Halo 2)
 - GOAP (FEAR)

Recapitulare



- Definirea unui joc
 - Starea inițială (cum sunt așezate inițial elementele în joc)
 - Acțiunile posibile (care sunt mutările permise)
 - Test terminal (care indică terminarea jocului)
 - Funcție utilitate (care spune cine și cât a câștigat)
- Strategii de rezolvare a jocurilor
 - Bazate pe explorarea (aproape) completă a arborelui de joc
 - AndOR → cine câștigă
 - MiniMax → cine și cât câștigă
 - MiniMax cu retezare α - β → cine și cât câștigă și ce mutări nu merită să fie efectuate
 - Bazate pe strategii inteligente
 - Strategia simetriei
 - Strategia perechilor
 - Strategia parității
 - Programare dinamică
 - A* (Pathfinding, Planning)

Cursul următor

- A. Scurtă introducere în Inteligența Artificială (IA)**

- B. Rezolvarea problemelor prin căutare**
 - Definirea problemelor de căutare
 - Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

- C. Sisteme inteligente**
 - Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
 - Sisteme bazate pe reguli
 - Sisteme hibride

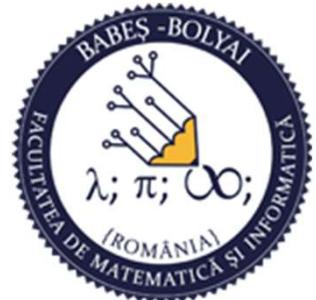
Cursul următor – Materiale de citit și legături utile

- capitolul III din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 4 și 5 din *H.F. Pop, G. Șerban, Inteligență artificială, Cluj Napoca, 2004*
- capitolul 2 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 6 și 7 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

-
- Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean –
www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan –
www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop –
www.cs.ubbcluj.ro/~hfpop



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ



Sisteme inteligente

Sisteme care învață singure

Laura Dioșan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Materiale de citit și legături utile

- capitolul VI (18) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 10 și 11 din *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- capitolul 3 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997* [link](#)
- capitolul 1 din *C. Bishop, Pattern Recognition and Machine Learning, Springer, 2006* [link](#)
- capitolul 1 din *S. Guido, A. C. Müller, Introduction to Machine Learning with Python, O'Reilly Media, 2016* [link](#)
- capitolele 1 și 2 din *A. Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly Media, 2019* [link](#)

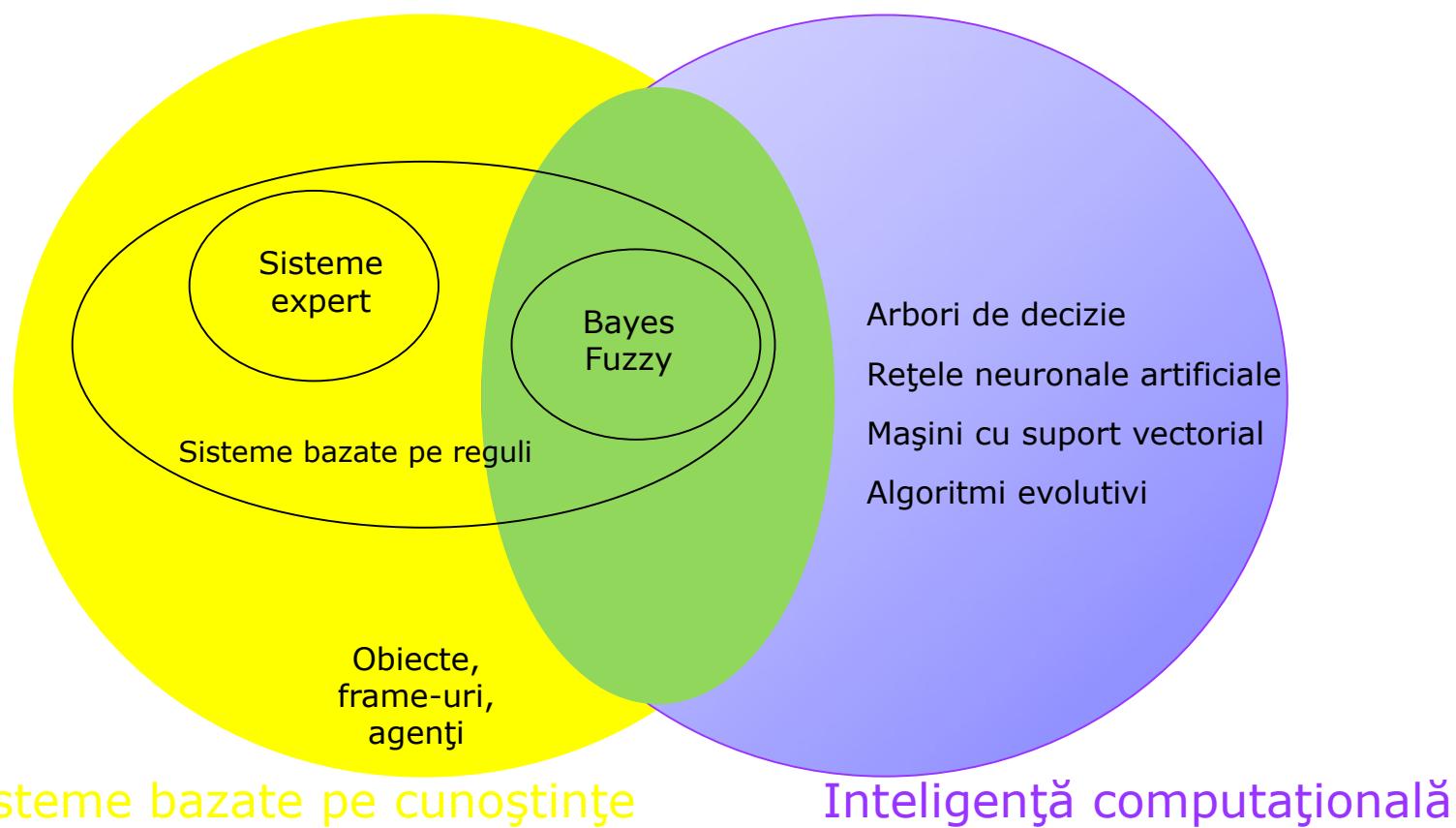
Conținut

□ Sisteme inteligente

■ Sisteme care învață singure (SIS)

- Instruire (învățare) automata (Machine Learning - ML)
 - Problematică
 - Proiectarea unui sistem de învățare automată
 - Tipologie
 - Învățare supervizată
 - Învățare nesupervizată
 - Învățare cu întărire
 - Teoria învățării
- Exemple de sisteme

Sisteme inteligente



Sisteme inteligente – SIS – Învățare automată

□ Problematica

- “How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?”

□ Aplicații

- Recunoaștere de imagini și semnal vocal
 - Recunoașterea scrisului de mâna
 - Detectia fețelor
 - Înțelegerea limbajului vorbit
- Computer vision
 - Detectia obstacolelor
 - Recunoașterea amprentelor
- Supraveghere bio
- Controlul roboților
- Predicția vremii
- Diagnosticare medicală
- Detectia fraudelor

Sisteme inteligente – SIS – Învățare automată

□ Definire

- Arthur Samuel (1959)
 - "field of study that gives computers the ability to learn without being explicitly programmed"
 - Înzestrarea computerelor cu abilitatea de a învăța pe baza experienței
- Herbert Simon (1970)
 - "Learning is any process by which a system improves performance from experience."
- Tom Mitchell (1998)
 - "a well-posed learning problem is defined as follows: He says that a computer program is set to learn from an experience E with respect to some task T and some performance measure P if its performance on T as measured by P improves with experience E"
- Ethem Alpaydin (2010)
 - Programming computers to optimize a performance criterion using example data or past experience.
- John L. Hennessy, President of Stanford (2000–2016)
 - Machine learning is the hot new thing
- Bill Gates (Microsoft co-founder)
 - A breakthrough in machine learning would be worth ten Microsofts

□ Necesitate

- Sisteme computaționale mai bune
 - Sisteme dificil sau prea costisitor de construit manual
 - Sisteme care se adaptează automat
 - Filtre de spam
 - Sisteme care descoperă informații în baze de date mari → data mining
 - Analize financiare
 - Analize de text/imagini
- Înțelegerea organismelor biologice



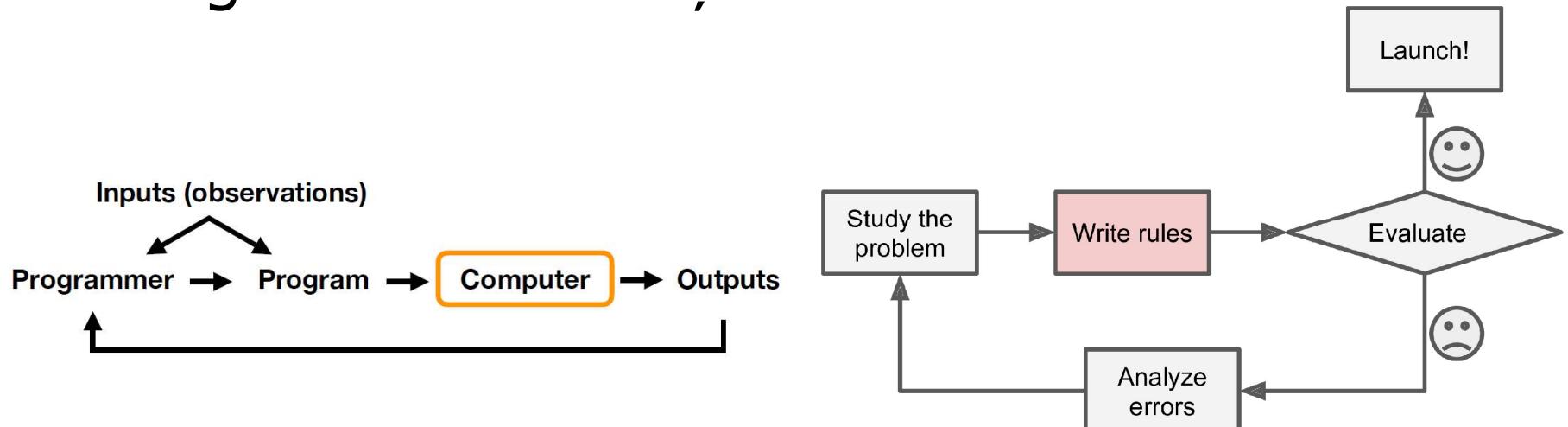
Sisteme inteligente – SIS – Învățare automată

❑ Persoane importante și/sau interesante

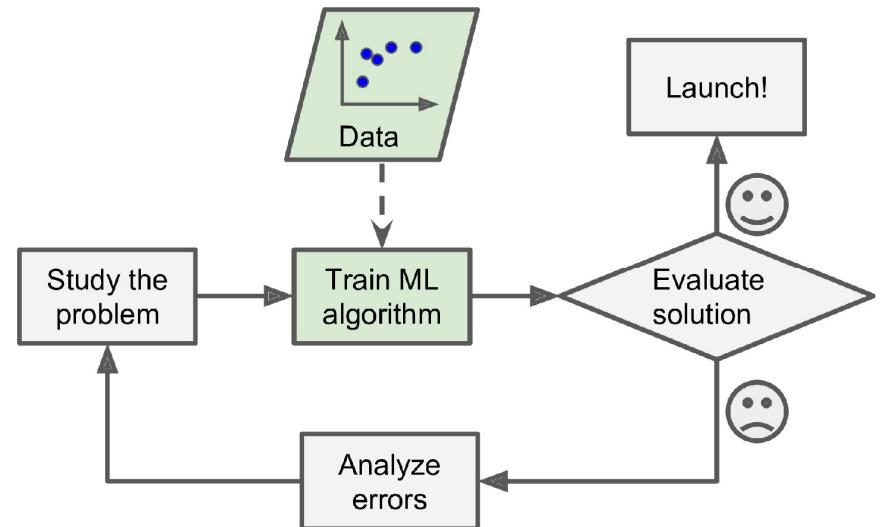
- Peter Norvig
- Stuart Russell
- Michael Jordan (Bayesian Nets), Andrew Ng
- Elon Musk, Andrej Karpathy (TESLA)
- Fei Fei Li (AI for social good)
- Richard Sutton (Reinforcement Learning)
- Jurgen Schmidhuber (LSTM)
- Geoffrey Hinton , Yann LeCun, and Yoshua Bengio (CNN and deep CNN)
- John Koza (Genetic Programming)
- Rana el Kaliouby (Affectiva)
- ...alții...

Sisteme inteligente – SIS – Învățare automată

❑ Programarea tradițională



❑ Machine Learning



Sisteme inteligente – SIS – Învățare automată

□ Proiectare

- Îmbunătățirea task-ului T
 - Stabilirea scopului (ceea ce trebuie învățat) - funcției obiectiv – și reprezentarea sa
 - Alegerea unui algoritm de învățare care să realizeze inferență (previziunea) scopului pe baza experienței
- respectând o metrică de performanță P
 - Evaluarea performanțelor algoritmului ales
- bazându-se pe experiența E
 - Alegerea bazei de experiență
- Exemplu
 - T: jucarea jocului de dame
 - P: procentul de jocuri câștigate împotriva unui oponent oarecare
 - E: exersarea jocului împotriva lui însuși
 - T: recunoașterea scrisului de mâнă
 - P: procentul de cuvinte recunoscute corect
 - E: baze de date cu imagini cu cuvinte corect adnotate
 - T: separarea spam-urilor de mesajele obișnuite
 - P: procentul de email-uri corect clasificate (spam sau normal)
 - E: baze de date cu email-uri adnotate

Sisteme inteligente – SIS – Învățare automată

❑ Stabilirea scopului (ceea ce trebuie învățat)

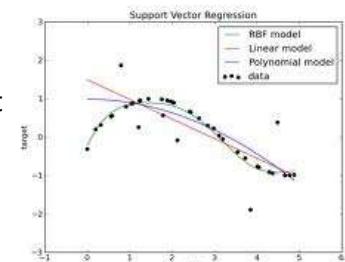
■ SI pentru predicții / regresii

- ❑ Scop: predicția ieșirii pentru o intrare nouă folosind un model învățat anterior
- ❑ Ex.: predicția vânzărilor dintr-un produs pentru un moment de timp viitor în funcție de preț, lună calendaristică, regiune, venit mediu pe economie



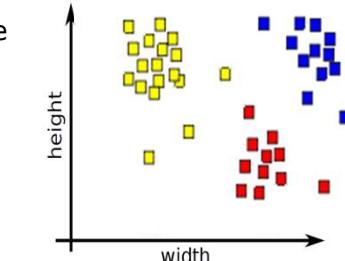
■ SI pentru regresii simbolice

- ❑ Scop: estimarea formei unei funcții uni sau multivariată folosind un model învățat anterior
- ❑ Ex.: estimarea funcției care modelează conturul unei suprafete



■ SI pentru clasificare

- ❑ Scop: clasificarea unui obiect într-una sau mai multe categorii (clase) – cunoscute anterior sau nu - pe baza caracteristicilor (atributelor, proprietăților) lui
- ❑ Ex.: sistem de diagnoză pentru un pacient cu tumoare: nevasculară, vasculară, angiogenă



■ SI pentru planificare

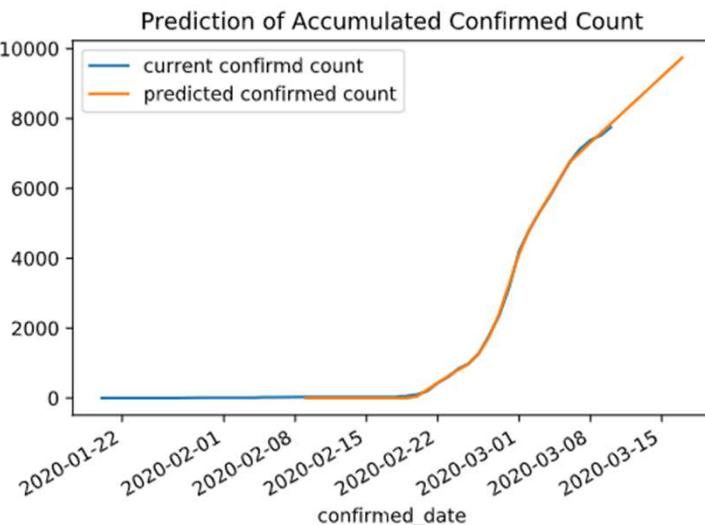
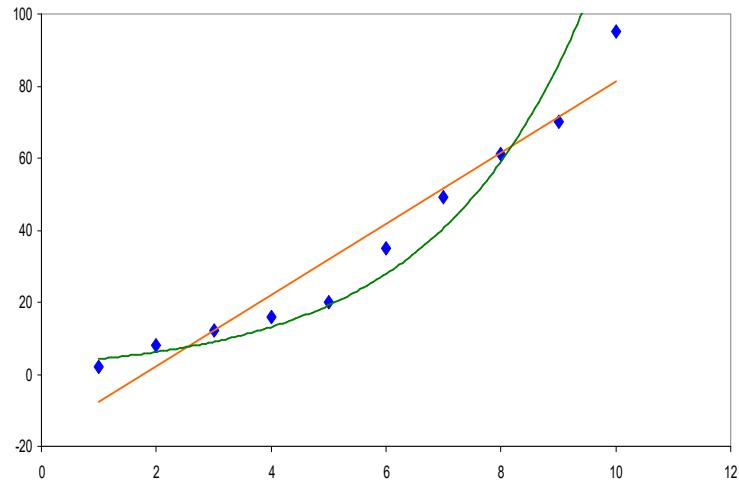
- ❑ Scop: generarea unei succesiuni optime de acțiuni pentru efectuarea unei sarcini
- ❑ Ex.: planificarea deplasării unui robot de la o poziție dată până la o sursă de energie (pentru alimentare)



Sisteme inteligente – SIS – Învățare automată

□ Stabilirea scopului (ceea ce trebuie învățat)

- Probleme de predicție / regresie
 - Se dau date (de intrare și ieșire) trecute
 - Numărul de persoane infectate cu SARS-CoV-2 pentru ultimele 3 luni
 - Se cer predicții viitoare (pentru anumite date de intrare)
 - Numărul de persoane care se vor infecta cu SARS-CoV-2 în urmatoarele 7 zile / 4 săptamani / 2 luni

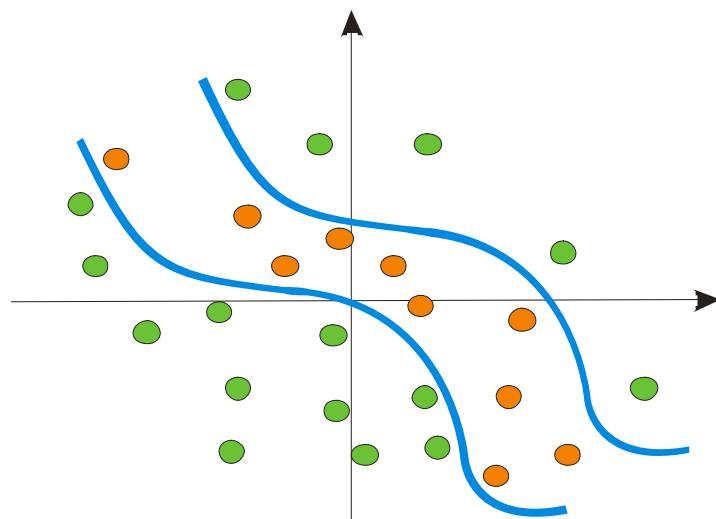


Sisteme inteligente – SIS – Învățare automată

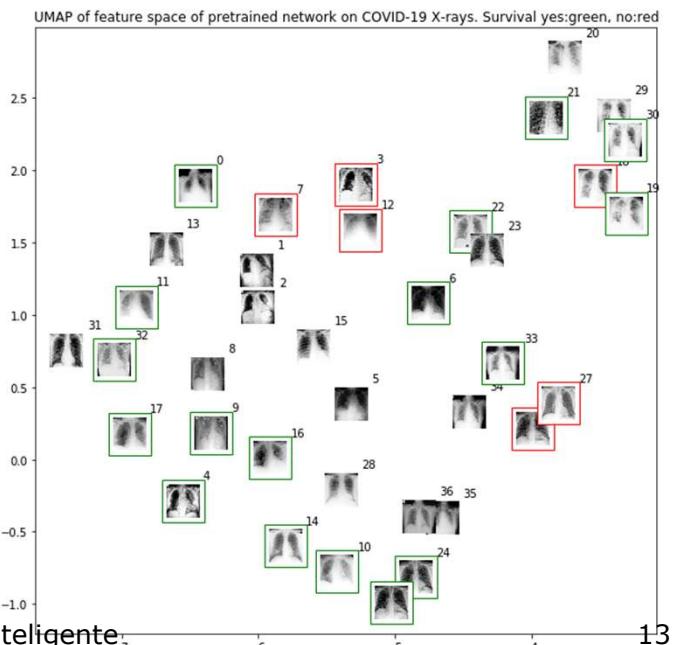
□ Stabilirea scopului (ceea ce trebuie învățat)

■ Probleme de clasificare

- Se dau date (de intrare și ieșire) trecute
 - Imagini RMN de la pacienți infectați cu SARS-CoV-2 și de la martori (sănătoși)
- Se cer predicții viitoare (pentru anumite date de intrare)
 - Să se prezică, pe baza RMN-ului, dacă o persoană este infectată sau nu cu SARS-CoV-2



Inteligență artificială - sisteme inteligente



13

Sisteme inteligente – SIS – Învățare automată

□ Stabilirea scopului (ceea ce trebuie învățat)

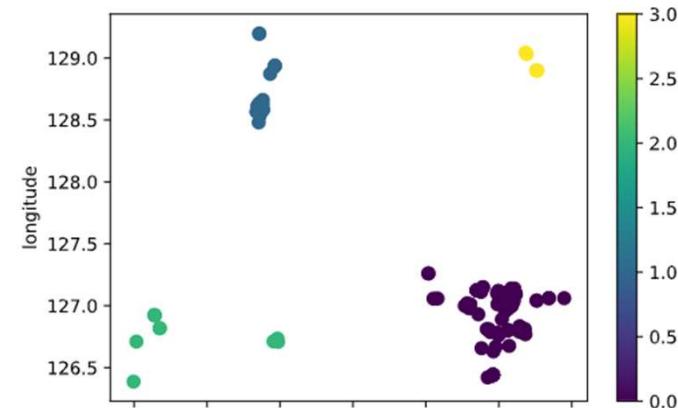
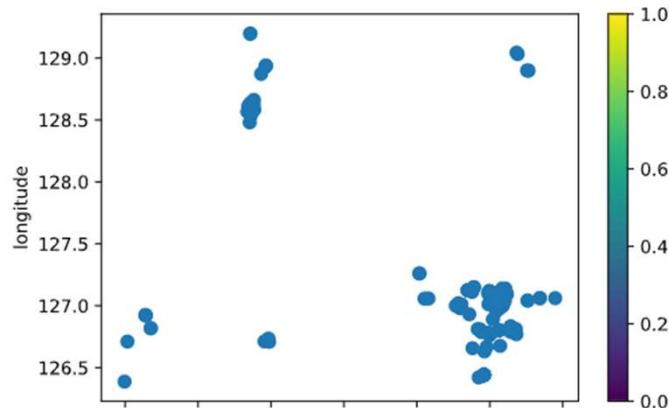
■ Probleme de clusterizare

□ Se dau date (de intrare)

- Localizarea geografică a unor persoane infectate cu SARS-CoV-2

□ Se cere identificarea anumitor structuri în aceste date

- Modul de grupare a celor infectați pe regiuni (Densitatea acestor regiuni)



Sisteme inteligente – SIS – Învățare automată

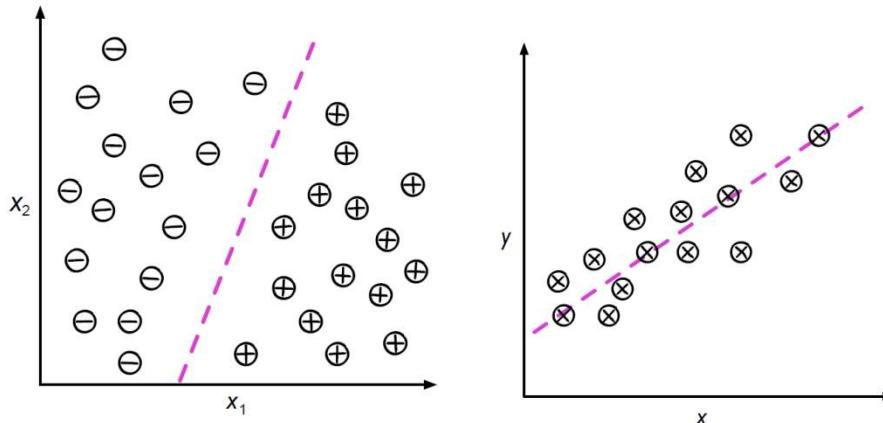
- Proiectare → Alegerea funcției obiectiv
 - Care este funcția care trebuie învățată?
 - Ex.: pentru jocul de dame → funcție care:
 - alege următoarea mutare
 - evaluează o mutare
 - obiectivul fiind alegerea celei mai bune mutări
 - Reprezentarea funcției obiectiv
 - Diferite reprezentări
 - Tablou (tabel)
 - Reguli simbolice
 - Funcție numerică
 - Funcții probabilistice
 - Ex. Jocul de dame
 - Combinăție liniară a nr. de piese albe, nr. de piese negre, nr. de piese albe compromise la următoarea mutare, r. de piese albe compromise la următoarea mutare
 - Există un compromis între
 - expresivitatea reprezentării și
 - ușurința învățării
 - Calculul funcției obiectiv
 - Timp polinomial
 - Timp non-polinomial

Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea unui algoritm de învățare
 - Algoritmul
 - folosind datele de antrenament
 - induce definirea unor ipoteze care
 - să se potrivească cu acestea și
 - să generalizeze cât mai bine datele ne-văzute (datele de test)
 - Principiul de lucru de bază
 - Minimizarea unei erori (funcție de cost – loss function)
 - Tipuri de algoritmi după metodologia de învățare automată
 - Învățare supervizată
 - Ex. regresie, clasificare
 - Învățare nesupervizată
 - Ex. clusterizare, reducerea numărului de dimensiuni
 - Învățare prin întărire
 - Ex. planning, gaming

Sisteme inteligente – SIS – Învățare automată

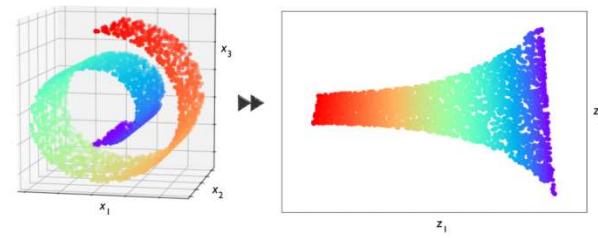
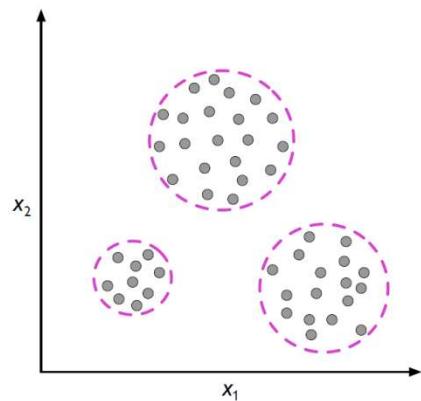
- Proiectare → Alegerea unui algoritm de învățare → Tipuri de algoritmi după metodologia de învățare automată
 - Învățare supervizată
 - Ex. regresie, clasificare
 - Caracteristici
 - Date etichetate (se cunosc o parte din datele de intrare și ieșire *)
 - Feedback direct în timpul învățării – algoritmul se adaptează la datele de intrare și ieșire
 - Predicție a datelor de ieșire (fiind cunoscute niște date de intrare diferite de cele din *)



)

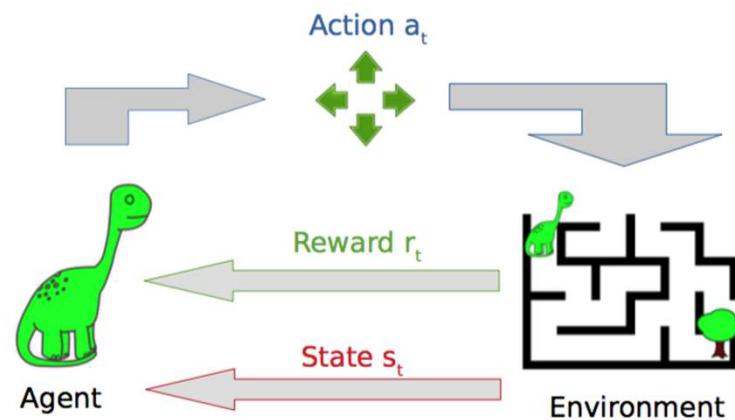
Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea unui algoritm de învățare → Tipuri de algoritmi după metodologia de învățare automată
 - Învățare supervizată
 - Învățare nesupervizată
 - Ex. clusterizare, reducerea numărului de dimensiuni
 - Caracteristici
 - Date neetichetate (se cunosc o parte din datele de intrare**)
 - Fără feedback direct în timpul învățării – pentru că nu se cunosc datele de ieșire
 - Identificarea unor structuri în date (generarea de date de ieșire pentru datele de intrare din **)



Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea unui algoritm de învățare → Tipuri de algoritmi după metodologia de învățare automată
 - Învățare supervizată
 - Învățare nesupervizată
 - Învățare prin întărire
 - Ex.
 - Caracteristici
 - Predicția unor secvențe de decizii / de acțiuni
 - Sistem de recompense (pentru fiecare decizie / acțiune)
 - Se învață un model de acțiune (o serie de acțiuni care trebuie efectuate)



Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea unui algoritm de învățare
 - Algoritmul
 - folosind datele de antrenament
 - induce definirea unor ipoteze care
 - să se potrivească cu acestea și
 - să generalizeze cât mai bine datele ne-văzute (datele de test)
 - Principiul de lucru de bază
 - Minimizarea unei erori (funcție de cost – loss function) pentru datele de antrenament
 - Eroarea de predicție (cât de departe sunt valorile prezise față de valorile reale)
 - Eroarea de clasificare (câte exemple au fost clasificate corect)
 - Eroarea creării unor structuri (cât de ne-omogene sunt structurile produse)
- Proiectare → Evaluarea unui sistem de învățare
 - Experimental
 - Compararea diferitelor metode pe diferite date (cross-validation)
 - Colectarea datelor pe baza performanței
 - Acuratețe, timp antrenare, timp testare
 - Aprecierea diferențelor dpdv statistic
 - Teoretic
 - Analiza matematică a algoritmilor și demonstrarea de teoreme
 - Complexitatea computațională
 - Abilitatea de a se potrivi cu datele de antrenament
 - Complexitatea eșantionului relevant pentru o învățare corectă

Sisteme inteligente – SIS – Învățare automată

- Proiectare → Evaluarea unui sistem de învățare
 - Compararea performanțelor a 2 algoritmi în rezolvarea unei probleme
 - Indicatori de performanță
 - Parametrii ai unei serii statistice (ex. media)
 - Proporție calculată pentru serie statistică (ex. acuratețea)
 - Comparare pe baza intervalelor de încredere
 - Pe o problemă și 2 algoritmi care o rezolvă
 - Performanțele algoritmilor: p_1 și p_2
 - Intervalele de încredere corespunzătoare celor 2 performanțe $I_1 = [p_1 - \Delta_1, p_1 + \Delta_1]$ și $I_2 = [p_2 - \Delta_2, p_2 + \Delta_2]$
 - Dacă $I_1 \cap I_2 = \emptyset \rightarrow$ algoritmul 1 este mai bun decât algoritmul 2 (pt problema dată)
 - Dacă $I_1 \cap I_2 \neq \emptyset \rightarrow$ nu se poate spune care algoritm este mai bun
 - Interval de încredere pentru medie
 - Pentru o serie statistică de volum n , cu media (calculată) m și dispersia σ să se determine intervalul de încredere al valorii medii μ
 - $P(-z \leq (m-\mu)/(\sigma/\sqrt{n}) \leq z) = 1 - \alpha \rightarrow \mu \in [m - z\sigma/\sqrt{n}, m + z\sigma/\sqrt{n}]$
 - $P = 95\% \rightarrow z = 1.96$
 - Ex. Problema rucsacului rezolvată cu ajutorul algoritmilor evolutivi
 - Interval de încredere pentru acuratețe
 - Pentru o performanță p (acuratețe) calculată pentru n date să se determine intervalul de încredere
 - $P \in [p - z(p(1-p)/n)^{1/2}, p + z(p(1-p)/n)^{1/2}]$
 - $P = 95\% \rightarrow z = 1.96$
 - Ex. Problema de clasificare rezolvată cu ajutorul Mașinilor cu suport vectorial

$P=1-\alpha$	z
99.9%	3.3
99.0%	2.577
98.5%	2.43
97.5%	2.243
95.0%	1.96
90.0%	1.645
85.0%	1.439
75.0%	1.151

Sisteme inteligente – SIS – Învățare automată

□ Proiectare → Alegerea bazei de experiență

■ Bazată pe

□ Experiență directă

- Perechi (intrare, ieșire) utile pt. funcția obiectiv
- Ex. Jocul de dame → table de joc etichetată cu mutare corectă sau incorectă

□ Experiență indirectă

- Feedback util (diferit de perechile I/O) pt funcția obiectiv
- Ex. Jocul de dame → secvențe de mutări și scorul final asociat jocului

■ Surse de date

□ Exemple generate aleator

- Exemple pozitive și negative

□ Exemple pozitive colectate de un "învățător" benevol

□ Exemple reale

■ Compoziție

□ Date de antrenament

□ Date de test

■ Caracteristici

□ Date independente

- Dacă nu → clasificare colectivă

□ Datele de antrenament și de test trebuie să urmeze aceeași lege de distribuție

- Dacă nu → învățare prin transfer (*transfer learning/inductive transfer*)
 - recunoașterea mașinilor → recunoașterea camioanelor
 - analiza textelor
 - filtre de spam

Sisteme inteligente – SIS – Învățare automată

□ Proiectare → Alegerea bazei de experiență

■ Tipuri de atrbute ale datelor

□ Cantitative → scară nominală sau rațională

- Valori continue → greutatea
- Valori discrete → numărul de computere
- Valori de tip interval → durata unor evenimente

□ Calitative

- Nominale → culoarea
- Ordinale → intensitatea sunetului (joasă, medie, înaltă)

□ Structurate

- Arboi – rădăcina e o generalizare a copiilor (vehicol → mașină, autobus, tractor, camion)

■ Transformări asupra datelor

□ Standardizare → atrbute numerice

- Înlăturarea efectelor de scară (scări și unități de măsură diferite)
- Valorile brute se transformă în scoruri z
 - $Z_{ij} = (x_{ij} - \mu_j)/\sigma_j$, unde x_{ij} – valoarea atrbutului al j -lea al instanței i , μ_j (σ_j) este media (abaterea) atrbutelor j pt. toate instanțele

□ Selectarea anumitor atrbute

Sisteme inteligente – SIS – Învățare automată

□ Învățare supervizată

- Definire
- Exemple
- Proces
- Calitatea învățării
 - Metode de evaluare
 - Măsuri de performanță
- Tipologie

Sisteme inteligente – SIS – Învățare automată

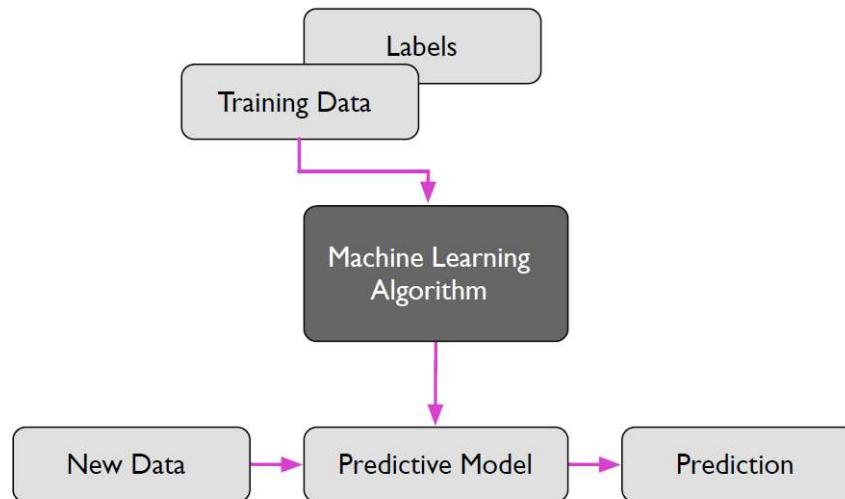
Învățare supervizată

- Scop
 - Furnizarea unei ieșiri corecte pentru o nouă intrare
- Definire
 - Se dă un set de date (exemple, instanțe, cazuri)
 - date de antrenament – sub forma unor perechi (atribute_data_i, ieșire_i), unde
 - i = 1, N (N = nr datelor de antrenament)
 - atribut_data_i = (atr_{i1}, atr_{i2}, ..., atr_{im}), m – nr atributelor (caracteristicilor, proprietăților) unei date
 - ieșire_i
 - o categorie dintr-o mulțime dată (predefinită) cu k elemente (k – nr de clase) → problemă de clasificare
 - un număr real → problemă de regresie
 - date de test - sub forma (atribute_data_i), i = 1, n (n = nr datelor de test).
 - Să se determine
 - o funcție (necunoscută) care realizează corespondența atribut – ieșire pe datele de antrenament
 - ieșirea (clasa/valoarea) asociată unei date (noi) de test folosind funcția învățată pe datele de antrenament
 - Alte denumiri
 - Clasificare (regresie), învățare inductivă

Sisteme inteligente – SIS – Învățare automată

Învățare supervizată

- ❑ Proces → 2 etape
 - Antrenarea
 - ❑ Învățarea, cu ajutorul unui algoritm, a modelului de predicție
 - Testarea
 - ❑ Testarea modelului folosind date de test noi (*unseen data*)



- ❑ Caracteristic
 - BD experimentală adnotată (pt. învățare)

Sisteme inteligente – SIS – Învățare automată

Învățare supervizată

□ Tip de probleme

- regresie
 - Scop: predicția output-ului pentru un input nou
 - Output continuu (nr real)
 - Ex.: predicția ratei șomajului în funcție de produsul intern brut și rata inflației
- clasificare
 - Scop: clasificarea (etichetarea) unui nou input
 - Output discret (etichetă dintr-o mulțime predefinită)
 - Ex.: detectarea tumorilor maligne în imagini RMN

□ Exemple de probleme

- Recunoașterea scrisului de mână
- Recunoașterea pietonilor în imagini
- Previziunea vremii
- Detecția spam-urilor

Sisteme inteligente – SIS – Învățare automată

■ Învățare supervizată

- Terminologie – e.g. Problema predicției consumului de înghețată pe baza temperaturii de afară și a sumei de bani avută la dispoziție
 - Exemplu (example, observation, instance, record)
 - o observație a datelor care trebuie procesate
 - dacă datele de intrare sunt tabelare, un exemplu este asociat cu o linie din tabel
 - format din proprietăți a datelor care trebuie procesate (de intrare și de ieșire)
 - Caracteristică (feature, property, attribute)
 - Proprietate cunoscută a unui exemplu, folosită drept dată de intrare pentru algoritmul de ML (variabilele independente din modelul de predicție)
 - dacă datele de intrare sunt tabelare, un proprietate are asociate valorile dintr-o coloana a tabelului (pentru toate exemplele)
 - E.g. Temperatura, banii
 - Valoare țintă (target or real value/label, ground-truth)
 - Proprietate a unui exemplu folosită ca variabilă dependentă
 - Cunoscută pentru exemplele de antrenament
 - Ne-cunoscută pentru exemplele de testare
 - E.g. Nr de inghetate
 - Valoare calculată (computed value/label)
 - Proprietate a unui exemplu estimată cu ajutorul algoritmului de ML
 - Se dorește a fi cât mai aproape de valoarea target

Exemplu	Temperatura	Banii	Nr de inghetate
Ex1	30	25	2
Ex2	5	100	0
Ex3	19	55	2
Ex4	35	75	4

Sisteme inteligente – SIS – Învățare automată

Învățare supervizată

□ Calitatea învățării

■ Definire

- o măsură de performanță a algoritmului de ML
 - ex. acuratețea ($Acc = \text{nr de exemple corect clasificate} / \text{nr total de exemple}$)
- Posibile măsuri:
 - Măsuri statistice
 - Eroarea de predicție
 - acuratețea
 - Precizia
 - Rapelul
 - Scorul F1

Sisteme inteligente – SIS – Învățare automată

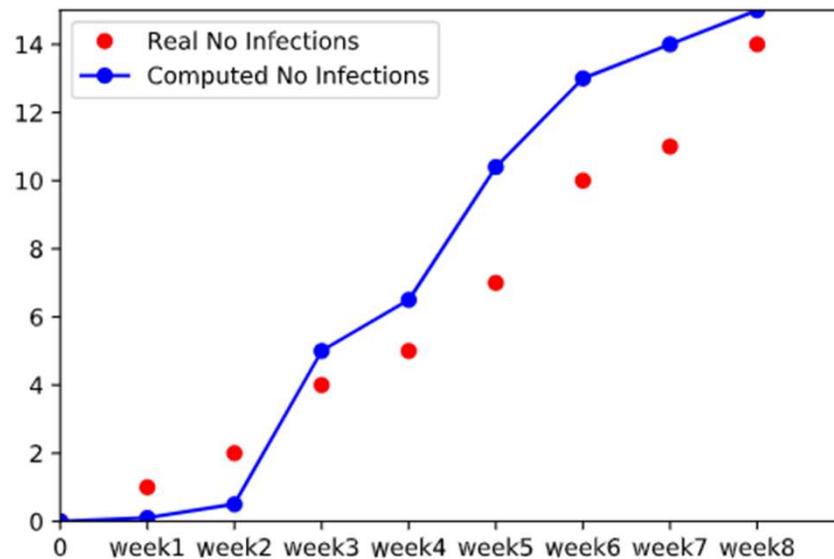
Învățare supervizată

- Calitatea învățării → Măsuri de performanță → Măsuri statistice
 - Eroarea de predicție
 - Suma diferențelor absolute între valorile reale și cele calculate

$$Err = \frac{1}{noSamples} \sum_{i=1}^{noSamples} abs(real_i - computed_i)$$

- Suma pătratelor diferențelor între valorile reale și cele calculate

$$Err = \sqrt{\frac{1}{noSamples} \sum_{i=1}^{noSamples} (real_i - computed_i)^2}$$



Sisteme inteligente – SIS – Învățare automată

Învățare supervizată

- Calitatea învățării → Măsuri de performanță → Măsuri statistice
 - Acuratețea
 - Nr de exemple corect clasificate / nr total de exemple
 - Opusul erorii
 - Calculată pe
 - Setul de validare
 - Setul de test
 - Uneori
 - Analiză de text
 - Detectarea intrușilor într-o rețea
 - Analize financiare
 - Este importantă doar o singură clasă (clasă pozitivă) → restul claselor sunt negative
 - Precizia (P)
 - nr. de exemple pozitive corect clasificate / nr. total de exemple clasificate ca pozitive
 - probabilitatea ca un exemplu clasificat pozitiv să fie relevant
 - $TP / (TP + FP)$
 - Rapelul (R)
 - nr. de exemple pozitive corect clasificate / nr. total de exemple pozitive
 - Probabilitatea ca un exemplu pozitiv să fie identificat corect de către clasificator
 - $TP / (TP + FN)$
 - Matrice de confuzie → rezultate reale vs. rezultate calculat
 - Scorul F1
 - Combină precizia și rapelul, facilitând compararea a 2 algoritmi
 - Media armonică a preciziei și rapelului
 - $2PR/(P+R)$

		Rezultate reale	
		Clasa pozitivă	Clasa(ele) negativă(e)
Rezultate calculate	Clasa pozitivă	<i>True positiv (TP)</i>	<i>False positiv (FP)</i>
	Clasa(ele) negativă(e)	<i>False negative (FN)</i>	<i>True negative (TN)</i>

Sisteme inteligente – SIS – Învățare automată

Învățare supervizată

- Calitatea învățării → Măsuri de performanță → Măsuri statistice

- Acuratețea
 - Nr de exemple corect clasificate / nr total de exemple
- Precizia (P)
 - nr. de exemple pozitive corect clasificate / nr. total de exemple clasificate ca pozitive
 - probabilitatea ca un exemplu clasificat pozitiv să fie relevant
 - $TP / (TP + FP)$
- Rapelul (R)
 - nr. de exemple pozitive corect clasificate / nr. total de exemple pozitive
 - Probabilitatea ca un exemplu pozitiv să fie clasificat pozitiv
 - $TP / (TP + FN)$
 - Matrice de confuzie → rezultate reale vs rezultate calculate
- Scorul F1
 - Combină precizia și rapelul, facilită compararea a 2 algoritmi
 - Media armonică a preciziei și rapelului
 - $2PR / (P+R)$

		Rezultate reale	
		Clasa pozitivă	Clasa(ele) negativă(e)
Rezultate calculate	Clasa pozitivă	<i>True positiv (TP)</i>	<i>False positiv (FP)</i>
	Clasa(ele) negativă(e)	<i>False negative (FN)</i>	<i>True negative (TN)</i>



Sisteme inteligente – SIS – Învățare automată

Învățare supervizată

□ Calitatea învățării

■ Definire

- o măsură de performanță a algoritmului
 - ex. acuratețea ($Acc = \text{nr de exemple corect clasificate} / \text{nr total de exemple}$)
- Posibile măsuri:
 - Măsuri statistice
 - acuratețea
 - Precizia
 - Rapelul
 - Scorul F1
 - Eficiența
 - În construirea modelului
 - În testarea modelului
 - Robustetea
 - Tratarea zgomotelor și a valorilor lipsă
 - Scalabilitatea
 - Eficiența gestionării seturilor mari de date
 - Interpretabilitatea
 - Modelului de clasificare
 - Proprietatea modelului de a fi compact
 - Scoruri

Sisteme inteligente – SIS – Învățare automată

Învățare supervizată

□ Calitatea învățării

■ Definire

- o măsură de performanță a algoritmului
 - ex. acuratețea ($Acc = \text{nr de exemple corect clasificate} / \text{nr total de exemple}$)
- calculată în
 - fază de antrenare
 - fază de testare

■ Metode de evaluare

- Seturi disjuncte de antrenare și testare
 - setul de antrenare poate fi împărțit în date de învățare și date de validare
 - setul de antrenare este folosit pentru estimarea parametrilor modelului (cei mai buni parametri obținuți pe validare vor fi folosiți pentru construcția modelului final)
 - pentru date numeroase
- Validare încrucișată cu mai multe (h) sub-seturi egale ale datelor (de antrenament)
 - separarea datelor de h ori în ($h-1$ sub-seturi pentru învățare și 1 sub-set pt validare)
 - dimensiunea unui sub-set = dimensiunea setului / h
 - performanța este dată de media pe cele h rulări (ex. $h = 5$ sau $h = 10$)
 - pentru date puține
- Leave-one-out cross-validation
 - similar validării încrucișate, dar $h = \text{nr de date} \rightarrow$ un sub-set conține un singur exemplu
 - pentru date foarte puține
- Dificultăți
 - Învățare pe derost (overfitting) \rightarrow performanță bună pe datele de antrenament, dar foarte slabă pe datele de test

Sisteme inteligente – SIS – Învățare automată

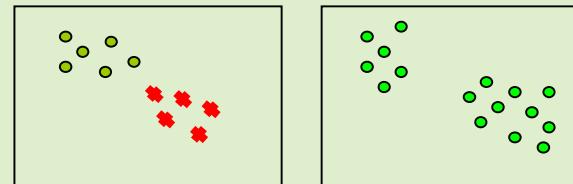
□ Învățare ne-supervizată

- Definire
- Exemple
- Proces
- Metode de evaluare și măsuri de performanță
- Tipologie

Sisteme inteligente – SIS – Învățare automată

Învățare ne-supervizată

- Scop
 - Găsirea unui model sau a unei structuri utile a datelor
 - Împărțirea unor exemple **neetichetate** în submulțimi disjuncte (clusteri) astfel încât:
 - exemplele din același cluster sunt foarte similare
 - exemplele din clusteri diferiți sunt foarte diferite
- Definire
 - Se dă un set de date (exemple, instanțe, cazuri)
 - Date de antrenament sub forma **atribute_data_i**, unde
 - $i = 1, N$ (N = nr datelor de antrenament)
 - **atribute_data_i** = $(atr_{i1}, atr_{i2}, \dots, atr_{im})$, m – nr atributelor (caracteristicilor, proprietăților) unei date
 - Date de test sub forma (**atribute_data_i**), $i = 1, n$ (n = nr datelor de test)
 - Se determină
 - o funcție (necunoscută) care realizează gruparea datelor de antrenament în mai multe clase
 - Nr de clase poate fi pre-definit (k) sau necunoscut
 - Datele dintr-o clasă sunt asemănătoare
 - clasa asociată unei date (noi) de test folosind gruparea învățată pe datele de antrenament
 - Învățare supervizată vs. Învățare ne-supervizată
 - Distanțe între 2 elemente p și $q \in R^m$
 - Euclideană $\rightarrow d(p,q) = \sqrt{\sum_{j=1,2,\dots,m} (p_j - q_j)^2}$
 - Manhattan $\rightarrow d(p,q) = \sum_{j=1,2,\dots,m} |p_j - q_j|$
 - Mahalanobis $\rightarrow d(p,q) = \sqrt{p^T S^{-1} p}$, unde S este matricea de variație și covariație ($S = E[(p - E[p])(q - E[q])^T]$)
 - Produsul intern $\rightarrow d(p,q) = \sum_{j=1,2,\dots,m} p_j q_j$
 - Cosine $\rightarrow d(p,q) = \sum_{j=1,2,\dots,m} p_j q_j / (\sqrt{\sum_{j=1,2,\dots,m} p_j^2} * \sqrt{\sum_{j=1,2,\dots,m} q_j^2})$
 - Hamming \rightarrow numărul de diferențe între p și q
 - Levenshtein \rightarrow numărul minim de operații necesare pentru a-l transforma pe p în q
 - Distanță vs. Similaritate
 - Distanță \rightarrow min
 - Similaritatea \rightarrow max



Sisteme inteligente – SIS – Învățare automată

Învățare ne-supervizată

- ❑ Alte denumiri
 - Clustering
- ❑ Procesul → 2 pași
 - Antrenarea → Învățarea (determinarea), cu ajutorul unui algoritm, a clusterilor existenți
 - Testarea → Plasarea unei noi date într-unul din clusterii identificați în etapa de antrenament
- ❑ Caracteristic
 - Datele nu sunt adnotate (etichetate)
- ❑ Tip de probleme
 - Identificarea unor grupuri (clusteri)
 - ❑ Analiza genelor
 - ❑ Procesarea imaginilor
 - ❑ Analiza rețelelor sociale
 - ❑ Segmentarea pieței
 - ❑ Analiza datelor astronomice
 - ❑ Clusteri de calculatoare
 - Reducerea dimensiunii
 - Identificarea unor cauze (explicații) ale datelor
 - Modelarea densității datelor
- ❑ Exemple de probleme
 - Gruparea genelor
 - Studii de piață pentru gruparea clienților (segmentarea pieței)
 - news.google.com

Sisteme inteligente – SIS – Învățare automată

Învățare ne-supervizată

□ Calitatea învățării (validarea clusterizări):

- Criterii interne → Similaritate ridicată în interiorul unui cluster și similaritate redusă între clusteri
 - Distanța în interiorul clusterului
 - Distanța între clusteri
 - Indexul Davies-Bouldin
 - Indexul Dunn
- Criterii externe → Folosirea unor benchmark-uri formate din date pre-grupate
 - Compararea cu date cunoscute – în practică este imposibil
 - Precizia
 - Rapelul
 - F-measure

Sisteme inteligente – SIS – Învățare automată

Învățare ne-supervizată

□ Calitatea învățării → Criterii interne

- Distanța în interiorul clusterului c_j care conține n_j instanțe
 - Distanța medie între instanțe (average distance) $D_a(c_j) = \sum_{x_{i1}, x_{i2} \in c_j} ||x_{i1} - x_{i2}|| / (n_j(n_j-1))$
 - Distanța între cei mai apropiati vecini $D_{nn}(c_j) = \sum_{x_{i1} \in c_j} \min_{x_{i2} \in c_j} ||x_{i1} - x_{i2}|| / n_j$
 - Distanța între centroizi $D_c(c_j) = \sum_{x_i \in c_j} ||x_i - \mu_j|| / n_j$, unde $\mu_j = 1/n_j \sum_{x_i \in c_j} x_i$
- Distanța între 2 clusteri c_{j1} și c_{j2}
 - Legătură simplă $d_s(c_{j1}, c_{j2}) = \min_{x_{i1} \in c_{j1}, x_{i2} \in c_{j2}} \{ ||x_{i1} - x_{i2}|| \}$
 - Legătură completă $d_{co}(c_{j1}, c_{j2}) = \max_{x_{i1} \in c_{j1}, x_{i2} \in c_{j2}} \{ ||x_{i1} - x_{i2}|| \}$
 - Legătură medie $d_a(c_{j1}, c_{j2}) = \sum_{x_{i1} \in c_{j1}, x_{i2} \in c_{j2}} \{ ||x_{i1} - x_{i2}|| \} / (n_{j1} * n_{j2})$
 - Legătură între centroizi $d_{ce}(c_{j1}, c_{j2}) = ||\mu_{j1} - \mu_{j2}||$
- Indexul Davies-Bouldin → min → clusteri compacti
 - $DB = 1/nc * \sum_{i=1,2,\dots,nc} \max_{j=1,2,\dots,nc, j \neq i} ((\sigma_i + \sigma_j)/d(\mu_i, \mu_j))$, unde:
 - nc – numărul de clusteri
 - μ_i – centroidul clusterului i
 - σ_i – media distanțelor între elementele din clusterul i și centroidul μ_i
 - $d(\mu_i, \mu_j)$ – distanța între centroidul μ_i și centroidul μ_j
- Indexul Dunn
 - Identifică clusterii denși și bine separați
 - $D = d_{min}/d_{max}$, unde:
 - d_{min} – distanța minimă între 2 obiecte din clusteri diferiți – distanța intra-cluster
 - d_{max} – distanța maximă între 2 obiecte din același cluster – distanța inter-cluster

Sisteme inteligente – SIS – Învățare automată

Învățare ne-supervizată

- Tipologie
 - După modul de formare al clusterilor
 - Ierarhic
 - se crează un arbore taxonomic (dendogramă)
 - crearea clusterilor → recursiv
 - nu se cunoaște k (nr de clusteri)
 - aglomerativ (de jos în sus) → clusteri mici spre clusteri mari
 - diviziv (de sus în jos) → clusteri mari spre clusteri mici
 - Ex. Clustering ierarhic aglomrativ
 - Ne-ierarhic
 - Partițional → se determină o împărțire a datelor → toți clusterii deodată
 - Optimizează o funcție obiectiv definită local (doar pe anumite atribute) sau global (pe toate atributele) care poate fi:
 - Pătratul erorii – suma patratelor distanțelor între date și centroizii clusterilor → min (ex. K-means)
 - Bazată pe grafuri (ex. Clusterizare bazată pe arborele minim de acoperire)
 - Pe modele probabilistice (ex. Identificarea distribuției datelor → Maximizarea așteptărilor)
 - Pe cel mai apropiat vecin
 - Necesită fixarea apriori a lui k → fixarea clusterilor inițiali
 - Algoritmii se rulează de mai multe ori cu diferiți parametri și se alege versiunea cea mai eficientă
 - Ex. K-means, ACO
 - bazat pe densitatea datelor
 - Densitatea și conectivitatea datelor
 - Formarea clusterilor de bază pe densitatea datelor într-o anumită regiune
 - Formarea clusterilor de bază pe conectivitatea datelor dintr-o anumită regiune
 - Funcția de densitate a datelor
 - Se încercă modelarea legii de distribuție a datelor
 - Avantaj:
 - Modelarea unor clusteri de orice formă
 - Bazat pe un grid
 - Nu e chiar o metodă nouă de lucru
 - Poate fi ierarhic, partițional sau bazat pe densitate
 - Pp segmentarea spațiului de date în zone regulate
 - Obiectele se plasează pe un grid multi-dimensional
 - Ex. ACO

Sisteme inteligente – SIS – Învățare automată

Învățare ne-supervizată

■ Tipologie

- După modul de lucru al algoritmului
 - Aglomerativ
 1. Fiecare instantă formează inițial un cluster
 2. Se calculează distanțele între oricare 2 clusteri
 3. Se reunesc cei mai apropiati 2 clusteri
 4. Se repetă pașii 2 și 3 până se ajunge la un singur cluster sau la un alt criteriu de stop
 - Diviziv
 1. Se stabilește numărul de clusteri (k)
 2. Se inițializează centrul fiecărui cluster
 3. Se determină o împărțire a datelor
 4. Se recalculează centrul clusterelor
 5. Se repetă pasul 3 și 4 până partitōnarea nu se mai schimbă (algoritmul a convergat)
- După atrbutele considerate
 - Monotetic – atrbutele se consideră pe rând
 - Politetic – atrbutele se consideră simultan
- După tipul de apartenență al datelor la clusteri
 - Clustering exact (*hard clustering*)
 - Asociază fiecarei intrări x_i o etichetă (clasă) c_j
 - Clustering fuzzy
 - Asociază fiecarei intrări x_i un grad (probabilitate) de apartenență f_{ij} la o anumită clasă $c_j \Rightarrow$ o instanță x_i poate apartine mai multor clusteri

Sisteme inteligente – SIS – Învățare automată

□ Tipologie

- În funcție de experiența acumulată în timpul învățării
 - SI cu învățare supervizată
 - SI cu învățare nesupervizată
 - **SI cu învățare activă**
 - SI cu învățare cu întărire
- În funcție de modelul învățat (algoritmul de învățare)
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Algoritmi evolutivi
 - Mașini cu suport vectorial
 - Modele Markov ascunse

Sisteme inteligente – SIS – Învățare automată

□ Învățare activă

- Algoritmul de învățare poate primi informații suplimentare în timpul învățării pentru a-și îmbunătăți performanța
 - ▣ Ex. pe care din datele de antrenament este mai ușor să se învețe modelul de decizie

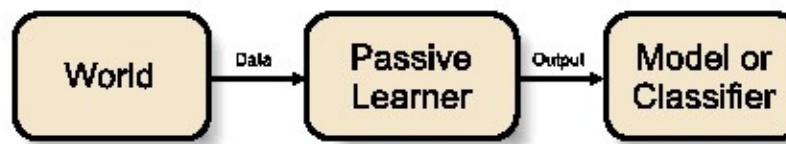


Figure 1.1: General schema for a passive learner.



Figure 1.2: General schema for an active learner.

Sisteme inteligente – SIS – Învățare automată

□ Tipologie

- În funcție de experiența acumulată în timpul învățării
 - SI cu învățare supervizată
 - SI cu învățare nesupervizată
 - SI cu învățare activă
 - SI cu învățare cu întărire
- În funcție de modelul învățat (algoritmul de învățare)
 - **Metoda celor mai mici pătrate**
 - Metoda gradient descent
 - Algoritmi evolutivi
 - Logistic regression
 - kNN
 - Arboi de decizie
 - Mașini cu suport vectorial
 - Rețele neuronale artificiale
 - Programare genetică
 - Modele Markov ascunse



Recapitulare

□ Sisteme care învață singure (SIS)

- Instruire (învățare) automata (Machine Learning - ML)
 - Învățare supervizată → datele de antrenament sunt deja etichetate cu elemente din E, iar datele de test trebuie etichetate cu una dintre etichetele din E pe baza unui model (învățat pe datele de antrenament) care face corespondență date-etichete
 - Învățare nesupervizată → datele de antrenament NU sunt etichetate, trebuie învățat un model de etichetare, iar apoi datele de test trebuie etichetate cu una dintre etichetele identificate de model
- Sisteme

Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Metoda celor mai mici patrate, Gradient Descent, Logistic regression
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

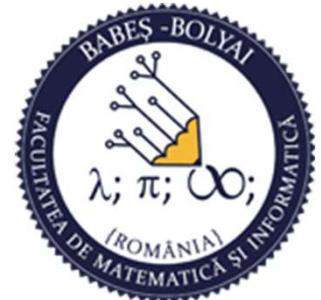
Cursul următor – Materiale de citit și legături utile

- Capitolul VI (19) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 8 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 12 și 13 din *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- Capitolul 4 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

-
- Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean –
www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan -
www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop -
www.cs.ubbcluj.ro/~hfpop



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ



Sisteme inteligente

Sisteme care învață singure

Laura Dioșan & Dragoș Dobrean

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Metoda celor mai mici patrate, Gradient Descent, Logistic regression
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Materiale de citit și legături utile

- capitolul VI (18) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 10 și 11 din *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- capitolul 3 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Conținut

□ Sisteme inteligente

■ Sisteme care învață singure (SIS)

- Instruire (învățare) automata (Machine Learning - ML)
 - Problematică
 - Proiectarea unui sistem de învățare automată
 - Tipologie
 - Învățare supervizată
 - Învățare nesupervizată
 - Învățare cu întărire
 - Teoria învățării
- Exemple de sisteme

Conținut

❑ Sisteme care învață singure (SIS)



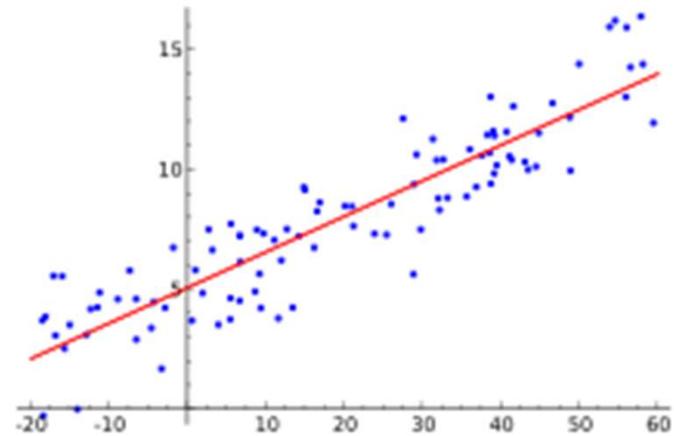
- ❑ "Field of study that gives computers the ability to learn without being explicitly programmed." -- Arthur Samuels (1959)
- Invățare
 - ❑ Supervizată
 - ❑ Nesupervizată
 - ❑ Reinforcement

Sisteme inteligente – SIS – Învățare automată

□ Metoda celor mai mici pătrate

■ Presupunem cazul unei probleme de regresie

- Date de intrare $x \in \mathbb{R}^d$
- Date de ieșire $y \in \mathbb{R}$



- Se cere un model **liniar** f care transformă x în y
- $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_d x_d$
- Invățare supervizată

Sisteme inteligente – SIS – Învățare automată

□ Metoda celor mai mici pătrate

■ Presupunem cazul unei probleme de regresie

- Date de intrare $x^i \in R^d$, $i=1,n$
- Date de ieșire $y^i \in R$
- Se cere un model **liniar** f care transformă orice x^i în y^i , $i=1,n$
- $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_d x_d$

- Se poate defini o funcție de cost
- $\text{Loss} = \sum_{i=1,n} (y^i - f(x^i))^2$ -- minimizată → valorile optime ale lui β

$$x = (1, x) = (1, x_1, x_2, \dots, x_d)^T \in R^{d+1}$$

$$\beta = (\beta_0, \beta_1, \beta_2, \dots, \beta_d)^T \in R^{d+1}$$

$$f(x) = x^T \beta$$

$$\text{Loss } (\beta) = \| y - X \beta \| ^2$$

$$X = 1 \ x_{1,1} \ x_{1,2} \ x_{1,3} \ \dots \ x_{1,d}$$

.

.

.

.

$$1 \ x_{n,1} \ x_{n,2} \ x_{n,3} \ \dots \ x_{n,d}$$

Sisteme inteligente – SIS – Învățare automată

□ Metoda celor mai mici pătrate

■ Presupunem cazul unei probleme de regresie

- Date de intrare $x^i \in R^d$, $i=1,n$
- Date de ieșire $y^i \in R$
- Se cere un model **liniar** f care transformă orice x^i în y^i , $i=1,n$
- $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_d x_d$

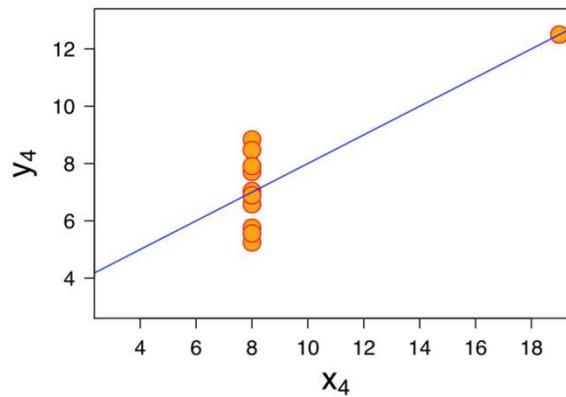
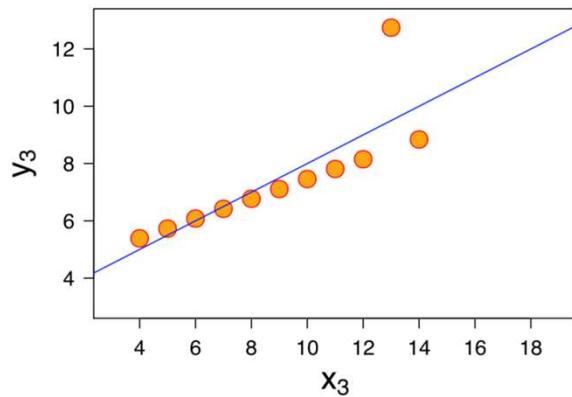
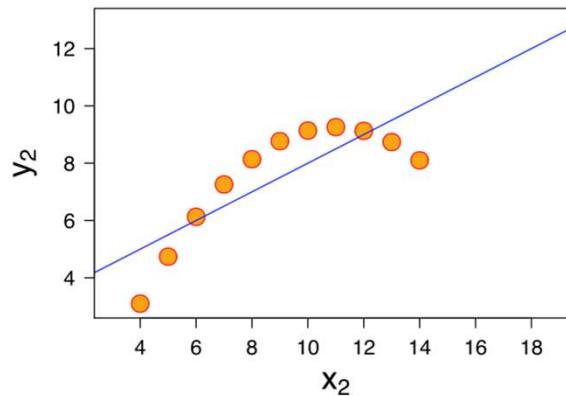
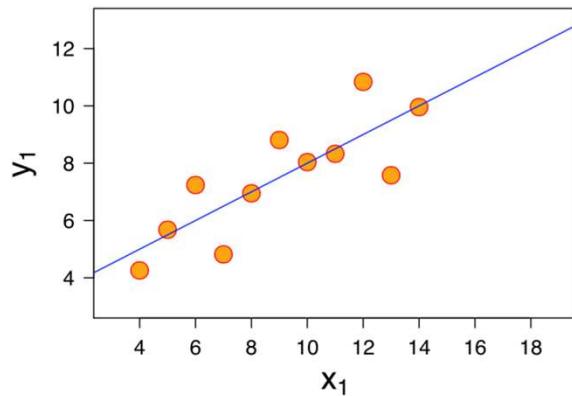
- Se poate defini o funcție de cost
- Loss = $\sum_{i=1,n} (y^i - f(x^i))^2$ -- minimizată → valorile optime ale lui β

- Derivarea loss-ului după β : $\beta = (X^T X)^{-1} X^T y$

- Dacă $d = 1$, $\beta_1 = \text{cov}(x,y)/\text{var}(x)$, $\beta_0 = y - \beta_1 x$

Sisteme inteligente – SIS – Învățare automată

- Metoda celor mai mici pătrate
 - Anscombe Quartet

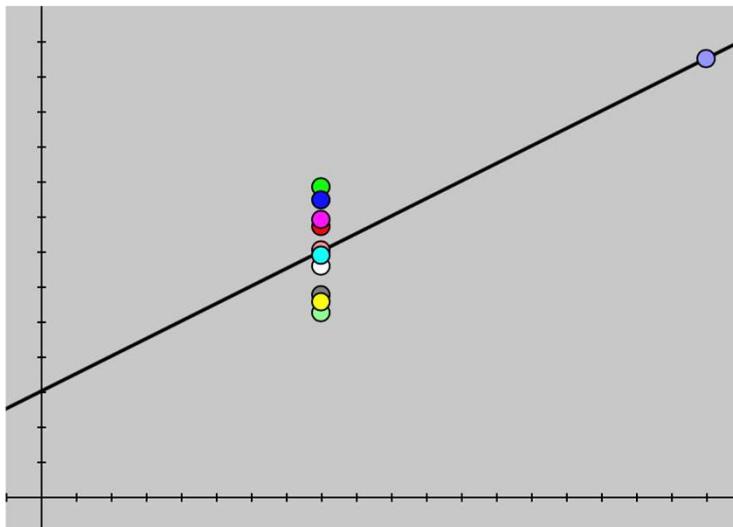


[Ref. imagine](#)

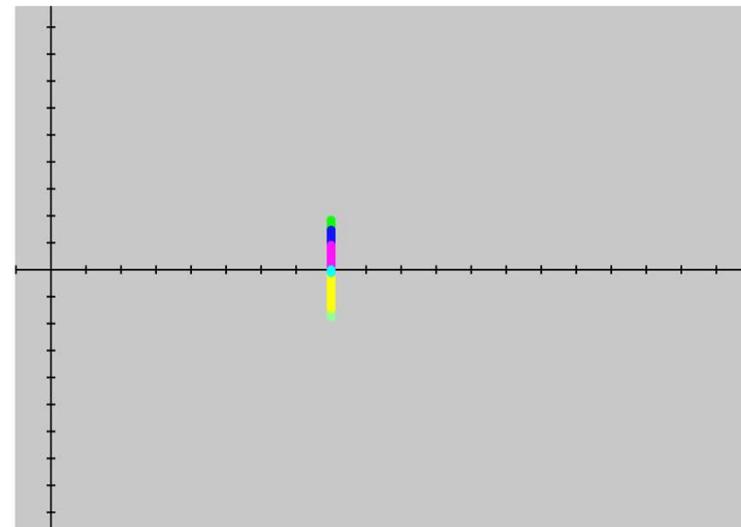
Sisteme inteligente – SIS – Învățare automată

- Metoda celor mai mici pătrate
 - Residual plot

Scatter plot



Residual plot

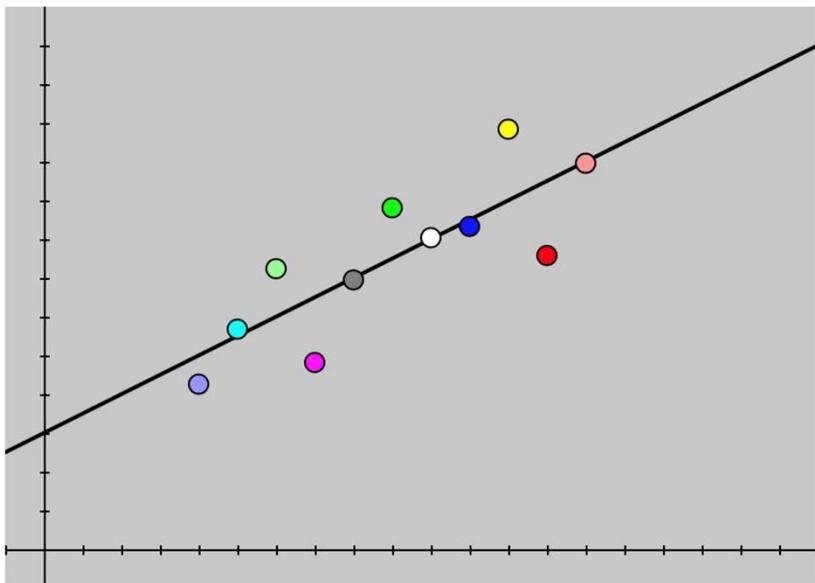


[Ref. imagine](#)

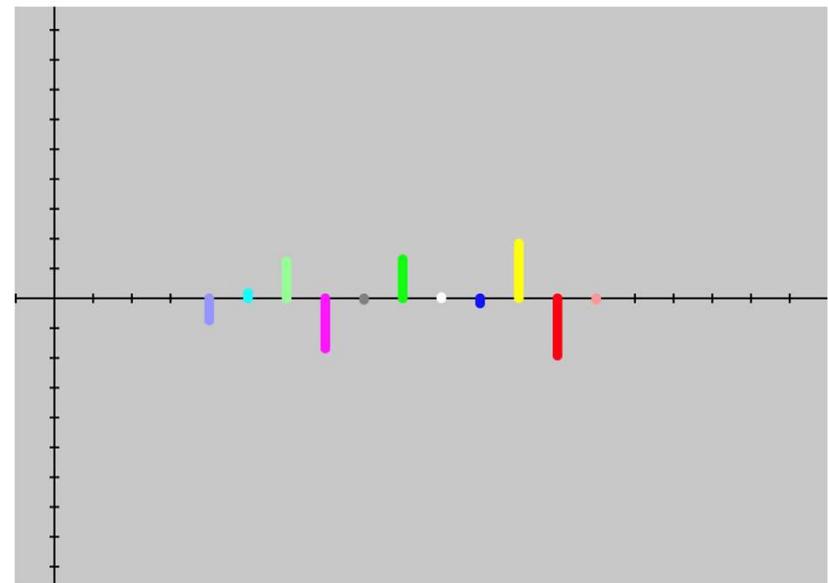
Sisteme inteligente – SIS – Învățare automată

- Metoda celor mai mici pătrate
 - Residual plot

Scatter plot



Residual plot

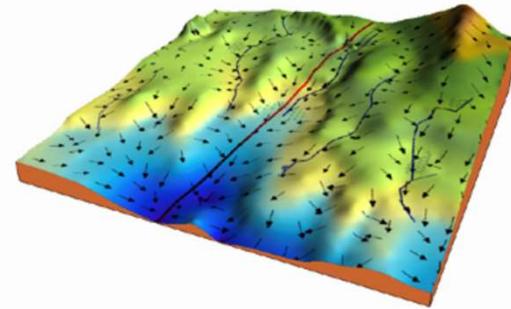


[Ref. imagine](#)

Sisteme inteligente – SIS – Învățare automată

□ Metoda *gradient descent*

- Presupunem cazul unei probleme de regresie
 - Date de intrare $x \in \mathbb{R}^d$
 - Date de ieșire $y \in \mathbb{R}$



- Se cere un model **liniar** f care transformă x în y
- $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_d x_d$
- Invățare supervizată

[Ref. imagine](#)

Sisteme inteligente – SIS – Învățare automată

□ Metoda *gradient descent*

■ Modelarea coeficienților β :

- la iterația 0: valori random (sau 0)
- la iterația $t + 1$ ($t = 0, 1, 2, \dots$)

$$\beta_k(t+1) = \beta_k(t) - \text{learning_rate} * \text{error}(t) * x_k, k=1,2,\dots,d$$

$$\beta_0(t+1) = \beta_0(t) - \text{learning_rate} * \text{error}(t)$$

- Unde

- $\text{error}(t) = \text{computed} - \text{realOutput}$

- $\text{error}(t) = \beta_0(t) + \beta_1(t)*x_1 + \beta_2(t)*x_2 + \dots + \beta_d(t)*x_d - y$

Sisteme inteligente – SIS – Învățare automată

□ Metoda *gradient descent* – versiuni:

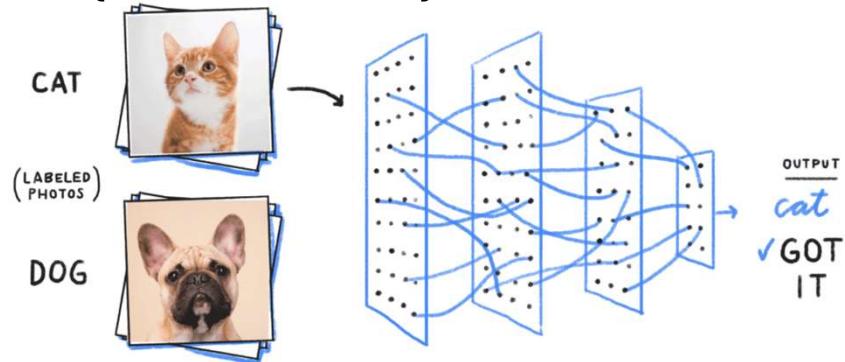
- Stochastic GD
 - Eroarea se calculează pentru fiecare exemplu de antrenament
 - Modelul se updatează pentru fiecare exemplu de antrenament (*online learning*)
- Batch GD
 - Eroarea se calculează pentru fiecare exemplu de antrenament
 - Modelul se updatează după ce toate exemplele de antrenament au fost evaluate (la finalul unei epoci)
- Mini-batch GD
 - Combinare a precedentelor două
 - Setul de date se împarte în mai multe părți (mini-batch-uri)
 - Eroarea se calculează pentru fiecare exemplu de antrenament dintr-un mini-batch
 - Modelul se updatează pentru fiecare exemplu de antrenament dintr-un mini-batch

Sisteme inteligente – SIS – Învățare automată

❑ Regresie Logistică (clasificare)

■ Presupunem cazul unei probleme de clasificare

- ❑ Date de intrare $x^i \in R^d$, $i=1,n$
- ❑ Date de ieșire $y^i \in \{0,1\}$ sau {label1, label2}



- ❑ Se cere un model **liniar** f care separă orice x^i în 2 clase (etichetate cu 0 și 1)
- ❑ $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_d x_d$
- ❑ Invățare supervizată

Sisteme inteligente – SIS – Învățare automată

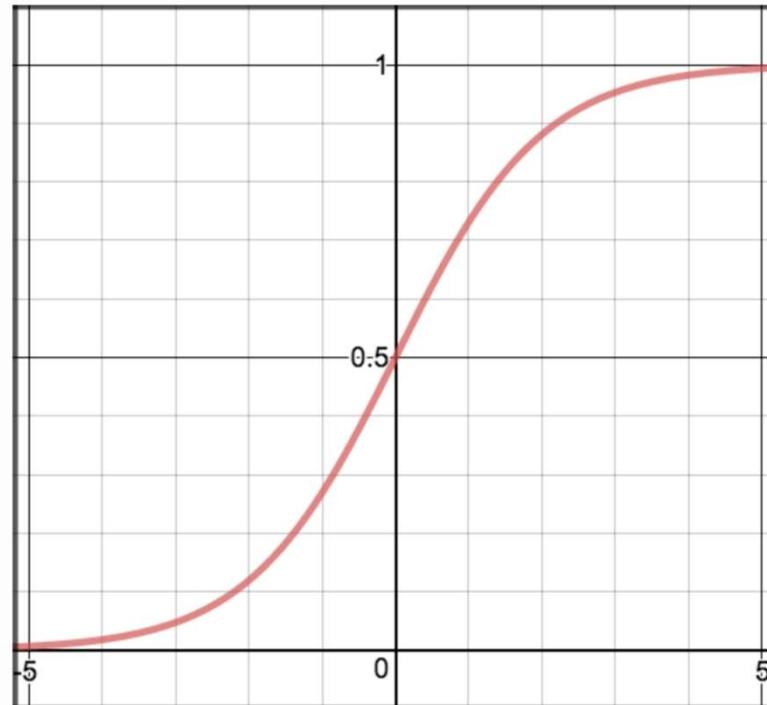
❑ Regresie Logistică (clasificare)

- Mapează datele într-un set discret de clase (label-uri)
- Tipuri:
 - ❑ Binar (Pass/Fail, True/False)
 - ❑ Multi (Cat,Dog,Panda)
 - ❑ Ordinal (Low,Medium,High)
- Folosește funcția sigmoid pentru a decide clasa de apartenență
- Putem folosi gradient descent pentru minimizarea erorii

Sisteme inteligente – SIS – Învățare automată

- Regresie Logistică – sigmoid:
 - Mapează orice numar real in intervalul (0,1)

$$S(z) = \frac{1}{1 + e^{-z}}$$



[Ref. imagine](#)

Sisteme inteligente – SIS – Învățare automată

❑ Regresie Logistica

■ Modelarea coeficienților β :

- ❑ la iterația 0: valori random (sau 0)
- ❑ la iterația $t + 1$ ($t = 0, 1, 2, \dots$)

$$\beta_k(t+1) = \beta_k(t) - \text{learning_rate} * \text{error}(t) * x_k, k=1,2,\dots,d$$

$$\beta_0(t+1) = \beta_0(t) - \text{learning_rate} * \text{error}(t)$$

- Unde

- $\text{error}(t) = \text{Sigmoid}(\text{computed}) - \text{realOutput}$
- $\text{error}(t) = \text{Sigmoid}(\beta_0(t) + \beta_1(t)*x_1 + \beta_2(t)*x_2 + \dots + \beta_d(t)*x_d) - y$

■ Clasificarea rezultatelor

- ❑ $(0,1) \rightarrow [\text{label}_0, \text{label}_1, \dots \text{label}_n]$

Sisteme inteligente – SIS – Învățare automată

- Metoda bazată pe algoritmi evolutivi
 - Modelarea coeficienților β cu ajutorul cromozomilor
 - Fitness-ul calitatea coeficienților β



Recapitulare

□ Sisteme care învață singure (SIS)

■ Instruire (învățare) automata (Machine Learning - ML)

- Învățare supervizată → datele de antrenament sunt deja etichetate cu elemente din E, iar datele de test trebuie etichetate cu una dintre etichetele din E pe baza unui model (învățat pe datele de antrenament) care face corespondență date-etichete
- Învățare nesupervizată → datele de antrenament NU sunt etichetate, trebuie învățat un model de etichetare, iar apoi datele de test trebuie etichetate cu una dintre etichetele identificate de model



Recapitulare

□ Sisteme care învață singure (SIS)

■ Metoda celor mai mici patrate

- Supervizată
- Output continu (vânzări, preț, etc.)
- Panta constantă

■ Gradient descent

- Supervizată
- Output continu (vânzări, preț, etc.)
- Optimizare

■ Regresie Logistică

- CLASIFICARE
- Supervizată
- Output – label-uri (clase) – set discret
- Folosește Gradient descent

Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

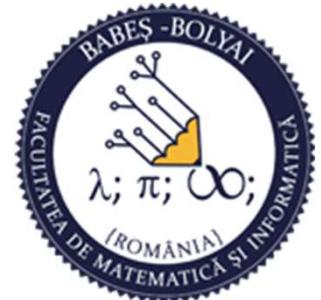
Cursul următor – Materiale de citit și legături utile

- Capitolul VI (19) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 8 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 12 și 13 din *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- Capitolul 4 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

-
- Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean –
www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan -
www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop -
www.cs.ubbcluj.ro/~hfpop



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ



Sisteme inteligente

Sisteme care învață singure
– rețele neuronale artificiale –

Laura Dioșan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

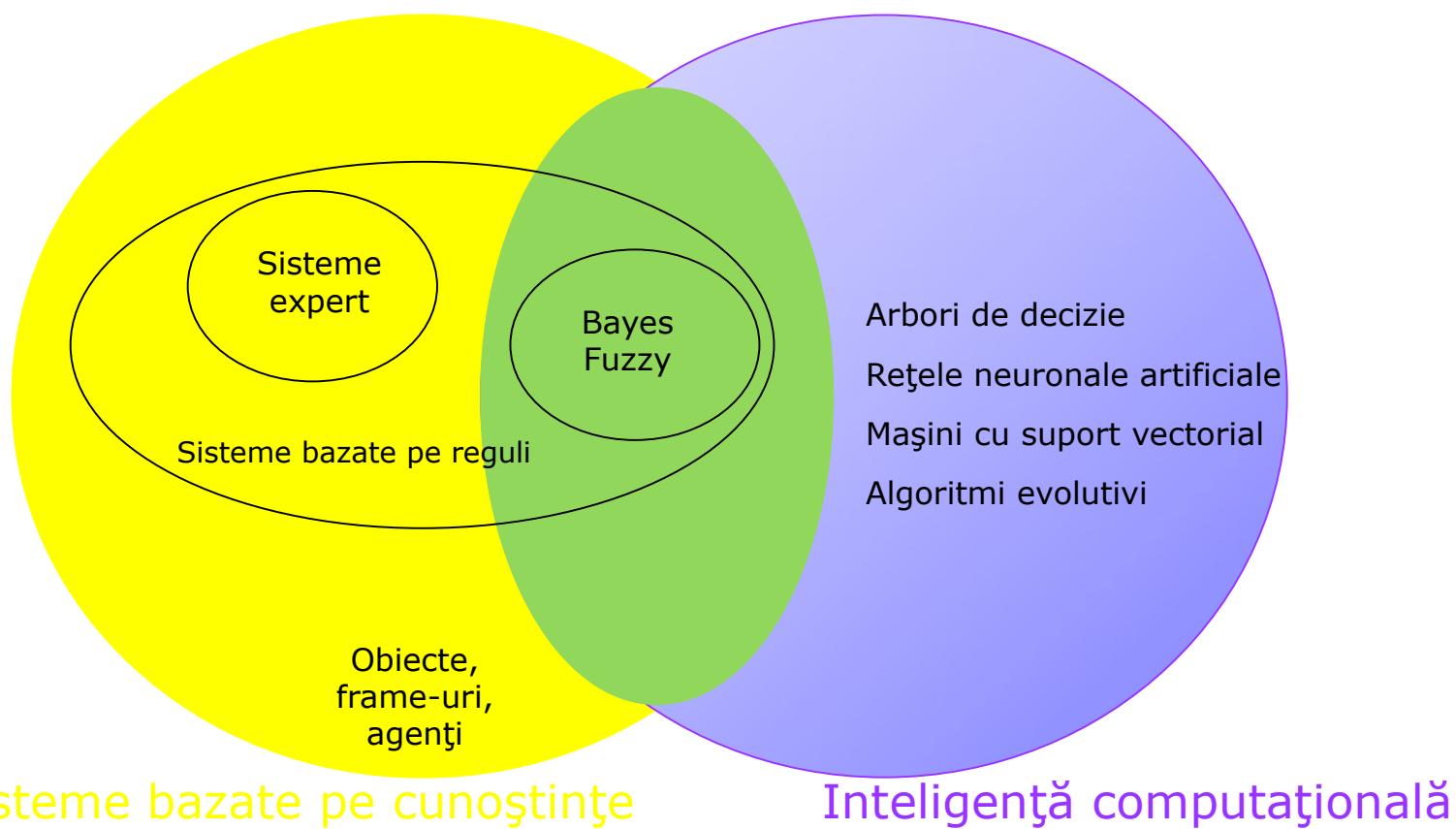
C. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Materiale de citit și legături utile

- Capitolul VI (19) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 8 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 12 și 13 din *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- Capitolul 4 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Sisteme inteligente



Sisteme inteligente – SIS – Învățare automată

□ Tipologie

- În funcție de experiența acumulată în timpul învățării:
 - SI cu învățare supervizată
 - SI cu învățare nesupervizată
 - SI cu învățare activă
 - SI cu învățare cu întărire

- În funcție de modelul învățat (algoritmul de învățare):
 - Arbori de decizie
 - **Rețele neuronale artificiale**
 - Algoritmi evolutivi
 - Mașini cu suport vectorial
 - Modele Markov ascunse

Sisteme inteligente – SIS – RNA

□ Rețele neuronale artificiale (RNA)

- Scop
- Definire
- Tipuri de probleme rezolvabile
- Caracteristici
- Exemplu
- Proiectare
- Evaluare
- Tipologie

Sisteme inteligente – SIS – RNA

□ Scop

- Clasificare binară pentru orice fel de date de intrare (discrete sau continue)
 - Datele pot fi separate de:
 - o dreaptă $\rightarrow ax + by + c = 0$ (dacă $m = 2$)
 - un plan $\rightarrow ax + by + cz + d = 0$ (dacă $m = 3$)
 - un hiperplan $\sum a_i x_i + b = 0$ (dacă $m > 3$)
 - Cum găsim valorile optime pt. a, b, c, d, a_i ?
 - Rețele neuronale artificiale (RNA)
 - Mașini cu suport vectorial (MSV)
- De ce RNA?
- Cum învață creierul?

Sisteme inteligente – SIS – RNA

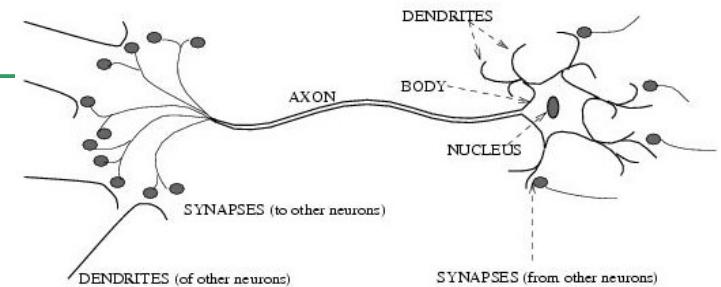
□ Scop → De ce RNA?

- Unele sarcini pot fi efectuate foarte ușor de către oameni, însă sunt greu de codificat sub forma unor algoritmi
 - Recunoașterea formelor
 - vechi prieteni
 - caractere scrise de mână
 - vocea
 - Diferite raționamente
 - conducerea autovehiculelor
 - cântatul la pian
 - jucarea baschetului
 - înnotul
- Astfel de sarcini sunt dificil de definit formal și este dificilă aplicarea unui proces de raționare pentru efectuarea lor

Sisteme inteligente – SIS – RNA

□ Scop → Cum învață creierul?

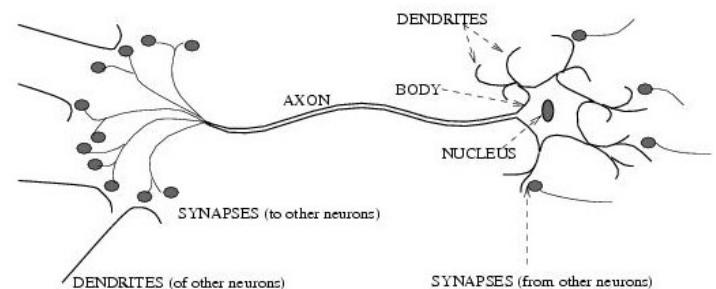
- Creierul uman - componență
 - Aproximativ 10.000.000.000 de neuroni conectați prin sinapse
 - Fiecare neuron
 - are un corp (soma), un axon și multe dendrite
 - poate fi într-una din 2 stări:
 - activ – dacă informația care intră în neuron depășește un anumit prag de stimulare –
 - pasiv – altfel
 - Sinapsă
 - Legătura între axon-ul unui neuron și dendritele altui neuron
 - Are rol în schimbul de informație dintre neuroni
 - 5.000 de conexiuni / neuron (în medie)
 - În timpul vieții pot să apară noi conexiuni între neuroni



Sisteme inteligente – SIS – RNA

□ Scop → Cum învață creierul?

- Cum "învață" (procesează informații)?
 - Conexiunile care de-a lungul trăirii unor experiențe s-au dovedit utile devin permanente (restul sunt eliminate)
 - Creierul este interesat de noutăți
 - Modelul de procesare a informației
 - Învățare
 - Depozitare
 - Amintire
 - Memoria
 - Tipologie
 - De scurtă durată
 - Imediată → 30 sec.
 - De lucru
 - De lungă durată
 - Capacitate
 - Crește odată cu vârsta
 - Limitată → învățarea unei poezii pe strofe
 - Influențată și de stările emoționale
 - Creierul
 - rețea de neuroni
 - sistem foarte complex, ne-liniar și paralel de procesare a informației
- Informația este depozitată și procesată de întreaga rețea, nu doar de o anumită parte a rețelei → informații și procesare globală
- Caracteristica fundamentală a unei rețele de neuroni → învățarea → rețele neuronale artificiale (RNA)



Sisteme inteligente – SIS – RNA

- Definire
 - Ce este o RNA?
 - RN biologice vs. RN artificiale
 - Cum învață rețeaua?

Sisteme inteligente – SIS – RNA

□ Definire → Ce este o RNA?

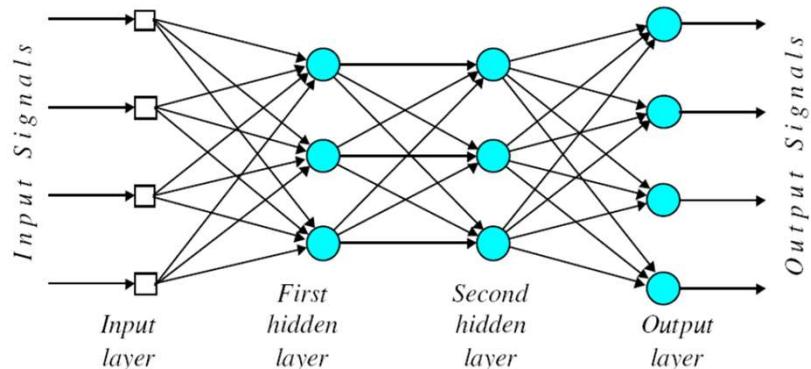
- O structură similară unei rețele neuronale biologice
- O mulțime de noduri (unități, neuroni, elemente de procesare) dispuse ca într-un graf pe mai multe straturi (*layere*)

□ Nodurile

- au intrări și ieșiri
- efectuează un calcul simplu prin intermediul unei funcții asociate → funcție de activare
- sunt conectate prin legături ponderate
 - Conexiunile între noduri conturează structura (arhitectura) rețelei
 - Conexiunile influențează calculele care se pot efectua

□ Straturile

- Strat de intrare
 - Conține m (nr de atrbute al unei date) noduri
- Strat de ieșire
 - Conține r (nr de ieșiri) noduri
- Straturi intermediare (ascunse) – rol în “complicarea” rețelei
 - Diferite structuri
 - Diferite mărimi



Sisteme inteligente – SIS – RNA

□ Definire → RN biologice vs. RN artificiale

RNB	RNA
Soma	Nod
Dendrite	Intrare
Axon	Ieșire
Activare	Procesare
Synapsă	Conexiune ponderată

□ Definire → Cum învață rețeaua?

- Plecând de la un set de n date de antrenament de forma

$$((x_{p1}, x_{p2}, \dots, x_{pm}, y_{p1}, y_{p2}, \dots, y_{pr}))$$

cu $p = 1, 2, \dots, n$, m – nr atributelor, r – nr ieșirilor

- se formează o RNA cu m noduri de intrare, r noduri de ieșire și o anumită structură internă
 - un anumit nr de nivele ascunse, fiecare nivel cu un anumit nr de neuroni
 - cu legături ponderate între oricare 2 noduri
- se caută valorile optime ale ponderilor între oricare 2 noduri ale rețelei prin minimizarea erorii
 - diferența între rezultatul real y și cel calculat de către rețea

Sisteme inteligente – SIS – RNA

- Tipuri de probleme rezolvabile cu RNA
 - Datele problemei se pot reprezenta prin numeroase perechi atribut-valoare
 - Funcția obiectiv poate fi:
 - Unicriterială sau multicriterială
 - Discretă sau cu valori reale
 - Datele de antrenament pot conține erori (zgomot)
 - Timp de rezolvare (antrenare) prelungit

Sisteme inteligente – SIS – RNA

□ Proiectare

- Construirea RNA pentru rezolvarea problemei P
- Inițializarea parametrilor RNA
- Antrenarea RNA
- Testarea RNA

Sisteme inteligente – SIS – RNA

□ Proiectare

- Construirea RNA pentru rezolvarea unei probleme P
 - pp. o problemă de clasificare în care avem un set de date de forma:
 - (x^d, t^d) , cu:
 - $x^d \in \mathbb{R}^m \rightarrow x^d = (x_{1^d}, x_{2^d}, \dots, x_{m^d})$
 - $t^d \in \mathbb{R}^R \rightarrow t^d = (t_{1^d}, t_{2^d}, \dots, t_{R^d})$,
 - cu $d = 1, 2, \dots, n, n+1, n+2, \dots, N$
 - primele n date vor fi folosite drept bază de antrenament a RNA
 - ultimele $N-n$ date vor fi folosite drept bază de testare a RNA
- se construiește o RNA astfel:
 - stratul de intrare conține exact m noduri (fiecare nod va citi una dintre proprietățile de intrare ale unei instanțe a problemei – $x_{1^d}, x_{2^d}, \dots, x_{m^d}$)
 - stratul de ieșire poate conține R noduri (fiecare nod va furniza una dintre proprietățile de ieșire ale unei instanțe a problemei $t_{1^d}, t_{2^d}, \dots, t_{R^d}$)
 - unul sau mai multe straturi ascunse cu unul sau mai mulți neuroni pe fiecare strat

Sisteme inteligente – SIS – RNA

□ Proiectare

- Construirea RNA pentru rezolvarea problemei P
- **Inițializarea parametrilor RNA**
- **Antrenarea RNA**
- Testarea RNA

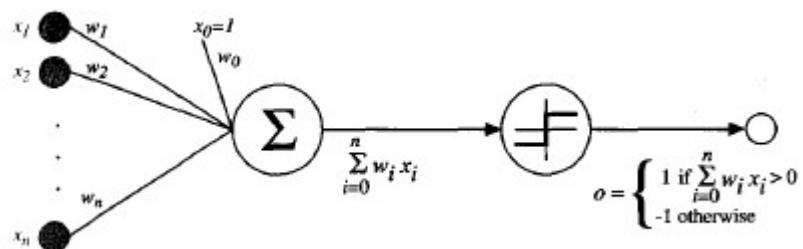
Sisteme inteligente – SIS – RNA

□ Proiectare

- Inițializarea parametrilor RNA
 - Inițializarea ponderile între oricare 2 noduri de pe straturi diferite
 - Stabilirea funcției de activare corespunzătoare fiecărui neuron (de pe straturile ascunse)
- Antrenarea (învățarea) RNA
 - Scop:
 - stabilirea valorii optime a ponderilor dintre 2 noduri
 - Algoritm
 - Se caută valorile optime ale ponderilor între oricare 2 noduri ale rețelei prin minimizarea erorii (diferența între rezultatul real y și cel calculat de către rețea)
 - Cum învață rețeaua?
 - Rețeaua = mulțime de unități primitive de calcul interconectate între ele →
 - Învățarea rețelei = \cup Învățarea unităților primitive
 - Unități primitive de calcul
 - Perceptron
 - Unitate liniară
 - Unitate sigmoidală

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață rețeaua?
 - Neuronul ca element simplu de calcul
 - Structura neuronului
 - Fiecare nod are intrări și ieșiri
 - Fiecare nod efectuează un calcul simplu
 - Procesarea neuronului
 - Se transmite informația neuronului
 - Neuronul procesează informația
 - Se citește răspunsul neuronului
 - Învățarea neuronului – algoritmul de învățare a ponderilor care procesează corect informațiile
 - Se pornește cu un set inițial de ponderi oarecare
 - Cât timp nu este îndeplinită o condiție de oprire
 - Se procesează informația și se stabilește calitatea ponderilor curente
 - Se modifică ponderile astfel încât să se obțină rezultate mai bune



Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață rețeaua?
 - Neuronul ca element simplu de calcul
 - Structura neuronului
 - Fiecare nod are intrări și ieșiri
 - Fiecare nod efectuează un calcul simplu prin intermediul unei funcții asociate
 - Procesarea neuronului
 - Se transmite informația neuronului → se calculează suma ponderată a intrărilor
$$net = \sum_{i=1}^n x_i w_i$$
 - Neuronul procesează informația → se folosește o funcție de activare:
 - Funcția constantă
 - Funcția prag
 - Funcția rampă
 - Funcția liniară
 - Funcția sigmoidală
 - Funcția Gaussiană
 - Funcția Relu

Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață rețeaua?

■ Funcția de activare a unui neuron

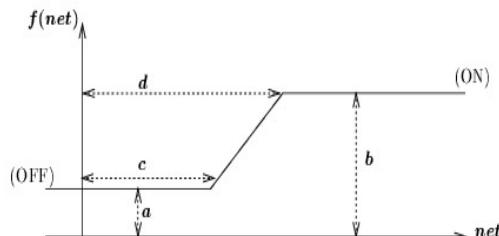
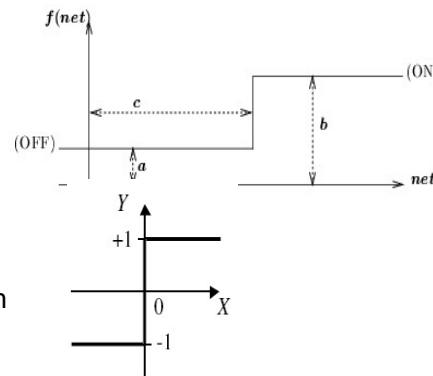
- Funcția constantă $f(\text{net}) = \text{const}$
- Funcția prag (c - pragul)

$$f(\text{net}) = \begin{cases} a, & \text{dacă } \text{net} < c \\ b, & \text{dacă } \text{net} > c \end{cases}$$

- Pentru $a=+1$, $b=-1$ și $c=0 \rightarrow$ funcția semn
- Funcție discontinuă

□ Funcția rampă

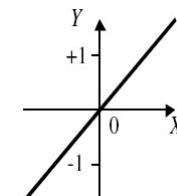
$$f(\text{net}) = \begin{cases} a, & \text{dacă } \text{net} \leq c \\ b, & \text{dacă } \text{net} \geq d \\ a + \frac{(\text{net}-c)(b-a)}{d-c}, & \text{altfel} \end{cases}$$



□ Funcția liniară

$$f(\text{net}) = a * \text{net} + b$$

- Pentru $a = 1$ și $b = 0 \rightarrow$ funcția identitate $f(\text{net})=\text{net}$
- Funcție continuă



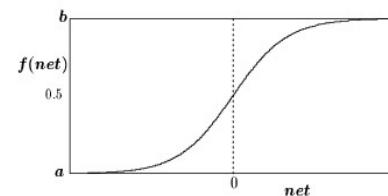
Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață rețeaua?

■ Funcția de activare a unui neuron

□ Funcția sigmoidală

- În formă de S
- Continuă și diferențiabilă în orice punct
- Simetrică rotațional față de un anumit punct ($net = c$)
- Atinge asymptotic puncte de saturare



$$\lim_{net \rightarrow -\infty} f(net) = a \quad \lim_{net \rightarrow \infty} f(net) = b$$

- Exemple de funcții sigmoidale:

$$f(net) = z + \frac{1}{1 + \exp(-x \cdot net + y)}$$

$$f(net) = \tanh(x \cdot net - y) + z$$

$$\text{unde } \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

- Pentru $y=0$ și $z = 0 \Rightarrow a=0, b = 1, c=0$
- Pentru $y=0$ și $z = -0.5 \Rightarrow a=-0.5, b = 0.5, c=0$
- Cu cât x este mai mare, cu atât curba este mai abruptă

Sisteme inteligente – SIS – RNA

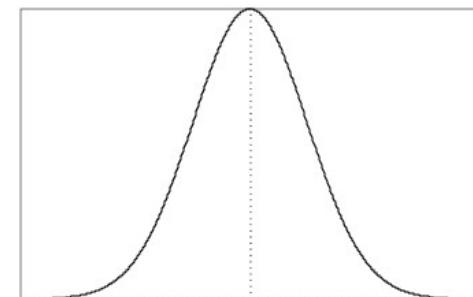
□ Proiectare → Antrenarea RNA → Cum învață rețeaua?

■ Funcția de activare a unui neuron

□ Functia Gaussiană

- În formă de clopot
- Continuă
- Atinge asimptotic un punct de saturatie

$$\lim_{net \rightarrow \infty} f(net) = a$$



- Are un singur punct de optim (maxim) – atins când net = μ
- Exemplu

$$f(net) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{net - \mu}{\sigma}\right)^2\right]$$

Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață rețeaua?

■ Funcția de activare a unui neuron

□ Funcția ReLU

- În formă de rampă
- Continuă, monotonă
- Derivata ei este monotonă
- Codomeniu pozitiv $[0, \infty)$

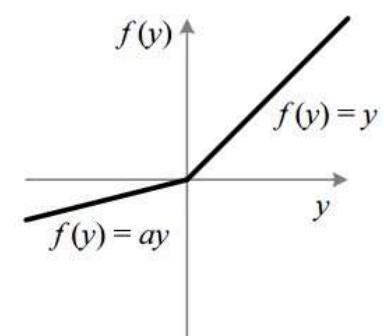
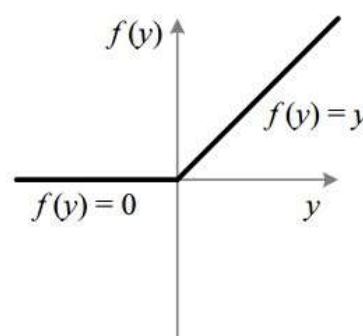
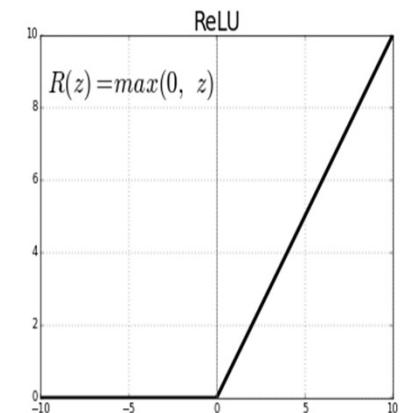
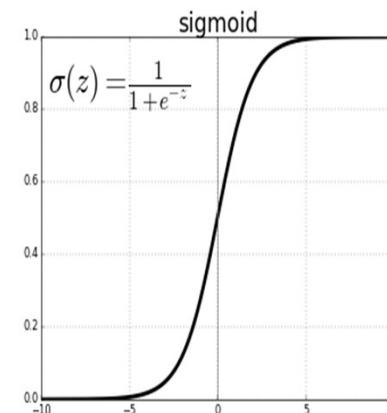
$$f(\text{net}) = \max(0, \text{net})$$

$$f(\text{net}) = \begin{cases} 0, & \text{dacă } \text{net} < 0 \\ \text{net}, & \text{dacă } \text{net} \geq 0 \end{cases}$$

▪ Variantă: Leaky ReLU

- Compensează problemele cu argumentele negative din ReLU

$$f(\text{net}) = \begin{cases} a \cdot \text{net}, & \text{dacă } \text{net} < 0 \\ \text{net}, & \text{dacă } \text{net} \geq 0 \end{cases}$$



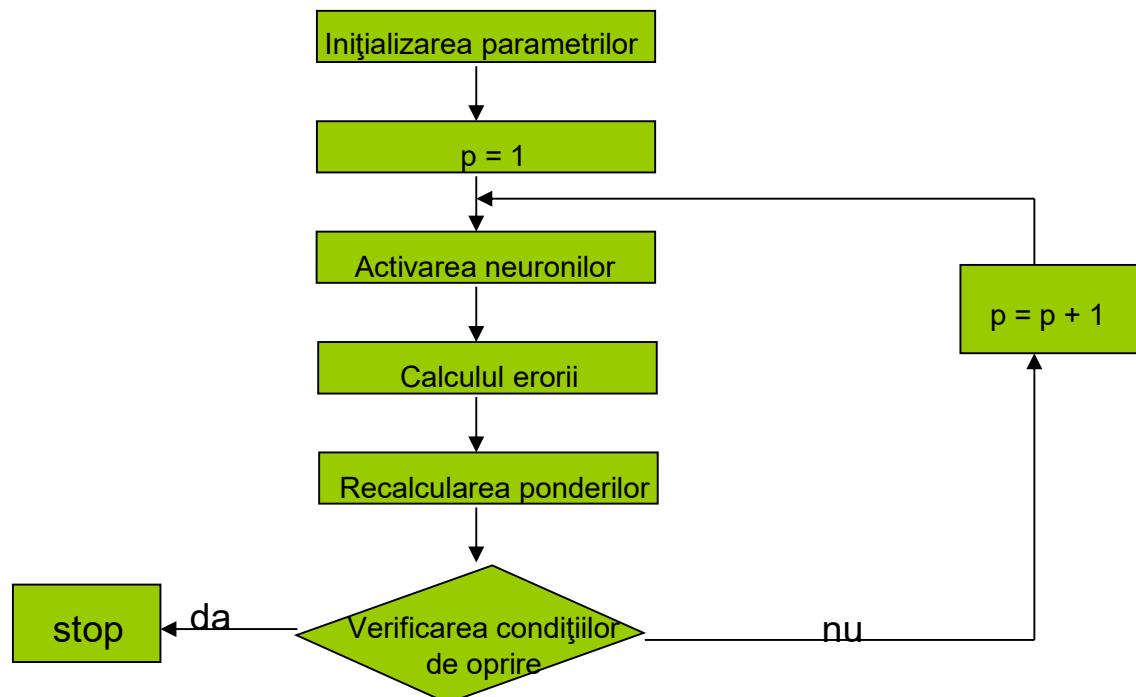
Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață rețeaua?
 - Neuronul ca element simplu de calcul
 - Structura neuronului
 - Procesarea neuronului
 - Se transmite informația neuronului → se calculează suma ponderată a intrărilor
 - Neuronul procesează informația → se folosește o funcție de activare:
 - Funcția constantă
 - Funcția prag
 - Funcția rampă
 - Funcția liniară
 - Funcția sigmoidală
 - Funcția Gaussiană
 - Se citește răspunsul neuronului → se stabilește dacă rezultatul furnizat de neuron coincide sau nu cu cel dorit (real)

Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață rețeaua?

- Neuronul ca element simplu de calcul
 - Structura neuronului
 - Procesarea neuronului
 - Învățarea neuronului
 - Algoritm



Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață RNA?

■ Învățarea neuronului

□ 2 reguli de bază

▪ Regula perceptronului → algoritmul perceptronului

1. Se porneste cu un set de ponderi oarecare
2. Se stabilește calitatea modelului creat pe baza acestor ponderi pentru **UNA** dintre datele de intrare
3. Se ajustează ponderile în funcție de calitatea modelului
4. Se reia algoritmul de la pasul 2 până când se ajunge la calitate maximă

▪ Regula Delta → algoritmul scăderii după gradient

1. Se porneste cu un set de ponderi oarecare
2. Se stabilește calitatea modelului creat pe baza acestor ponderi pentru **TOATE** dintre datele de intrare
3. Se ajustează ponderile în funcție de calitatea modelului
4. Se reia algoritmul de la pasul 2 până când se ajunge la calitate maximă

- Similar regulii perceptronului, dar calitatea unui model se stabilește în funcție de toate datele de intrare (tot setul de antrenament)

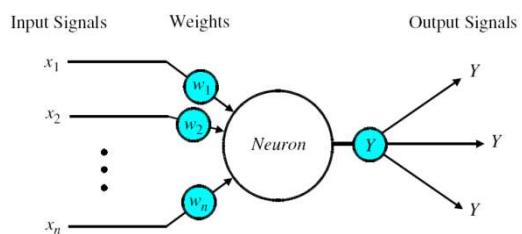
Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață RNA?

■ Învățarea neuronului

□ Pp că avem un set de date de antrenament de forma:

- (x^d, t^d) , cu:
 - $x^d \in \mathbb{R}^m \rightarrow x^d = (x_1^d, x_2^d, \dots, x_m^d)$
 - $t^d \in \mathbb{R}^R \rightarrow t^d = (t_1^d, t_2^d, \dots, t_R^d)$, și $R = 1$ (adică $t^d = (t_1^d)$)
 - cu $d = 1, 2, \dots, n$
- RNA = unitate primitivă de calcul (un neuron) \rightarrow o rețea cu:
 - m noduri de intrare
 - legate de neuronul de calcul prin ponderile w_i , $i = 1, 2, \dots, m$ și
 - cu un nod de ieșire



Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învăță RNA?

■ Învățarea neuronului

□ Algoritmul perceptronului

- Se bazează pe minimizarea erorii asociată unei instanțe din setul de date de antrenament
- Modificarea ponderilor pe baza erorii asociate unei instanțe din setul de antrenament

Inițializare ponderi din rețea

$w_i = \text{random}(a, b)$, unde $i=1,2,\dots,m$

$d = 1$

Cât timp mai există exemple de antrenament clasificate incorect

Se activează neuronul și se calculează ieșirea

Perceptron → funcția de activare este funcția semn (funcție prag de tip discret, nediferențiabil)

$$o^d = \text{sign}(\mathbf{wx}) = \text{sign}\left(\sum_{i=1}^m w_i x_i\right)$$

Se stabilește ajustarea ponderilor $\Delta w_i = \eta(t^d - o^d)x_i^d$, unde $i=1,2,\dots,m$

unde η - rată de învățare

Se ajustează ponderile $w'_i = w_i + \Delta w_i$

Dacă $d < n$ atunci $d++$

Altfel $d = 1$

SfCâtTimp

Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață RNA?

■ Învățarea neuronului

□ Algoritmul scădere după gradient

- Se bazează pe eroarea asociată întregului set de date de antrenament
- Modificarea ponderilor în direcția dată de cea mai abruptă pantă a reducerii erorii $E(\mathbf{w})$ pentru tot setul de antrenament

$$E(\mathbf{w}) = \frac{1}{2} \sum_{d=1}^n (t^d - o^d)^2$$

- Cum se determină cea mai abruptă pantă? \rightarrow se derivează E în funcție de w (se stabilește gradientul erorii E)

$$\nabla E(\mathbf{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_m} \right)$$

- Gradientul erorii E se calculează în funcție de funcția de activare a neuronului (care trebuie să fie diferențialabilă, deci continuă)

- Funcția liniară $f(\text{net}) = \sum_{i=1}^m w_i x_i^d$

- Funcția sigmoidală $f(\text{net}) = \frac{1}{1 + e^{-\mathbf{w}\mathbf{x}}} = \frac{1}{1 + e^{-\sum_{i=1}^m w_i x_i^d}}$

- Cum se ajustează ponderile?

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \text{ unde } i = 1, 2, \dots, m$$

Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață RNA?

■ Învățarea neuronului

- Algoritmul scădere după gradient → calcularea gradientului erorii
 - Funcția liniară

$$\begin{aligned}f(\text{net}) &= \sum_{i=1}^m w_i x_i^d \\ \frac{\partial E}{\partial w_i} &= \frac{\partial \frac{1}{2} \sum_{d=1}^n (t^d - o^d)^2}{\partial w_i} = \frac{1}{2} \sum_{d=1}^n \frac{\partial (t^d - o^d)^2}{\partial w_i} = \frac{1}{2} \sum_{d=1}^n 2(t^d - o^d) \frac{\partial (t^d - o^d)}{\partial w_i} \\ \frac{\partial E}{\partial w_i} &= \sum_{d=1}^n (t^d - o^d) \frac{\partial (t^d - w_1 x_1^d - w_2 x_2^d - \dots - w_m x_m^d)}{\partial w_i} = \sum_{d=1}^n (t^d - o^d)(-x_i^d)\end{aligned}$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_{d=1}^n (t^d - o^d)x_i^d$$

- Funcție sigmoidală

$$f(\text{net}) = \frac{1}{1 + e^{-\mathbf{w}\mathbf{x}}} = \frac{1}{1 + e^{-\sum_{i=1}^m w_i x_i^d}}$$

$$y = s(z) = \frac{1}{1 + e^{-z}} \Rightarrow \frac{\partial s(z)}{\partial z} = s(z)(1 - s(z))$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial \frac{1}{2} \sum_{d=1}^n (t^d - o^d)^2}{\partial w_i} = \frac{1}{2} \sum_{d=1}^n \frac{\partial (t^d - o^d)^2}{\partial w_i} = \frac{1}{2} \sum_{d=1}^n 2(t^d - o^d) \frac{\partial (t^d - \text{sig}(\mathbf{w}\mathbf{x}^d))}{\partial w_i} = \sum_{d=1}^n (t^d - o^d)(1 - o^d)o^d(-x_i^d)$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_{d=1}^n (t^d - o^d)(1 - o^d)o^d x_i^d$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață RNA?
 - Învățarea neuronului
 - Algoritmul scădere după gradient (ASG)

ASG simplu	ASG stocastic
<p>Inițializare ponderi din rețea $w_i = \text{random}(a,b)$, unde $i=1,2,\dots,m$</p> <p>$d = 1$</p> <p>Cât timp nu este îndeplinită condiția de oprire $\Delta w_i = 0$, unde $i=1,2,\dots,m$</p> <p>Pentru fiecare exemplu de antrenament (x^d, t^d), unde $d=1,2,\dots,n$</p> <ul style="list-style-type: none">Se activează neuronul și se calculează ieșirea o^d<ul style="list-style-type: none">funcția de activare = funcția liniară $\rightarrow o^d = \mathbf{w} \mathbf{x}^d$funcția de activare = funcția sigmoid $\rightarrow o^d = \text{sig}(\mathbf{w} \mathbf{x}^d)$ <p>Pentru fiecare pondere w_i, unde $i=1,2,\dots,m$</p> <p>Se stabilește ajustarea ponderii $\Delta w_i = \Delta w_i - \eta \frac{\partial E}{\partial w_i}$</p> <p>Pentru fiecare pondere w_i, unde $i=1,2,\dots,m$</p> <p>Se ajustează fiecare pondere w_i $w_i = w_i + \Delta w_i$</p>	<p>Inițializare ponderi din rețea $w_i = \text{random}(a,b)$, unde $i=1,2,\dots,m$</p> <p>$d = 1$</p> <p>Cât timp nu este îndeplinită condiția de oprire $\Delta w_i = 0$, unde $i=1,2,\dots,m$</p> <p>Pentru fiecare exemplu de antrenament (x^d, t^d), unde $d=1,2,\dots,n$</p> <p>Se activează neuronul și se calculează ieșirea o^d<ul style="list-style-type: none">funcția de activare = funcția liniară $\rightarrow o^d = \mathbf{w} \mathbf{x}^d$funcția de activare = funcția sigmoid $\rightarrow o^d = \text{sig}(\mathbf{w} \mathbf{x}^d)$</p> <p>Pentru fiecare pondere w_i, unde $i=1,2,\dots,m$</p> <p>Se stabilește ajustarea ponderilor $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$</p> <p>Se ajustează ponderea w_i $w_i = w_i + \Delta w_i$</p>

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învăță RNA?
 - Învățarea neuronului

Diferențe	Algoritmul perceptronului	Algoritmul scădere după gradient (regula Delta)
Ce reprezintă o^d	$o^d = \text{sign}(\mathbf{w}\mathbf{x}^d)$	$o^d = \mathbf{w}\mathbf{x}^d$ sau $o^d = \text{sig}(\mathbf{w}\mathbf{x}^d)$
Cum converge	Într-un nr finit de pași (până la separarea perfectă)	Asimtotic (spre eroarea minimă)
Ce fel de probleme se pot rezolva	Cu date liniar separabile	Cu orice fel de date (separabile liniar sau neliinar)
Ce tip de ieșire are neuronul	Discretă și cu prag	Continuă și fără prag

Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA

■ Cum învață rețeaua?

- Rețeaua = mulțime de unități primitive de calcul interconectate între ele →
 - Învățarea rețelei = \cup învățarea unităților primitive
- Rețea cu mai mulți neuroni așezăți pe unul sau mai multe straturi → RNA este capabilă să învețe un model mai complicat (nu doar liniar) de separare a datelor
- Algoritmul de învățare a ponderilor → backpropagation
 - Bazat pe algoritmul scădere după gradient
 - Îmbogățit cu:
 - Informația se propagă în RNA înainte (dinspre stratul de intrare spre cel de ieșire)
 - Eroarea se propagă în RNA înapoi (dinspre stratul de ieșire spre cel de intrare)

Se inițializează ponderile

Cât timp nu este îndeplinită condiția de oprire

Pentru fiecare exemplu (x^d, t^d)

Se activează fiecare neuron al rețelei

Se propagă informația înainte și se calculează ieșirea corespunzătoare fiecărui neuron al rețelei

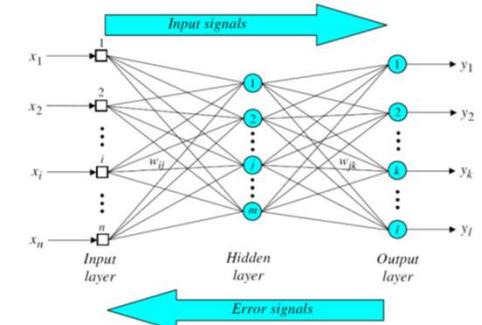
Se ajustează ponderile

Se stabilește și se propagă eroarea înapoi

Se stabilesc erorile corespunzătoare neuronilor din stratul de ieșire

Se propagă aceste erori înapoi în toată rețeaua → se distribuie erorile pe toate conexiunile existente în rețea proporțional cu valorile ponderilor asociate acestor conexiuni

Se modifică ponderile



Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA

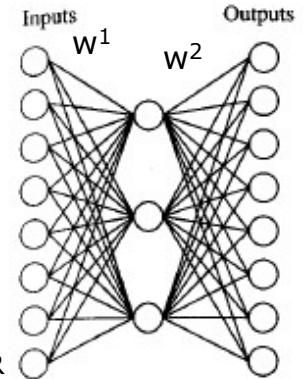
■ Cum învață o întreagă RNA?

□ Pp că avem un set de date de antrenament de forma:

- (x^d, t^d) , cu:
 - $x^d \in \mathbb{R}^m \rightarrow x^d = (x_{d_1}^d, x_{d_2}^d, \dots, x_{d_m}^d)$
 - $t^d \in \mathbb{R}^R \rightarrow t^d = (t_{d_1}^d, t_{d_2}^d, \dots, t_{d_R}^d)$
 - cu $d = 1, 2, \dots, n$

□ Presupunem 2 cazuri de RNA

- O RNA cu un singur strat ascuns cu H neuroni \rightarrow RNA_1
 - m neuroni pe stratul de intrare,
 - R neuroni pe stratul de ieșire,
 - H neuroni pe stratul ascuns
 - Ponderile între stratul de intrare și cel ascuns w_{ih}^1 cu $i=1, 2, \dots, m$, $h = 1, 2, \dots, H$
 - Ponderile între stratul ascuns și cel de ieșire w_{hr}^2 cu $h = 1, 2, \dots, H$ și $r = 1, 2, \dots, R$
- O RNA cu p straturi ascunse, fiecare strat cu h_i ($i = 1, 2, \dots, p$) neuroni \rightarrow RNA_p
 - m neuroni pe stratul de intrare,
 - R neuroni pe stratul de ieșire,
 - P straturi ascunse
 - H_p neuroni pe stratul ascuns p , $p = 1, 2, \dots, P$
 - Ponderile între stratul de intrare și primul strat ascuns w_{ih}^1 cu $i=1, 2, \dots, m$, $h_1 = 1, 2, \dots, H_1$
 - Ponderile între primul strat ascuns și cel de-al doilea strat ascuns $w_{h_1 h_2}^2$ cu $h_1 = 1, 2, \dots, H_1$ și $h_2 = 1, 2, \dots, H_2$
 - Ponderile între cel de-al doilea strat ascuns și cel de-al treilea strat ascuns $w_{h_2 h_3}^3$ cu $h_2 = 1, 2, \dots, H_2$ și $h_3 = 1, 2, \dots, H_3$
 - ...
 - Ponderile între cel de-al $p-1$ strat ascuns și ultimul strat ascuns $w_{h_{p-1} h_p}^p$ cu $h_{p-1} = 1, 2, \dots, H_{p-1}$ și $h_p = 1, 2, \dots, H_p$
 - Ponderile între ultimul strat ascuns și cel de ieșire $w_{h_p r}^{p+1}$ cu $h_p = 1, 2, \dots, H_p$ și $r = 1, 2, \dots, R$



Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață o întreagă RNA?
 - Algoritmul backpropagation pentru RNA₁

Se inițializează ponderile w_{ih}^1 și w_{hr}^2 cu $i=1,2,\dots,m$, $h=1,2,\dots,H$ și $r=1,2,\dots,R$

Cât timp nu este îndeplinită condiția de oprire

Pentru fiecare exemplu (x^d, t^d)

Se activează fiecare neuron al rețelei

Se propagă informația înainte și se calculează ieșirea corespunzătoare fiecărui neuron al rețelei

$$o_h^d = \sum_{i=1}^m w_{ih}^1 x_i^d \text{ sau } o_h^d = \text{sig}\left(\sum_{i=1}^m w_{ih}^1 x_i^d\right), \text{ cu } h=1,2,\dots,H$$

$$o_r^d = \sum_{h=1}^H w_{hr}^2 o_h^d \text{ sau } o_r^d = \text{sig}\left(\sum_{h=1}^H w_{hr}^2 o_h^d\right), \text{ cu } r=1,2,\dots,R$$

Se ajustează ponderile

Se stabilește și se propagă eroarea înapoi

Se stabilesc erorile corespunzătoare neuronilor din stratul de ieșire

$$\delta_r^d = t_r^d - o_r^d \text{ sau } \delta_r^d = o_r^d(1-o_r^d)(t_r^d - o_r^d), \text{ cu } r=1,2,\dots,R$$

Se modifică ponderile între nodurile de pe stratul ascuns și stratul de ieșire

$$w_{hr}^2 = w_{hr}^2 + \eta \delta_r^d o_h^d, \text{ unde } h=1,2,\dots,H \text{ și } r=1,2,\dots,R$$

Se propagă erorile nodurilor de pe stratul de ieșire înapoi în toată rețeaua → se distribuie erorile pe toate conexiunile existente în rețea proporțional cu valorile ponderilor asociate acestor conexiuni

$$\delta_h^d = \sum_{r=1}^R w_{hr}^2 \delta_r^d \text{ sau } \delta_h^d = o_h^d(1-o_h^d) \sum_{r=1}^R w_{hr}^2 \delta_r^d$$

Se modifică ponderile între nodurile de pe stratul de intrare și stratul ascuns

$$w_{ih}^1 = w_{ih}^1 + \eta \delta_h^d x_i^d, \text{ unde } i=1,2,\dots,m \text{ și } h=1,2,\dots,H$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață o întreagă RNA?
 - Algoritmul backpropagation pentru RNA_p

Se inițializează ponderile $w_{ih_1}^1, w_{h_1 h_2}^2, \dots, w_{h_{p-1} h_p}^p, w_{h_p r}^{p+1}$

Cât timp nu este îndeplinită condiția de oprire

Pentru fiecare exemplu (x^d, t^d)

Se activează fiecare neuron al rețelei

Se propagă informația înainte și se calculează ieșirea corespunzătoare fiecărui neuron al rețelei

$$o_{h_1}^d = \sum_{i=1}^m w_{ih_1}^1 x_i^d \text{ sau } o_{h_1}^d = \text{sig}\left(\sum_{i=1}^m w_{ih_1}^1 x_i^d\right), \text{ cu } h_1 = 1, 2, \dots, H_1$$

$$o_{h_2}^d = \sum_{h_1=1}^{H_1} w_{h_1 h_2}^2 o_{h_1}^d \text{ sau } o_{h_2}^d = \text{sig}\left(\sum_{h_1=1}^{H_1} w_{h_1 h_2}^2 o_{h_1}^d\right), \text{ cu } h_2 = 1, 2, \dots, H_2$$

...

$$o_{h_p}^d = \sum_{h_{p-1}=1}^{H_{p-1}} w_{h_{p-1} h_p}^p o_{h_{p-1}}^d \text{ sau } o_{h_p}^d = \text{sig}\left(\sum_{h_{p-1}=1}^{H_{p-1}} w_{h_{p-1} h_p}^p o_{h_{p-1}}^d\right), \text{ cu } h_p = 1, 2, \dots, H_p$$

$$o_r^d = \sum_{h_p=1}^{H_p} w_{h_p r}^{p+1} o_{h_p}^d \text{ sau } o_r^d = \text{sig}\left(\sum_{h_p=1}^{H_p} w_{h_p r}^{p+1} o_{h_p}^d\right), \text{ cu } r = 1, 2, \dots, R$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață o întreagă RNA?
 - Algoritmul backpropagation pentru RNA_p

Se inițializează ponderile $w_{ih_1}^1, w_{h_1 h_2}^2, \dots, w_{h_{p-1} h_p}^p, w_{h_p r}^{p+1}$

Cât timp nu este îndeplinită condiția de oprire

Pentru fiecare exemplu (x^d, t^d)

Se activează fiecare neuron al rețelei

Se ajustează ponderile

Se stabilește și se propagă eroarea înapoi

Se stabilesc erorile corespunzătoare neuronilor din stratul de ieșire

$$\delta_r^d = t_r^d - o_r^d \text{ sau } \delta_r^d = o_r^d(1 - o_r^d)(t_r^d - o_r^d), \text{ cu } r = 1, 2, \dots, R$$

Se modifică ponderile între nodurile de pe ultimul strat ascuns și stratul de ieșire

$$w_{h_p r}^{p+1} = w_{h_p r}^{p+1} + \eta \delta_r^d o_{h_p}^d, \text{ unde } h_p = 1, 2, \dots, H_p \text{ și } r = 1, 2, \dots, R$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învăță o întreagă RNA?
 - Algoritmul backpropagation pentru RNA_p

Se inițializează ponderile $w_{ih_1}^1, w_{h_1 h_2}^2, \dots, w_{h_{p-1} h_p}^p, w_{h_p r}^{p+1}$

Cât timp nu este îndeplinită condiția de oprire

Pentru fiecare exemplu (x^d, t^d)

Se activează fiecare neuron al rețelei

Se ajustează ponderile

Se stabilește și se propagă eroarea înapoi

Se stabilesc erorile corespunzătoare neuronilor din stratul de ieșire

Se modifică ponderile între nodurile de pe ultimul strat ascuns și stratul de ieșire

Se propagă (pe starturi) aceste erori înapoi în toată rețea → se distribuie erorile pe toate conexiunile existente în rețea proporțional cu valorile ponderilor asociate acestor conexiuni și se modifică ponderile corespunzătoare

$$\delta_{h_p}^d = \sum_{r=1}^R w_{h_p r}^{p+1} \delta_r^d \text{ sau } \delta_{h_p}^d = o_{h_p}^d (1 - o_{h_p}^d) \sum_{r=1}^R w_{h_p r}^{p+1} \delta_r^d$$

$$w_{h_p r}^{p+1} = w_{h_p r}^{p+1} + \eta \delta_r^d o_{h_p}^d, \text{ unde } h_p = 1, 2, \dots, H_p \text{ și } r = 1, 2, \dots, R$$

$$\delta_{h_{p-1}}^d = \sum_{h_p=1}^{H_p} w_{h_{p-1} h_p}^p \delta_{h_p}^d \text{ sau } \delta_{h_{p-1}}^d = o_{h_{p-1}}^d (1 - o_{h_{p-1}}^d) \sum_{h_p=1}^{H_p} w_{h_{p-1} h_p}^p \delta_{h_p}^d$$

$$w_{h_{p-1} h_p}^p = w_{h_{p-1} h_p}^p + \eta \delta_{h_p}^d o_{h_{p-1}}^d, \text{ unde } h_{p-1} = 1, 2, \dots, H_{p-1} \text{ și } h_p = 1, 2, \dots, H_p$$

...

$$\delta_{h_1}^d = \sum_{h_2=1}^{H_2} w_{h_1 h_2}^2 \delta_{h_2}^d \text{ sau } \delta_{h_1}^d = o_{h_1}^d (1 - o_{h_1}^d) \sum_{h_2=1}^{H_2} w_{h_1 h_2}^2 \delta_{h_2}^d$$

$$w_{i h_i}^1 = w_{i h_i}^1 + \eta \delta_{h_i}^d x_i^d, \text{ unde } i = 1, 2, \dots, m \text{ și } h_i = 1, 2, \dots, H_1$$

Sisteme inteligente – SIS – RNA

□ Cum se masoara calitatea algoritmului de invatare?

■ Probleme de regresie

- Stratul de iesire are un nr de neuroni egal cu nr de variabile care trebuie prezise
- Identificarea erorii: pentru fiecare neuron de output
 - Suma diferenelor intre valorile prezise si cele reale
 - Diferentele in valoare absoluta (L1, MAE)
 - Robustete la valori extreme (outliers)
 - Dar, gradienti mari \leftrightarrow eroare mica
 - Solutii: rata de invatare dinamica sau MSE
 - Patratul diferenelor (L2, MSE)
 - Stabilitatea solutiilor
 - Dar sensibilitate la valori extreme
 - Huber

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for } |y - f(x)| \leq \delta, \\ \delta |y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

- Pentru
 - Un exemplu din date
 - Pentru mai multe exemple din date (batch)
- Functia de cost (loss) = eroarea

Sisteme inteligente – SIS – RNA

□ Cum se masoara calitatea algoritmului de invatare?

■ Probleme de clasificare – clasificarea fructelor in imagini

□ Binara

examples labels

	fruit
	non-fruit
	non-fruit
	fruit

examples labels

	apple
	pear
	grappe
	pear

□ Multi-clasa

examples labels
[apple? pear? grappes?]



[1 1 0]



[1 0 1]



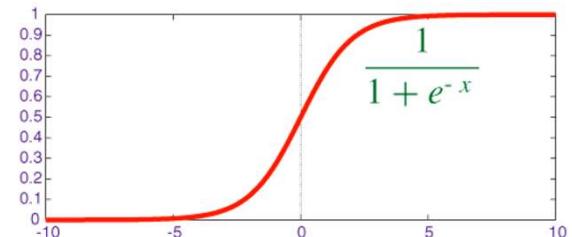
[1 1 1]

Sisteme inteligente – SIS – RNA

examples	labels
	fruit
	non-fruit
	non-fruit
	fruit

- Cum se masoara calitatea algoritmului de invatare? → Problem clasificare – clasificarea fructelor in imagini → Binara

- Stratul de iesire are un singur neuron cu functie de activare sigmoid
 - Outputul neuronului este transformat prin sigmoid → o valoare intre 0 si 1 → probabilitatea apartenentei la una din cele 2 clase (→ acuratetea, precizia, rapelul)



- Functia de cost != acuratete (precizie, recall)
 - Functia de cost = functia de loss = Cross-entropy loss (Logistic Loss or Multinomial Logistic Loss)
 - Cross Entropy pentru 2 distributii p si q
 - $CE(p, q) = -\sum(p_i \log_2(q_i))$, $i = 1, 2, \dots$
 - Cross-entropy pt clasificare binara
 - Pentru fiecare exemplu din date
 - Se transforma fiecare eticheta reala intr-o lista cu 2 probabilitati → **p** [$1 - \text{eticheta_clasa}$, eticheta_clasa]
 - se transforma outputul sigmoidat al neuronului de iesire intr-o lista de 2 probabilitati → **q** [$1 - \text{out_sigm}$, out_sigm]
 - se calculeaza entropia conditionata intre **p** si **q** : $CE(\text{exemplul crt})$
 - Se calculeaza media entropiilor conditionate pt fiecare exemplu

Sisteme inteligente – SIS – RNA

examples	labels
	apple
	pear
	grappe
	pear

- Cum se masoara calitatea algoritmului de invatare? → Probleme de clasificare – clasificarea fructelor in imagini → multi-clasa

■ Abordarea one-vs-all

- O problema cu C-clase → C probleme de clasificare binara
 - Apple vs. others, Pear vs. others, Grape vs others
- Se antreneaza C modele de clasificare binara (neuronul de output are activare sigmoidala)
- La testare, se aplica toate cele C modele antrenate si se alege clasa indicata de cel cu outputul cel mai mare

■ Abordarea soft-max

- O retea cu C neuroni in stratul de iesire si activare softmax

$$out'(neuron_i) = \frac{e^{out(neuron_i)}}{\sum_{i=1}^C e^{out(neuron_i)}}$$

Sisteme inteligente – SIS – RNA

examples	labels
	apple
	pear
	grappe
	pear

- Cum se masoara calitatea algoritmului de invatare? → Probleme de clasificare – clasificarea fructelor in imagini → multi-clasa → Abordarea soft-max

- Stratul de iesire are C neuroni (C – nr de clase)
 - Outputurile neuronilor (s) se transforma prin softmax → o valoare intre 0 si 1 → probabilitatea apartenenței la una din cele C clase (→ acuratetea, precizia, rapelul)

$$q = \left[\frac{e^{s_1}}{\sum_{j=1}^C e^{s_j}}, \frac{e^{s_2}}{\sum_{j=1}^C e^{s_j}}, \dots, \frac{e^{s_C}}{\sum_{j=1}^C e^{s_j}} \right]$$

- Functia de cost != acuratete (precizie, recall)
 - Functia de cost = functia de loss = categorical cross-entropy loss (Softmax Loss)
 - Cross Entropy pentru 2 distributii p si q
 - $CE(p,q) = -\sum(p_i \log_2(q_i))$, $i = 1, 2, \dots$
 - Cross-entropy pt clasificare multi-clasa
 - Pentru fiecare exemplu din date
 - Se transforma fiecare eticheta reala intr-o lista cu C probabilitati, probabilitatea clasei corekte fiind 1, restul 0 (one-hot encoding) → \mathbf{p}
 $\mathbf{p} = [1, 0, 0], \mathbf{p} = [0, 1, 0], \mathbf{p} = [0, 0, 1]. \mathbf{p} = [0, 1, 0]$
 - Pentru fiecare exemplu din date, se transforma outputurile neuronilor cu softmax → \mathbf{q}
 - Se calculeaza entropia conditionata intre \mathbf{p} si \mathbf{q} : $CE(\text{exemplul crt})$
 - $CE = -\sum_{i=1}^C p_i \log(q_i) = -\log(q_{\text{correctClass}})$
 - Se calculeaza media entropiilor conditionate pt fiecare exemplu

Sisteme inteligente – SIS – RNA

examples	labels
	[1 1 0]
	[1 0 1]
	[1 1 1]

- Cum se masoara calitatea algoritmului de invatare? → Probleme de clasificare – clasificarea fructelor in imagini → multi-label
 - Stratul de iesire are C neuroni (C – nr de clase)
 - Outputurile neuronilor (s) se transforma prin sigmoid → o valoare intre 0 si 1 → probabilitatea apartenientei la una din cele C clase (→ acuratetea, precizia, rapelul)
 - Functia de cost != acuratete (precizie, recall)
 - Functia de cost = functia de loss = Binary cross entropy (sigmoid cross entropy) pt fiecare eticheta posibila
 - Cross Entropy pentru 2 distributii p si q
 - $CE(p,q) = -\sum(p_i \log_2(q_i))$, $i = 1, 2, \dots$
 - Cross-entropy pt clasificare multi-label
 - Pentru fiecare exemplu din date
 - Se transforma fiecare eticheta reala intr-o lista cu C probabilitati, probabilitatea clasei corecte fiind 1, restul 0 (one-hot encoding) → \mathbf{p}
 - $\mathbf{p} = [1, 0, 0], \mathbf{p} = [0, 1, 0], \mathbf{p} = [0, 0, 1]. \mathbf{p} = [0, 1, 0]$
 - Pentru fiecare exemplu din date, se transforma outputurile neuronilor cu sigmoid → \mathbf{q}
 - $qi = \text{sigm}(si)$, $i = 1, 2, \dots, C$, $\mathbf{q} = [qi, 1 - qi]$
 - Se calculeaza entropia conditionata intre \mathbf{p} si \mathbf{q} : $CE(\text{exemplul crt})$
 - $CE_i = -p_i \log(q_i) - (1 - p_i) \log(1 - q_i)$
 - $CE = \sum_{i=1}^C CE_i$
 - Se calculeaza media entropiilor conditionate pt fiecare exemplu

Sisteme inteligente – SIS – RNA

- Cum se masoara calitatea algoritmului de invatare? → Probleme de clasificare – clasificarea fructelor in imagini → multi-class & multi-label

- Cross-entropy

- Classic loss

$$CE = - \sum_{i=1}^C p_i \log(q_i)$$

- Focal loss

$$CE = - \sum_{i=1}^C (1 - q_i)^\gamma p_i \log(q_i)$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață o întreagă RNA?
 - Algoritmul backpropagation
 - Condiții de oprire
 - S-a ajuns la eroare 0
 - S-au efectuat un anumit număr de iterații
 - La o iterație se procesează un singur exemplu
 - n iterații = o epocă

Sisteme inteligente – SIS – RNA

□ Proiectare

- Construirea RNA pentru rezolvarea problemei P
- Inițializarea parametrilor RNA
- Antrenarea RNA
- **Testarea RNA**

Sisteme inteligente – SIS – RNA

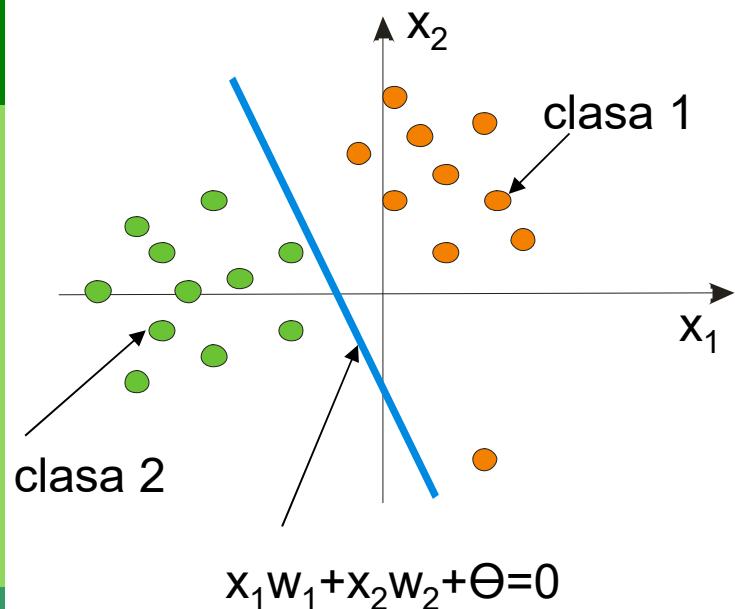
□ Proiectare

■ Testarea RNA

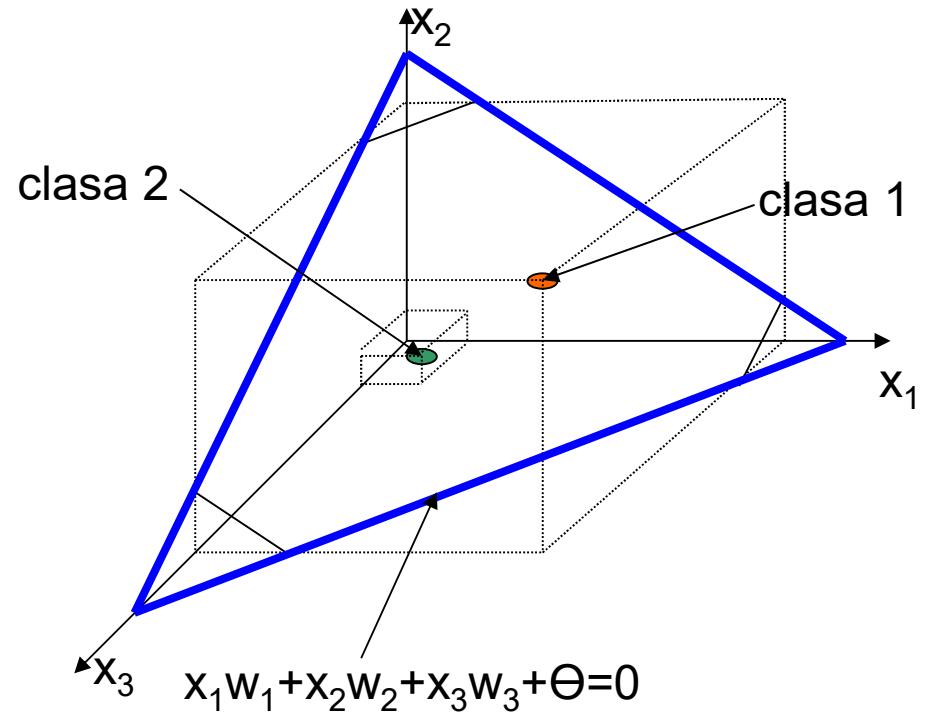
- Se decodifică modelul învățat de RNA
 - prin combinarea ponderilor cu intrările
 - ținând cont de funcțiile de activare a neuronilor și de structura rețelei

Sisteme inteligente – SIS – RNA

□ Exemplu



Clasificare binară cu m=2 intrări



Clasificare binară cu m=3 intrări

Sisteme inteligente – SIS – RNA

□ Exemplu

■ Perceptron pentru rezolvarea problemei *SI logic*

Epoch	Inputs		Desired output Y_d	Initial weights		Actual output Y	Error e	Final weights	
	x_1	x_2		w_1	w_2			w_1	w_2
1	0	0	0	0.3	-0.1	0	0	0.3	-0.1
	0	1	0	0.3	-0.1	0	0	0.3	-0.1
	1	0	0	0.3	-0.1	1	-1	0.2	-0.1
	1	1	1	0.2	-0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	-1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	-1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	-1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

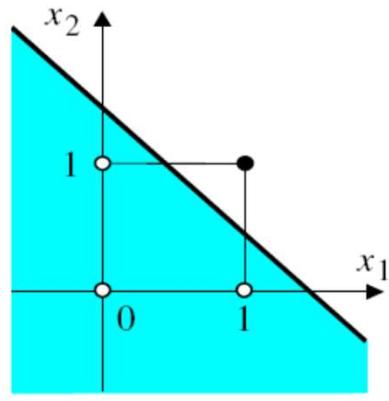
Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$

Sisteme inteligente – SIS – RNA

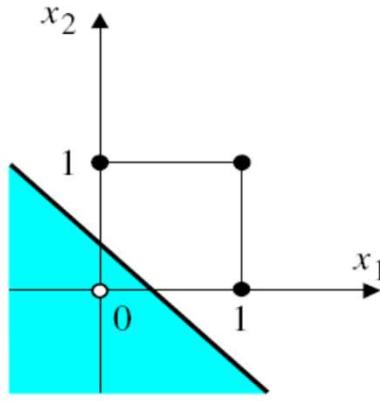
□ Exemplu

■ Perceptron - limitări

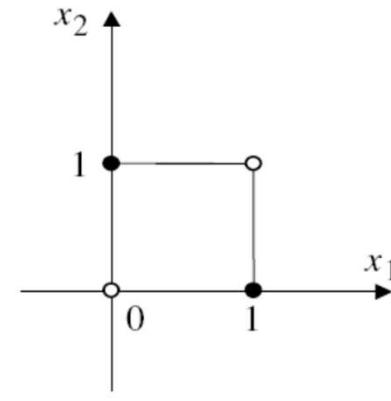
- Un perceptron poate învăța operațiile AND și OR, dar nu poate învăța operația XOR (nu e liniar separabilă)



(a) $AND (x_1 \cap x_2)$



(b) $OR (x_1 \cup x_2)$



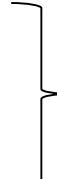
(c) $Exclusive-OR$
 $(x_1 \oplus x_2)$

- Nu poate clasifica date non-liniar separabile

- soluții
 - Neuron cu un prag continu
 - Mai mulți neuroni

Sisteme inteligente – SIS – RNA

□ Tipologie

- RNA feed-forward
 - Informația se procesează și circulă de pe un strat pe altul
 - Conexiunile între noduri nu formează cicluri
 - Se folosesc în special pentru învățarea supervizată
 - Funcțiile de activare a nodurilor → liniare, sigmoidale, gaussiene
- RNA recurente (cu feedback)
 - Pot conține conexiuni între noduri de pe același strat
 - Conexiunile între noduri pot forma cicluri
 - RNA de tip Jordan
 - RNA de tip Elman
 - RNA de tip Hopfield
 - RNA auto-organizate → pentru învățarea nesupervizată
 - De tip Hebbian
 - De tip Kohonen (*Self organised maps*)

} pentru învățarea supervizată

Sisteme inteligente – SIS – RNA

□ Avantaje

- Pot rezolva atât probleme de învățare super-vizată, cât și nesupervizată
- Pot identifica relații dinamice și neliniare între date
- Pot rezolva probleme de clasificare cu oricâte clase (multi-clasă)
- Se pot efectua calcule foarte rapid (în paralel și distribuit)

□ Dificultăți și limite

- RNA se confruntă cu problema overfitting-ului chiar și când modelul se învăță prin validare încrucișată
- RNA pot găsi (uneori) doar optimele locale (fără să identifice optimul global)



Recapitulare

□ Sisteme care învață singure (SIS)

■ Rețele neuronale artificiale

- Modele computaționale inspirate de rețelele neuronale artificiale
- Grafe speciale cu noduri așezate pe straturi
 - Strat de intrare → citește datele de intrare ale problemei de rezolvat
 - Strat de ieșire → furnizează rezultate problemei date
 - Strat(uri) ascunse → efectuează calcule
- Nodurile (neuronii)
 - Au intrări ponderate
 - Au funcții de activare (liniare, sigmoidale, etc)
 - necesită antrenare → prin algoritmi precum:
 - Perceptron
 - Scădere după gradient
- Algoritm de antrenare a întregii RNA → Backpropagation
 - Informația utilă se propagă înainte
 - Eroarea se propagă înapoi

Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor – Materiale de citit și legături utile

- capitolul 15 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*
- Documentele din directorul *12_svm* și *13_GP*

-
- Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean –
www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan -
www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop -
www.cs.ubbcluj.ro/~hfpop

INTELIGENȚĂ ARTIFICIALĂ



Sisteme inteligente

Sisteme care învață singure
– kNN și programare genetică –

Laura Dioșan

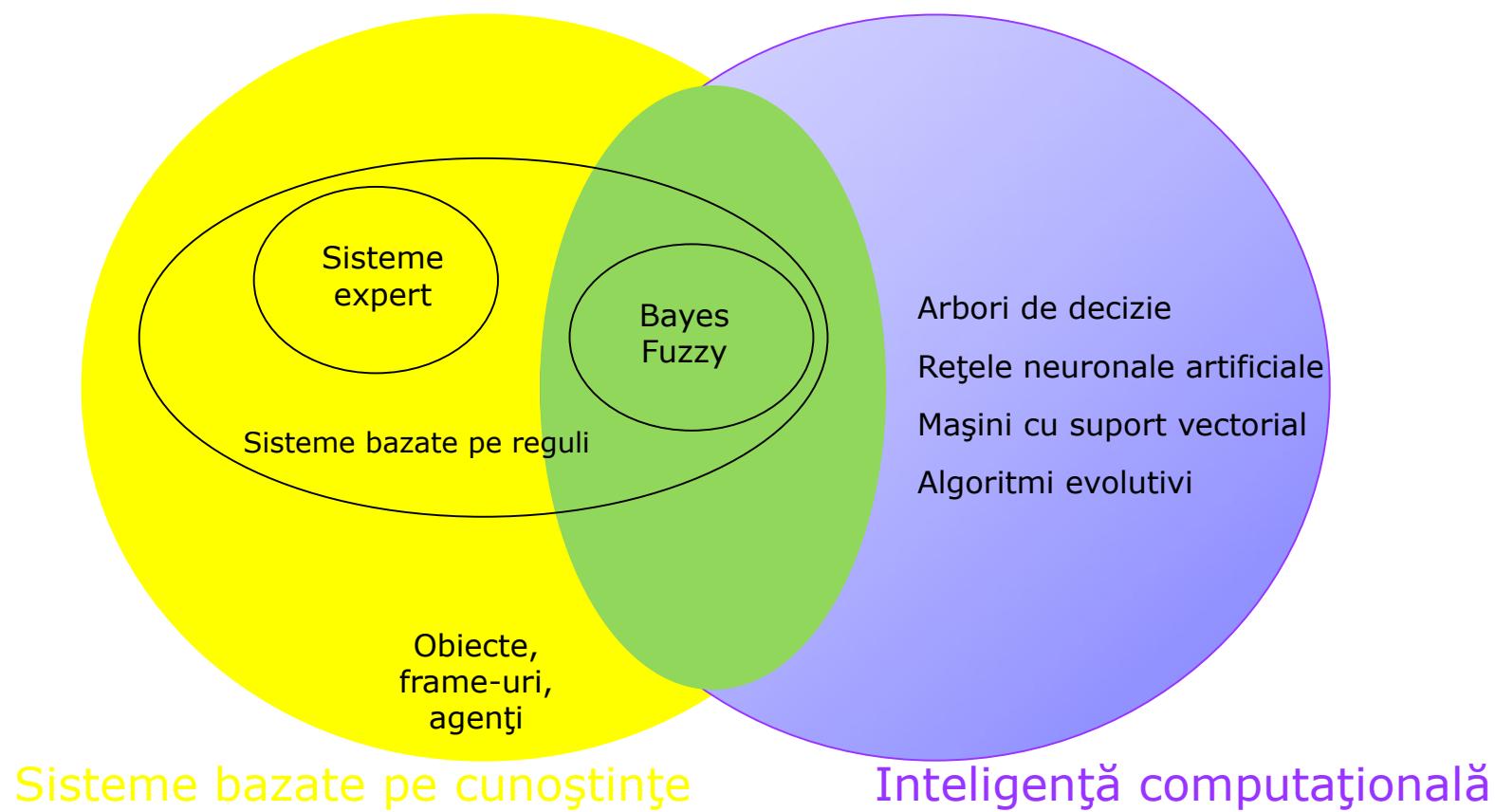
Sumar

- A. Scurtă introducere în Inteligența Artificială (IA)**
- B. Rezolvarea problemelor prin căutare**
 - Definirea problemelor de căutare
 - Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversarială
- C. Sisteme inteligente**
 - Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - kNN
 - Algoritmi evolutivi
 - Mașini cu suport vectorial
 - Sisteme bazate pe reguli
 - Sisteme hibride

Materiale de citit și legături utile

- capitolul 15 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*
- capitolul VI (18) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 10 și 11 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- capitolul 3 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Sisteme inteligente



Sisteme inteligente – SIS – Învățare automată

□ Tipologie

- În funcție de experiența acumulată în timpul învățării:
 - SI cu învățare supervizată
 - SI cu învățare nesupervizată
 - SI cu învățare activă
 - SI cu învățare cu întărire
- În funcție de modelul învățat (algoritmul de învățare):
 - Rețele neuronale artificiale
 - **Mașini cu suport vectorial (MSV)**
 - **Algoritmi evolutivi**
 - **kNN**
 - Arbori de decizie
 - Modele Markov ascunse

Materiale de citit și legături utile

- capitolul VI (18) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 10 și 11 din *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- capitolul 3 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Sisteme inteligente – SIS – Învățare automată

□ Tipologie

- În funcție de experiența acumulată în timpul învățării:
 - SI cu învățare supervizată
 - SI cu învățare nesupervizată
 - SI cu învățare activă
 - SI cu învățare cu întărire

- În funcție de modelul învățat (algoritmul de învățare):
 - Arbori de decizie
 - Rețele neuronale artificiale
 - **Mașini cu suport vectorial (MSV)**
 - Algoritmi evolutivi
 - Modele Markov ascunse

Sisteme inteligente – SIS – MSV

□ Mașini cu suport vectorial (MSV)

- Definire
- Tipuri de probleme rezolvabile
- Avantaje
- Dificultăți
- Tool-uri

Sisteme inteligente – SIS – MSV

□ Definire

- Dezvoltate de Vapnik în 1970
- Popularizate după 1992
- Clasificatori liniari care identifică un hiperplan de separare a clasei pozitive de clasa negativă
- Au o fundamentare teoretică foarte riguroasă
- Funcționează foarte bine pentru date de volum mare (analiza textelor, analiza imaginilor)

■ **Reamintim**

- Problemă de învățare supervizată în care avem un set de date de forma:
 - (x^d, t^d) , cu:
 - $x^d \in \mathbb{R}^m \rightarrow x^d = (x_1^d, x_2^d, \dots, x_m^d)$
 - $t^d \in \mathbb{R} \rightarrow t^d \in \{1, -1\}$, 1 → clasă pozitivă, -1 → clasă negativă
 - cu $d = 1, 2, \dots, n, n+1, n+2, \dots, N$
- Primele n date (se cunosc x^d și t^d) vor fi folosite drept bază de antrenament a MSV
- Ultimele $N-n$ date (se cunosc doar x^d , fără t^d) vor fi folosite drept bază de testare a MSV

Sisteme inteligente – SIS – MSV

□ Definire

- MSV găsește o funcție liniară de forma $f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b$, (\mathbf{w} -vector pondere) a.î.

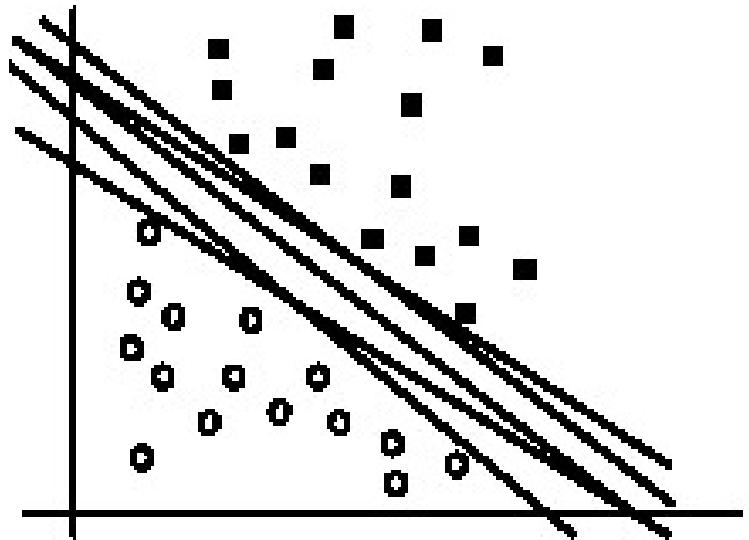
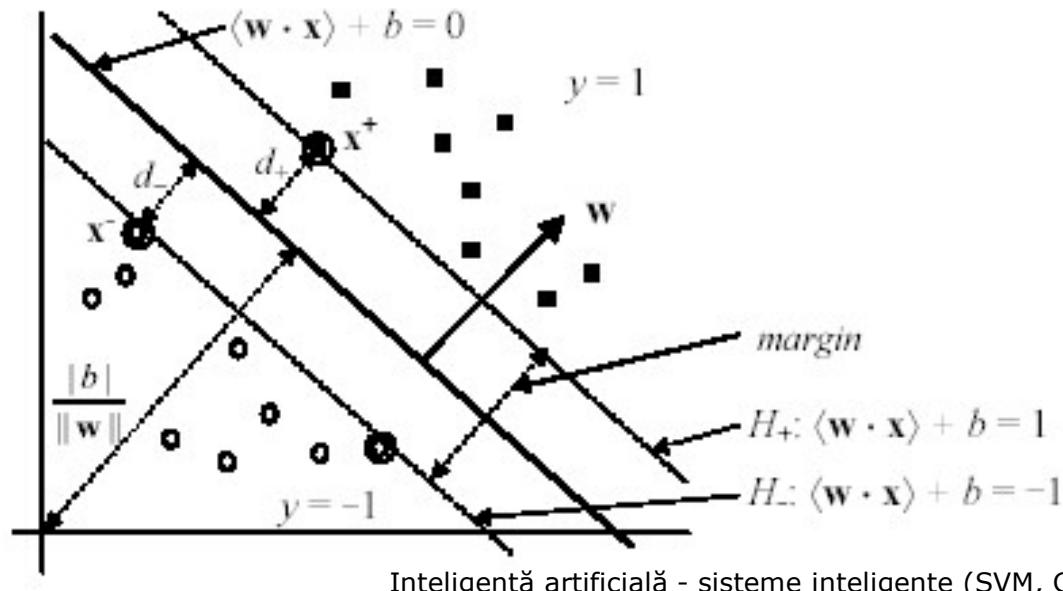
$$y_i = \begin{cases} 1 & \text{if } \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 0 \\ -1 & \text{if } \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b < 0 \end{cases}$$

- $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0 \rightarrow$ hiperplanul de decizie care separă cele 2 clase

Sisteme inteligente – SIS – MSV

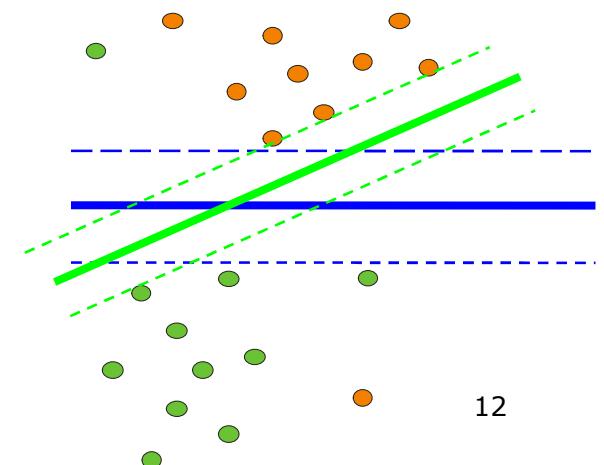
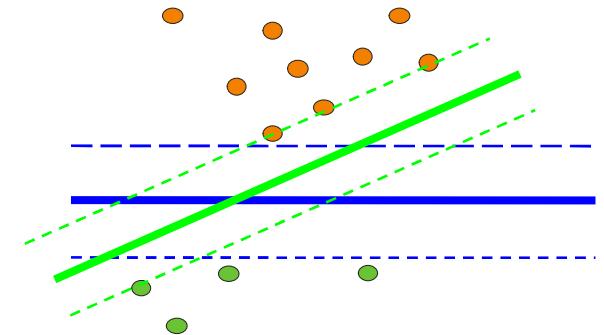
□ Definire

- Pot exista mai multe hiperplane
 - Care este cel mai bun hiperplan?
- MSV caută hiperplanul cu cea mai largă margine (cel care micșorează eroarea de generalizare)
 - Algoritmul SMO (*Sequential minimal optimization*)



Sisteme inteligente – SIS – MSV

- Tipuri de probleme rezolvabile
 - Probleme de clasificare → Cazuri de date
 - Liniar separabile
 - Separabile
 - Eroarea = 0
 - Ne-separabile
 - Se relaxează constrângerile → se permit unele erori
 - C – coeficient de penalizare

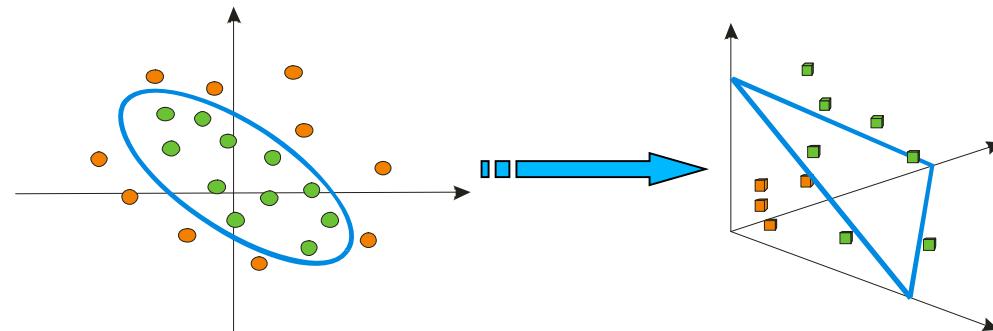


Sisteme inteligente – SIS – MSV

Cazuri de date

□ Non-liniar separabile

- Spațiul de intrare se transformă într-un spațiu cu mai multe dimensiuni (*feature space*), cu ajutorul unei funcții kernel, unde datele devin liniar separabile



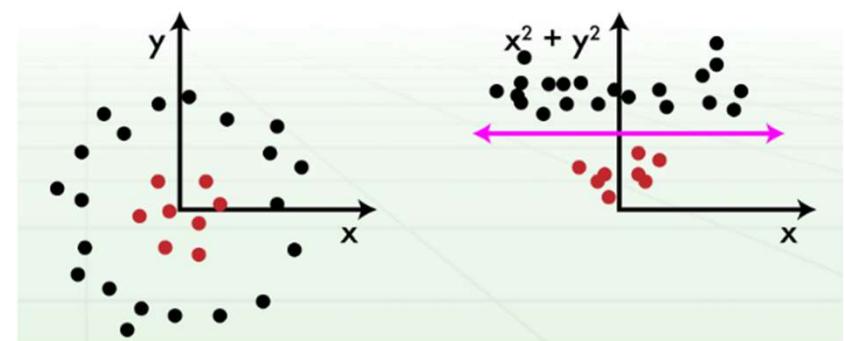
■ Kernele posibile

□ Clasice

- Polynomial kernel: $K(\mathbf{x}_1, \mathbf{x}_2) = (\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + 1)^d$
- RBF kernel: $K(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\sigma |\mathbf{x}_1 - \mathbf{x}_2|^2)$

□ Kernele multiple

- Liniare: $K(\mathbf{x}_1, \mathbf{x}_2) = \sum w_i K_i$
- Ne-liniare
 - Fără coeficienți: $K(\mathbf{x}_1, \mathbf{x}_2) = K_1 + K_2 * \exp(K_3)$
 - Cu coeficienți: $K(\mathbf{x}_1, \mathbf{x}_2) = K_1 + c_1 * K_2 * \exp(c_2 + K_3)$



Sisteme inteligente – SIS – MSV

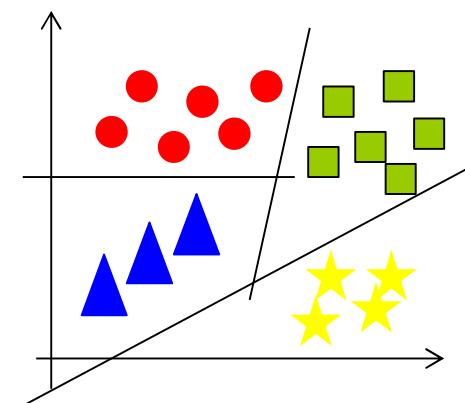
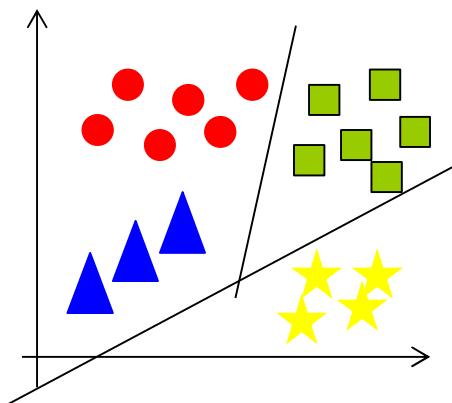
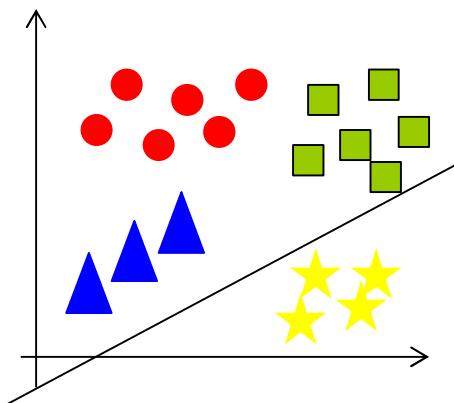
□ Configurarea MSV

■ Parametrii unei MSV

- Coeficientul de penalizare C
 - C – mic \rightarrow convergență lentă
 - C – mare \rightarrow convergență rapidă
- Parametrii funcției kernel (care kernel și cu ce parametri)
 - Dacă m (nr de atribute) este mult mai mare decât n (nr de instanțe)
 - MSV cu kernel liniar (MSV fără kernel) $\rightarrow K(\mathbf{x}^{d1}, \mathbf{x}^{d2}) = \mathbf{x}^{d1} \cdot \mathbf{x}^{d2}$
 - Dacă m (nr de atribute) este mare, iar n (nr de instanțe) este mediu
 - MSV cu kernel Gaussian $K(\mathbf{x}^{d1}, \mathbf{x}^{d2}) = \exp(-||\mathbf{x}^{d1} - \mathbf{x}^{d2}||^2 / 2\sigma^2)$
 - σ – dispersia datelor de antrenament
 - Atributele instanțelor trebuie normalize (scalate la $(0,1)$)
 - m (nr de atribute) este mic, iar n (nr de instanțe) este mare
 - Se adaugă noi atribute, iar apoi
 - MSV cu kernel liniar

Sisteme inteligente – SIS – MSV

- MSV pentru probleme de clasificare supervizate cu mai mult de 2 clase
 - Una vs. restul (one vs. all)



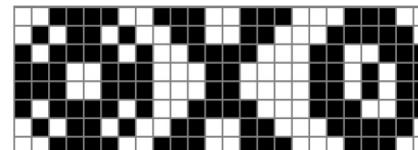
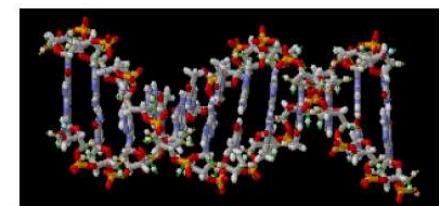
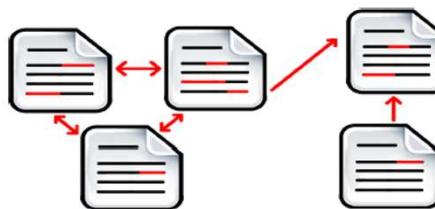
Sisteme inteligente – SIS – MSV

□ MSV structurate

- Învățare automată

- Normală $f: \mathcal{X} \rightarrow \mathbb{R}$
 - Intrări de orice fel
 - Ieșiri numerice (naturale, întregi, reale)
- Structurată: $\mathcal{X} \rightarrow \mathcal{Y}$
 - Intrări de orice fel
 - Ieșiri de orice fel (simple sau structurate)

- Informații structurate
 - Texte și hiper-texte
 - Molecule și structuri moleculare
 - Imagini



Sisteme inteligente – SIS – MSV

□ MSV structurate

■ Aplicații

- Procesarea limbajului natural
 - Trăduceri automate (ieșiri → propoziții)
 - Analiza sintactică și/sau morfologică a propozițiilor (ieșiri → arborele sintactic și/sau morfologic)
- Bioinformatică
 - Predicția unor structuri secundare (ieșirile → grafe bi-partite)
 - Predicția funcționării unor enzime (ieșirile → *path*-uri în arbori)
- Procesarea vorbirii
 - Transcrieri automate (ieșiri → propoziții)
 - Transformarea textelor în voce (ieșiri → semnale audio)
- Robotică
 - Planificare (ieșirile → secvențe de acțiuni)

Sisteme inteligente – SIS – MSV

□ Avantaje

- Pot lucra cu orice fel de date (liniar separabile sau nu, distribuit uniform sau nu, cu distribuție cunoscută sau nu)
 - Funcțiile kernel care crează noi atrbute (features) → straturile ascunse dintr-o RNA
- Dacă problema e convexă oferă o soluție unică → optimul global
 - RNA pot asocia mai multe soluții → optime locale
- Selectează automat mărimea modelului învățat (prin vectorii suport)
 - În RNA straturile ascunse trebuie configurate de către utilizator *apriori*
- Nu învață pe derost datele (overfitting)
 - RNA se confruntă cu problema overfitting-ului chiar și cand modelul se învață prin validare încrucișată

□ Dificultăți

- Doar atrbute reale
- Doar clasificare binară
- Background matematic dificil

□ Tool-uri

- LibSVM → <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- Weka → SMO
- SVMLight → <http://svmlight.joachims.org/>
- SVM Torch → <http://www.torch.ch/>
- <http://www.support-vector-machines.org/>

Sisteme inteligente – SIS – Învățare automată

□ Programare genetică

- Definire
- Proiectare
- Avantaje
- Limite
- Versiuni

Sisteme inteligente – SIS – PG

Reamintim

- Învățare supervizată → problemă de regresie (Studiul legăturii între variabile)
 - Se dă un set de n date (exemple, instanțe, cazuri)
 - date de antrenament – sub forma unor perechi (atribute_data_i , ieșire_i), unde
 - $i = 1, n$ (n = nr datelor de antrenament)
 - $\text{atribute_data}_i = (\text{atr}_{i1}, \text{atr}_{i2}, \dots, \text{atr}_{im})$, m – nr atributelor (caracteristicilor, proprietăților) unei date
 - ieșire_i – un număr real
 - date de test
 - sub forma (atribute_data_i), $i = n+1, N$ ($N-n$ = nr datelor de test)
 - Să se determine
 - o funcție (necunoscută) care realizează corespondența atribut – ieșire pe datele de antrenament
 - Ieșirea (valoarea) asociată unei date (noi) de test folosind funcția învățată pe datele de antrenament
- Cum găsim forma (expresia) funcției?
 - Algoritmi evolutivi → Programare genetică

Sisteme inteligente – SIS – PG

Reamintim

□ Algoritmi evolutivi

- Inspirați din natură (biologie)
- Iterativi
- Bazați pe
 - populații de potențiale soluții
 - căutare aleatoare ghidată de
 - Operații de selecție naturală
 - Operații de încrucișare și mutație

■ Care procesează în paralel mai multe soluții

□ Metafora evolutivă

Evoluție naturală	Rezolvarea problemelor
Individ	Soluție potențială (candidat)
Populație	Mulțime de soluții
Cromozom	Codarea (reprezentarea) unei soluții
Genă	Parte a reprezentării
Fitness (măsură de adaptare)	Calitate
Încrucișare și mutație	- Operații de căutare
Mediu	Inteligenta artificială (SVM, GP, kNN, arbori de decizie) - Sisteme Inteligente Spațiul de căutare al problemei

Sisteme inteligente – SIS – PG

Reamintim

□ Algoritmi evolutivi

Initializare populație $P(0)$

Evaluare $P(0)$

$g := 0$; //generația

CâtTimp (not condiție_stop) execută

Repetă

Selectează 2 părinți p_1 și p_2 din $P(g)$

Încrucișare(p_1, p_2) $\Rightarrow o_1$ și o_2

Mutatie(o_1) $\Rightarrow o_1^*$

Mutatie(o_2) $\Rightarrow o_2^*$

Evaluare(o_1^*)

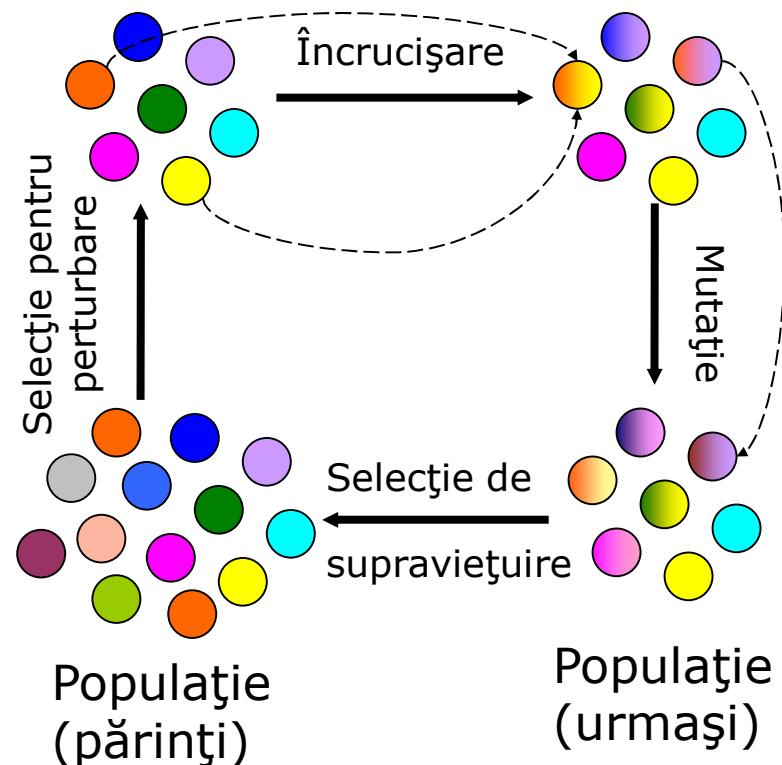
Evaluare(o_2^*)

adăugare o_1^* și o_2^* în $P(g+1)$

Până când $P(g+1)$ este completă

$g := g + 1$

Sf CâtTimp



Sisteme inteligente – SIS – PG

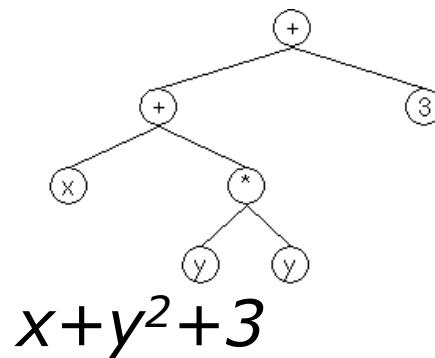
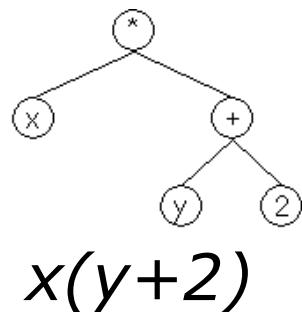
□ Definire

- Propusă de john Koza în 1988
- <http://www.genetic-programming.org/>
- Un tip particular de algoritmi evolutivi
- Cromozomi
 - sub formă de arbore care codează mici programe
- Fitness-ul unui cromozom
 - Performanța programului codat în el
- Scopul PG
 - Evoluarea de programe de calculator
 - AG evoluează doar soluții pentru probleme particulare

Sisteme inteligente – SIS – PG

□ Proiectare

- Reprezentarea cromozomilor
 - Foarte importantă, dar este o sarcină dificilă
 - Cromozomul = un arbore cu noduri de tip
 - Funcție → operatori matematici ($+, -, *, /, \sin, \log, \text{if}, \dots$)
 - Terminal → atrbute ale datelor problemei sau constante (x, y, z, a, b, c, \dots)
 - care codează expresia matematică a unui program (problema regresiei → a unei funcții)



Sisteme inteligente – SIS – PG

□ Proiectare

■ Fitness

- Eroarea de predicție – diferența între ceea ce dorim să obținem și ceea ce obținem de fapt

- pp o problemă de regresie cu următoarele date de intrare (2 atribute și o ieșire) și 2 cromozomi:

- $c_1 = 3x_1 - x_2 + 5 = (((3, x1, *), x2, -), 5, +)$

- $c_2 = 3x_1 + 2x_2 + 2$ $f^*(x_1, x_2) = 3x_1 + 2x_2 + 1$ – necunoscută

x_1	x_2	$f^*(x_1, x_2)$	$f_1(x_1, x_2)$	$f_2(x_1, x_2)$	$ f^* - f_1 $	$ f^* - f_2 $
1	1	6	7	7	1	1
0	1	3	4	4	1	1
1	0	4	8	5	4	1
-1	1	0	1	1	1	1
					$\Sigma = 7$	$\Sigma = 4$

→ c_2 e mai bun ca c_1

Sisteme inteligente – SIS – PG

□ Proiectare

■ Fitness

- Eroarea de predicție – diferența între ceea ce dorim să obținem și ceea ce obținem de fapt
- pp o problemă de clasificare cu următoarele date de intrare (2 atribute și o ieșire) și 2 cromozomi:
 - $c_1 = 3x_1 - x_2 + 5 \quad f1(1,1) = 7$ -: $\text{sigm}(7) - (0,1)$ -: Theta
 - $c_2 = 3x_1 + 2x_2 + 2$

x_1	x_2	$f^*(x_1, x_2)$	$f_1(x_1, x_2)$	$f_2(x_1, x_2)$	$ f^* - f_1 $	$ f^* - f_2 $
1	1	Yes	Yes	Yes	0	0
0	1	No	Yes	No	1	0
1	0	Yes	No	No	1	1
-1	1	Yes	No	yes	1	0
					$\Sigma = 3$	$\Sigma = 1$

→ c_2 e mai bun ca c_1

Sisteme inteligente – SIS – PG

□ Proiectare

■ Inițializarea cromozomilor

- Generare aleatoare de arbori corecți → programe valide (expresii matematice valide)
- Se stabilește o adâncime maximă a arborilor D_{max}
- 3 metode de inițializare
 - *Full* → fiecare ramură a rădăcinii are adâncimea D_{max}
 - Nodurile aflate la o adâncime $d < D_{max}$ se inițializează cu una dintre funcțiile din F
 - Nodurile aflate la o adâncime $d = D_{max}$ se inițializează cu unul dintre terminalele din T
 - *Grow* → fiecare ramură a rădăcinii are o adâncime $< D_{max}$
 - Nodurile aflate la o adâncime $d < D_{max}$ se inițializează cu un element din $F \cup T$
 - Nodurile aflate la o adâncime $d = D_{max}$ se inițializează cu unul dintre terminalele din T
 - *Ramped half and half* → $\frac{1}{2}$ din populația de cromozomi se inițializează folosind metoda *full*, $\frac{1}{2}$ din populația de cromozomi se inițializează folosind metoda *grow*

Sisteme inteligente – SIS – PG

□ Proiectare

- Operatori genetici → Selecția pentru recombinare
 - similar oricărui algoritm evolutiv
 - recomandare → selecție proporțională
 - over-selection → pentru populații f mari
 - Se ordonează populația pe baza fitness-ului și se împarte în 2 grupuri:
 - Grupul 1: cei mai buni x% cromozomi din populație
 - Grupul 2: restul de (100-x)% cromozomi din populație
 - Pentru populații cu 1000, 2000, 4000, 8000 de cromozomi, x este stabilit la 32%, 16%, 8%, respectiv 4%
 - 80% din operațiile de selecție vor alege cromozomi din grupul 1, 20% din grupul 2

Sisteme inteligente – SIS – PG

□ Proiectare

■ Operatori genetici → Selecția de supraviețuire

□ Scheme

- Generațională
- steady-state

□ Probleme

- *Bloat* → supraviețuirea celui mai “gras” individ (dimensiunea cromozomilor crește de-a lungul evoluției)
- Soluții
 - Interzicerea operatorilor de variație care produc descendenți prea mari
 - Presiunea economiei (zgârceniei) – penalizarea cromozomilor prea mari

Sisteme inteligente – SIS – PG

□ Proiectare

- Operatori genetici → Încrucișare și mutație
 - Parametri
 - O probabilitate p de alegere între încrucișare și mutație
 - $p = 0$ (cf. Koza) sau $p = 0.05$ (cf. Banzhaf)
 - O probabilitate p_c și respectiv p_m de stabilire a nodului care urmează a fi supus modificării
 - Dimensiunea descendenților diferă de dimensiune părinților

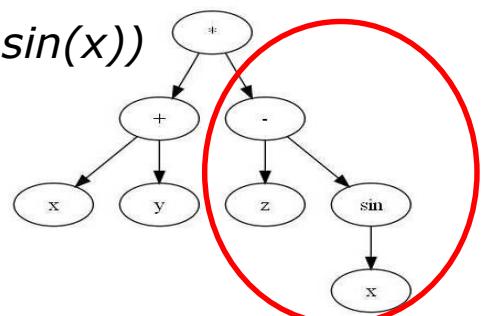
Sisteme inteligente – SIS – PG

□ Proiectare

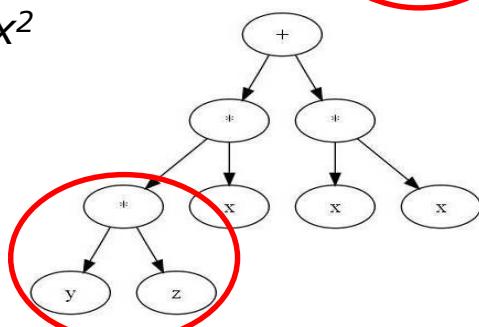
■ Operatori genetici → Încrucișare

- Cu punct de tăietură – se interchimbă doi sub-arbori
- Punctul de tăietură se generează aleator

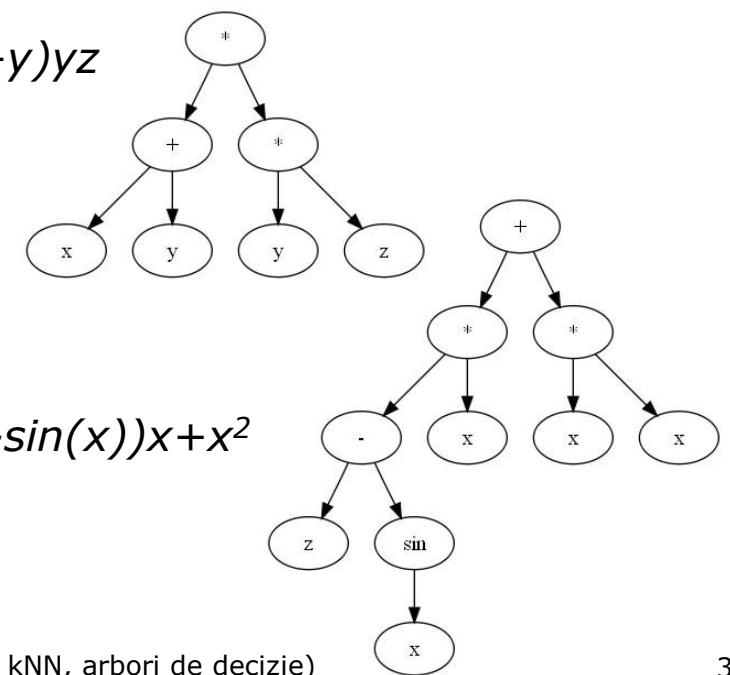
$$p_1 = (x+y)*(z-\sin(x))$$



$$p_2 = xyz + x^2$$



$$f_1 = (x+y)yz$$



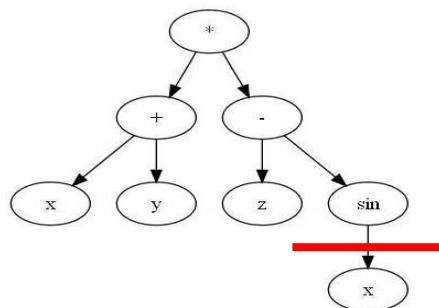
Sisteme inteligente – SIS – PG

□ Proiectare

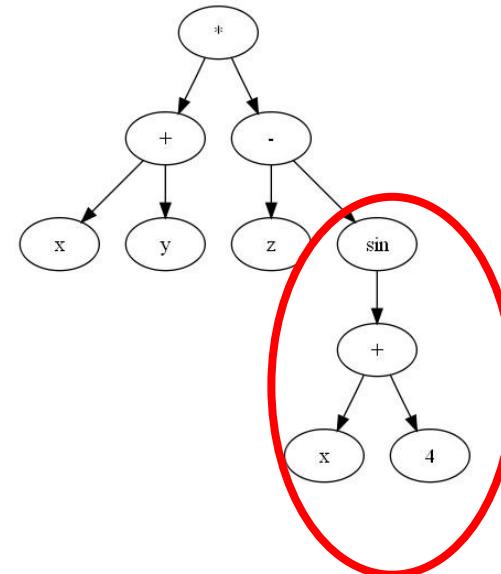
■ Operatori genetici → Mutăție

- Mutăție de tip *grow* → Înlocuirea unei frunze cu un nou sub-arbore

$$p = (x+y)*(z-\sin(x))$$



$$f = (x+y)*(z-\sin(x+4))$$



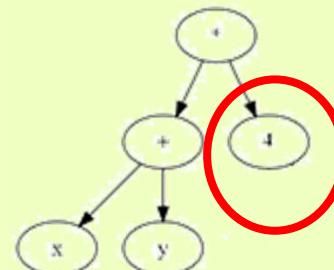
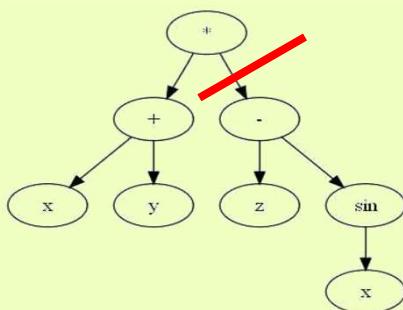
Sisteme inteligente – SIS – PG

□ Proiectare

- Operatori genetici → Mutăție
 - Mutăție de tip *shrink* → Înlocuirea unui sub-arbore cu o frunză

$$p = (x+y)*(z-\sin(x))$$

$$f = (x+y)*4$$



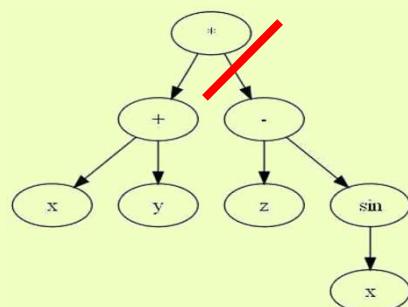
Sisteme inteligente – SIS – PG

□ Proiectare

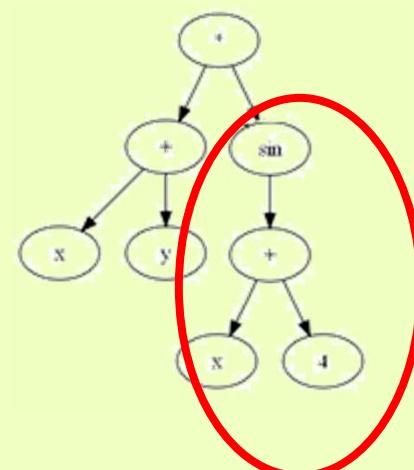
■ Operatori genetici → Mutăție

- Mutăție de tip Koza → Înlocuirea unui nod (intern sau frunză) cu un nou sub-arbore

$$p = (x+y)*(z-\sin(x))$$



$$f = (x+y)*\sin(x+4)$$



Sisteme inteligente – SIS – PG

□ Proiectare

- Operatori genetici → Mutătie
 - Mutătie de tip *switch*
 - selectarea unui nod intern și re-ordonarea subarborilor săi
 - Mutătie de tip *cycle*
 - selectarea unui nod și înlocuirea lui cu unul nou de același tip (intern – cu o funcție – sau frunză – cu un terminal)

Sisteme inteligente – SIS – PG

□ Comparație AG și PG

■ Forma cromozomilor

- AG – cromozomi liniari
- PG – cromozomi ne-liniari

■ Dimensiunea cromozomilor

- AG – fixă
- PG – variabilă (în adâncime sau lățime)

■ Schema de creare a descendenților

- AG – încrucișare și mutație
- PG – încrucișare sau mutație

Sisteme inteligente – SIS – PG

□ Avantaje

- PG găsește soluții problemelor care nu au o soluție optimă
 - Un program pentru conducerea mașinii → nu există o singură soluție
 - Unele soluții implică un condus sigur, dar lent
 - Alte soluții implică o viteză mare, dar un risc ridicat de accidente
 - Coducerea mașinii \leftrightarrow compromis între viteză mare și siguranță
- PG este utilă în problemele a căror variabile se modifică frecvent
 - Conducerea mașinii pe autostradă
 - Conducerea mașinii pe un drum forestier

□ Limite

- Timpul mare necesar evoluției pentru identificarea soluției

Sisteme inteligente – SIS – PG

□ Versiuni ale PG

- PG liniară (Cramer, Nordin)
- Gene Expression Programming (Ferreira)
- Multi Expression Programing (Oltean)
- Gramatical Evolution (Ryan, O'Neill)
- Cartesian Genetic Programming (Miller)

Sisteme inteligente – SIS – PG

□ PG liniară

- Evoluarea de programe scrise într-un limbaj imperativ (calculul fitness-ului nu necesită interpretare) → viteză mare de lucru
- Reprezentare
 - Vector de instrucțiuni, fiecare instrucțiune fiind de forma (pp ca aritatea maximală a unei funcții din F este n):
 - Index_op, registru_out, registru_in₁, registru_in₂, ..., registru_in_n
 - $v_i = v_j * v_k$ // instruction operating on two registers
 - $v_i = v_j * c$ // instruction operating on one register and one constant
 - $v_i = \sin(v_j)$ // instruction operating on one register

```
void LGP_program (double v[11])
{
    ...
    v[8] = v[0] - 10;
    v[6] = v[2] * v[0];
    v[5] = v[8] * 7;
    v[4] = v[2] - v[0];
    v[10] = v[1]/v[4];
    v[3] = sin(v[1]);
    v[1] = v[8] - v[6];
    v[7] = v[10] * v[3];
    v[9] = v[0] + v[7];
    v[2] = v[7] + 3;
    ...
}
```

```
void LGP_effective_program (double v[11])
{
    ...
    v[4] = v[2] - v[0];
    v[10] = v[1]/v[4];
    v[3] = sin(v[1]);
    v[7] = v[10] * v[3];
    v[9] = v[0] + v[7];
    ...
}
```

Sisteme inteligente – SIS – PG

□ PG liniară

■ Initializare

- Aleatoare
- Restricții
 - Lungimea inițială a cromozomului (nr de instrucțiuni)

■ Operatori genetici de variație

- Încrucișare – cu 2 puncte de tăietură
- Mutăție
 - Micro mutație → schimbarea unui operand sau operator (nu se modifică dimensiunea cromozomului)
 - Macro mutație → inserarea sau eliminarea unei instrucțiuni (se modifică dimensiunea cromozomului)

Sisteme inteligente – SIS – PG

□ PG liniară

- Avantaje
 - Evoluare într-un limbaj de nivel redus (low-level)
- Dezavantaje
 - Numărul de registri necesari (numărul de attribute ale problemei)
- Resurse
 - Register Machine Learning Technologies <http://www.aimlearning.com>
 - Peter Nordin's home page <http://fy.chalmers.se/~pnordin>
 - Wolfgang Banzhaf's home page <http://www.cs.mun.ca/~banzhaf>
 - Markus Brämeier's home page <http://www.daimi.au.dk/~brämeier>

Sisteme inteligente – SIS – PG

□ Gene Expression Programming (GEP)

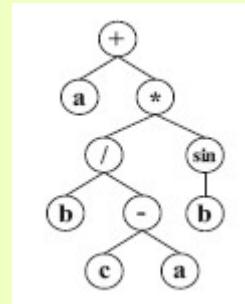
■ Ideea de bază

- Reprezentarea liniară a expresiilor codabile în arbori (prin parcuregerea în lățime a acestora – breadth-first)

$$C = +a * /Sb - bcacabbc$$

■ Reprezentare

- Un cromozom este format din mai multe gene
 - Legate între ele prin + sau *
- Fiecare genă este formată din:
 - Cap
 - conține h elemente → funcții și terminale
 - Coadă
 - conține doar terminale, în număr de $t = (n-1)*h+1$, unde n – aritatea maximă a unei funcții din F



Sisteme inteligente – SIS – PG

□ GEP

- Inițializare
 - Aleatoare, cu elemente din F și T conform regulilor precizate anterior
- Operatori de variație
 - Încrucișare
 - La nivel de alelă
 - Cu un punct de tăietură
 - Cu două puncte de tăietură
 - Încrucișare la nivel de genă
 - Cromozomii schimbă între ei anumite gene (plasate pe aceeași poziție)
 - Mutăție
 - La nivel de alelă
 - se modifică un element din cap sau coadă, respectând regulile de la inițializare
 - Transpoziții

Sisteme inteligente – SIS – PG

□ GEP

■ Avantaje

- Codarea în cromozomi a unor programe corecte datorită separării unei gene în cap și coadă

■ Dezavantaje

- Cromozomii multi-genă
 - Câte gene?
 - Cum se leagă genele între ele?

■ Resurse

- Gene Expression Programming website, <http://www.gepsoft.com>
- *Heitor Lopes's home page* <http://www.cpgei.cefetpr.br/~hslopes/index-english.html>
- *Xin Li's home page* <http://www.cs.uic.edu/~xli1>
- *GEP in C#* <http://www.c-sharpcorner.com/Code/2002/Nov/GEPAlgorithm.asp>

Sisteme inteligente – SIS – PG

□ Multi Expression Programming (MEP)

■ Ideea de bază

- Cromozomul este format din mai multe gene, fiecare genă → cod cu 3 adrese
 - Similar PG liniară, dar mai rapid

■ Reprezentare

- Liniară
- O genă conține o funcție (unară sau binară) și pointeri spre argumentele sale
- Cromozomul codează mai multe potențiale soluții → fiecare soluție corespunde unei gene
 - Calitatea unei soluții (gene) = suma (peste datele de antrenament) între ceea ce trebuia obținut și ceea ce se obține
 - Calitatea unui cromozom = fitness-ul celei mai bune gene

Sisteme inteligente – SIS – PG

□ MEP

■ Inițializare

- Prima genă trebuie să fie un terminal
- Restul genelor pot conține
 - un terminal sau
 - o funcție (unară sau binară) și pointeri spre argumentele sale
 - Argumentele unei funcții poziționată în a i-a genă trebuie să fie poziționate în cromozom la indici mai mici decât i

■ Operatori de variație

- Încrucișare → schimbarea unor gene între părinți
 - Cu un punct de tăietură
 - Cu două puncte de tăietură
 - Uniformă
- Mutăție → modificarea unei gene
 - Prima genă → generarea unui nou terminal
 - Restul genelor → generarea unui terminal sau a unei funcții (simbolul funcției și argumentele funcției)
 - Generarea are loc la fel ca la inițializare

Sisteme inteligente – SIS – PG

□ MEP

■ Avantaje

- Ieșire dinamică corespunzătoare unui cromozom
 - Complexitatea programului (expresiei) căutat(e)
 - Programe (expresii) de lungime variabilă obținute fără operatori speciali
 - Programe de lungime exponențială codate în cromozomi de lungime polinomială

■ Dezavantaje

- Complexitatea decodării pt date de antrenament necunoscute → evoluarea strategiilor de joc

■ Resurse

- Mihai Oltean's home page <http://www.cs.ubbcluj.ro/~>
- Crina Grosan's home page <http://www.cs.ubbcluj.ro/~cgrosan>
- MEP web page <http://www.mep.cs.ubbcluj.ro>
- MEP in C# <http://www.c-sharpcorner.com>

Sisteme inteligente – SIS – PG

□ Grammatical Evolution (GE)

- Ideea de bază
 - Evoluarea de programe în forma Backus-Naur (program exprimat sub forma unei gramici cu simboluri terminale și non-terminale, simbol de start, reguli/producții)
- Reprezentare
 - String binar de codons (grupuri de 8 biți) → care regulă a gramicii tb aplicată
 - Exemplu
 - $G = \{N, T, S, P\}$, $N = \{+, -, *, /, \sin, (,)\}$, $T = \{\text{expr}, \text{op2}, \text{op1}\}$, $S = \langle \text{expr} \rangle$, iar P este:
 - $\langle \text{expr} \rangle ::= a | b | c | \langle \text{expr} \rangle \langle \text{op2} \rangle \langle \text{expr} \rangle | (\langle \text{expr} \rangle \langle \text{op2} \rangle \langle \text{expr} \rangle) | \langle \text{op1} \rangle \langle \text{expr} \rangle$
 - $\langle \text{op2} \rangle ::= + | - | * | /$,
 - $\langle \text{op1} \rangle ::= \sin$
 - $C^*_{GE} = (9 \ 12 \ 12 \ 3 \ 15 \ 7 \ 11 \ 4 \ 2 \ 5 \ 0 \ 6 \ 11 \ 0 \ 1 \ 7 \ 12)$
 - $S = \langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle \langle \text{op2} \rangle \langle \text{expr} \rangle \rightarrow a \langle \text{op2} \rangle \langle \text{expr} \rangle \rightarrow a + \langle \text{expr} \rangle \rightarrow a + \langle \text{expr} \rangle \langle \text{op2} \rangle \langle \text{expr} \rangle \rightarrow a + b \langle \text{op2} \rangle \langle \text{expr} \rangle \langle \text{op2} \rangle \langle \text{expr} \rangle \rightarrow a + b / \langle \text{op2} \rangle \langle \text{expr} \rangle$

$$C_{GE} = (00001001 \ 00001100 \ 00001100 \ 00000011 \ 00001111 \ 00000111 \\ 00001011 \ 00000100 \ 00000010 \ 00000101 \ 00000000 \ 00000110 \\ 00001011 \ 00000000 \ 00000001 \ 00000111 \ 00001100)$$

$a + b / \langle \text{op2} \rangle \langle \text{expr} \rangle$
 $a + b / (\langle \text{expr} \rangle \langle \text{op2} \rangle \langle \text{expr} \rangle) \langle \text{op2} \rangle \langle \text{expr} \rangle$
 $a + b / (c \langle \text{op2} \rangle \langle \text{expr} \rangle) \langle \text{op2} \rangle \langle \text{expr} \rangle$
 $a + b / (c - \langle \text{expr} \rangle) \langle \text{op2} \rangle \langle \text{expr} \rangle$
 $a + b / (c - a) \langle \text{op2} \rangle \langle \text{expr} \rangle$
 $a + b / (c - a) * \langle \text{expr} \rangle$
 $a + b / (c - a) * \langle \text{op1} \rangle \langle \text{expr} \rangle$
 $a + b / (c - a) * \sin \langle \text{expr} \rangle$

$$E = a + b / (c - a) * \sin(b)$$

Sisteme inteligente – SIS – PG

□ GE

■ Inițializare

- Stringul binar este inițializat aleator cu 0 și 1 fără restricții → programe valide
- Decodarea se termină când s-a obținut un program complet
 - Dacă s-au terminat codons și încă nu s-a format tot programul, se reiau codons de la primul element → wrapping

■ Operatori de variație

- Încrucișare
 - Cu un punct de tăietură
- Mutăție
 - Schimbarea probabilistică a unui bit în opusul său
- Duplicare
 - O secvență de gene este copiată la sfârșitul cromozomului
- Pruning
 - Eliminarea genelor ne-folosite în procesul de transformare (decodare) a cromozomului

Sisteme inteligente – SIS – PG

□ GE

- Avantaje
 - Evoluarea de programe scrise în limbaje a căror instrucțiuni pot fi exprimate ca reguli de tip BNF
 - Reprezentarea poate fi schimbată prin modificarea gramaticii
- Dezavantaje
 - Wrapping-ul la infinit → limitarea repetărilor și penalizarea cromozomilor care depășesc un anumit prag de repetări
- Resurse
 - Grammatical Evolution web page, <http://www.grammatical-evolution.org>
 - Conor Ryan's home page, <http://www.csis.ul.ie/staff/conorryan>
 - Michael O'Neill's home page, <http://ncra.ucd.ie/members/oneillm.html>
 - John James Collins's home page, <http://www.csis.ul.ie/staff/jjcollins>
 - Maarten Keijzer's home page, <http://www.cs.vu.nl/~mkeijzer>
 - Anthony Brabazon's home page <http://ncra.ucd.ie/members;brabazont.html>

Sisteme inteligente – SIS – PG

□ Cartesian Genetic Programming (CGP)

■ Ideea de bază

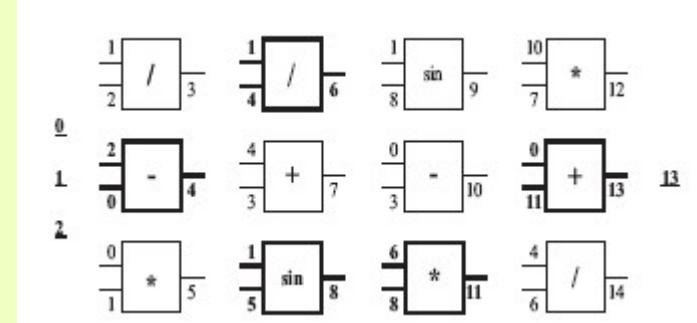
- Cromozomi sub formă de graf (matrice) → programe mai complexe decât cele din arbori

■ Reprezentare

- În sistem cartezian (matrice de noduri)

- Un nod are asociate
 - O funcție
 - Intrări
 - Ieșiri

- Ouputul cromozomului
 - Outputul oricărui nod



$$C = (1, 2, 3, \mathbf{2}, 0, 1, 0, 1, 2, 1, 4, 3, 4, 3, 0, 1, 5, 4, 1, 8, 4, 0, 3, 1, \mathbf{6}, \mathbf{8}, \mathbf{2}, 10, 7, 2, 0, 11, 0, 4, 6, 3, 13)$$

Sisteme inteligente – SIS – PG

□ CGP

- Inițializare
 - Aleatoare
 - Intrările oricărui nod trebuie să fie noduri de pe coloanele anterioare
 - Nodurile de pe prima coloană au ca intrări caracteristicile datelor de antrenament
- Operatori de variație
 - Încrucișare
 - Nu se aplică
 - Mutăție
 - Modificarea elementelor unui nod

Sisteme inteligente – SIS – PG

□ CGP

■ Avantaje

- Evoluarea indicelui nodului care furnizează ieșirea programului codat în cromozom
- Programul evoluat poate avea una sau mai multe ieșiri

■ Dezavantaje

- Stabilirea numărului de coloane influențează rezultatele obținute

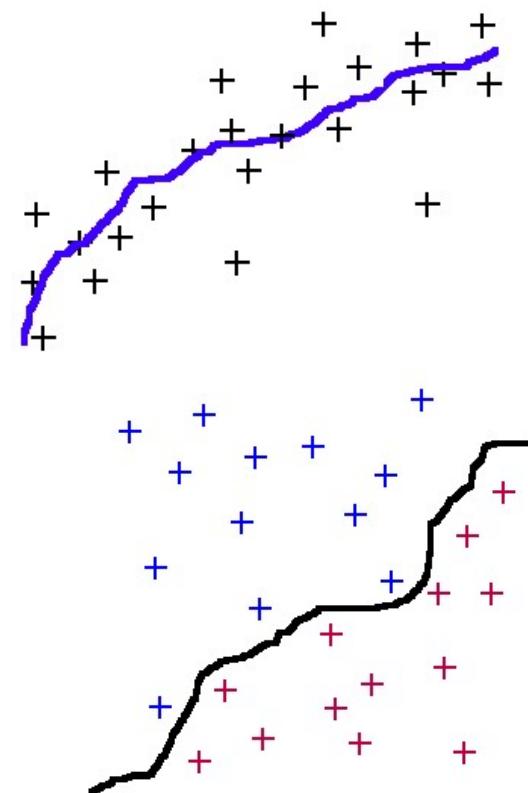
■ Resurse

- Julian. F. Miller's home page
<http://www.elec.york.ac.uk/intsys/users/jfm7>
- Lukás Sekanina's home page <http://www.fit.vutbr.cz/~sekanina/>

Sisteme inteligente – SIS – PG

□ Aplicații

- Probleme în care există o relație între intrări și ieșiri
- Probleme de regresie
- Probleme de clasificare

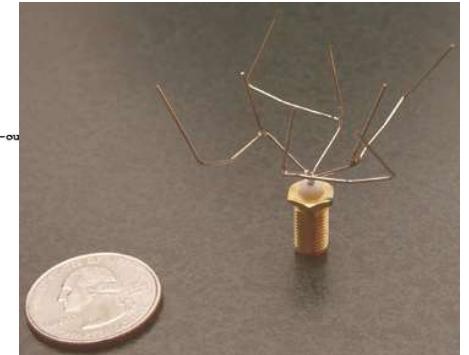
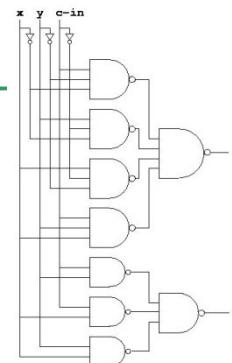


Sisteme inteligente – SIS – PG

□ Aplicații

■ Probleme de design

- Evoluarea de circuite digitale



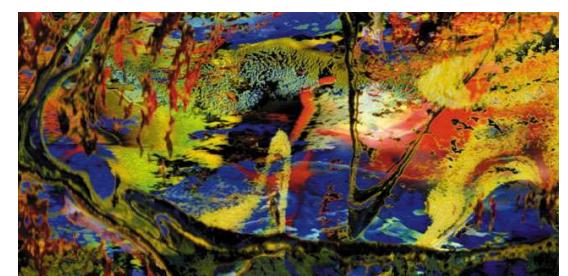
- Evoluarea de antene

- http://idesign.ucsc.edu/projects/evo_antenna.html

- Evoluarea de programe (scrise într-un anumit limbaj)

- Evoluarea de picturi și muzică

- <http://www.cs.vu.nl/~gusz/>



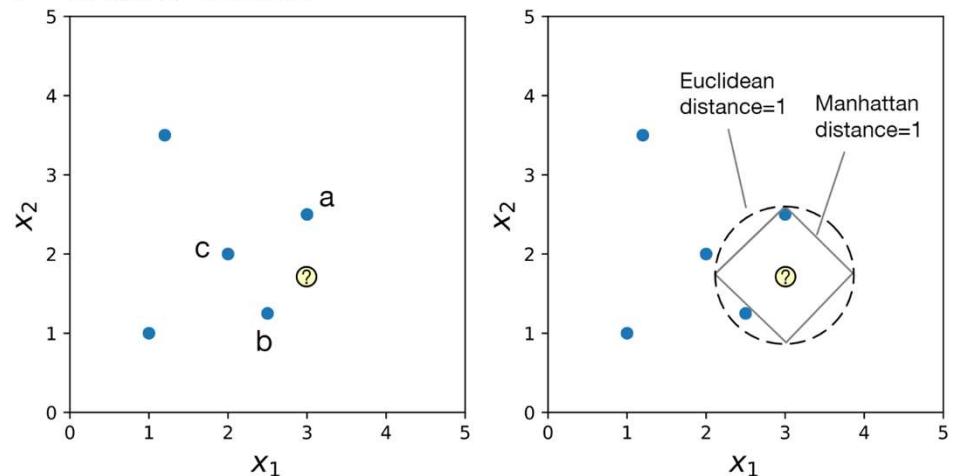
- Altele

- <http://www.genetic-programming.com/humancompetitive.html>

Sisteme inteligente – SIS - kNN

Cei mai apropiati k vecini (*k-nearest neighbours - kNN*)

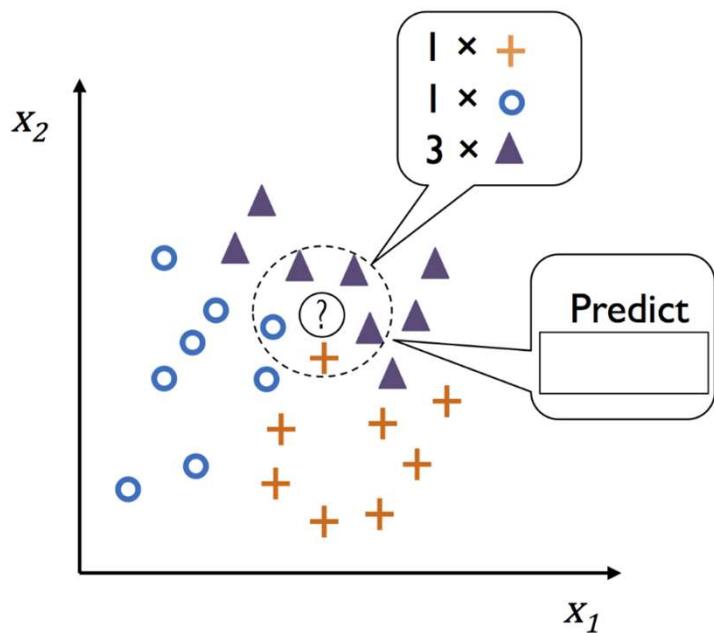
- Unul dintre cei mai simpli algoritmi de clasificare
- În etapa de antrenament, algoritmul doar citește datele de intrare (atributele și clasa fiecărei instanțe)
- În etapa de testare, pentru o nouă instanță (fără clasă):
 - se caută (printre instanțele de antrenament) **cei mai apropiati k vecini**
 - distanța Minkowski (Manhattan, Euclidiană) – atrbute continue
 - distanța Hamming, Levensthein – analiza textelor
 - alte distanțe (funcții kernel)



- se preia clasa majoritară a acestor k vecini

Sisteme inteligente – SIS - kNN

- ❑ Unul dintre cei mai simpli algoritmi de clasificare
- ❑ În etapa de antrenament, algoritmul doar citește datele de intrare (atributele și clasa fiecărei instanțe)
- ❑ În etapa de testare, pentru o nouă instanță (fără clasă)
 - se caută (printre instanțele de antrenament) cei mai apropiati k vecini și
 - **se preia clasa majoritară a acestor k vecini**



○ ○ ○ + + + + +

Majority vote: +

Plurality vote: +

○ ○ ○ + + + ▲ ▲ ▲ ▲

Majority vote: None

Plurality vote: ▲

Sisteme inteligente – SIS - kNN

□ Tool-uri

■ Sklearn (python)

- <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

■ Weka (java)

- <https://weka.sourceforge.io/doc.dev/weka/classifiers/lazy/IBk.html>

□ Biblio

■ https://github.com/rasbt/stat479-machine-learning-fs19/tree/master/02_knn

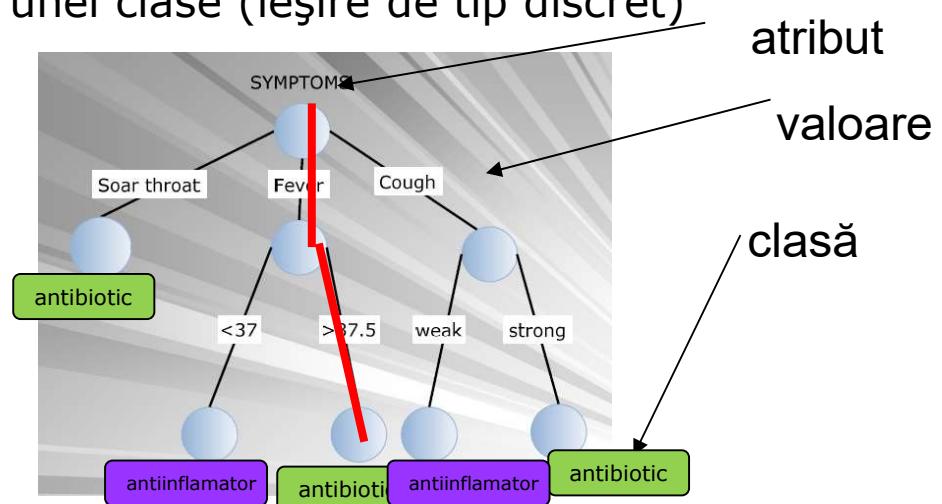
Sisteme inteligente – SIS – Arbori de decizie

□ Scop

- Divizarea unei colecții de articole în seturi mai mici prin aplicarea succesivă a unor reguli de decizie → adresarea mai multor întrebări
 - Fiecare întrebare este formulată în funcție de răspunsul primit la întrebarea precedentă
- Elementele se caracterizează prin informații non-metrice

□ Definire

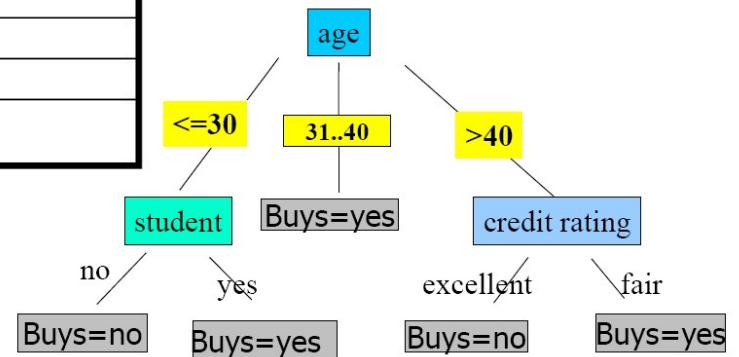
- Fiecare nod intern corespunde unui atribut
- Fiecare ramură de sub un nod (atribut) corespunde unei valori a atributului
- Fiecare frunză corespunde unei clase (ieșire de tip discret)



Sisteme inteligente – SIS – Arbori de decizie

□ Exemplu

rec	Age	Income	Student	Credit_rating	Buys_computer(CLASS)
r1	<=30	High	No	Fair	No
r2	<=30	High	No	Excellent	No
r3	31...40	High	No	Fair	Yes
r4	>40	Medium	No	Fair	Yes
r5	>40	Low	Yes	Fair	Yes
r6	>40	Low	Yes	Excellent	No
r7	31...40	Low	Yes	Excellent	Yes
r8	<=30	Medium	No	Fair	No
r9	<=30	Low	Yes	Fair	Yes
r10	>40	Medium	Yes	Fair	Yes
r11	<=30	Medium	Yes	Excellent	Yes
r12	31...40	Medium	No	Excellent	Yes
r13	31...40	High	Yes	Fair	Yes
r14	>40	Medium	No	Excellent	No



Sisteme inteligente – SIS – Arbori de decizie

□ Definire

- Arborele de decizie
 - Un graf special → arbore orientat bicolor
 - Contine noduri de 3 tipuri:
 - Noduri de decizie → posibilitățile decidentului (ex. Diversele examinări sau tratamente la care este supus pacientul) și indică un test pe un atribut al articoului care trebuie clasificat
 - Noduri ale hazardului – evenimente aleatoare în afara controlului decidentului (rezultatul examinărilor, efectul terapiilor)
 - Noduri rezultat – situațiile finale cărora li se asociază o utilitate (apreciată aprioric de către un pacient generic) sau o etichetă
 - Nodurile de decizie și cele ale hazardului alternează pe nivelele arborelui
 - Nodurile rezultat – noduri terminale (frunze)
 - Muchiile arborelui (arce orientate) → consecințele în timp (rezultate) ale decizilor, respectiv ale realizării evenimentelor aleatoare (pot fi însotite de probabilități)
- Fiecare nod intern corespunde unui atribut
- Fiecare ramură de sub un nod (atribut) corespunde unei valori a atributului
- Fiecare frunză corespunde unei clase

Sisteme inteligente – SIS – Arbori de decizie

□ Tipuri de probleme

- Exemplele (instanțele) sunt reprezentate printr-un număr fix de atribute, fiecare atribut putând avea un număr limitat de valori
- Funcția obiectiv ia valori de tip discret
- AD reprezintă o disjuncție de mai multe conjuncții, fiecare conjuncție fiind de forma atributul a_i are valoarea v_j
- Datele de antrenament pot conține erori
- Datele de antrenament pot fi incomplete
 - Anumitor exemple le pot lipsi valorile pentru unele atribute

■ Probleme de clasificare

- Binară
 - exemple date sub forma $[(\text{atribut}_{ij}, \text{valoare}_{ij}), \text{clasă}_i, i=1,2,\dots,n, j=1,2,\dots,m, \text{clasă}_i \text{ putând lua doar 2 valori}]$
- Multi-clasă
 - exemple date sub forma $[(\text{atribut}_{ij}, \text{valoare}_{ij}), \text{clasă}_i, i=1,2,\dots,n, j=1,2,\dots,m, \text{clasă}_i \text{ putând lua } k \text{ valori}]$

■ Probleme de regresie

- AD se construiesc similar cazului problemei de clasificare, dar în locul etichetării fiecărui nod cu eticheta unei clase se associază nodului o valoare reală sau o funcție dependentă de intrările nodului respectiv
- Spațiul de intrare se împarte în regiuni de decizie prin tăieturi paralele cu axele Ox și Oy
- Are loc o transformare a ieșirilor discrete în funcții continue
- Calitatea rezolvării problemei
 - Eroare (pătratică sau absolută) de predicție

Sisteme inteligente – SIS – Arbori de decizie

❑ Proces

- Construirea (creșterea, inducția) arborelui
 - Se bazează pe un set de date de antrenament
 - Lucrează de jos în sus sau de sus în jos (prin divizare – *splitting*)
- Utilizarea arborelui ca model de rezolvare a problemelor
 - Ansamblul decizilor efectuate de-a lungul unui drum de la rădăcină la o frunză formează o regulă
 - Regulile formate în AD sunt folosite pentru etichetarea unor noi date
- Tăierea (curățirea) arborelui (pruning)
 - Se identifică și se mută/elimină ramurile care reflectă zgomote sau excepții

Sisteme inteligente – SIS – Arbori de decizie

□ Proces → Construirea AD

- Divizarea datelor de antrenament în subseturi pe baza caracteristicilor datelor
 - Un nod → întrebare legată de o anumită proprietate a unui obiect dat
 - Ramurile ce pleacă din nod → etichetate cu posibilele răspunsuri la întrebarea din nodul curent
 - La început toate exemplele sunt plasate în rădăcină
 - La pornire, un atribut va fi rădăcina arborelui, iar valorile atributului vor deveni ramuri ale rădăcinii
 - Pe următoarele nivele exemplele sunt partiționate în funcție de atrbute → ordinea considerării atributelor
 - Pentru fiecare nod se alege în mod recursiv câte un atribut (cu valorile lui pe ramurile descendente din nodul curent)
 - Divizarea → greedy în luarea decizilor

■ Proces iterativ

- Reguli de oprire
 - toate exemplele aferente unui nod fac parte din aceeași clasă → nodul devine frunză și este etichetat cu C_i
 - Nu mai sunt exemple → nodul devine frunză și este etichetat cu clasa majoritară în setul de date de antrenament
 - nu mai pot fi considerate noi atrbute

Sisteme inteligente – SIS – Arbori de decizie

□ Proces → Construirea AD

■ Exemplu

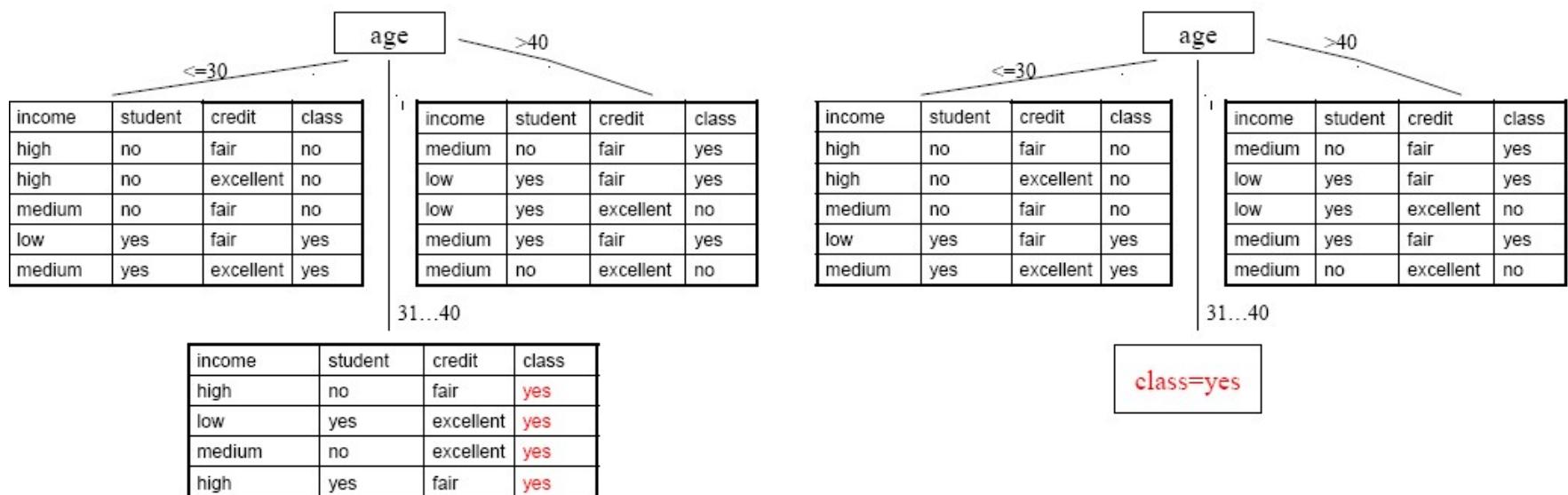
rec	Age	Income	Student	Credit_rating	Buys_computer(CLASS)
r1	<=30	High	No	Fair	No
r2	<=30	High	No	Excellent	No
r3	31...40	High	No	Fair	Yes
r4	>40	Medium	No	Fair	Yes
r5	>40	Low	Yes	Fair	Yes
r6	>40	Low	Yes	Excellent	No
r7	31...40	Low	Yes	Excellent	Yes
r8	<=30	Medium	No	Fair	No
r9	<=30	Low	Yes	Fair	Yes
r10	>40	Medium	Yes	Fair	Yes
r11	<=30	Medium	Yes	Excellent	Yes
r12	31...40	Medium	No	Excellent	Yes
r13	31...40	High	Yes	Fair	Yes
r14	>40	Medium	No	Excellent	No

Sisteme inteligente – SIS – Arbori de decizie

□ Proces → Construirea AD

■ Exemplu

- Pentru rădăcină se alege atributul *age*

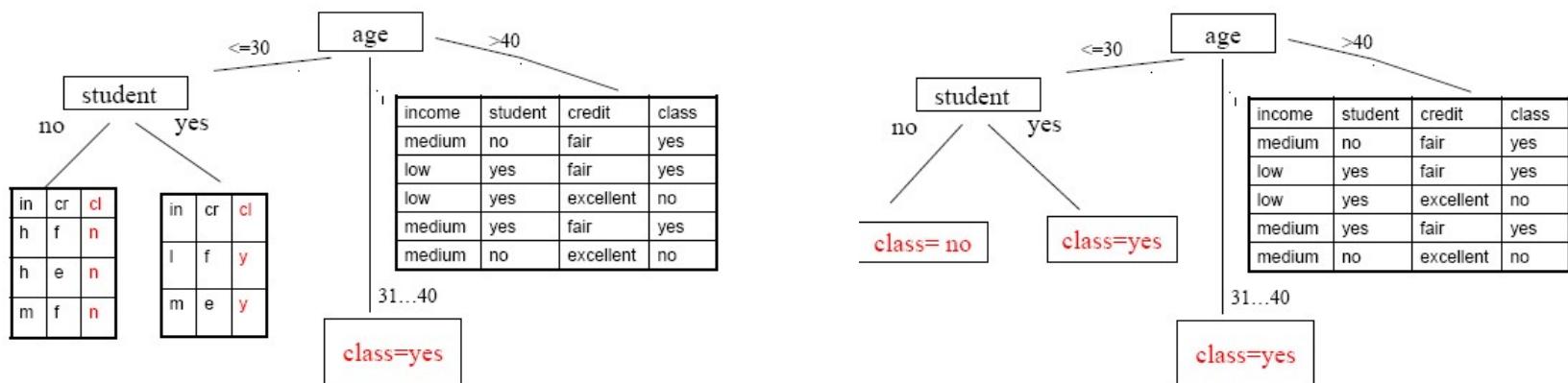


Sisteme inteligente – SIS – Arbori de decizie

□ Proces → Construirea AD

■ Exemplu

- Pentru rădăcină se alege atributul *age*
- Pe ramura ≤ 30 se alege atributul *student*

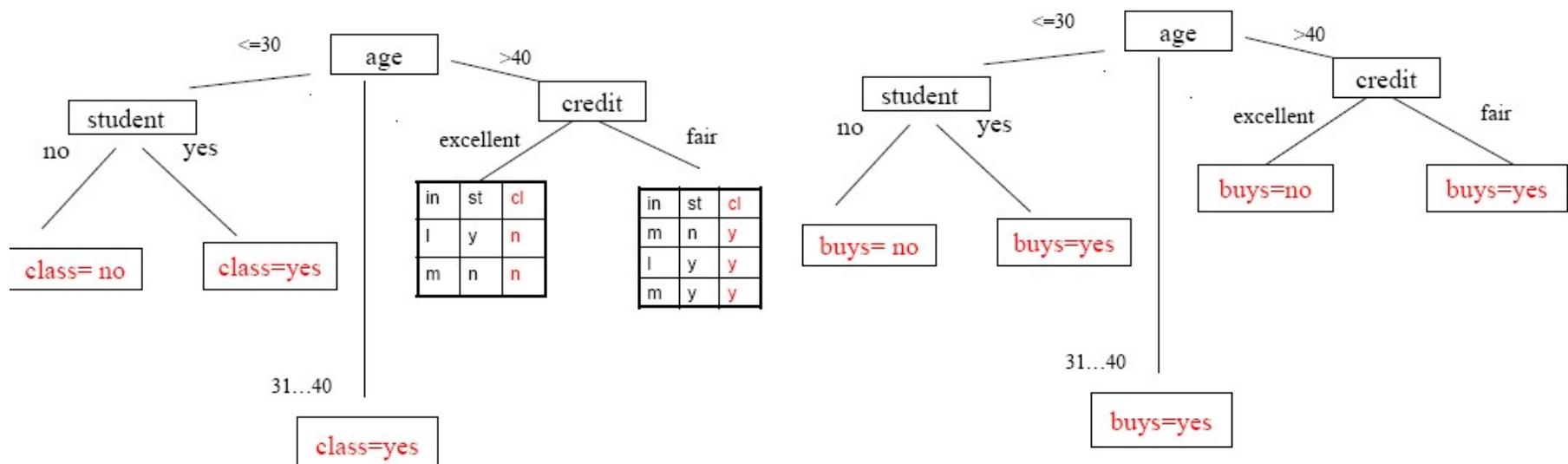


Sisteme inteligente – SIS – Arbori de decizie

□ Proces → Construirea AD

■ Exemplu

- Pentru rădăcină se alege atributul *age*
- Pe ramura ≤ 30 se alege atributul *student*
- Pe ramura > 40 se alege atributul *credit*



Sisteme inteligente – SIS – Arbori de decizie

- Proces → Construirea AD → Algoritmul ID3/C4.5

- Greedy, recursiv, top-down, divide-and-conquer

```
generare(D, A){    //D – partiționare a exemplelor de antrenament, A – lista de atribute
    Crearea unui nod nou N
    Dacă exemplele din D fac parte dintr-o singură clasă C atunci
        nodul N devine frunză și este etichetat cu C
        returnează nodul N
    Altfel
        Dacă A=∅ atunci
            nodul N devine frunză și este etichetat cu clasa majoritară în D
            returnează nodul N
        Altfel
            atribut_separare = Selectează_atribut(D, A)
            Etichetează nodul N cu atribut_separare
            Pentru fiecare valoare posibilă vj a lui atribut_separare
                Fie Dj multimea exemplelor din D pentru care atribut_separare = vj
                Dacă Dj =∅ atunci
                    Atașează nodului N o frunză etichetată cu clasa majoritară în Dj
                Altfel
                    Atașează nodului N un nod returnat de generare(Dj, A –atribut_separare)
            Returnează nodul N
}
```

Sisteme inteligente – SIS – Arbori de decizie

- Proces → Construirea AD → Algoritmul ID3/C4.5
 - Selectează_atribut(D, A) → Alegerea atributului aferent unui nod (rădăcină sau intern)
 - Aleatoare
 - Atributul cu cele mai puține/multe valori
 - Pe baza unei ordini prestabilite a atributelor
 - Câștigul de informație
 - Rata câștigului
 - Indicele Gini
 - Distanța între partiiile create de un atribut

Sisteme inteligente – SIS – Arbori de decizie

- Proces → Construirea AD → Algoritmul ID3/C4.5 → Selectare atribut
 - Câștigul de informație
 - O măsură de impuritate
 - 0 (minimă) dacă toate exemplele fac parte din aceeași clasă
 - 1 (maximă) dacă avem număr egal de exemple din fiecare clasă
 - Se bazează pe entropia datelor
 - măsoară impuritatea datelor
 - numărul sperat (așteptat) de biți necesari pentru a coda clasa unui element oarecare din setul de date
 - clasificare binară (cu 2 clase): $E(S) = - p_+ \log_2 p_+ - p_- \log_2 p_-$, unde
 - p_+ - proporția exemplelor pozitive în setul de date S
 - p_- - proporția exemplelor negative în setul de date S
 - clasificare cu mai multe clase: $E(S) = \sum_{i=1, 2, \dots, k} p_i \log_2 p_i$ – entropia datelor relativ la atributul țintă (atributul de ieșire), unde
 - p_i – proporția exemplelor din clasa i în setul de date S
 - câștigul de informație (*information gain*) al unei caracteristici a (al unui atribut al) datelor
 - Reducerea entropiei setului de date ca urmare a eliminării atributului a
 - $\text{Gain}(S, a) = E(S) - \sum_{v \in \text{valori}(a)} |S_v| / |S| E(S_v)$
 - $\sum_{v \in \text{valori}(a)} |S_v| / |S| E(S_v)$ – *informația scontată*

Sisteme inteligente – SIS – Arbori de decizie

- Proces → Construirea AD → Algoritmul ID3/C4.5 → Selectare atribut
 - Câștigul de informație
 - exemplu

	a1	a2	a3	Clasa
d1	mare	roșu	cerc	clasa 1
d2	mic	roșu	pătrat	clasa 2
d3	mic	roșu	cerc	clasa 1
d4	mare	albastru	cerc	clasa 2

$$S = \{d1, d2, d3, d4\} \rightarrow p_+ = 2 / 4, p_- = 2 / 4 \rightarrow E(S) = - p_+ \log_2 p_+ - p_- \log_2 p_- = 1$$

$$S_{v=mare} = \{d1, d4\} \rightarrow p_+ = 1/2, p_- = 1/2 \rightarrow E(S_{v=mare}) = 1$$

$$S_{v=mic} = \{d2, d3\} \rightarrow p_+ = 1/2, p_- = 1/2 \rightarrow E(S_{v=mic}) = 1$$

$$S_{v=rosu} = \{d1, d2, d3\} \rightarrow p_+ = 2/3, p_- = 1/3 \rightarrow E(S_{v=rosu}) = 0.923$$

$$S_{v=albastru} = \{d4\} \rightarrow p_+ = 0, p_- = 1 \rightarrow E(S_{v=albastru}) = 0$$

$$S_{v=cerc} = \{d1, d3, d4\} \rightarrow p_+ = 2/3, p_- = 1/3 \rightarrow E(S_{v=cerc}) = 0.923$$

$$S_{v=patrat} = \{d2\} \rightarrow p_+ = 0, p_- = 1 \rightarrow E(S_{v=patrat}) = 0$$

$$\text{Gain}(S, a) = E(S) - \sum_{v \in \text{valori}(a)} |S_v| / |S| E(S_v)$$

$$\text{Gain}(S, a_1) = 1 - (|S_{v=mare}| / |S| E(S_{v=mare}) + |S_{v=mic}| / |S| E(S_{v=mic})) = 1 - (2/4 * 1 + 2/4 * 1) = 0$$

$$\text{Gain}(S, a_2) = 1 - (|S_{v=rosu}| / |S| E(S_{v=rosu}) + |S_{v=albastru}| / |S| E(S_{v=albastru})) = 1 - (3/4 * 0.923 + 1/4 * 0) = 0.307$$

$$\text{Gain}(S, a_3) = 1 - (|S_{v=cerc}| / |S| E(S_{v=cerc}) + |S_{v=patrat}| / |S| E(S_{v=patrat})) = 1 - (3/4 * 0.923 + 1/4 * 0) = 0.307$$

Sisteme inteligente – SIS – Arbori de decizie

- Proces → Construirea AD → Algoritmul ID3/C4.5 → Selectare atribut
 - Rata câștigului
 - Penalizează un atribut prin încorporarea unui termen – *split information* – sensibil la gradul de împrăștiere și uniformitate în care atributul separă datele
 - *Split information* – entropia relativ la valorile posibile ale atributului *a*
 - S_v – proporția exemplelor din setul de date *S* care au atributul *a* evaluat cu valoarea *v*
 - $$splitInformation(S,a) = - \sum_{v=value(a)} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}$$

Sisteme inteligente – SIS – Arbori de decizie

□ Studiu de caz - Problema vampirilor

Items	Shadow	Complexion	Garlic	Accent	Vampire
i1	?	Pale	Yes	None	No
i2	Yes	Ruddy	Yes	None	No
i3	?	Ruddy	No	None	Yes
i4	No	Average	No	Heavy	Yes
i5	?	Average	No	Odd	Yes
i6	Yes	Pale	No	Heavy	No
i7	Yes	Average	No	Heavy	No
i8	?	Ruddy	Yes	Odd	No

Sisteme inteligente – SIS – Arbori de decizie

□ Proces

- Construirea arborelui
- Utilizarea arborelui ca model de rezolvare a problemelor
 - Ideea de bază
 - Se extrag regulile formate în arborele anterior construit → Reguli extrase din arborele dat în exemplul anterior:
 - IF $age = "<=30"$ AND $student = "no"$ THEN $buys_computer = "no"$
 - IF $age = "<=30"$ AND $student = "yes"$ THEN $buys_computer = "yes"$
 - IF $age = "31...40"$ THEN $buys_computer = "yes"$
 - IF $age = ">40"$ AND $credit_rating = "excellent"$ THEN $buys_computer = "no"$
 - IF $age = ">40"$ AND $credit_rating = "fair"$ THEN $buys_computer = "yes"$
 - Regulile sunt folosite pentru a clasifica datele de test (date noi). Fie x o dată pentru care nu se știe clasa de apartenență → Regulile se pot scrie sub forma unor predicate astfel:
 - IF $age(x, <=30)$ AND $student(x, no)$ THEN $buys_computer(x, no)$
 - IF $age(x, <=30)$ AND $student(x, yes)$ THEN $buys_computer(x, yes)$

Sisteme inteligente – SIS – Arbori de decizie

□ Proces

- Construirea arborelui
- Utilizarea arborelui ca model de rezolvare a problemelor
 - Dificultăți
 - *Underfitting* (sub-potrivire) → AD indus pe baza datelor de antrenament este prea simplu → eroare de clasificare mare atât în etapa de antrenare, cât și în cea de testare
 - *Overfitting* (supra-potrivire, învățare pe derost) → AD indus pe baza datelor de antrenament se potrivește prea accentuat cu datele de antrenament, nefiind capabil să generalizeze pentru date noi
 - Soluții:
 - fasonarea arborelui (pruning) → Îndepărarea ramurilor nesemnificative, redundante → arbore mai puțin stufoș
 - validare cu încrucișare

Sisteme inteligente – SIS – Arbori de decizie

□ Proces

- Construirea arborelui
- Utilizarea arborelui ca model de rezolvare a problemelor
- Tăierea (fasonarea) arborelui

□ Necesitate

- Odată construit AD, se pot extrage reguli (de clasificare) din AD pentru a putea reprezenta cunoștințele sub forma regulilor *if-then* atât de ușor de înțeles de către oameni
- O regulă este creată (extrasă) prin parcurgerea AD de la rădăcină până la o frunză
- Fiecare pereche (atribut, valoare), adică (nod, muchie), formează o conjuncție în ipoteza regulii (partea dacă), mai puțin ultimul nod din drumul parcurs care este o frunză și reprezintă consecința (ieșirea, partea atunci) regulii

□ Tipologie

- Prealabilă (*pre-pruning*)
 - Se oprește creșterea arborelui în timpul inducției prin sistarea divizării unor noduri care devin astfel frunze etichetate cu clasa majoritară a exemplelor aferente nodului respectiv
- Ulterioară (*post-pruning*)
 - După ce AD a fost creat (a crescut) se elimină ramurile unor noduri care devin astfel frunze → se reduce eroarea de clasificare (pe datele de test)

Sisteme inteligente – SIS – Arbori de decizie

□ Tool-uri

- <http://webdocs.cs.ualberta.ca/~aixplore/learning/DecisionTrees/Applet/DecisionTreeApplet.html>
- WEKA → J48
- <http://id3alg.altervista.org/>
- <http://www.rulequest.com/Personal/c4.5r8.tar.gz>

□ Biblio

- <http://www.public.asu.edu/~kirkwood/DASstuff/decisiontrees/index.html>

Sisteme inteligente – SIS – Arbori de decizie

□ Avantaje

- Ușor de înțeles și interpretat
- Permit utilizarea datelor nominale și categoriale
- Logica deciziei poate fi urmărită ușor, regulile fiind vizibile
- Lucrează bine cu seturi mari de date

□ Dezavantaje

- Instabilitate → modificarea datelor de antrenament
- Complexitate → reprezentare
- Greu de manevrat
- Costuri mari pt inducerea AD
- Inducerea AD necesită multă informație

Sisteme inteligente – SIS – Arbori de decizie

□ Dificultăți

- Existența mai multor arbori
 - Cât mai mici
 - Cu o acuratețe cât mai mare (ușor de "citat" și cu performanțe bune)
 - Găsirea celui mai bun arbore → problemă NP-dificilă
- Alegerea celui mai bun arbore
 - Algoritmi euristici
 - ID3 → cel mai mic arbore acceptabil
 - → teorema lui Occam: "always choose the simplest explanation"
- Atribute continue
 - Separarea în intervale
 - Câte intervale?
 - Cât de mari sunt intervalele?
- Arbori prea adânci sau prea stufoși
 - Fasonarea prealabilă (pre-pruning) → oprirea construirii arborelui mai devreme
 - Fasonarea ulterioară (post-pruning) → înlăturarea anumitor ramuri

Învățare supervizată – algoritmi

Arbore de decizie

□ Tool-uri

- <http://webdocs.cs.ualberta.ca/~aixplore/learning/DecisionTrees/Applet/DecisionTreeApplet.html>
- WEKA → J48
- <http://id3alg.altervista.org/>
- <http://www.rulequest.com/Personal/c4.5r8.tar.gz>
- <https://scikit-learn.org/stable/modules/tree.html>

□ Biblio

- <http://www.public.asu.edu/~kirkwood/DASstuff/decisiontrees/index.html>
- https://github.com/rasbt/stat479-machine-learning-fs19/tree/master/06_trees



Recapitulare

□ Sisteme care învață singure (SIS)

■ Algoritmi de programare genetică (PG)

- Algoritmi evolutivi cu cromozomi sub formă de arbore
- Cromozomii
 - Arborescenți
 - Matriciali
 - Liniari
- codează potențiale soluții de tipul
 - Expresiilor matematice → probleme de regresie/clasificare
 - Expressiilor de tip Boolean → probleme de tip EvenParity / proiectare de circuite digitale
 - Programelor → evoluarea de cod sursă pentru rezolvarea unor probleme

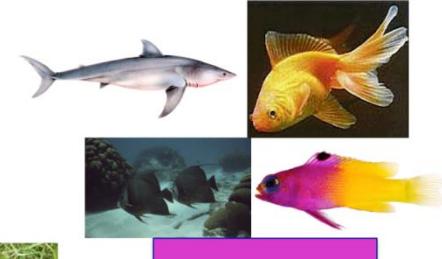
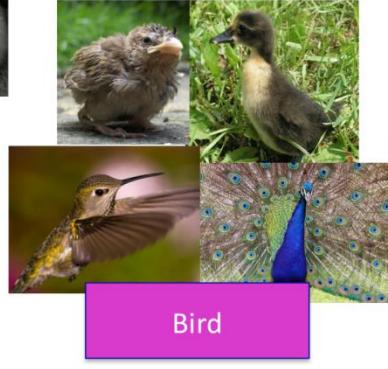
Cursul următor

- A. Scurtă introducere în Inteligența Artificială (IA)
- B. Rezolvarea problemelor prin căutare
 - Definirea problemelor de căutare
 - Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială
- C. Sisteme inteligente
 - **Sisteme care învață singure**
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
 - Sisteme bazate pe reguli
 - Sisteme hibride

Cursul următor – Materiale de citit și legături utile

- capitolul 15 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

-
- Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean –
www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan -
www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop -
www.cs.ubbcluj.ro/~hfpop



ÎNVĂȚAREA NESUPERVIZATĂ

LIMBOI SERGIU

AGENDA

- Învățarea automată-recapitulare
- Învățarea nesupervizată-generalități
- Gruparea datelor (Clustering)
 - Generalități
 - Concepțe de bază
 - Măsuri de similaritate
 - Algoritmi
 - Evaluarea rezultatelor
- Aplicații ale clustering-ului
- Concluzii



ÎNVĂȚAREA AUTOMATĂ

- Învățarea poate fi explicată ca fiind activitatea de a obține cunoștințe sau de a le înțelege, precum și abilitatea de a-ți însuși noțiuni prin studiu, instruire sau experiență [1].
- Învățarea automată (*Machine Learning*) [1] se referă la modificări din sisteme care realizează sarcini asociate cu diverse teme, concepte din Inteligența Artificială, sarcini precum diagnoză, predicție, planificare, control sau detectare.



ÎNVĂȚAREA AUTOMATĂ

- Scop -> proiectarea și dezvoltarea unor algoritmi și metode utilizate pentru ca un sistem computațional să „învețe”.
- De ce să învețe sistemele informaticice?
 - unele sarcini se pot defini doar prin exemple; de aceea trebuie să putem furniza perechi de intrări- ieșiri, în absența unei legături concrete între datele de intrare și rezultate;
 - este posibil ca anumite informații ascunse, nedescifrate, să reflecte corelații sau legături;
 - cantitatea mare de informații poate fi dificil de codificat de către mintea umană.



ÎNVĂȚAREA AUTOMATĂ

- Pentru a defini conceptele utilizate în problematică învățării se folosește o funcție f , iar sarcina celui care învață este de a “ghici” această funcție.
- Tipuri de învățare
 - **Supervizată-** știm (uneori doar aproximativ) valorile funcției f pentru m cazuri ale unui set de antrenare; sistemul poate intui, după etapa de antrenare, care ar fi funcția și pentru un alt set de date
 - **Nesupervizată-** avem doar o mulțime de vectori (un set de date) pentru care nu știm funcția. Presupune divizarea (partiționarea) setului de date în submulțimi sau grupuri. În acest caz valoarea funcției va fi numele subgrupului (clasei) din care vectorul de intrare face parte
 - **Prin întărire (reinforcement)-** sistemul interacționează cu mediul și poate primi recompense sau penalizări

ÎNVĂȚAREA NESUPERVIZATĂ

GENERALITĂȚI

- Ideea de bază- sistemul primește informații (secvențe de forma x_1, x_2, \dots), dar nu î se furnizează rezultate prestabilite, obținute anterior și nici nu primește răsplată din partea mediului
- Stilul nesupervizat este mai comun creierului decât cel supervizat.
- Oamenii și animalele învață să-și analizeze mediul și să identifice obiectele și evenimentele din jurul lor.



ÎNVĂȚAREA NESUPERVIZATĂ

GENERALITĂȚI

- Putem obține grupuri de fructe pe baza anumitor caracteristici?

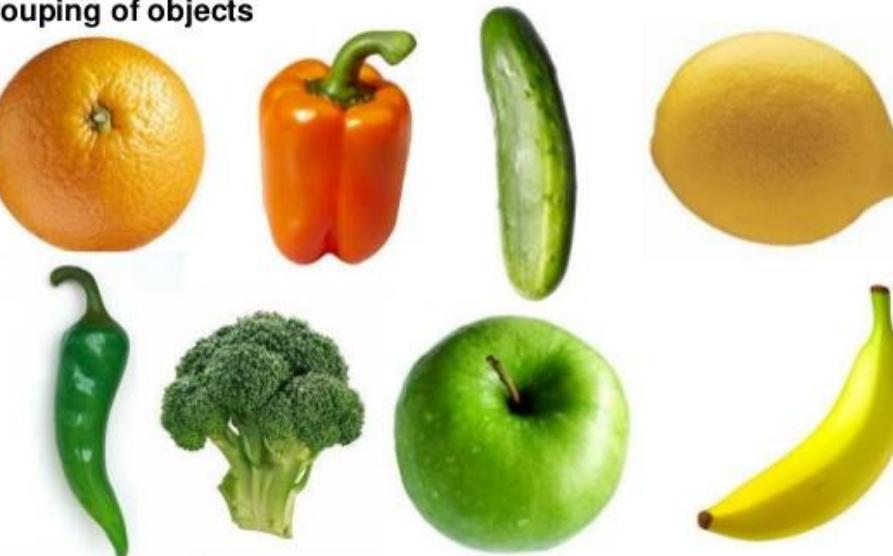


ÎNVĂȚAREA NESUPERVIZATĂ-EXEMPLE [2]

- Dar dacă avem fructe și legume?

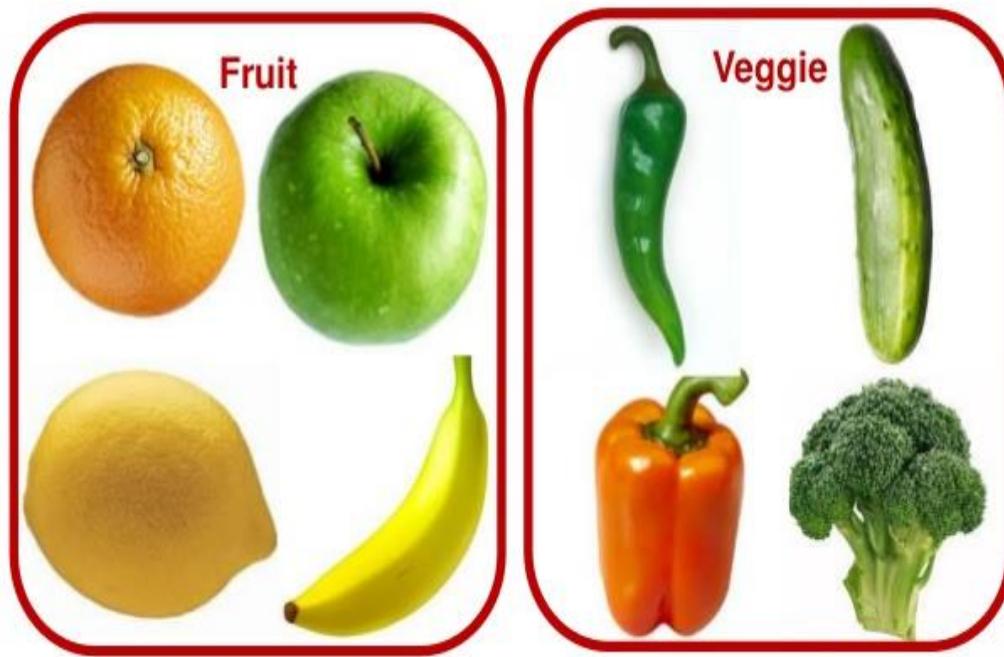
WHAT IS CLUSTERING?

Grouping of objects



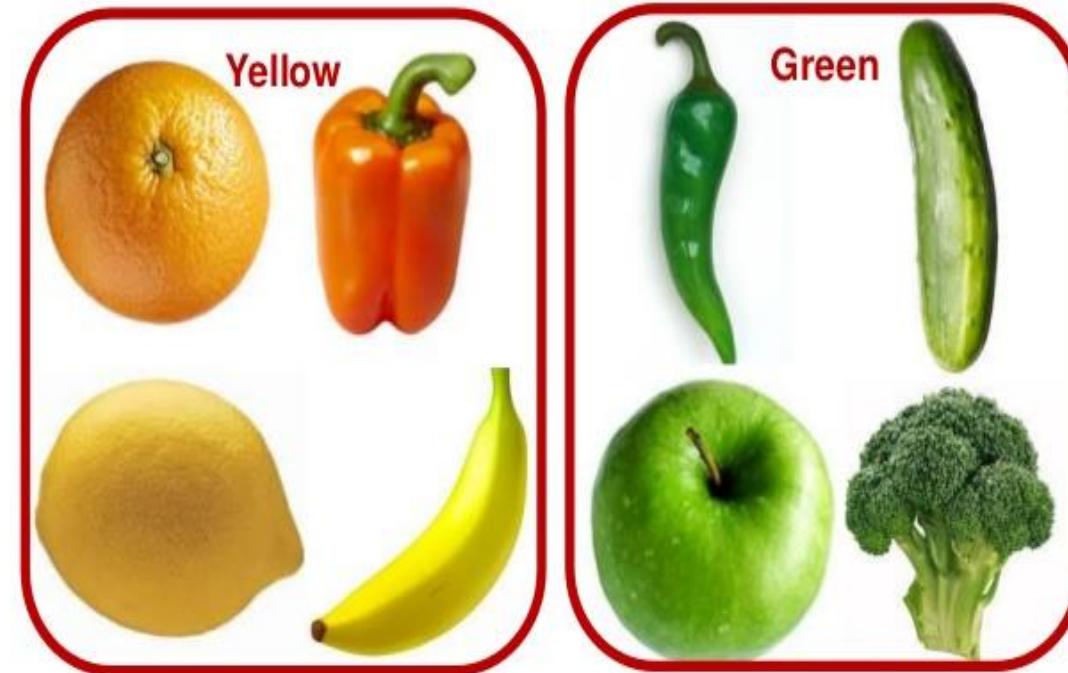
ÎNVĂȚAREA NESUPERVIZATĂ-EXEMPLE [2]

CLUSTERING I. (BY TYPE)



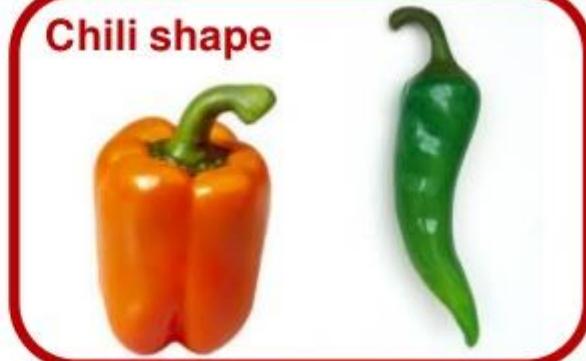
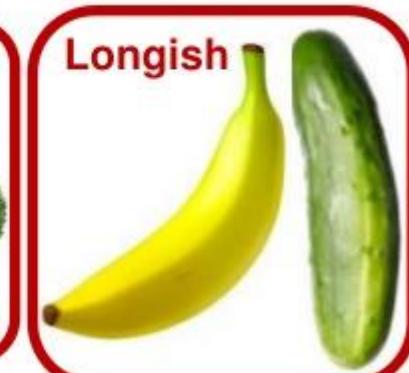
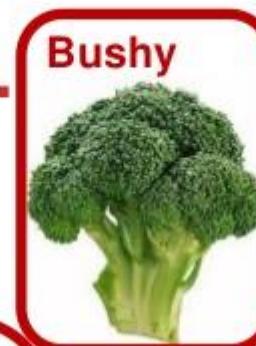
ÎNVĂȚAREA NESUPERVIZATĂ-EXEMPLE [2]

CLUSTERING II. (BY COLOR)

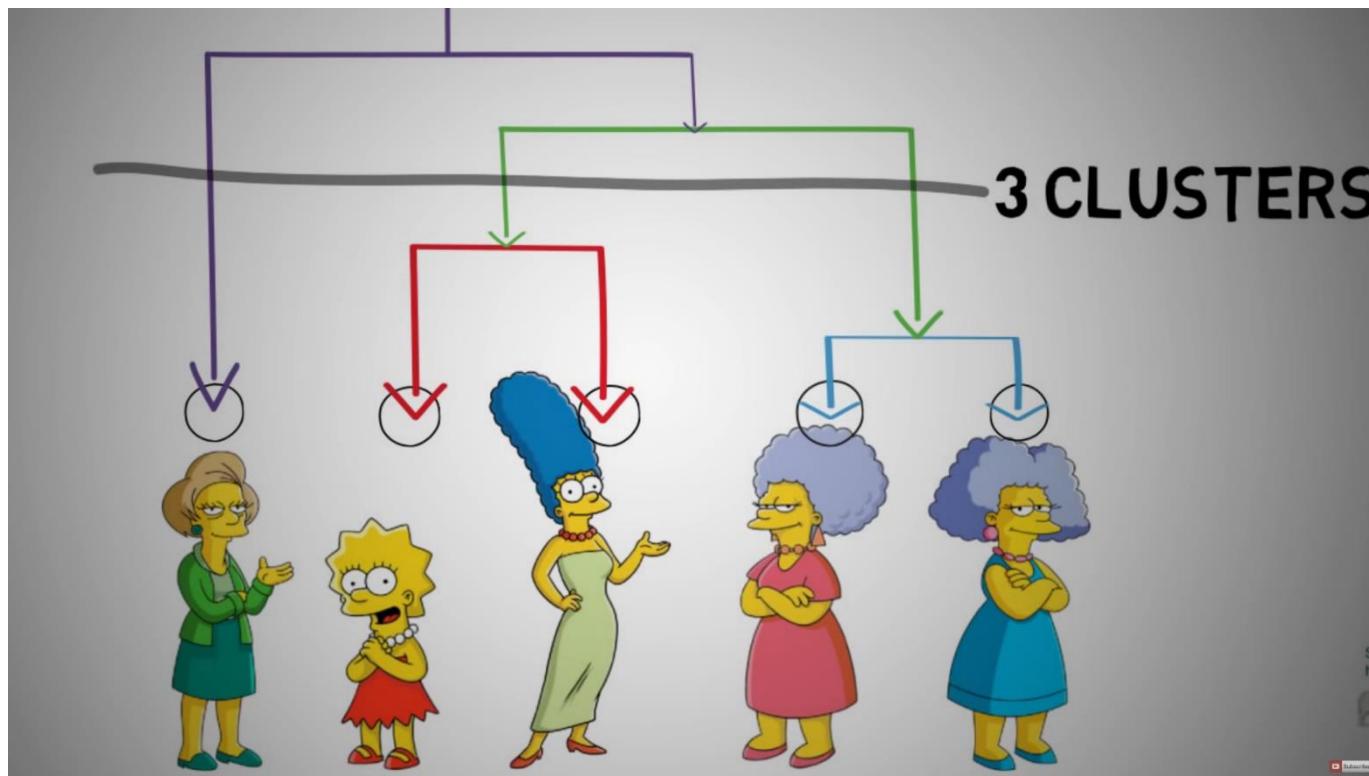


ÎNVĂȚAREA NESUPERVIZATĂ-EXEMPLE [2]

CLUSTERING III. (BY SHAPE)



ÎNVĂȚAREA NESUPERVIZATĂ- EXEMPLE



ÎNVĂȚAREA NESUPERVIZATĂ

GENERALITĂȚI

- “Ce dorim să învețe sistemul, dacă nu îi livrăm informație externă?”
 - descoperirea unor grupuri (clase) din setul de date inițial;
 - extragerea unor atribute ce caracterizează datele de intrare într-un mod mai compact;
 - identificarea unor coincidențe (similitudini) naturale în cadrul datelor primite.



ÎNVĂȚAREA NESUPERVIZATĂ

GENERALITĂȚI

- Utilitate în clasificarea de obiecte neetichetate.
- Plecând de la o mulțime de instanțe, se dorește găsirea unor mecanisme de a grupa obiectele pe baza unor similitudini, astfel încât obiectele din același grup să aibă o similitudine maximă.



ÎNVĂȚAREA NESUPERVIZATĂ

GENERALITĂȚI

○ Abordări

- gruparea datelor (*clustering*);
- rețele cu auto-organizare (*self-organizing maps*);
- reguli de asociere;
- algoritmi de maximizare a așteptărilor;
- analiza componentelor independente;
- analiza componentelor principale;
- descompunerea în valori singulare.
- metoda momentelor



GRUPAREA DATELOR

CLUSTERING

- *Clustering-ul* (gruparea datelor) [3] este procesul de grupare a obiectelor (instanțe, observații) pe baza caracteristicilor lor.
- Scopul acestei activități este ca în cadrul aceluiași grup, obiectele să fie similare unele cu altele și diferite de obiectele din alte grupuri.
- Omogenitatea (similaritatea) din cadrul unui grup, denumit și *cluster*, precum și diferența cât mai mare între grupuri caracterizează acest proces.



GRUPAREA DATELOR

CLUSTERING

- În clasificarea supervizată se dă o colecție de date etichetate (preclasificate), problema fiind de a putea eticheta un nou set de date.
- În cazul *clustering-ului*, problema este de a grupa o colecție de date neetichetate în grupuri cu o anumită semnificație.
- Într-o anumită măsură, etichetele sunt asociate grupurilor, dar aceste etichete sunt date pe seama informațiilor initiale (nu se oferă etichete din sursă externă), aşa numitele *data driven*.



GRUPAREA DATELOR

CLUSTERING

- Componentele (pașii) procesului de grupare a datelor [4] :
 - stabilirea datelor de intrare (optional selectarea atributelor—*feature selection*);
 - definirea unei măsuri de proximitate (vecinătate) adecvată domeniul datelor;
 - gruparea propriu-zisă (*clustering*);
 - abstractizare (dacă este cazul);
 - evaluarea rezultatelor.



CLUSTERING

CONCEPTE

- Stabilirea datelor se referă la identificarea numărului de clase (*clusteri*), numărului și tipurilor atributelor din cadrul setului de date.
- În cadrul acestei etape se poate aplica și selectarea atributelor (*feature selection*), proces de identificare a submulțimii, din mulțimea de attribute care va fi efectiv folosită pentru grupare.
- Tipuri de date
 - După nr de attribute- binare, discrete, continue
 - După tipul valorilor- calitative și cantitative



CLUSTERING CONCEPTE

- O măsură de proximitate sau de similaritate reprezentă, de fapt, o funcție distanță, pentru a determina cât de similari/diferiți sunt doi clusteri
- Abstractizarea este procesul de extragere a unei reprezentări simple și compacte a setului de date. Se dorește simplicitate (să fie ușor de înțeles), reliefată printr-o descriere concisă a fiecărui grup de obiecte, cu ajutorul unor termeni corespunzători acestui domeniu de *clustering* (ex: medoizi, centroizi).



CLUSTERING CONCEPTE

- Evaluarea rezultatelor
 - Se poate folosi o examinare externă care presupune compararea structurii obținute cu cea *a priori* sau o examinare internă care evaluează structura intrinsecă potrivită pentru setul de date inițial.



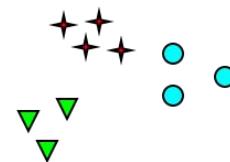
CE ESTE UN CLUSTER?



So tell me how
many clusters do
you see?



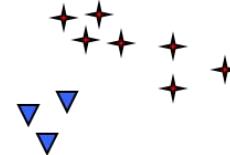
How many clusters?



Six Clusters



Two Clusters

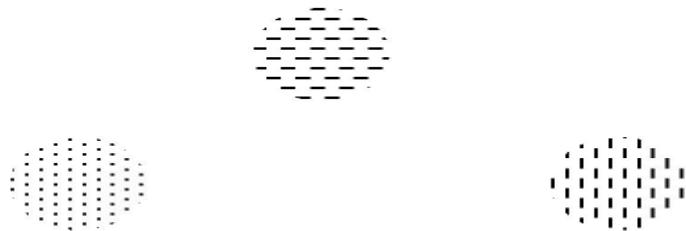


Four Clusters



CE ESTE UN CLUSTER?

- Cormack (1971) și Gordon (1999) [5] au stabilit că un *cluster* poate fi definit în funcție de două proprietăți interne: omogenitate și izolare externă (separare).
- Vipin Kumar [3] prezintă câteva definiții ale unui *cluster*:
 - definiție bazată pe delimitări/separări: *cluster* = o mulțime de puncte în care orice punct este mai apropiat de un altul din mulțime, decât de un punct din afara ei



CE ESTE UN CLUSTER?

- definiție bazată pe centru: *cluster* = o mulțime de obiecte în care un obiect este mai apropiat de centrul mulțimii decât de centrul unei alte mulțimi



- definiție bazată pe similaritate: *cluster*= mulțime de obiecte care sunt “similară”, dar diferite de obiectele din alte grupuri



MĂSURI DE SIMILARITATE

- O funcție de similaritate “măsoară” cât de bun este un anumit grup. În general termenul utilizat pentru astfel de măsuri este proximitate sau vecinătate. Două instanțe sunt “apropiate”, atunci când disimilaritatea (distanța) este mică sau similaritatea este mare.
- Distanța-minimă, similaritatea-maximă
- Exemple:
- Distanța euclideană $d(i,j)=\sqrt{\sum_{k=1}^n |x_{ik} - x_{jk}|^2}$
- Distanța *Manhattan*: $d(i,j)=\sum_{k=1}^n |x_{ik} - x_{jk}|$.



MĂSURI DE SIMILARITATE

- Similaritatea cosinus

- $\cos(A, B) = (A \cdot B) / \|A\| \|B\|$,
 - unde \cdot este produsul vectorial, iar $\|A\|$ lungimea vectorului A.

- Similaritate Jaccard – unde A, B multimi

- între 0 și 1

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$



TEHNICI DE CLUSTERING

- metode ierarhice
 - algoritmi aglomerativi;
 - algoritmi divizivi.
- metode partitioñale
 - *k-medoids*;
 - *k-means*;
 - algoritmi bazañi pe densitate.
- metode bazate pe grilă /retea
- algoritmi bazañi pe scalabilitate
- algoritmi pentru date de dimensiuni mari
 - *clustering* de subspañiu;
 - tehnici de proiecþie;



TEHNICI DE CLUSTERING

- Jain [4] consideră că metodele de grupare a datelor pot fi organizate în funcție de anumite criterii:
 - **aglomerativ** vs. **diviziv**- abordare ce se referă la modul de formare a grupurilor;
 - **hard** (strictă) vs **fuzzy** (maleabilă/permisivă) –se referă la modul de atribuire a instanțelor la un anumit grup.
 - Atribuirea *hard* indică faptul că un obiect poate apartine unui singur grup, de-a lungul procesului. Atribuirea *fuzzy* (*soft*) definește grade de apartenență la diferite grupuri, pentru fiecare instanță;

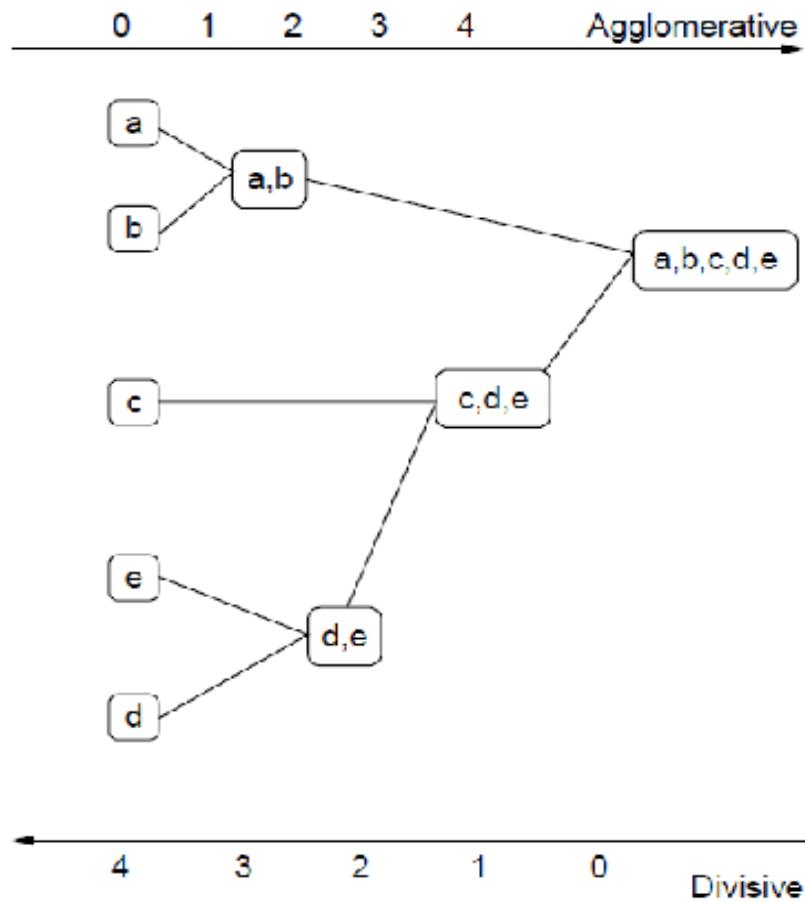


CLUSTERING IERARHIC

- În clasificarea ierarhică datele nu sunt partitionate într-un anumit număr de grupuri dintr-un singur pas.
- Tehnicile ierarhice se împart în metode agglomerative și divizive.
- Cele agglomerative procesează o serie de fuziuni succesive, anumite, în grupuri. Cele divizive separă, succesiv, individii în grupuri fine, mai precise.



CLUSTERING IERARHIC [5]



CLUSTERING IERARHIC

- Aglomerativ: Se pornește cu obiecte individuale. Se unifică, gradual, obiecte care au similaritate maximă (distanță minimă). Se continuă până când toate obiectele vor fi conținute într-un singur grup sau până când se ajunge la număr dorit de *clusteri*.
- Diviziv: Se pornește cu un grup ce conține toate obiectele. Se separă, gradual, grupul în două, atribuind obiectele celor doi noi *clusteri* astfel încât să se maximizeze similaritatea din cadrul fiecăruia grup. Se continuă divizarea până când se obțin *clusteri* ce conțin un singur obiect sau până când se obține numărul dorit de *clusteri*.



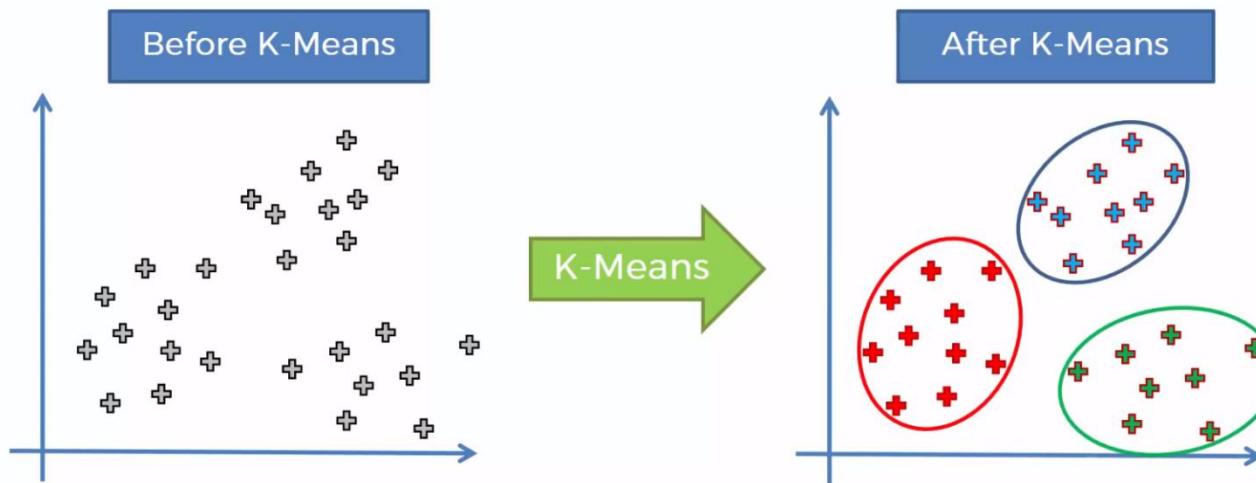
CLUSTERINGUL PARTIȚIONAL

- Un algoritm de *clustering* partițional obține o partiție a setului de date, comparativ cu *clustering-ul* ierarhic care produce o structură
- O problemă a algoritmilor partiționali este alegerea numărului de *clusteri* doriti a fi obținuți în urma procesului.
- Metodele partiționale au ca scop construirea sau găsirea unei partiții formate din k *clusteri*, plecând de la un set de date cu n instanțe.
- Aceste tehnici cuprind și câteva metode euristică: *k-means* și *k-medoids* [5]. Algoritmul *k-means* presupune că fiecare grup este reprezentat de centrul lui, iar algoritmul *k-medoids* implică faptul că fiecare grup este reprezentat de unul dintre obiectele lui.

K-MEANS

- Denumirea provine de la reprezentarea a *k clusteri* pe baza mediei obiectelor din cadrul lor.
- Un astfel de obiect, definit ca fiind media instanțelor unui *cluster*, poartă denumirea de centroid [5]

What K-Means does for you



K-MEANS

○ Algoritmul

- selectează k instanțe ca fiind centroizii inițiali;
- atribuie obiectele la cel mai apropiat centroid;
- recalculează centroidul fiecărui grup;
- revenire la pasul 2, oprirea realizându-se atunci când nu mai au loc modificări.



K-MEANS

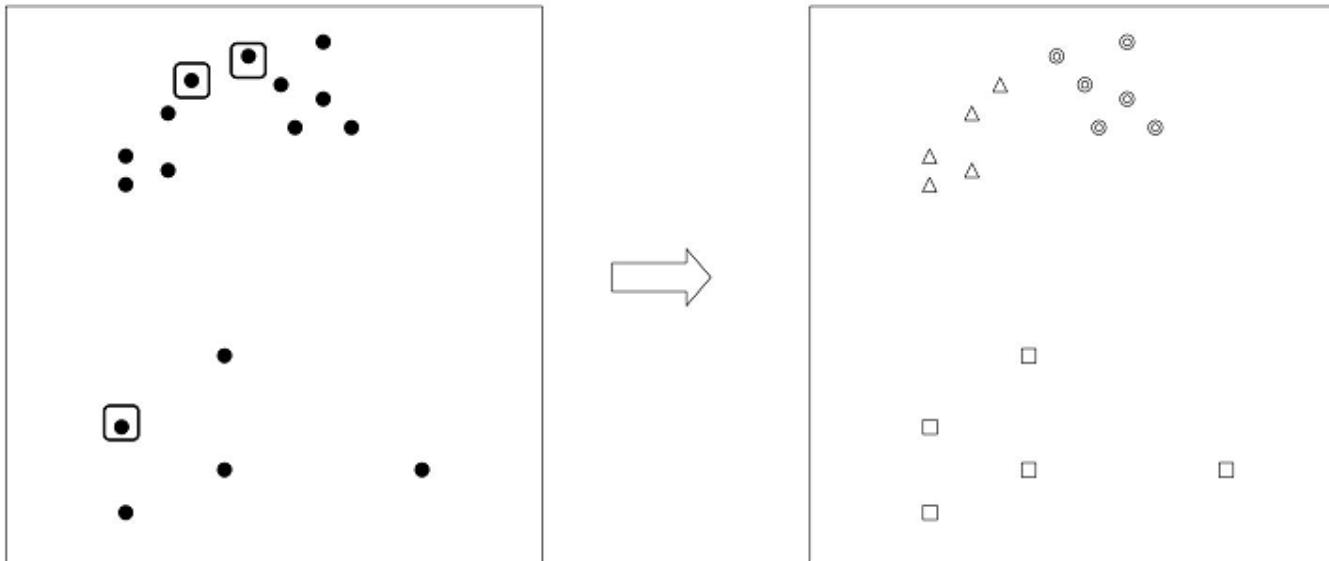
- Alegerea centroizilor inițiali

- etapa de bază a procedurii *k-means*. Este foarte ușor să alegi aleator k obiecte care să fie centroizi, dar rezultatele, de regulă, nu sunt mulțumitoare.
- Este posibilă și abordarea ce implică mai multe rulări, fiecare cu un număr diferit de obiecte alese aleator ca reprezentând centroizi



K-MEANS

- Alegerea naturală a centroizilor [5]



- centroizii inițiali sunt aleși, adesea, din regiuni dense, în aşa fel încât obiectele să fie foarte bine separate, pentru ca doi centroizi să nu fie aleşi din cadrul aceluiași grup.

K-MEANS EXEMPLU

- Se dau punctele $P_1(2,3)$, $P_2(3,1)$, $P_3(4,2)$,
 $P_4(11,5)$, $P_5(12,4)$, $P_6(12,6)$, $P_7(7,5)$, $P_8(8,4)$,
 $P_9(8,6)$
- Aplicați k-means pornind de la centroizii $K_1=P_2$
și $K_2=P_8$
- Se folosește distanța euclideană

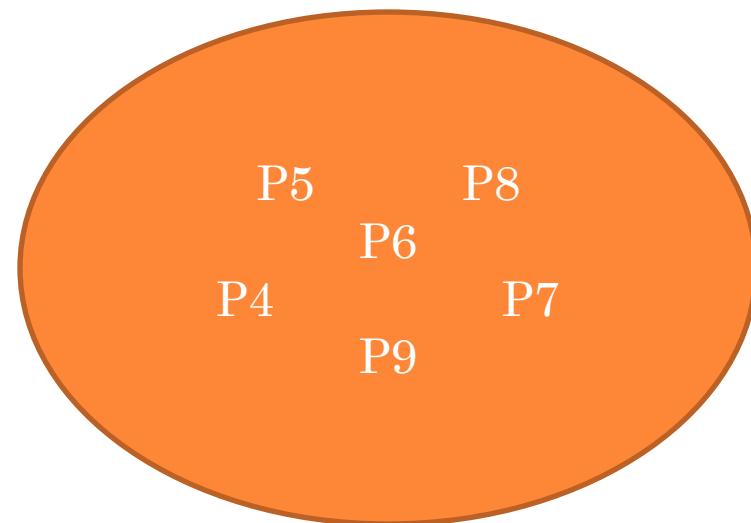
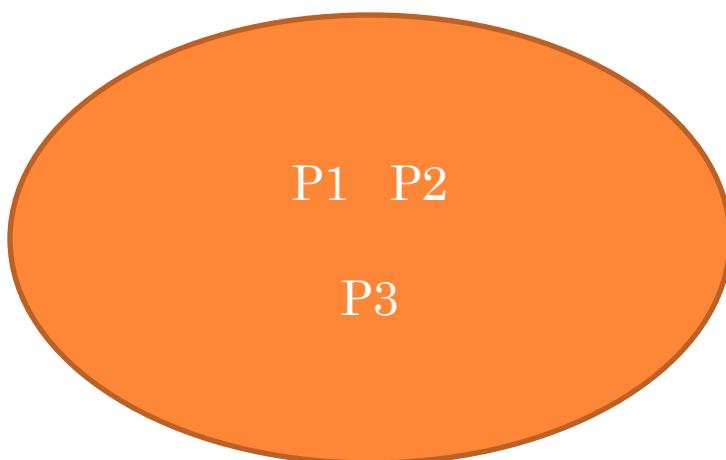


REZOLVARE

- Calculăm distanța dintre fiecare obiect neselectat și centroizi
- $d(P_1, K_1) = 2,23$ și $d(P_1, K_2) = 6,08 \rightarrow P_1$ se atribuie lui K_1
- $d(P_3, K_1) = 1,41$ $d(P_3, K_2) = 4,47$ deci P_3 în K_1
- $d(P_4, K_1) = 8,94$ $d(P_4, K_2) = 3,16$, P_4 în K_2
-



REZOLVARE



- Calculăm noii centroizi
 - $K_1 = (3,2)$ $K_{1x} = 9/3$ $K_{1y} = 6/3$
 - $K_2 = \dots$



K-MEDOIDS

1. se selectează k obiecte reprezentative numite medoizi
2. se înlocuiește unul dintre obiectele selectate (medoizi) cu unul dintre obiectele neselectate. Se calculează distanța dintre fiecare obiect neselectat și cel mai apropiat medoid candidat, iar apoi distanța este însumată pentru toate punctele/ instanțele. Această distanță reprezintă costul configurației curente
3. se selectează configurația cu costul cel mai mic. Dacă există o nouă configurație, se repetă pasul 2;
4. dacă nu există o configurație nouă, se atribuie fiecare obiect neselectat celui mai apropiat medoid și algoritmul se oprește.

EVALUAREA REZULTATELOR

- De ce să evaluăm procesul de *clustering*?
 - pentru a compara diferiți algoritmi de *clustering*;
 - pentru a compara două grupuri;
 - pentru a compara multimi de *clusteri*.
- Aspecte legate de validarea *clustering-ului*:
 - determinarea numărului ideal de *clusteri*;
 - găsirea unor legături între structurile identificate în urma procesului, din cadrul setului de date, și informații externe legate de datele de intrare;
 - evaluatează cât de bine se potrivesc rezultatele produse de analiza de *clustering* cu setul de date, fără a referi informații externe (utilizează doar datele).

EVALUAREA REZULTATELOR

- Există două abordări referitoare la activitatea de evaluare a *clustering-ului*:
 - externă-bazată pe informații anterioare despre date (posibil să cunoaștem etichetele datelor);
 - internă-bazată doar pe informații intrinseci (doar setul de date), fără să utilizăm surse externe.



EVALUAREA INTERNA

- Indexul Dunn [6] - acest index determină raportul dintre cea mai mică distanță *intercluster* și cea mai mare distanță *intracluster*, în cadrul partiției.

$$D = \frac{\min_{1 \leq i < j \leq n} d(i, j)}{\max_{1 \leq k \leq n} d'(k)},$$

- unde $d(i, j)$ reprezintă distanța *intercluster* (distanța între *clusterul i* și *clusterul j*), iar $d'(k)$ reprezintă *distanța intracluster* (distanța în *clusterul k*).
- Ca distanță între doi *clusteri* se poate folosi orice distanță, de exemplu distanța între centroizii grupurilor. Distanța *intracluster* (d') poate fi cea mai mare distanță între orice pereche de elemente din *clusterul k*. Valorile mari pentru indexul Dunn, indică soluții bune.

EVALUAREA INTERNA

- Coeficientul Silhouette [6]

$$s(i) = \frac{(b(i) - a(i))}{\text{Max}\{a(i), b(i)\}}$$

- unde $a(i)$ este distanța medie între obiectul i și celelalte obiecte din X_j , iar $b(i)$ reprezintă cea mai mică distanță medie între obiectul i și obiectele din ceilalți *clusteri*, din care i nu face parte.
- Valorile pentru $s(i)$ sunt din $[-1, 1]$, soluții bune când indexul este cât mai aproape de 1

EVALUAREA INTERNA

- Indexul Davies-Bouldin [6]

$$BD = \frac{1}{c} \sum_{i=1}^c \text{Max}_{i \neq j} \left\{ \frac{d(X_i) + d(X_j)}{d(c_i, c_j)} \right\}$$

- unde c reprezintă numărul de *clusteri*, $d(X_i)$ și $d(X_j)$ sunt distanțele între toate obiectele din *clusterul* X_i , respectiv X_j , și centroizii acelor *clusteri*. $d(c_i, c_j)$ este distanța între centroizii celor doi *clusteri*, c_i și c_j . Cu cât acest index produce valori mai mici, cu atât se obțin soluții mai bune.



EVALUARE EXTERNĂ

- Acuratețea
- Precizia
- Specificitatea
- Senzitivitatea (recall)



APLICAȚII ALE CLUSTERINGULUI

- Marketing și comerț

- segmentare=modalitatea de organizare a clienților în grupuri, în funcție de preferințe pentru anumite produse, trăsături sau așteptări

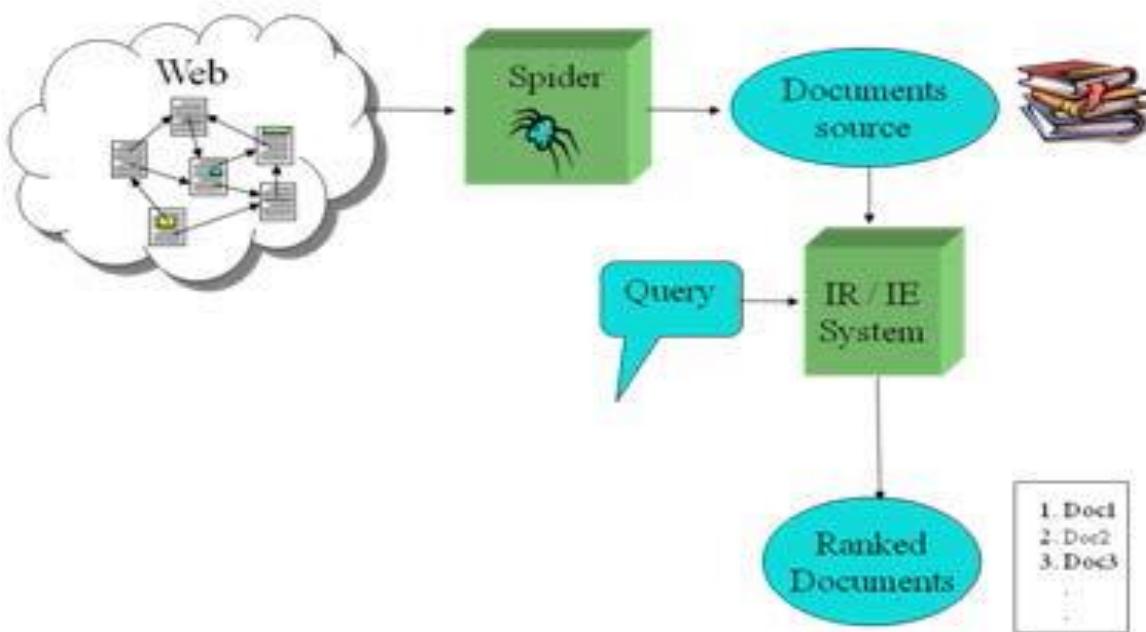


APLICAȚII ALE CLUSTERINGULUI

- Geico [7] - planificări pentru personalizarea ofertelor de asigurări auto și voia să înțeleagă exact ce dorințe și așteptări au clienții de la furnizorul de asigurări
- Respondenții implicați în chestionare au fost grupați în clustere/segmente
- acțiunile luate de clienții din același segment ilustrează răspunsuri similare, dar diferite de cele date de clienții din restul segmentelor. Așadar, aspectele privind asigurarea auto pe care un client le consideră importante vor fi importante și pentru restul clienților din același *cluster*/segment

APLICAȚII ALE CLUSTERINGULUI

- Motoare de căutare [9]



APLICAȚII ALE CLUSTERINGULUI

- Analiza crimelor [8]
- Pentru aplicarea *clustering-ului* în analiza crimelor s-au utilizat seturi de date înregistrare de poliția din Anglia și Țara Galilor în perioada 1990-2011. Seturile de date se refereau la omucideri (uciderea unei alte persoane).



APLICAȚII ALE CLUSTERINGULUI

- *k-means* a avut drept obiectiv urmărirea ratei criminaliătii
- s-au luat în considerare omuciderile și s-au format *clusteri* ce ofereau informații legate de numărul de omucideri din diferiți ani.
- *Clusterul 0* cuprindea între 0 și 15 crime, *clusterul 1* între 0 și 60, *clusterul 2* între 0 și 600, etc.
- De pe graficele corespunzătoare fiecărui *cluster* se observa anul cu număr de omucideri minim și anul cu număr maxim. La finalul analizei s-a observat că omuciderile au scăzut din 1990 până în 2011.
- tendințele legate de crime, infracționalități pentru următorii ani și se pot projecța tehnici de preveneție și de reducere a acestor acțiuni.

APLICAȚII ALE CLUSTERINGULUI

- K-Means în predicția performanței studenților [10]
 - monitorizarea performanțelor studenților
 - rezultatele studenților de la un institut privat din Nigeria
- Economie
- Finanțe – identificarea unor categorii de clienți
- Bio-arheologie



CONCLUZII

- *Clustering-ul* reprezintă o ramură importantă a învățării nesupervizate, aplicarea lui având rezultate semnificative în multe domenii
- *clustering-ul* simplifică mult o muncă manuală care, uneori, se dovedește a fi anevoieioasă.



BIBLIOGRAFIE

- [1] Nils J.Nilsson, *Introduction to Machine Learning*, Stanford University, Stanford, 1996
- [2] Gabor Veress, Clustering training, 2013, <https://www.slideshare.net/gveress/cluster-training-2013>
- [3] Vipin Kumar, *An introduction to cluster analysis for data mining*, course, 2000
- [4] A. K. Jain, M. N. Murty and P. J. Flynn, *Data clustering: A review.*, ACM Comput.Surv., 31(3):264-323, 1999
- [5] Brian S.Everitt, Sabine Landau, Morven Leese, Daniel Stahl, *Cluster Analysis*, 5th edition, Wiley, Londra, 2011

BIBLIOGRAFIE

- [6] E.Rendón, I.Abundez, A.Arizmendi and E. M. Quiroz, *Internal versus External cluster validation indexes*, International Journal of Computers and Communications, Vol.5: 27-34, 2011
- [7] R. Venkatesan, *Cluster analysis for segmentation*, Darden Business Publishing, University of Virginia, 2007
- [8] Jyoti Agarwal, Renuka Nagpal, Rajni Sehgal, *Crime Analysis using K-Means Clustering*, International Journal of Computer Applications, 83(4):1-4, 2013

BIBLIOGRAFIE

- [9] L.V. Bijuraj, *Clustering and its Applications*, National Conference on New Horizons in IT, 169-172, 2013
- [10] O.J. Oyelade, O.O. Oladipupo, I.C. Obagbuwa, *Application of k-means clustering algorithm for prediction of students' academic performance*, International Journal of Computer Science and Information Security, 1:292-295, 2010





UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ



Sisteme inteligente

Sisteme bazate pe reguli în medii certe

Laura Dioșan

Sumar

- A. Scurtă introducere în Inteligența Artificială (IA)**
- B. Rezolvarea problemelor prin căutare**
 - Definirea problemelor de căutare
 - Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială
- C. Sisteme inteligente**
 - Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
 - Sisteme hibride
 - Sisteme bazate pe reguli în medii certe
 - Sisteme bazate pe reguli în medii incerte (Bayes, factori de certitudine, Fuzzy)

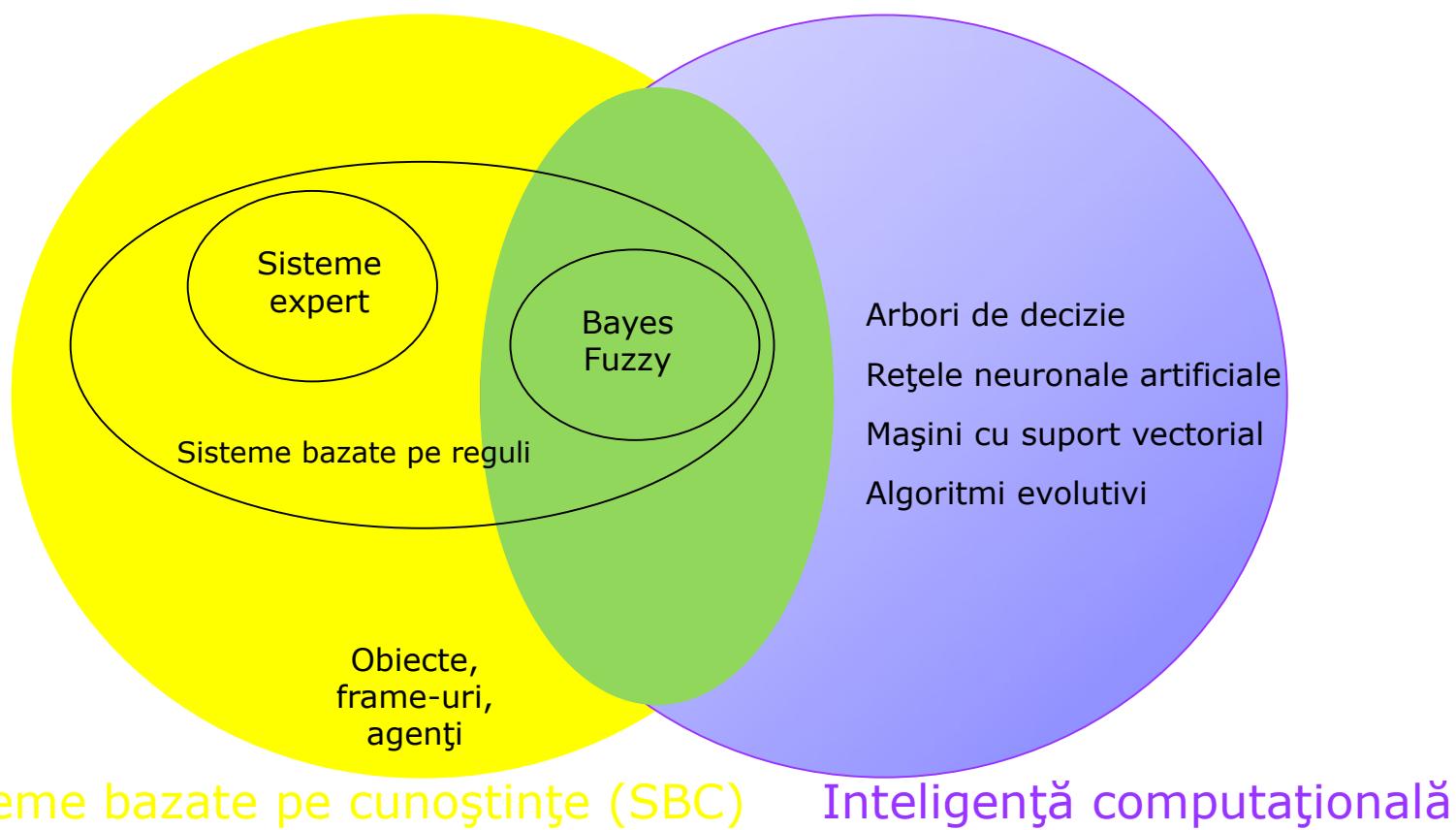
Materiale de citit și legături utile

- capitolul III din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 4 și 5 din *H.F. Pop, G. Șerban, Inteligență artificială, Cluj Napoca, 2004*
- capitolul 2 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 6 și 7 din *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

Conținut

- Sisteme inteligente
 - Sisteme bazate pe cunoștințe
 - Sisteme bazate pe logică
 - Sisteme bazate pe reguli în medii certe

Sisteme inteligente



Sisteme inteligente – sisteme bazate pe cunoștințe (SBC)

- sistemele computaționale – alcătuite din 2 module principale (roluri):
 - Domeniul de cunoștințe (baza de cunoștințe – BC – *knowledge base*)
 - Informațiile specifice despre un domeniu
 - Modulul de control (MC – *inference engine*)
 - Regulile prin care se pot obține informații noi
 - Algoritmi independenți de domeniu

Sisteme inteligente – SBC

Baza de cunoștințe (BC)

- Conținut
- Tipologie
- Modalități de reprezentare a cunoștințelor
- Modalități de stocare a cunoștințelor

Sisteme inteligente – SBC

Baza de cunoștințe (BC)

□ Conținut

- Informații (exprimate într-o anumită reprezentare – ex. propoziții) despre mediu
- informații necesare pentru înțelegerea, formularea și rezolvarea problemelor
- mulțime de propoziții (exprimate/reprezentate într-un limbaj formal) care descriu mediul
 - reprezentare ușor interpretabilă de către calculator → limbaj de reprezentare a cunoștințelor
 - mecanismul de obținere a unor propoziții noi pe baza celor vechi → inferență/raționare

□ Tipologie

- cunoștințe exacte (perfecte)
- cunoștințe imperfecte (nesigure, incerte)
 - Inexacte
 - Incomplete
 - Incomensurabile

□ Modalități de reprezentare a cunoștințelor

- Logica formală (limbaje formale)
- Reguli
- Rețele semantice

Sisteme inteligente – SBC

Baza de cunoștințe (BC)

- ❑ Modalități de reprezentare a cunoștințelor
 - Logica formală (limbaje formale)
 - ❑ Definiție
 - Știința principiilor formale de raționament
 - ❑ Componete
 - Syntaxă – simbolurile atomice folosite de către limbaj și regulile de construcție a expresiilor (structurilor/propozițiilor) limbajului
 - Semantică – asociază un înțeles simbolurilor și o valoare de adevăr (Adevărat sau Fals) regulilor (propozițiilor) limbajului
 - Metodă de inferență sintactică – regulile necesare determinării unei submulțimi de expresii logice → teoreme (folosite pentru obținerea de noi expresii)
 - ❑ Tipologie
 - În funcție de numărul valorilor de adevăr:
 - logică duală
 - logică polivalentă
 - În funcție de tipul elementelor de bază:
 - clasică → primitivele = propoziții (predicate)
 - probabilistică → primitivele = variabile aleatoare
 - În funcție de obiectul de lucru:
 - logica propozițională → se lucrează doar cu propoziții declarative, iar obiectele descrise sunt fixe sau unice (Ionică este student)
 - logica predicatorilor de ordin I → se lucrează cu propoziții declarative, cu predicate și cuantificări, iar obiectele descrise pot fi unice sau variabile asociate unui obiect unic (Toți studenții sunt prezenți)
 - Reguli
 - Rețele semantice

Sisteme inteligente – SBC

Baza de cunoștințe (BC)

□ Modalități de reprezentare a cunoștințelor

- Logica formală (limbaje formale)
- Reguli
 - Euristici speciale care generează informații (cunoștințe)
 - O modalitate de exprimare (reprezentare) a cunoștințelor
 - Ex. dacă Ionică lucrează la Facebook, atunci el câștigă mulți bani și are puțin timp liber
 - Interdependențele între reguli → rețea de inferență
 - Fac legătura între cauză și efect - memorate în calculator sub forma unor structuri de control
IF cauză THEN efect
- Rețelele semantice
 - Grafuri orientate cu noduri care conțin concepte și arce care reprezintă relații semantice între concepte precum:
 - *Meronymy* (A este meronym al lui B dacă A este o parte a lui B)
 - Ex. Degetul este un meronym al mâinii, roata este un meronym al mașinii
 - *Holonymy* (A este holonym al lui B dacă B este o parte a lui A)
 - Ex. Copacul este un holonym al scoarței
 - *Hyponymy* (A este hyponym al lui B dacă A este un fel de B)
 - Ex. Tractorul este un hyponym al autovehiculului
 - *Hypernymy* (A este hypernym al lui B dacă A este o generalizare al lui B)
 - Ex. Fructul este un hypernym al portocalei
 - *Synonymy* (A este sinonim al lui B dacă A denotă același lucru ca B)
 - Ex. A alerga este sinonim cu a fugi
 - *Antonymy* (A este antonim al lui B dacă A denotă lucruri opuse ca B)
 - Ex. Uscat este antonim cu ud

Sisteme inteligente – SBC

Baza de cunoștințe (BC)

- Modalități de stocare a cunoștințelor
 - Relații
 - Simple → baze de date
 - Ierarhice → ierarhii de concepte (rețele semantice)
 - Logică formală
 - Reguli
 - Logică procedurală
 - Algoritmi

Sisteme inteligente – SBC

Modulul de control (MC)

□ Conținut

- Responsabil cu inferență
- A ajunge la o concluzie plecând de la anumite premise (cunoștințe) și aplicând anumite reguli de inferență
- MC depinde de complexitate și tipul cunoștințelor cu care are de-a face

□ Tipologie

- În funcție de direcția inferenței:
 - MC cu legătură înainte (*forward chaining*)
 - Pornesc de la informația disponibilă (fapte date, condiții) și încearcă să ajungă la o concluzie (fapte derivate)
 - Se bazează pe date (*data driven*)
 - MC cu legătură înapoi (*backward chaining*)
 - Pornesc de la o concluzie potențială (ipoteză) și caută evidențe care să o suporte-contrazică (explicații)
 - Se bazează pe scop (*goal driven*)

□ Tehnici de raționare (tehnici de inferență)

- În medii certe
 - bazate pe logică
 - bazate pe reguli
- În medii incerte
 - bazate pe teoria probabilităților
 - bazate pe teoria posibilității

Sisteme inteligente – SBC

Tipologia SBC

- **Sisteme bazate pe logică (SBL)**
- **Sisteme bazate pe reguli (SBR)**
- *Case-based reasoning*
- *Hypertext manipulating systems*
- *Data bases and intelligent UI*
- *Intelligent tutoring systems*

Sisteme inteligente – SBC

Sisteme bazate pe logică (SBL)

- ❑ Conținut și obiective
- ❑ Arhitectură
- ❑ Tipologie
- ❑ Tool-uri
- ❑ Avantaje și limite

Sisteme inteligente – SBC – SBL

Conținut și obiective

□ Conținut

- explorează o multitudine de cunoștințe date pentru a obține concluzii noi despre activități dificil de examinat, folosind metode specifice logicii formale
- Un sistem logic este alcătuit din:
 - limbaj (sintaxă + semantică)
 - metodă de deducție (inferență)

□ Scopul SBL

- Rezolvarea de probleme cu ajutorul programării declarative
 - descriind ceea ce este adevărat sau nu în rezolvarea problemelor
 - permitând tehnici de raționare automată
- Exemple de probleme rezolvate de SBRL
 - demonstrarea automată a teoremelor

□ De ce se studiază SBL?

- Logica formală este precisă și definită

Sisteme inteligente – SBC – SBL

Arhitectură

❑ Baza de cunoștințe (BC)

- Sintaxă
 - ❑ simbolurile atomice folosite de către limbaj și regulile de construcție a expresiilor (structurilor/propozițiilor) limbajului
- Semantică
 - ❑ asociază un înțeles simbolurilor și o valoare de adevăr (Adevărat sau Fals) regulilor (propozițiilor) limbajului

❑ Modulul ce control (MC)

- Metodă de inferență sintactică – regulile necesare determinării unei submulțimi de expresii logice → teoreme (folosite pentru obținerea de noi expresii)

Sisteme inteligente – SBC – SBL

Tipologie

- **Sisteme bazate pe logica propozițională**
 - se lucrează doar cu propoziții declarative
 - obiectele descrise sunt unice și fixe (*Ionică este student*)
- **Sisteme bazate pe logica predicatelor de ordin I**
 - se lucrează cu propoziții declarative, cu predicate (*Student(a)*) și cuantificări (variabile cuantificabile → *pentru orice a, Student(a) → AccesWiFi(a)*)
 - obiectele descrise pot fi unice sau dinamice (variabile asociate unui obiect unic – *Toți studenții sunt prezenți*)
 - predicatele au argumente simple (*Student(a)*)
- **Sisteme bazate pe logica predicatelor de ordin superior (≥ 2)**
 - se lucrează cu propoziții declarative, cu predicate și cuantificări (variabile cuantificabile)
 - permit variabilelor să reprezinte mai multe relații între obiecte
 - predicatele pot avea argumente simple, argumente de tip predicat (*StudentSenator(Student(a))*) sau argumente de tip funcție (*Bursier(a are media peste 9.50)*)
- **Sisteme temporale**
 - Reprezintă valoarea de adevăr a faptelor de-a lungul timpului (*Ionică este uneori grăbit*)
- **Sisteme modale**
 - Reprezintă și fapte îndoioanelnice (*Ionică poate să promoveze examenul*)

Sisteme inteligente – SBC – SBL

Sisteme bazate pe logica propozițională

□ Baza de cunoștințe

- Poate fi alcătuită din:
 - Simboluri (A, B, P, Q, ...)
 - Propoziții (formule)
 - definite astfel:
 1. Un simbol
 2. Dacă P este o propoziție, atunci și $\neg P$ este tot propoziție
 3. Dacă P și Q sunt propoziții, atunci $P \wedge Q$, $P \vee Q$, $P \Rightarrow Q$, $P \Leftrightarrow Q$ sunt tot propoziții
 4. Un număr finit de aplicări ale regulilor (1) – (3)
 - Interpretare a unei propoziții \rightarrow stabilirea valorii de adevăr
 - Model \rightarrow interpretare a unei multimi de propoziții astfel încât toate propozițiile să fie adevărate

□ Modulul de control

- realizează inferență
 - stabilirea valorii de adevăr a unei propoziții obiectiv pe baza informațiilor din BC
- în mai multe moduri
 - verificarea modelului
 - enumerarea tuturor combinațiilor posibile pentru valorile de adevăr ale simbolurilor și propozițiilor implicate în SBL
 - deducția modelului cu ajutorul regulilor de inferență

Sisteme inteligente – SBC – SBL

Sisteme bazate pe logica propozițională – modulul de control

□ Problemă

- Se dă o BC = {P₁, P₂, ..., P_m} formată din simbolurile {X₁, X₂, ..., X_n} și o propoziție obiectiv O.
- Se poate deduce O din BC?

□ Verificarea modelului

■ Etape

- Se construiește tabelul corespunzător tuturor combinațiilor posibile pentru valorile de adevăr ale simbolurilor
- Se determină dacă toate modelele BC sunt și modele ale lui O
 - Modelele BC – acele linii în care toate propozițiile din BC sunt *adevărate*

■ Exemplu

- P = Afară este foarte cald
- Q = Afară este umezeală
- R = Afară plouă
- BC = {P \wedge Q \Rightarrow R, Q \Rightarrow P, Q}
- R = Va ploua?

P	Q	R	P \wedge Q \Rightarrow R	Q \Rightarrow P	Q	BC	R	BC \Rightarrow R
T	T	T	T	T	T	T	T	T
T	T	F	F	T	T	F	F	T
T	F	T	T	T	F	F	T	T
T	F	F	T	T	F	F	F	T
F	T	T	T	F	T	F	T	T
F	T	F	T	F	T	F	F	T
F	F	T	T	T	F	F	T	T
F	F	F	T	T	F	F	F	T

■ Dificultăți

- Nr tuturor combinațiilor crește exponențial cu n \rightarrow timp mare de calcul
- Soluția: deducerea prin folosirea regulilor de inferență

Sisteme inteligente – SBC – SBL

Sisteme bazate pe logica propozițională – modulul de control

□ Problemă

- Se dă o BC = {P₁, P₂, ..., P_m} formată din simbolurile {X₁, X₂, ..., X_n} și o propoziție obiectiv O.
- Se poate deduce O din BC?

□ Deducția modelului cu ajutorul regulilor de inferență

■ Etape

- Construirea unei demonstrații a valorii de adevăr a propoziției obiectiv pe baza:
 - propozițiilor
 - originale din BC
 - derivate
 - regulilor de inferență

Regulă de inferență	Premisă	Propoziția derivată
Modus ponens	A, A⇒B	B
Și introductiv	A, B	A ∧ B
Și eliminativ	A ∧ B	A
Negație dublă	¬¬A	A
Rezoluție unitară	A∨B, ¬B	A
Rezoluție	A∨B, ¬B∨C	A∨C

■ Exemplu

□ Problemă

- P = Afară este foarte cald
- Q = Afară este umezeală
- R = Afară plouă
- BC = {P∧Q⇒R, Q⇒P, Q}
- R = Va ploua?

Soluție

1. Q Premisă
2. Q⇒P Premisă
3. P Modus Ponens (1,2)
4. (P∧Q)⇒R Premisă
5. P∧Q Și introductiv (1,3)
6. R Modus Ponens (4,5)

Sisteme inteligente – SBC

Tipologia SBC

- **Sisteme bazate pe logică (SBL)**
- **Sisteme bazate pe reguli (SBR)**
- *Case-based reasoning*
- *Hypertext manipulating systems*
- *Data bases and intelligent UI*
- *Intelligent tutoring systems*

Sisteme inteligente – SBC

□ Sisteme bazate pe reguli (SBR)

- Conținut și obiective
- Proiectare
- Arhitectură
- Tool-uri și exemple
- Avantaje și limite

Sisteme inteligente – SBC – SBR

Conținut și obiective

- Conținut
 - explorează o multitudine de cunoștințe date pentru a obține concluzii noi despre activități dificil de examinat, folosind metode asemănătoare cu experții umani
 - pot avea succes la problemele fără soluție algoritmică deterministă
 - încearcă să imite un expert uman (într-un anumit domeniu)
 - SBR nu înlocuiesc experiența umană, dar îi largesc sfera disponibilității permitând ne-expertilor să lucreze mai bine → **Sisteme expert (SE)**
- Scopul SBR
 - Rezolvarea acelor tipuri de probleme care, de obicei, necesită experți umani prin
 - Transferul expertizei de la un expert la un sistem computațional și
 - Apoi la alții oameni (ne-experti)
 - Exemple de probleme rezolvate de SBR → Probleme de recomandare/consultare
 - Consultant medical – aplicație care înlocuiește medicul (dându-se simptomele, SE sugerează un diagnostic și un tratament)
 - Detector al problemelor de funcționare ale unei mașini
 - Detector de probleme în sistemele de operare - Microsoft Windows troubleshooting
 - Consultant finanicar
- De ce se studiază SBR?
 - Pentru a înțelege metodele umane de raționare
 - Expertii umani au nevoie de vacanțe, pot pleca la alte companii, se pot îmbolnăvi, cer măririi de salar, etc.
 - Au foarte mare succes comercial

Sisteme inteligente – SBC – SBR

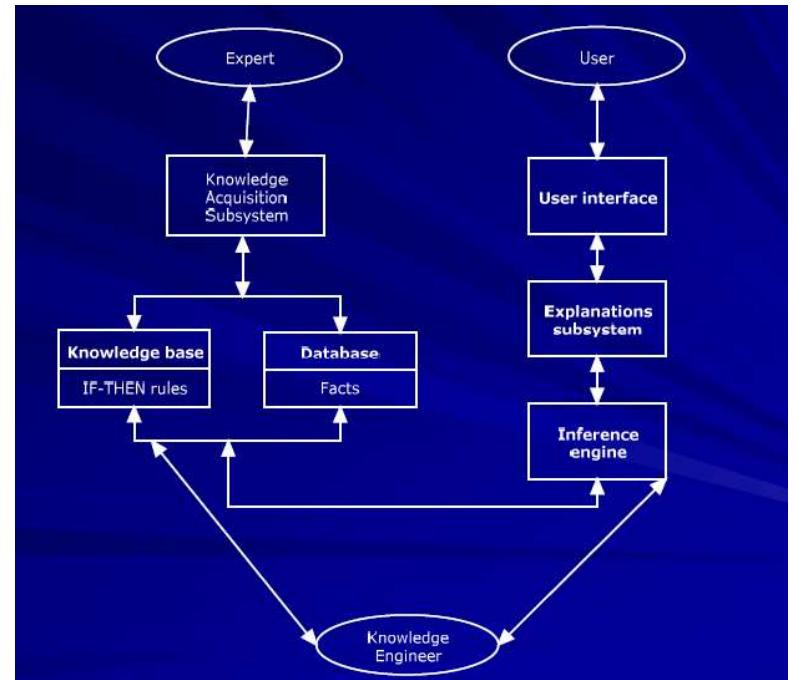
Proiectare

- Achiziționarea informațiilor (cunoștințelor)
- Reprezentarea cunoștințelor
- Inferență cunoștințelor
- Transmiterea către utilizator a cunoștințelor

Sisteme inteligente – SBC – SBR

Arhitectură

- ❑ Baza de cunoștințe (BC)
 - Informațiile specifice despre un domeniu
- ❑ Modulul de control (MC)
 - Regulile prin care se pot obține informații noi
- ❑ Interfața cu utilizatorul
 - permite dialogul cu utilizatorii în timpul sesiunilor de consultare, precum și accesul acestora la faptele și cunoștințele din BC pentru adăugare sau actualizare
- ❑ Modulul de îmbogățire a cunoașterii
 - ajută utilizatorul expert să introducă în bază noi cunoștințe într-o formă acceptată de sistem sau să actualizeze baza de cunoștințe.
- ❑ Modulul explicativ
 - are rolul de a explica utilizatorilor atât cunoștințele de care dispune sistemul, cât și raționamentele sale pentru obținerea soluțiilor în cadrul sesiunilor de consultare. Explicațiile într-un astfel de sistem, atunci când sunt proiectate corespunzător, îmbunătățesc la rândul lor modul în care utilizatorul percepe și acceptă sistemul



Sisteme inteligente – SBC – SBR

Arhitectură → baza de cunoștințe

❑ Conține

- Informațiile specifice despre un domeniu sub forma unor
 - ❑ fapte – afirmații corecte
 - ❑ reguli - euristici speciale care generează informații (cunoștințe)

❑ Rol

- stocarea tuturor elementelor cunoașterii (fapte, reguli, metode de rezolvare, euristici) specifice domeniului de aplicație, preluate de la experții umani sau din alte surse

Sisteme inteligente – SBC – SBR

Arhitectură → baza de cunoștințe

□ Fapte

- Definiție
 - Afirmații necondiționate corecte (propoziții)
 - Memorate în calculator sub forma unor structuri de date
- Exemplu
 - *Ionică lucrează la Facebook*
- Tipologie
 - În funcție de persistență (ritmul de modificare)
 - Fapte statice – aprox. permanente (*Ionică lucrează la Facebook*)
 - Fapte tranzitive – specifice unei instanțe/rulări (*Ionică este în pauza de masă*)
 - În funcție de modul de generare
 - Fapte date (*Ionică participă la ședință*)
 - Fapte deriveate – rezultate prin aplicarea unor reguli (*Dacă Ionică este PM, atunci el trebuie să conducă ședință*)

Sisteme inteligente – SBC – SBR

Arhitectură → baza de cunoștințe

□ Reguli

■ Definiție

- Euristici speciale care generează informații (cunoștințe)
- O modalitate de exprimare (reprezentare) a cunoștințelor
- Interdependențele între reguli → rețea de inferență
- Fac legătura între cauză și efect - memorate în calculator sub forma unor structuri de control
IF cauză THEN efect
 - Deducre – cauză + regulă → efect
 - Abducție – efect + regulă → cauză
 - Inducție – cauză + efect → regulă

■ Exemplu

- O cauză și mai multe consecințe (combinare cu řI)
 - *DACĂ Ionică lucrează la Facebook, ATUNCI el câștigă mulți bani řI are puțin timp liber*
- O cauză și mai multe consecințe (combinare cu SAU)
 - *DACĂ anotimpul este iarnă ATUNCI vremea este rece SAU este zăpadă*
- Mai multe cauze/antecedente (combinare cu řI) și un efect/o consecință
 - *DACĂ anotimpul este iarnă řI temperatura este sub 0 grade řI bate vântul ATUNCI nu mergem la plimbare*
- Mai multe cauze/antecedente (combinare cu SAU) și un efect/o consecință
 - *DACĂ anotimpul este iarnă SAU temperatura este sub 0 grade SAU bate vântul ATUNCI vremea este rece*
- Mai multe cauze/antecedente (combinare cu řI și SAU) și un efect/o consecință
 - *DACĂ anotimpul este iarnă řI temperatura este sub 0 grade SAU bate vântul ATUNCI avioanele nu pot ateriza*

Sisteme inteligente – SBC – SBR

Arhitectură → baza de cunoștințe

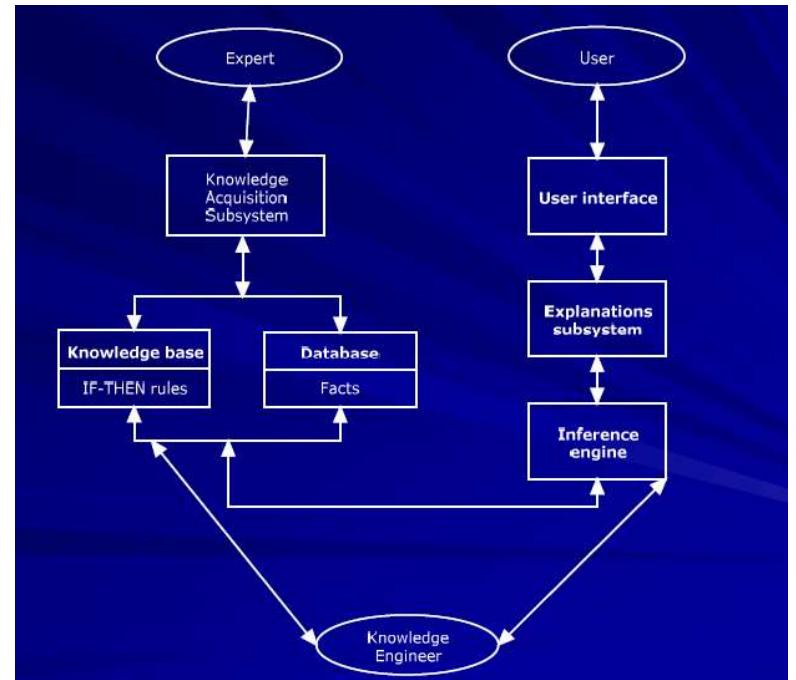
❑ Reguli

- Tipologie
 - ❑ În funcție de gradul de incertitudine
 - Reguli sigure – *Dacă ești angajat, atunci primești salar*
 - Reguli nesigure – *Dacă este iarnă, temperatura este sub 0 grade*
 - ❑ În funcție de ceea ce exprimă
 - Relații – ex. *Dacă studentul are media peste 9.50, atunci el primește bursă*
 - Recomandări – ex. *Dacă plouă, atunci să luăm umbrela*
 - Directive – ex. *Dacă bateria telefonului este gata, atunci trebuie pusă la încărcat*
 - Euristici – ex. *Dacă lumina telefonului este stinsă, atunci bateria este plină*
- Avantajul lucrului cu reguli
 - ❑ Ușor de înțeles (o formă naturală a cunoștințelor)
 - ❑ Simplu de explicat
 - ❑ Simplu de modificat și întreținut
- Limitări ale regulilor
 - ❑ Cunoștințele complexe necesită exprimarea prin foarte multe reguli
 - ❑ Căutarea în sistemele cu numeroase reguli devine greoaie

Sisteme inteligente – SBC – SBR

Arhitectură

- ❑ Baza de cunoștințe (BC)
 - Informațiile specifice despre un domeniu
- ❑ Modulul de control (MC)
 - Regulile prin care se pot obține informații noi
- ❑ Interfața cu utilizatorul
 - permite dialogul cu utilizatorii în timpul sesiunilor de consultare, precum și accesul acestora la faptele și cunoștințele din BC pentru adăugare sau actualizare
- ❑ Modulul de îmbogățire a cunoașterii
 - ajută utilizatorul expert să introducă în bază noi cunoștințe într-o formă acceptată de sistem sau să actualizeze baza de cunoștințe.
- ❑ Modulul explicativ
 - are rolul de a explica utilizatorilor atât cunoștințele de care dispune sistemul, cât și raționamentele sale pentru obținerea soluțiilor în cadrul sesiunilor de consultare. Explicațiile într-un astfel de sistem, atunci când sunt proiectate corespunzător, îmbunătățesc la rândul lor modul în care utilizatorul percepe și acceptă sistemul



Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control

- Conținut
 - Regulile prin care se pot obține informații noi
 - Algoritmi independenti de domeniu
 - Creierul SBR – un algoritm de deducere bazat pe BC și specific metodei de raționare
 - un program în care s-a implementat cunoașterea de control, procedurală sau operatorie, cu ajutorul căruia se exploatează baza de cunoștințe pentru efectuarea de raționamente în vederea obținerii de soluții, recomandări sau concluzii.
 - depinde de complexitate și tipul cunoștințelor cu care are de-a face
- Rol
 - cu ajutorul lui se exploatează baza de cunoștințe pentru efectuarea de raționamente în vederea obținerii de soluții, recomandări sau concluzii
- Tipologie – în funcție de direcția inferenței:
 - MC cu legătură înainte (*forward chaining*)
 - MC cu legătură înapoi (*backward chaining*)

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control cu legătură înainte (*forward chaining*)

□ Ideea de bază

- Se pornește de la informația disponibilă (fapte date, condiții) și se încearcă ajungerea la o concluzie (fapte derivate) folosind regulile disponibile
- Regulile sunt de forma:
 - partea stângă (PS) => partea dreaptă (PD)
 - partea de condiții => partea de consecințe (efecte)
- Se bazează pe date (*data driven*)

□ Exemplu

- **Întrebare (problemă):** *Angajatul Popescu are telefon?*
- **Regulă:** *Dacă Popescu este angajat, atunci el are telefon.*
- **Fapt curent:** *Popescu este angajat.*
- **Concluzie:** *Popescu are telefon.*

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control cu legătură înainte
(forward chaining)

□ Algoritm

■ Ciclul de execuție

- Repetă
 - Se selectează o regulă a cărei condiții din PS sunt satisfăcute de starea curentă a faptelor stocate în memoria curentă
 - Se execută PD a regulii anterior selectate (schimbând starea curentă)
- Până când nu se mai poate aplica nici o regulă

■ Observații

- Faptele sunt reprezentate în memoria curentă (de lucru) care este continuu actualizată
- Regulile reprezintă acțiuni care pot fi executate atunci când condițiile specificate sunt satisfăcute de elementele stocate în memoria curentă
- Condițiile sunt, de obicei, şabloane care se potrivesc cu elementele din memoria curentă
- Acțiunile implică, de obicei, adăugarea sau eliminarea unor elemente în memoria curentă

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control cu legătură înapoi
(backward chaining)

- Ideea de bază
 - Se pornește de la o concluzie potențială (ipoteză) și se caută evidențe care să o suporte/contrazică (explicații)
 - Regulile sunt de forma:
 - partea stângă (PS) ==>partea dreaptă (PD)
 - Se bazează pe scop (goal driven)
- Exemplu
 - **Întrebare (problemă):** *Angajatul Popescu are calculator?*
 - **Afirmație:** *Popescu are calculator.*
 - **Fapt curent:** *Popescu este programator*
 - **Regulă:** *Dacă Popescu este programator, atunci el are calculator*
 - Se verifică setul de reguli și se caută ce trebuie să fie Adevărat (în PS) pentru ca Popescu să aibă calculator: un programator. Popescu este programator este un fapt, deci atunci el are calculator.

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control cu legătură înapoi (*backward chaining*)

- Algoritm
 - Ciclul de execuție
 - Se începe cu starea obiectiv
 - Se verifică dacă obiectivul nu se potrivește cu unul din faptele inițiale. Dacă da, atunci STOP. Altfel, se caută acele reguli a căror concluzie se potrivește cu starea obiectiv.
 - Se alege una dintre reguli și se încearcă demonstrarea faptelor din precondiție (folosind același mecanism), care devin noi obiective.
 - Observații
 - este necesară memorarea obiectivelor urmărite

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

■ Baza de cunoștințe

■ Fapte

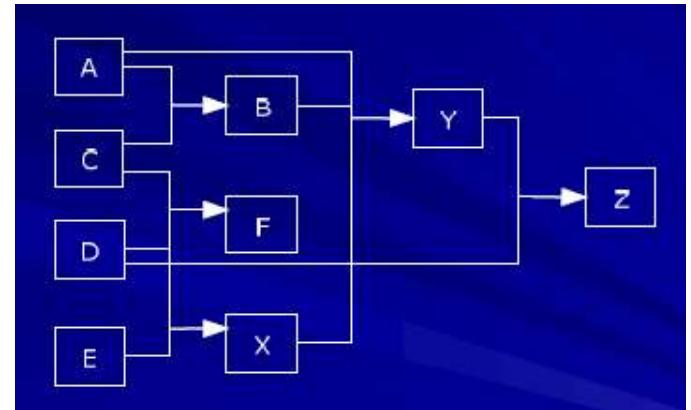
- A – secreții nazale
- B – sinuzită
- C – dureri de cap
- D - amețeli
- E – febră
- F – probleme cu tensiunea
- X – infecție
- Y – antibiotic
- Z – repaus la pat

■ Reguli

- R1: dacă A este adevărat și C este adevărat atunci B este adevărat
- R2: dacă C este adevărat și D este adevărat atunci F este adevărat
- R3: dacă C este adevărat și D este adevărat și E este adevărat atunci X este adevărat
- R4: dacă A este adevărat și B este adevărat și X este adevărat atunci Y este adevărat
- R5: dacă D este adevărat și Y este adevărat atunci Z este adevărat

■ Scop

- faptul Z



Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

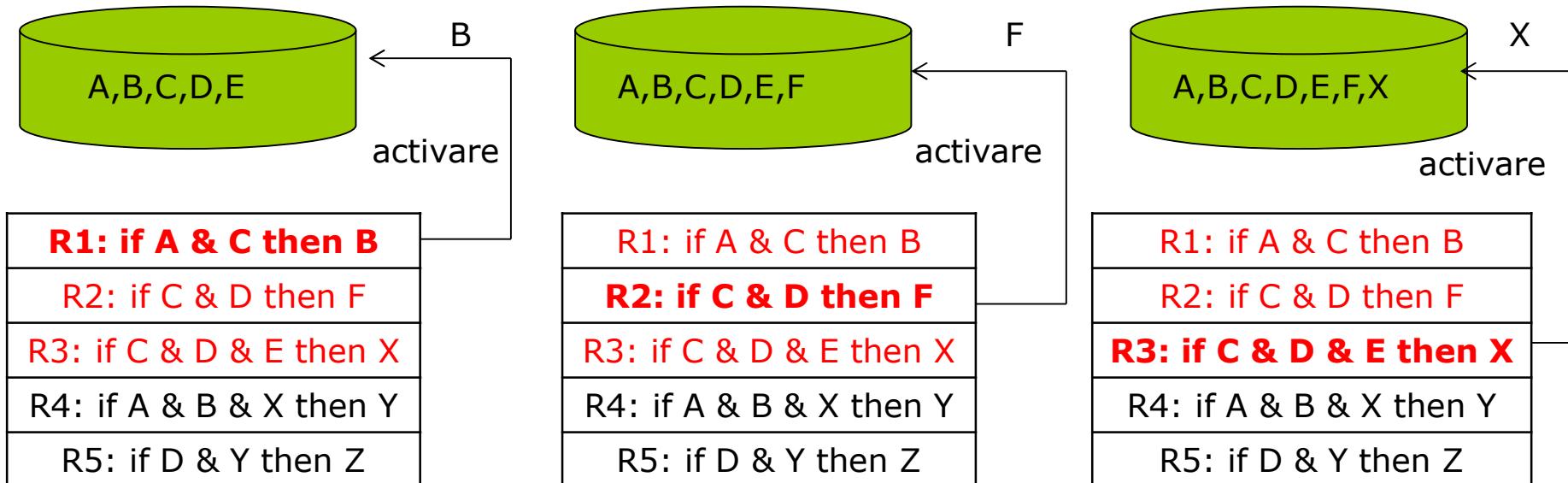
□ Algoritm *forward chaining*

- Se repetă
 - Se selectează regulile **aplicabile** pentru faptele existente în BC
 - Regulile care conțin în PS a lor doar fapte deja existente în BC
 - Dacă pentru un fapt se pot aplica mai multe reguli, se alege doar una dintre ele (care nu a mai fost folosită)
 - Se aplică regulile selectate, iar faptele noi obținute se adaugă în BC
- Până când se ajunge la concluzie sau la o regulă care indică oprirea procesului

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

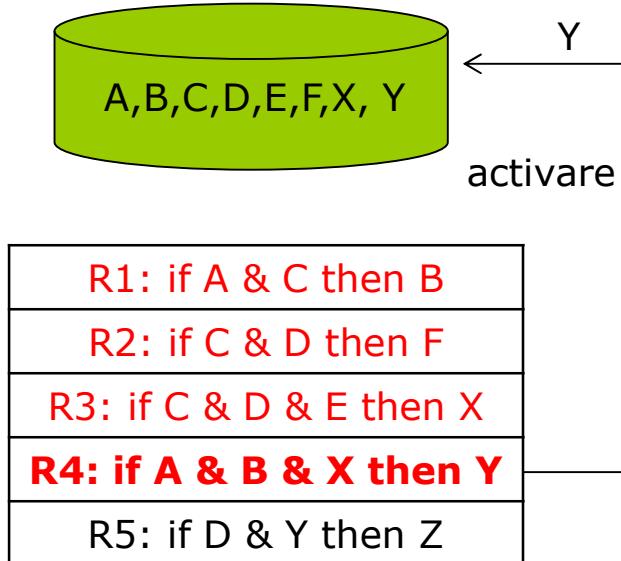
□ Iterația 1



Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

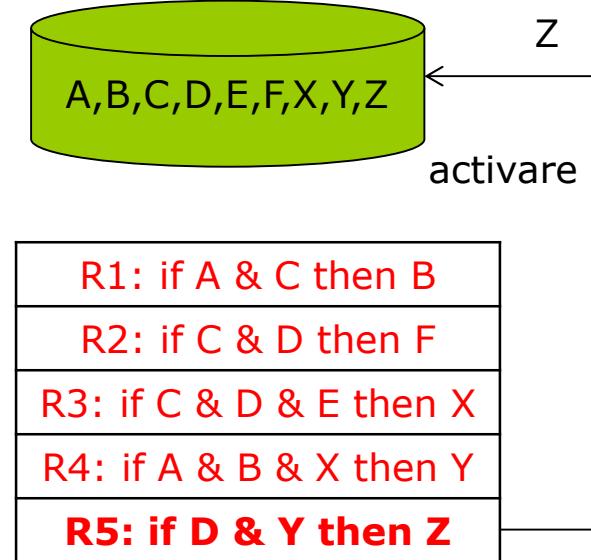
❑ Iterația 2



Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

❑ Iterația 3



Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

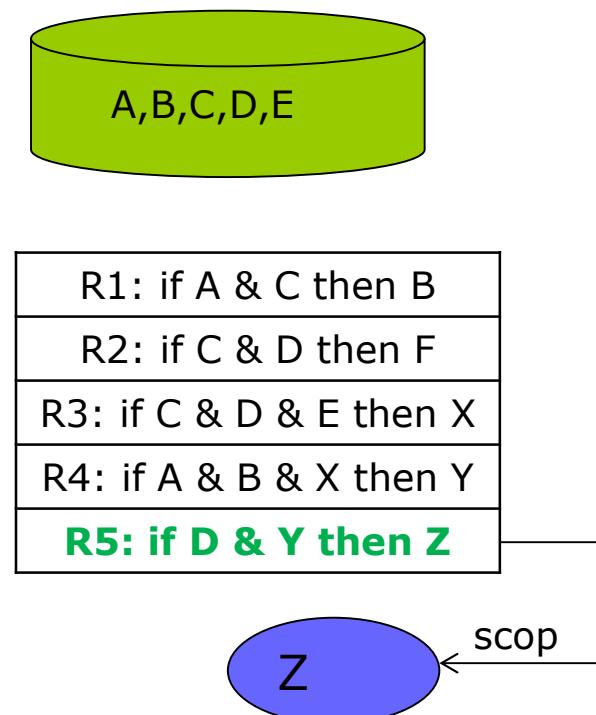
□ Algoritm *backward chaining*

- Se repetă
 - Se selectează regulile **care se potrivesc** cu scopul
 - Regulile care conțin în PD a lor scopul urmărit
 - Dacă pentru un scop se pot aplica mai multe reguli, se alege doar una dintre ele
 - Se **verifică** regulile selectate
 - Se înlocuiește scopul cu premisele (cauzele) regulii selectate, acestea devenind sub-scopuri
- Până când toate sub-scopurile sunt adevărate
 - Sunt fapte cunoscute (existente inițial în BC)
 - sunt informații oferite de utilizator
- Se repetă
 - Se **aplică** regulile anterior verificate în ordine inversă
- Până la aplicarea tuturor regulilor și obținerea scopului urmărit (ca fapt în BC)

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

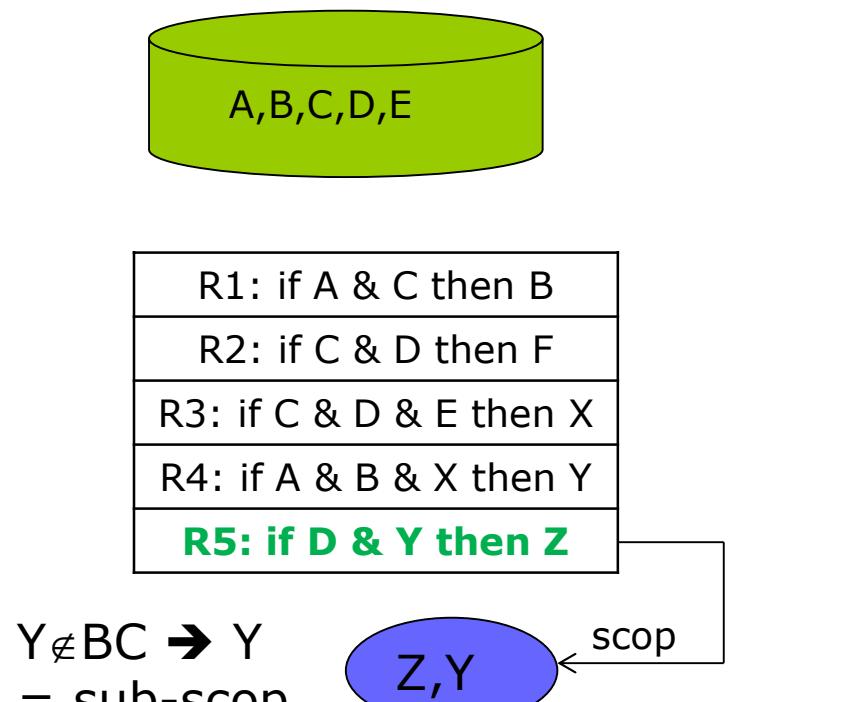
❑ Iterația 1.1



Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

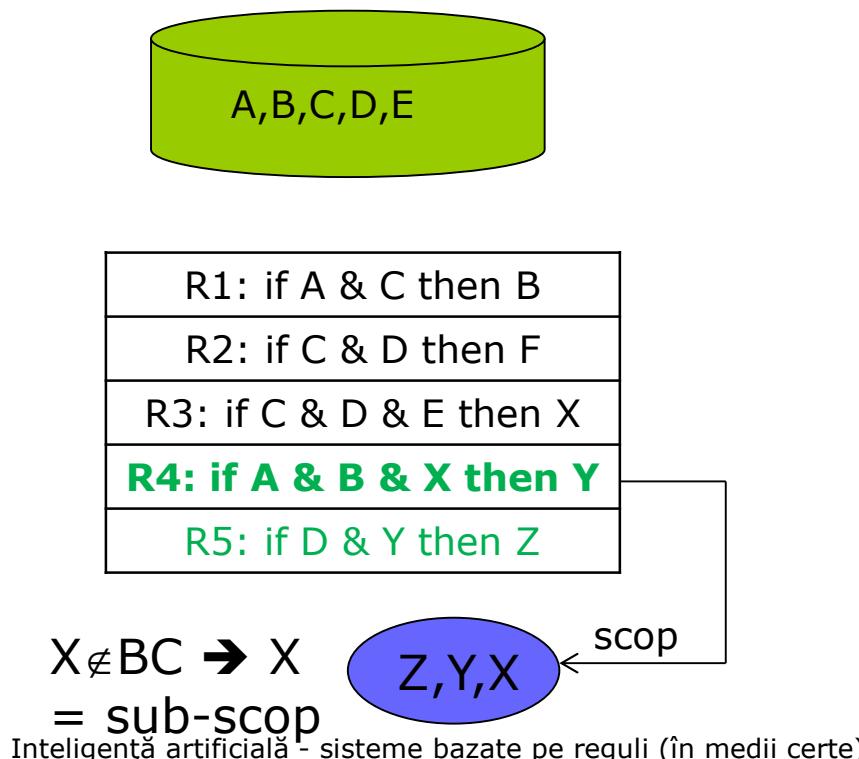
❑ Iterația 1.1



Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

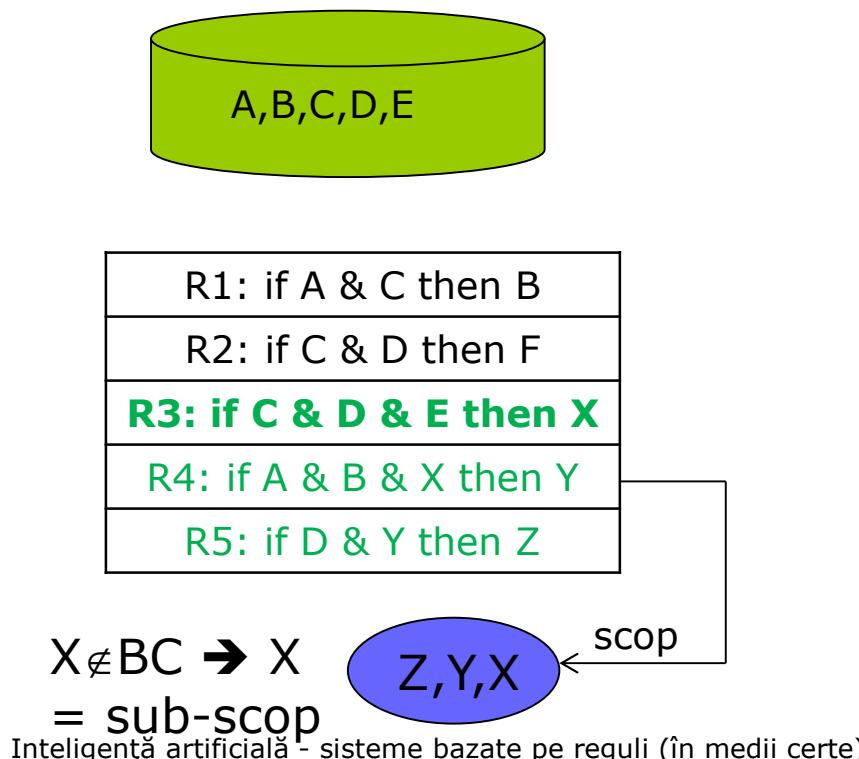
❑ Iterația 1.2



Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

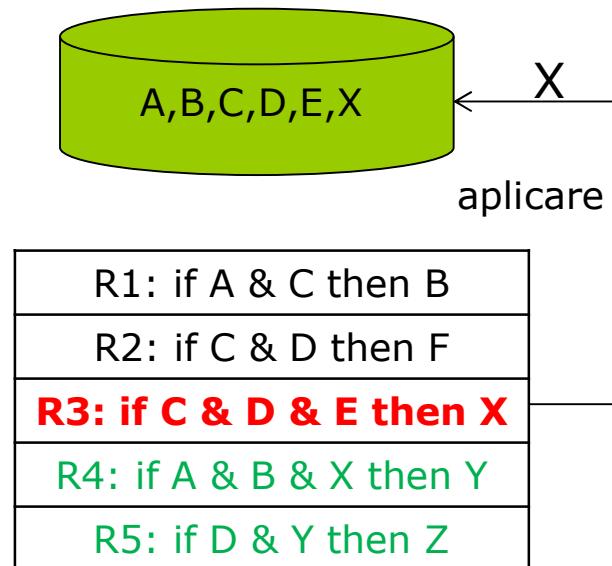
❑ Iterația 1.2



Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

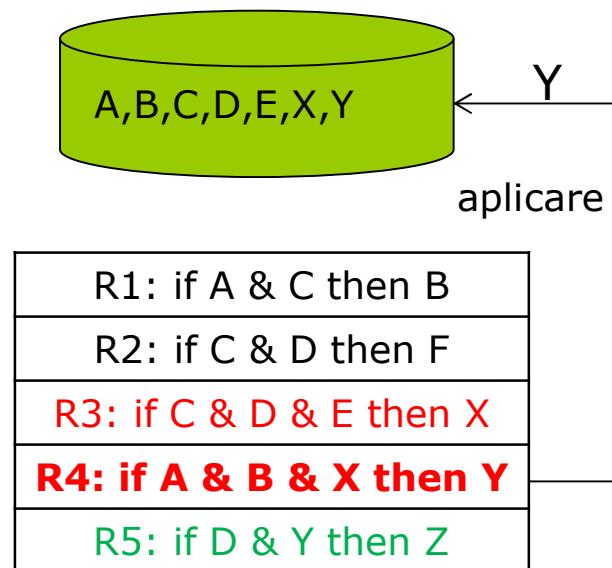
❑ Iterația 2.1



Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

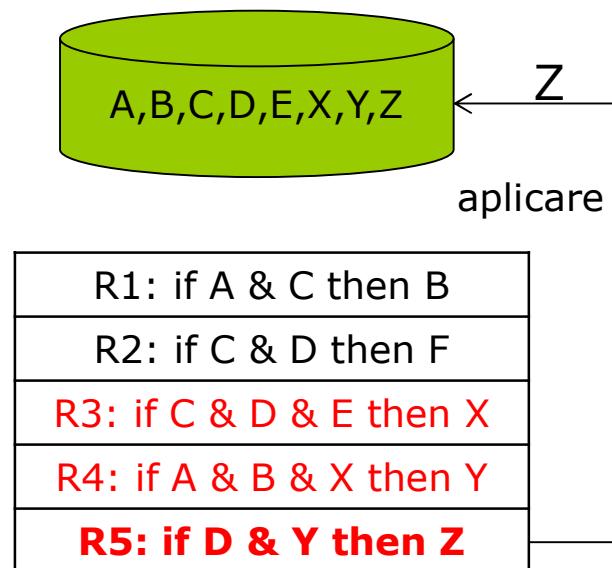
❑ Iterația 2.1



Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

❑ Iterația 2.1



Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control

❑ Dificultăți

- *Forward Chaining* (FC) sau *Backward chaining* (BC)?
- Rezolvarea conflictelor

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control – dificultăți

- ❑ *Forward Chaining* (FC) sau *Backward chaining* (BC)?
 - FC se recomandă a fi folosit atunci când:
 - ❑ Toate (sau aproape toate) informațiile se dau de la început în problemă
 - ❑ Există un număr mare de scopuri potențiale, dar doar o parte din ele sunt realizabile pentru o instanță dată a problemei
 - ❑ Este dificilă formularea unui scop sau a unor ipoteze
 - BC se recomandă a fi folosit atunci când:
 - ❑ Scopul sau ipotezele se dau în problemă sau sunt ușor de formulat
 - ❑ Există numeroase reguli care se potrivesc cu faptele din BC, producând numeroase concluzii
 - ❑ Datele problemei nu se dau (sau nu sunt ușor accesibile), dar trebuie achiziționate (în anumite sisteme)

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control – dificultăți

□ Rezolvarea conflictelor

- Dacă se pot aplica mai multe reguli, care regulă este aleasă?
 - De ex.
 - R1: *dacă culoarea este galben atunci fructul este măr*
 - R2: *dacă culoarea este galben și forma este lunguiată atunci fructul este banană*
 - R3: *dacă forma este rotundă atunci fructul este măr*
 - Strategii de alegere a regulii care se aplică
 - **prima** regulă
 - o regulă **aleatoare**
 - regula **cea mai specifică**
 - **cea mai veche** regulă
 - **cea mai bună** regulă

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control – dificultăți

- Rezolvarea conflictelor
 - Alegerea **primei** reguli care se potrivește (***First in first serve***)
 - Exemplu
 - R1: dacă culoarea este galben atunci fructul este măr
 - R2: dacă culoarea este galben și forma este lunguiată atunci fructul este banană
 - R3: dacă forma este rotundă atunci fructul este măr
 - Observații
 - Regulile sunt ordonate doar în sistemele mici

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control – dificultăți

□ Rezolvarea conflictelor

■ Alegerea **aleatoare** a unei reguli care se potrivește

□ Exemplu

- R1: dacă culoarea este galben atunci fructul este măr
- R2: dacă culoarea este galben și forma este lunguiată atunci fructul este banană
- R3: dacă forma este rotundă atunci fructul este măr

□ Observații

- Alegerea poate fi bună sau mai puțin bună

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control – dificultăți

- Rezolvarea conflictelor
 - Alegerea **celei mai specifice** reguli (*Specificity*)
 - Cea cu cele mai multe condiții, fiind cea mai relevantă pentru datele existente
 - Exemplu
 - R1: dacă culoarea este galben atunci fructul este măr
 - R2: dacă culoarea este galben și forma este lunguiată atunci fructul este banană
 - R3: dacă forma este rotundă atunci fructul este măr
 - Observații
 - O regulă specifică procesează mai multă informație decât o regulă generală
→ *longest matching strategy*

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control – dificultăți

□ Rezolvarea conflictelor

■ Alegerea **celei mai vechi** reguli utilizate (**Recency**):

- Fiecare regulă are asociată o marcă temporală – ultima dată când a fost folosită
- Exemplu
 - R1: dacă culoarea este galben atunci fructul este măr [12.01.2012 – 13.45]
 - R2: dacă culoarea este galben și forma este lunguiată atunci fructul este banană [7.02.2012 – 21.10]
 - R3: dacă forma este rotundă atunci fructul este măr [10.01.2012 – 10.25]
- Observații
 - Noile regule au fost adăugate de experți mai puțin pregătiți decât vechile regule (adăugate de experți mai bine pregătiți – cu mai multe cunoștințe în domeniu)

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control – dificultăți

□ Rezolvarea conflictelor

■ Alegerea **celei mai bune** reguli (**Prioritization**)

- Fiecare regulă are asociată o pondere care specifică importanța ei (relativ la alte reguli)

□ Exemplu

- R1: dacă culoarea este galben atunci fructul este măr [30%]
- R2: dacă culoarea este galben și forma este lunguiată atunci fructul este banană [30%]
- R3: dacă forma este rotundă atunci fructul este măr [40%]

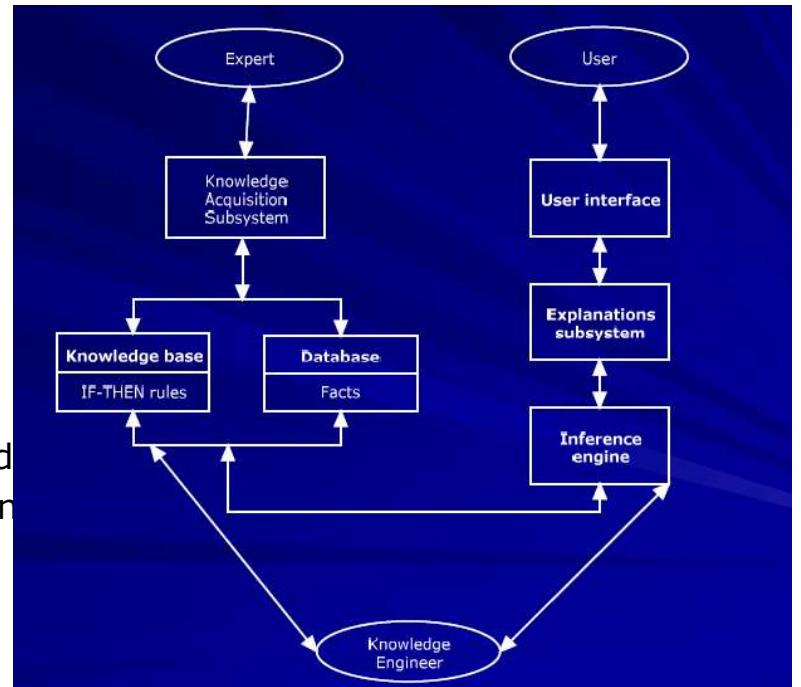
□ Observații

- Necesită expertiză umană pentru stabilirea importanței

Sisteme inteligente – SBC – SBR

Arhitectură

- ❑ Baza de cunoștințe (BC)
 - Informațiile specifice despre un domeniu
- ❑ Modulul de control (MC)
 - Regulile prin care se pot obține informații noi
- ❑ Interfața cu utilizatorul
 - permite dialogul cu utilizatorii în timpul sesiunilor de consultare, precum și accesul acestora la faptele și cunoștințele din bază pentru adăugarea sau actualizarea bazei.
- ❑ Modulul de îmbogățire a cunoașterii
 - ajută utilizatorul expert să introducă în bază noi cunoștințe într-o formă acceptată de sistem sau să actualizeze baza de cunoștințe.
- ❑ Modulul explicativ
 - are rolul de a explica utilizatorilor atât cunoștințele de care dispune sistemul, cât și raționamentele sale pentru obținerea soluțiilor în cadrul sesiunilor de consultare. Explicațiile într-un astfel de sistem, atunci când sunt proiectate corespunzător, îmbunătățesc la rândul lor modul în care utilizatorul percepe și acceptă sistemul



Sisteme inteligente – SBC – SBR

Arhitectură

- Interfața cu utilizatorul
 - Procesarea limbajului de dialog
 - Tehnici de procesare a limbajului
 - Meniuri
 - Elemente grafice, etc
- Modulul de îmbogățire a cunoașterii
 - ajută utilizatorul expert să introducă în bază noi cunoștințe într-o formă acceptată de sistem sau să actualizeze baza de cunoștințe

Sisteme inteligente – SBC – SBR

Arhitectură

- ❑ Modulul explicativ
 - are rolul de a explica utilizatorilor
 - ❑ cunoștințele de care dispune sistemul,
 - ❑ raționamentele sale pentru obținerea soluțiilor în cadrul sesiunilor de consultare.
 - explicațiile într-un astfel de sistem, atunci când sunt proiectate corespunzător, îmbunătățesc la rândul lor modul în care utilizatorul percepe și acceptă sistemul
- Exemplu
 - ❑ Un expert medical care prescrie un tratament unui pacient trebuie să explice
 - motivele pentru care a ajuns la acea recomandare
 - risurile unui astfel de tratament
 - alternative la acest tratament

Sisteme inteligente – SBC – SBR

- Conținut și obiective
- Proiectare
- Arhitectură
- Tool-uri și exemple
- Avantaje și limite

Sisteme inteligente – SBC – SBR

Tool-uri și limbaje existente

- PROLOG
 - Limbaj de programare care utilizează *backward chaining*
- ART (Inference Corporation)
 - În 1984, Inference Corporation a dezvoltat Automated Reasoning Tool (ART), un sistem expert bazat pe *forward chaining*
- CLIPS
 - NASA preia abilitățile de *forward chaining* ale sintaxei ART și dezvoltă "C Language Integrated Production System" (CLIPS)
- ART-IM (Inference Corporation)
 - Inference Corporation implementează o versiunea *forward-chaining* a ART/CLIPS, numită ART-IM.
- OPS5 (Carnegie Mellon University)
 - Primul limbaj de IA utilizat pentru sisteme de producție (XCON)
- Eclipse (The Haley Enterprise, Inc.)
 - Eclipse este singurul modul de control pentru C/C++ care suportă atât *forward chaining*, cât și *backward chaining*

Sisteme inteligente – SBC – SBR

Exemple

- DENDRAL (1965-1983)
 - Analizează structura moleculelor și propune structuri plauzibile pentru compuși chimici noi sau necunoscuți
- MYCIN (1972-1980)
 - Program interactiv pentru
 - diagnosticarea unor boli infecțioase sanguine
 - Recomandări terapeutice antimicrobiene
- EMYCIN, HEADMED, CASNET și INTERNIST
 - pentru domeniul medical
- PROSPECTOR (1974-1983)
 - Oferă recomandări pentru explorările mineralelor
- TEIRESIAS
 - pentru achiziția intelligentă a cunoașterii
- XCON (1978-1999)
 - Oferă recomandări pentru configurarea calculatoarelor
- SBR financiare
 - ExpertTAX, Risk Advisor (Coopers & Lybrand), Loan Probe, Peat/1040 (KPMG), VATIA, Flow Eval (Ernst & Young), Planet, Compas, Comet (Price Waterhouse), Rice (Arthur Andersen), Audit Planning Advisor, World Tax Planner (Deloitte Touche)

Sisteme inteligente – SBC – SBR

Avantaje și limite

□ Avantaje

- Oferă recomandări celor mai puțin experți în anumite domenii
- Permit companiilor replicarea celor mai buni angajați
 - Preia informația și cunoștințele intelectuale ale experților și le pun la dispoziția celorlalți oameni
- Se reduc erorile date de proceselor de automatizare a sarcinilor monotone, repetitive sau critice
- Se reduce necesarul de forță umană și de timp pentru testarea și analizarea datelor
- Se reduc costurile prin accelerarea procesului de observare a greșelilor
- Se elimină munca pe care oamenii nu ar trebui să o facă (dificilă, consumatoare de timp, susceptibilă de erori, care necesită antrenare lungă și costisitoare)
- Se elimină munca pe care oamenii nu și-o doresc să o facă (luarea unor decizii care nu-i pot mulțumi pe toți – sistemele expert nu pot fi acuzate de favoritsime)

□ Dezavantaje

- Domeniu îngust de aplicare a unui SBR
- Focus limitat la anumite obiective
- Lipsa capacității de învățare și adaptare
- Probleme de întreținere
- Costuri de dezvoltare mari



Recapitulare

□ SBC

- Sisteme computaționale în care
- baza de cunoștințe și modulul de control se suprapun

□ SBC pot fi

- SBL
 - ▣ explorează o multitudine de cunoștințe date pentru a obține concluzii noi despre activități dificil de examinat, folosind metode specifice logicii formale
 - ▣ Componență
 - limbaj (sintaxă + semantică) și
 - metoda de deducție (inferență)
- SBR
 - ▣ explorează o multitudine de cunoștințe date pentru a obține concluzii noi despre activități dificil de examinat, folosind metode asemănătoare cu experții umani
 - ▣ pot avea succes la problemele fără soluție algoritmică deterministică
 - ▣ încearcă să imite un expert uman (într-un anumit domeniu)
 - ▣ Componență
 - Baza de cunoștințe → fapte și reguli
 - Modulul de control → inferență înainte sau înapoi

Cursul următor

- A. Scurtă introducere în Inteligența Artificială (IA)**
- B. Rezolvarea problemelor prin căutare**
 - Definirea problemelor de căutare
 - Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială
- C. Sisteme inteligente**
 - Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
 - Sisteme hibride
 - Sisteme bazate pe reguli în medii certe
 - Sisteme bazate pe reguli în medii incerte (Bayes, factori de certitudine, Fuzzy)

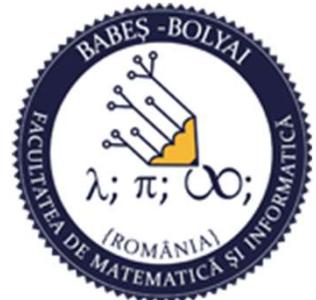
Cursul următor – Materiale de citit și legături utile

- Capitolul V din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 3 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 8 și 9 din *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

-
- Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean –
www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan -
www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop -
www.cs.ubbcluj.ro/~hfpop



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ



Sisteme inteligente

Sisteme bazate pe reguli în medii
incerte

Laura Dioșan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme hibride
- Sisteme bazate pe reguli în medii certe
- Sisteme bazate pe reguli în medii incerte (Bayes, factori de certitudine, Fuzzy)

Materiale de citit și legături utile

- capitolul V din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 3 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 8 și 9 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

Conținut

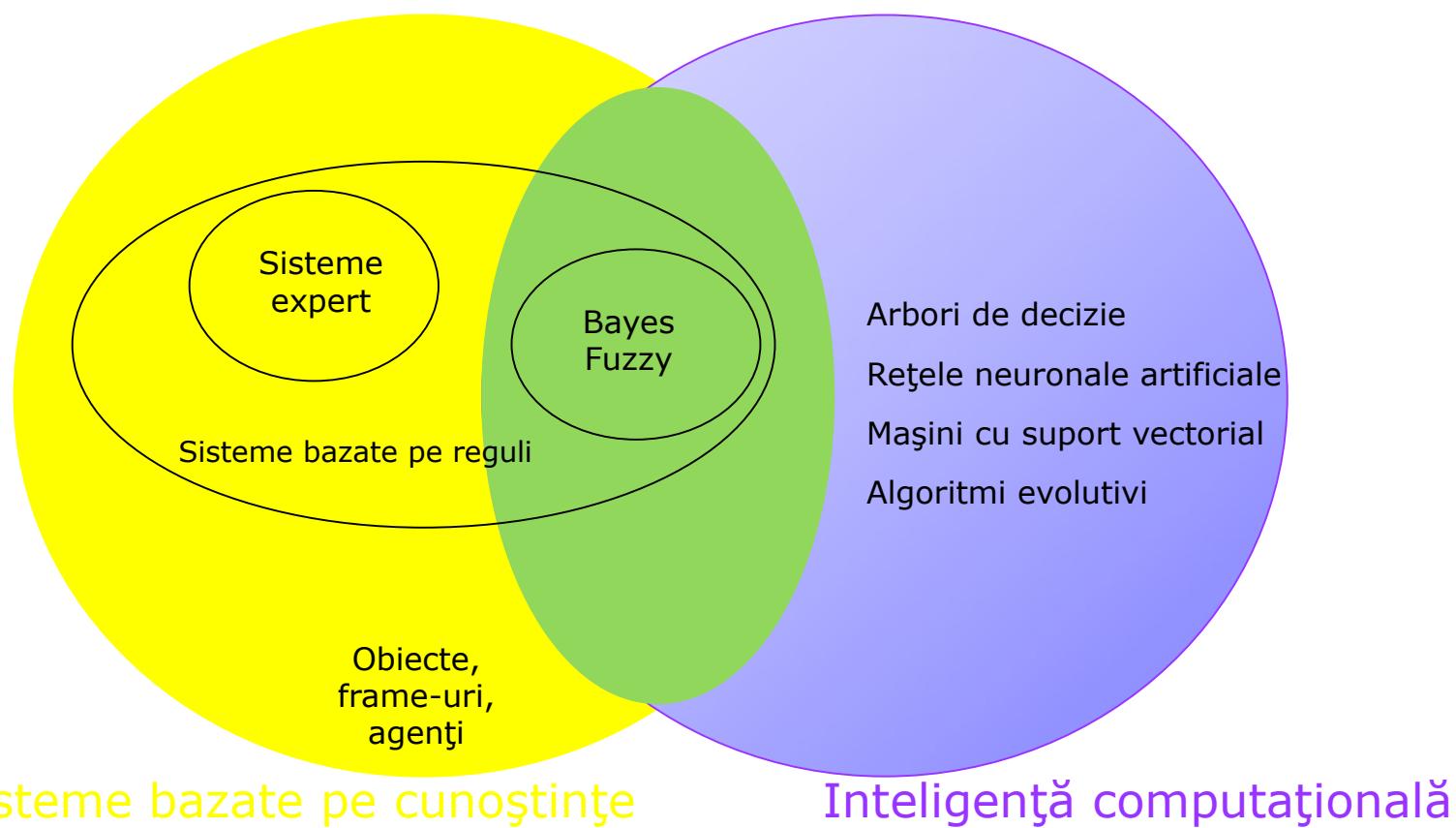
□ Sisteme inteligente

■ Sisteme bazate pe cunoștințe

□ Sisteme bazate pe reguli în medii incerte



Sisteme inteligente



Sisteme inteligente–sisteme bazate pe cunoștințe (SBC)

- sistemele computaționale – alcătuite din 2 module principale (roluri):
 - Domeniul de cunoștințe (baza de cunoștințe – BC – *knowledge base*)
 - Informațiile specifice despre un domeniu
 - Modulul de control (MC – *inference engine*)
 - Regulile prin care se pot obține informații noi
 - Algoritmi independenți de domeniu

Sisteme inteligente – SBC

Baza de cunoștințe (BC)

❑ Conținut

- Informații (exprimate într-o anumită reprezentare – ex. propoziții) despre mediu
- informații necesare pentru înțelegerea, formularea și rezolvarea problemelor
- mulțime de propoziții (exprimate/reprezentate într-un limbaj formal) care descriu mediul
 - reprezentare ușor interpretabilă de către calculator → limbaj de reprezentare a cunoștințelor
 - mecanismul de obținere a unor propoziții noi pe baza celor vechi → inferență/raționare

❑ Tipologie

- cunoștințe exacte (perfecte)
 - Raționarea exactă se bazează pe logica clasică
 - *IF A is true THEN A is \neg false*
 - *IF B is false THEN B is \neg true*
- cunoștințe imperfecte (nesigure, incerte)
 - Inexacte
 - Incomplete
 - Incomensurabile

Sisteme inteligente – SBC

- Surse ale incertitudinii
 - Imperfecțiunea regulii
 - Nesiguranța evidenței (dovezii)
 - Încrederea în concluzie trebuie scalată
 - Utilizarea unui limbaj vag, imprecis
- Moduri de exprimare a incertitudinii
 - Probabilități
 - Logica fuzzy
 - Teorema lui Bayes
 - Teoria Dempster-Shafer
- Moduri de reprezentare a incertitudinii
 - Cu ajutorul unui singur număr → factor de certitudine, confidență (încredere), valoare de adevăr
 - Cât de siguri suntem că respectivele date sunt valide
 - Cu ajutorul a 2 numere → logica intervalelor
 - Min → limita inferioară a certitudinii (încredere, necesitate)
 - Max → limita superioară a certitudinii (plauzabilitate, posibilitate)

Sisteme inteligente – SBC

□ Tehnici de raționare în medii nesigure

- Teoria Bayesiana – metodă probabilistică
- Teoria certitudinii
- Teoria posibilității (logica fuzzy)

} Metode euristice

Sisteme inteligente – SBC – factori de certitudine

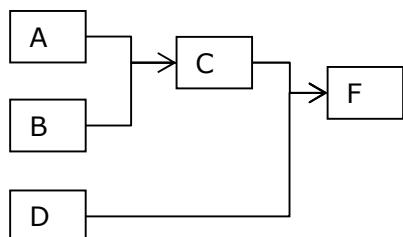
□ sisteme de tip Bayes

- SBR (Sisteme expert) în care faptele și regulile sunt probabilistice

□ sisteme cu factori de certitudine

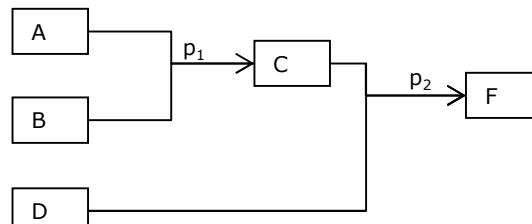
- SBR (sisteme expert) în care faptele și regulile au asociate câte un factor de certitudine (FC)/coeficient de încredere
- Un fel de sisteme de tip Bayes în care probabilitățile sunt înlocuite cu factori de certitudine

- Dacă A și B atunci C
- Dacă C și D atunci F



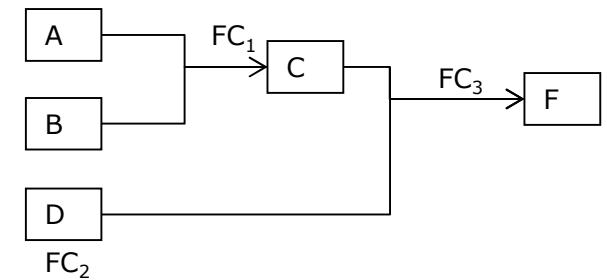
SBR classic

- Dacă A și B atunci C [cu prob p_1]
- Dacă C și D atunci F [cu prob p_2]



SBR de tip Bayes

- Dacă A și B atunci C [FC_1]
- Dacă C și D [FC_2] atunci F [FC_3]



SBR cu FC

Sisteme inteligente – SBC – factori de certitudine

□ SBR de tip Bayes vs. SBR cu FC

Bayes	FC
Teorie probabilităților este veche și fundamentată matematic	Teoria FC este nouă și fără demonstrații matematice
Necesită existența unor informații statistice	Nu necesită existența unor date statistice
Propagarea încrederei crește în timp exponențial	Informația circulă repede și eficient în SBR
Este necesar calculul apriori a probabilităților	Nu este necesar calculul apriori a probabilităților
Ipotezele susținute de probe pot să fie independente sau nu	Ipotezele susținute de probe sunt independente

Sisteme inteligente – SBC – sisteme de tip Bayes

- Elemente de teoria probabilităților
- Conținut și arhitectură
- Tipologie
- Tool-uri
- Avantaje și limite

Sisteme inteligente – SBC – sisteme de tip Bayes

Amintim componența unui SBC

- Baza de cunoștințe (BC) → Modalități de reprezentare a cunoștințelor
 - Logica formală (limbaje formale)
 - Definiție
 - Știința principiilor formale de raționament
 - Componente
 - Sintaxă
 - Semantică
 - Metodă de inferență sintactică
 - Tipologie
 - În funcție de numărul valorilor de adevăr:
 - logică duală
 - logică polivalentă
 - În funcție de tipul elementelor de bază:
 - clasică → primitivele = propoziții (predicate)
 - probabilistică → primitivele = variabile aleatoare
 - În funcție de obiectul de lucru:
 - logica propozițională → se lucrează doar cu propoziții declarative, iar obiectele descrise sunt fixe sau unice (Ionică este student)
 - logica predicatelor de ordin I → se lucrează cu propoziții declarative, cu predicate și cuantificări , iar obiectele descrise pot fi unice sau variabile asociate unui obiect unic (Toți studenții sunt prezenti)
 - Reguli
 - Rețele semantice
- Modulul de control (MC – pentru inferență)

Sisteme inteligente – SBC – sisteme de tip Bayes

Elemente de teoria probabilităților

- ❑ Teorii ale probabilităților
- ❑ Concepte de bază
 - Teoria clasică și teoria modernă
 - Eveniment
 - Probabilitate simplă
 - Probabilitate condiționată
 - Axiome

Sisteme inteligente – SBC – sisteme de tip Bayes

Elemente de teoria probabilităților

□ Teorii ale probabilităților

■ Teoria clasică (*a priori*)

- Propusă de Pascal și Fermat în 1654
- Lucrează cu sisteme ideale:
 - toate posibilele evenimente sunt cunoscute
 - toate evenimentele se pot produce cu aceeași probabilitate (sunt uniform distribuite)
- evenimente discrete
- metode combinatoriale
- spațiul rezultatelor posibile este continuu

■ Teoria modernă

- evenimente continue
- metode combinatoriale
- spațiul rezultatelor posibile este cuantificabil

Sisteme inteligente – SBC – sisteme de tip Bayes

Elemente de teoria probabilităților

□ Concepte de bază

- Considerăm un experiment care poate produce mai multe ieșiri (rezultate)
 - Ex. *Ev1: Aruncarea unui zar poate produce apariția uneia din cele 6 fețe ale zarului (deci 6 rezultate)*
- Eveniment
 - Definiție
 - producerea unui anumit rezultat
 - Ex. *Ev2: Apariția feței cu nr 3*
 - Ex. *Ev3: Apariția unei fețe cu un nr par (2,4,6)*
 - Tipologie
 - Evenimente independente și mutual exclusive
 - Nu se pot produce simultan
 - Ex. *Ev4: Apariția feței 1 la aruncarea unui zar și Ev5: Apariția feței 3 la aruncarea unui zar*
 - Dependente
 - Producerea unor evenimente afectează producerea altor evenimente
 - Ex. *Ev6: Apariția feței 6 la prima aruncare a unui zar și Ev7: Apariția unor fețe a căror numere însumate să dea 8 la 2 aruncări succesive ale unui zar*
- Multimea tuturor rezultatelor = *sample space* al experimentului
 - Ex. pentru *Ev1*: (1,2,3,4,5,6)
- Multimea tuturor rezultatelor tuturor evenimentelor posibile = *power set* (multimea părților)

Sisteme inteligente – SBC – sisteme de tip Bayes

Elemente de teoria probabilităților

□ Concepte de bază

■ Probabilitate simplă $p(A)$

- probabilitatea producerii unui eveniment A independent de alte evenimente (B)
- șansa ca acel eveniment să se producă
- proporția cazurilor de producere a evenimentului în mulțimea tuturor cazurilor posibile
- nr cazurilor favorabile / nr cazurilor posibile
- un număr real în $[0,1]$
 - 0 – imposibilitate absolută
 - 1 – posibilitate absolută
- Ex. $P(Ev1) = 1/6$, $P(Ev3) = 3/6$

■ Probabilitate condiționată $p(A|B)$

- probabilitatea producerii unui eveniment A dependentă de producerea altor evenimente (B)
- proporția cazurilor de producere a evenimentului A și a evenimentului B în mulțimea tuturor cazurilor producerii evenimentului B
- probabilitatea comună /probabilitatea lui B

$$p(A|B) = \frac{p(A \cap B)}{p(B)}$$

Sisteme inteligente – SBC – sisteme de tip Bayes

Elemente de teoria probabilităților

□ Concepte de bază

■ Axiome

- $0 \leq p(E) \leq 1$ pentru orice eveniment E
- $p(Adevărat) = 1, p(Fals) = 0$

- $\sum_i p(E_i) = 1$

- Dacă A și B sunt independente
 - $p(A \cup B) = p(A) + p(B)$
 - $p(A \cap B) = p(A) * p(B)$

- Dacă A și B nu sunt independente
 - Dacă A depinde de B
 - $p(A \cup B) = p(A) + p(B) - p(A \cap B)$
 - $p(A \cap B) = p(A|B) * p(B)$
 - $p(B \cap A) = p(A \cap B)$
 - $p(A|B) = \frac{p(B|A)p(A)}{p(B)}$ (b)

- Dacă A depinde de B_1, B_2, \dots, B_n (evenimente mutual exclusive)
 - $p(A) = \sum_{i=1}^n p(A | B_i) p(B_i)$ (a)

Sisteme inteligente – SBC – sisteme de tip Bayes

Elemente de teoria probabilităților

□ Concepte de bază

■ Exemplu

- Dacă A depinde de 2 evenimente mutual exclusive (B și $\neg B$), FC ec.

$$p(A) = \sum_{i=1}^n p(A|B_i)p(B_i) \quad \text{avem:}$$

- $p(B) = p(B|A)p(A) + p(B|\neg A)p(\neg A)$

- Înlocuind pe $p(B)$ în ec. $p(A|B) = \frac{p(B|A)p(A)}{p(B)}$ se obține ec.:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B|A)p(A) + p(B|\neg A)p(\neg A)} \quad (\text{c})$$

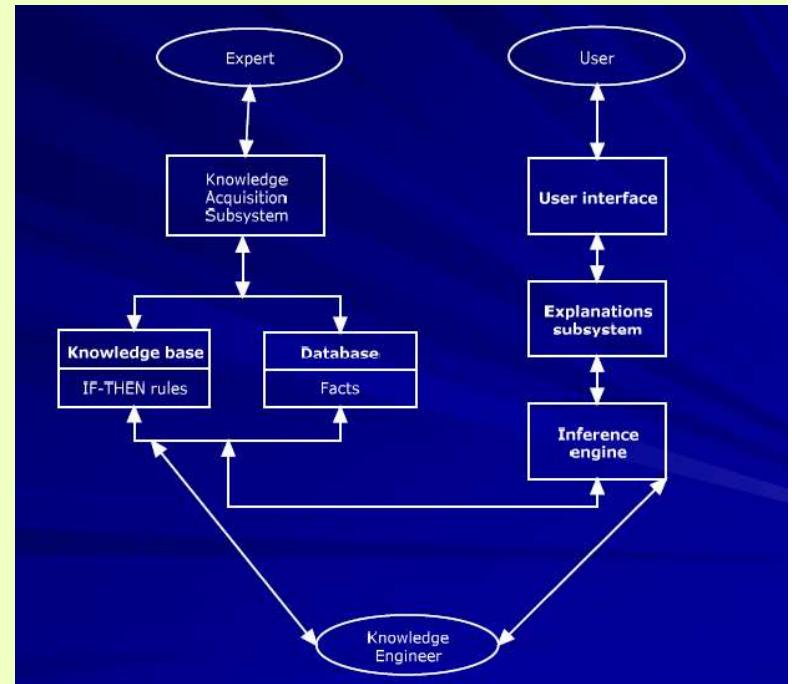
- Ecuația (c) se folosește pentru controlul incertitudinii în sistemele expert

Sisteme inteligente – SBC – sisteme de tip Bayes

Reamintim ca un SBR are următoarea

Arhitectură

- Baza de cunoștințe (BC)
 - Informațiile specifice despre un domeniu
- Modulul de control (MC)
 - Regulile prin care se pot obține informații noi
- Interfața cu utilizatorul
 - permite dialogul cu utilizatorii în timpul sesiunilor de consultare, precum și accesul acestora la faptele și cunoștințele din BC pentru adăugare sau actualizare
- Modulul de îmbogățire a cunoașterii
 - ajută utilizatorul expert să introducă în bază noi cunoștințe într-o formă acceptată de sistem sau să actualizeze baza de cunoștințe.
- Modulul explicativ
 - are rolul de a explica utilizatorilor atât cunoștințele de care dispune sistemul, cât și raționamentele sale pentru obținerea soluțiilor în cadrul sesiunilor de consultare. Explicațiile într-un astfel de sistem, atunci când sunt proiectate corespunzător, îmbunătățesc la rândul lor modul în care utilizatorul percepă și acceptă sistemul



Sisteme inteligente – SBC – sisteme de tip Bayes

Reamintim: SBR – arhitectură

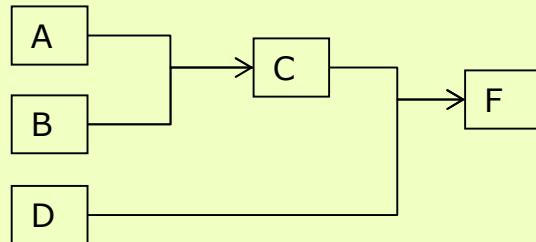
- baza de cunoștințe
 - Conținut
 - Informațiile specifice despre un domeniu sub forma unor
 - fapte – afirmații corecte
 - reguli - euristici speciale care generează informații (cunoștințe)
 - Rol
 - stocarea tuturor elementelor cunoașterii (fapte, reguli, metode de rezolvare, euristici) specifice domeniului de aplicație, preluate de la experții umani sau din alte surse
- modulul de control
 - Conținut
 - regulile prin care se pot obține informații noi
 - algoritmi independenți de domeniu
 - creierul SBR – un algoritm de deducere bazat pe BC și specific metodei de raționare
 - un program în care s-a implementat cunoașterea de control, procedurală sau operatorie, cu ajutorul căruia se exploatează baza de cunoștințe pentru efectuarea de raționamente în vederea obținerii de soluții, recomandări sau concluzii.
 - depinde de complexitate și tipul cunoștințelor cu care are de-a face
 - Rol
 - cu ajutorul lui se exploatează baza de cunoștințe pentru efectuarea de raționamente în vederea obținerii de soluții, recomandări sau concluzii

Sisteme inteligente – SBC – sisteme de tip Bayes

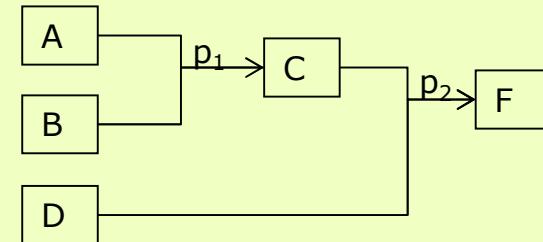
Conținut și arhitectură

□ Ideea de bază

- SBR (Sisteme expert) în care faptele și regulile sunt probabilistice
- Dacă A și B atunci C
- Dacă C și D atunci F
- Dacă A și B atunci C [cu probabilitatea p_1]
- Dacă C și D atunci F [cu probabilitatea p_2]



SBR classic



SBR de tip Bayes

Sisteme inteligente – SBC – sisteme de tip Bayes

Conținut și arhitectură

❑ Regulile din BC sunt (în general) de forma:

- Dacă evenimentul (faptul) I este adevărat, atunci evenimentul (faptul) D este adevărat [cu probabilitatea p]
- Dacă evenimentul I s-a produs, atunci evenimentul D se va produce cu probabilitatea p
 - ❑ I – ipoteza (asertiune, concluzie)
 - ❑ D – dovada (premisa) care susține ipoteza

$$p(I|D) = \frac{p(D|I)p(I)}{p(D|I)p(I) + p(D|\neg I)p(\neg I)} \quad (d)$$

■ unde:

- ❑ $p(I)$ – probabilitatea apriori ca ipoteza I să fie adevărată
- ❑ $p(D|I)$ – probabilitatea ca ipoteza I fiind adevărată să implice dovada D
- ❑ $p(\neg I)$ – probabilitatea apriori ca ipoteza I să fie falsă
- ❑ $p(D|\neg I)$ - probabilitatea găsirii dovezii D chiar dacă ipoteza I este falsă

❑ Cum și cine calculează aceste probabilități? → modulul de control

Sisteme inteligente – SBC – sisteme de tip Bayes

Conținut și arhitectură

□ Cum calculează MC aceste probabilități într-un SBR?

$$p(I|D) = \frac{p(D|I)p(I)}{p(D|I)p(I) + p(D|\neg I)p(\neg I)} \quad (d)$$

- utilizatorul furnizează informații privind dovezile observate
 - experții determină probabilitățile necesare rezolvării problemei
 - Probabilități apriori pentru posibile ipoteze (adevărate sau false) – $p(I)$ și $p(\neg I)$
 - Probabilitățile condiționate pentru observarea dovezii D dacă ipoteza I este adevărată $p(D|I)$, respectiv falsă $p(D|\neg I)$
 - SBR calculează probabilitatea posteriori $p(I|D)$ pentru ipoteza I în condițiile dovezilor D furnizate de utilizator
-
- Actualizare de tip Bayes
 - O tehnică de actualizare a probabilității p asociate unei reguli care susține o ipoteză pe baza dovezilor (pro sau contra)
 - Inferență (raționament) de tip Bayes

Sisteme inteligente – SBC – sisteme de tip Bayes

Conținut și arhitectură

□ Actualizare de tip Bayes

- O tehnică de actualizare a probabilității p asociate unei reguli care susține o ipoteză pe baza dovezilor (pro sau contra)

- Actualizarea poate să conteze:

- una sau mai multe (m) ipoteze (exclusive și exhaustive)
 - una sau mai multe (n) dovezi (exclusive și exhaustive)

- Cazuri:

- Mai multe ipoteze și o singură dovedă

$$p(I_i | D) = \frac{p(D | I_i)p(I_i)}{\sum_{k=1}^m p(D | I_k)p(I_k)}$$

- Mai multe ipoteze și mai multe dovezi

$$p(I_i | D_1D_2...D_n) = \frac{p(D_1D_2...D_n | I_i)p(I_i)}{\sum_{k=1}^m p(D_1D_2...D_n | I_k)p(I_k)} = \frac{p(D_1 | I_i)p(D_2 | I_i)...p(D_n | I_i)p(I_i)}{\sum_{k=1}^m p(D_1D_2...D_n | I_k)p(I_k)}$$

Sisteme inteligente – SBC – sisteme de tip Bayes

Conținut și arhitectură

□ Exemplu numeric

■ Pp. un SBR în care:

- utilizatorul
 - furnizează 3 dovezi condiționate independente D_1 , D_2 și D_3
- expertul
 - crează 3 ipoteze mutual exclusive și exhaustive I_1 , I_2 și I_3 și stabilește probabilitățile asociate lor – $p(I_1)$, $p(I_2)$ și $p(I_3)$
 - determină probabilitățile condiționate pentru observarea fiecărei dovezi pentru toate ipotezele posibile

probabilitatea	Ipotezele		
	$i = 1$	$i = 2$	$i = 3$
$p(I_i)$	0.40	0.35	0.25
$p(D_1 I_i)$	0.30	0.80	0.50
$p(D_2 I_i)$	0.90	0.00	0.70
$p(D_3 I_i)$	0.60	0.70	0.90

Sisteme inteligente – SBC – sisteme de tip Bayes

Conținut și arhitectură

❑ Exemplu numeric

- Presupunem că prima dovardă observată este D_3
- SE calculează probabilitățile posterioare $p(I_i | D_3)$ pentru toate ipotezele:

$$p(I_1 | D_3) = \frac{0.60 \cdot 0.40}{0.60 \cdot 0.40 + 0.70 \cdot 0.35 + 0.90 \cdot 0.25} = 0.34$$

$$p(I_2 | D_3) = \frac{0.70 \cdot 0.35}{0.60 \cdot 0.40 + 0.70 \cdot 0.35 + 0.90 \cdot 0.25} = 0.34$$

$$p(I_3 | D_3) = \frac{0.90 \cdot 0.25}{0.60 \cdot 0.40 + 0.70 \cdot 0.35 + 0.90 \cdot 0.25} = 0.32$$

- După observarea dovezii D_3
 - ❑ Încrederea în ipoteza I_2 este aceeași cu încrederea în ipoteza I_1
 - ❑ Încrederea în ipoteza I_3 crește

probabilitatea	Ipotezele		
	$i = 1$	$i = 2$	$i = 3$
$p(I_i)$	0.40	0.35	0.25
$p(D_1 I_i)$	0.30	0.80	0.50
$p(D_2 I_i)$	0.90	0.00	0.70
$p(D_3 I_i)$	0.60	0.70	0.90

Sisteme inteligente – SBC – sisteme de tip Bayes

Conținut și arhitectură

□ Exemplu numeric

- Presupunem că a doua dovedă observată este D_1
- SE calculează probabilitățile posterioare $p(I_i|D_1D_3)$ pentru toate ipotezele:

$$p(I_1 | D_1D_3) = \frac{0.30 \cdot 0.60 \cdot 0.40}{0.30 \cdot 0.60 \cdot 0.40 + 0.80 \cdot 0.70 \cdot 0.35 + 0.50 \cdot 0.90 \cdot 0.25} = 0.19$$

$$p(I_2 | D_1D_3) = \frac{0.80 \cdot 0.70 \cdot 0.35}{0.30 \cdot 0.60 \cdot 0.40 + 0.80 \cdot 0.70 \cdot 0.35 + 0.50 \cdot 0.90 \cdot 0.25} = 0.52$$

$$p(I_3 | D_1D_3) = \frac{0.50 \cdot 0.90 \cdot 0.25}{0.30 \cdot 0.60 \cdot 0.40 + 0.80 \cdot 0.70 \cdot 0.35 + 0.50 \cdot 0.90 \cdot 0.25} = 0.29$$

- După observarea dovezii D_1
 - Încrederea în ipoteza I_1 scade
 - Încrederea în ipoteza I_2 crește (fiind cea mai probabilă de a fi adevărată)
 - Încrederea în ipoteza I_3 crește

probabilitatea	Ipotezele		
	$i = 1$	$i = 2$	$i = 3$
$p(I_i)$	0.40	0.35	0.25
$p(D_1 I_i)$	0.30	0.80	0.50
$p(D_2 I_i)$	0.90	0.00	0.70
$p(D_3 I_i)$	0.60	0.70	0.90

Sisteme inteligente – SBC – sisteme de tip Bayes

Conținut și arhitectură

❑ Exemplu numeric

- Presupunem că ultima dovedă observată este D_2
- Se calculează probabilitățile posterioare $p(I_i | D_2 D_1 D_3)$ pentru toate ipotezele:

$$p(I_1 | D_2 D_1 D_3) = \frac{0.90 \cdot 0.30 \cdot 0.60 \cdot 0.40}{0.90 \cdot 0.30 \cdot 0.60 \cdot 0.40 + 0.00 \cdot 0.80 \cdot 0.70 \cdot 0.35 + 0.70 \cdot 0.50 \cdot 0.90 \cdot 0.25} = 0.45$$

$$p(I_2 | D_2 D_1 D_3) = \frac{0.00 \cdot 0.80 \cdot 0.70 \cdot 0.35}{0.90 \cdot 0.30 \cdot 0.60 \cdot 0.40 + 0.00 \cdot 0.80 \cdot 0.70 \cdot 0.35 + 0.70 \cdot 0.50 \cdot 0.90 \cdot 0.25} = 0.00$$

$$p(I_3 | D_2 D_1 D_3) = \frac{0.70 \cdot 0.50 \cdot 0.90 \cdot 0.25}{0.90 \cdot 0.30 \cdot 0.60 \cdot 0.40 + 0.00 \cdot 0.80 \cdot 0.70 \cdot 0.35 + 0.70 \cdot 0.50 \cdot 0.90 \cdot 0.25} = 0.55$$

- După observarea dovezii D_2
 - ❑ Încrederea în ipoteza I_1 crește
 - ❑ Încrederea în ipoteza I_2 e nulă (ipoteza e falsă)
 - ❑ Încrederea în ipoteza I_3 crește

probabilitatea	Ipotezele		
	$i = 1$	$i = 2$	$i = 3$
$p(I_i)$	0.40	0.35	0.25
$p(D_1 I_i)$	0.30	0.80	0.50
$p(D_2 I_i)$	0.90	0.00	0.70
$p(D_3 I_i)$	0.60	0.70	0.90

Sisteme inteligente – SBC – sisteme de tip Bayes

Conținut și arhitectură

□ Exemplu practic

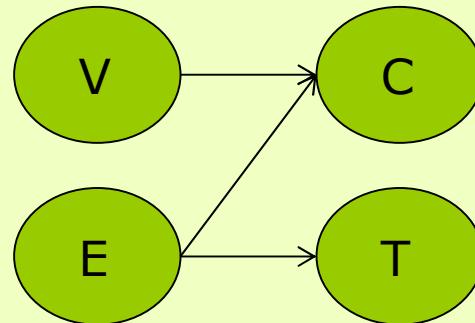
- Presupunem cazul unei mașini care nu pornește când este accelerată, dar scoate fum
 - Dacă scoate fum, atunci acceleratia este defectă [cu probabilitatea 0.7]
 - $P(I_1|D_1) = 0.7$
- Pe baza unor observări statistice, experții au constatat:
 - următoarea regulă:
 - Dacă acceleratia este defectă, atunci mașina scoate fum [cu probabilitatea 0.85]
 - probabilitatea ca mașina să pornească din cauză că acceleratia este defectă = 0.05 (probabilitate *apriori*)
 - deci avem
 - 2 ipoteze:
 - I_1 : acceleratia este defectă
 - I_2 : acceleratia nu este defectă
 - o dovedă
 - D_1 : mașina scoate fum
 - probabilitatea că acceleratia este defectă dacă mașina scoate fum
 - $P(I_1|D_1)=p(D_1|I_1)*p(I_1)/(p(D_1|I_1)*p(I_1)+p(D_1|I_2)*p(I_2))$
 - $P(I_1|D_1)=0.23 < 0.7$

	I_1	I_2
$p(I_i)$	0.05	$1-0.05=0.95$
$P(D_1 I_i)$	0.85	$1-0.85=0.15$

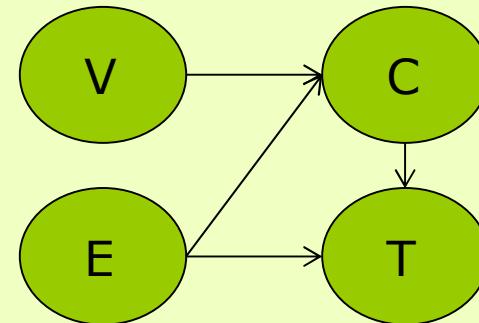
Sisteme inteligente – SBC – sisteme de tip Bayes

Tipologie

- ❑ Sisteme simple de tip Bayes
 - Consecințele unei ipoteze nu sunt corelate
- ❑ Rețele de tip Bayes
 - Consecințele unei ipoteze pot fi corelate
- ❑ De exemplu, reținem informații despre vârstă (V), educația (E), câștigurile (C) și preferința pentru teatru (T) ale unor persoane



Sistem Bayes simplu (naiv)



Rețea Bayes simplu

Sisteme inteligente – SBC – sisteme de tip Bayes

□ Tool-uri

- MSBNx – [view](#)
- JavaBayes – [view](#)
- BNJ – [view](#)

Sisteme inteligente – SBC – sisteme de tip Bayes

□ Avantaje ale inferenței de tip Bayes

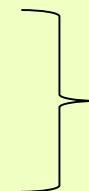
- Tehnică bazată pe teoreme statistice
- Probabilitatea dovezilor (simptomelor) în ipotezele (cauzele) date sunt posibil de furnizat
- Probabilitatea unei ipoteze se poate modifica datorită uneia sau mai multor dovezi

□ Dezavantaje ale inferenței de tip Bayes

- Trebuie cunoscute (sau ghicite) probabilitățile apriori ale unor ipoteze

Sisteme inteligente – SBC

- Tehnici de raționare în medii nesigure
 - Teoria Bayesiană – metodă probabilistică
 - **Teoria certitudinii**
 - Teoria posibilității (logica fuzzy)



Metode
euristice

Sisteme inteligente – SBC – factori de certitudine

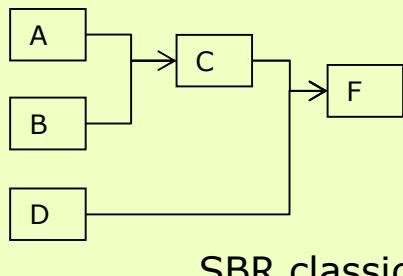
- Conținut și arhitectură
- Tipologie
- Tool-uri
- Avantaje și dezavantaje

Sisteme inteligente – SBC – factori de certitudine

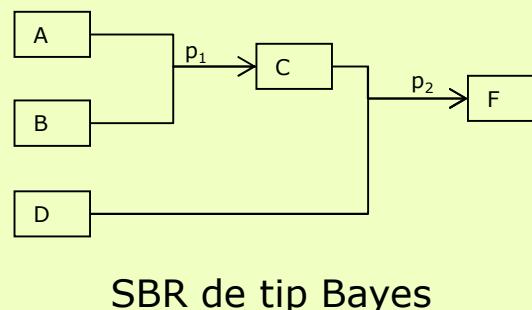
Conținut și arhitectură

□ Ideea de bază

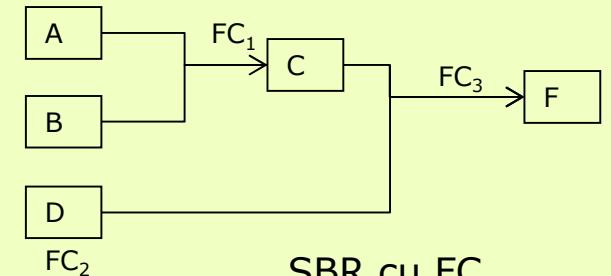
- SBR (sisteme expert) în care faptele și regulile au asociate câte un factor de certitudine (FC)/coeficient de încredere
 - Un fel de sisteme de tip Bayes în care probabilitățile sunt înlocuite cu factori de certitudine
-
- Dacă A și B atunci C
 - Dacă C și D atunci F
 - Dacă A și B atunci C [cu prob p_1]
 - Dacă C și D atunci F [cu prob p_2]
 - Dacă A și B atunci C [FC_1]
 - Dacă C și D [FC_2] atunci F [FC_3]



SBR classic



SBR de tip Bayes



SBR cu FC

Sisteme inteligente – SBC – factori de certitudine

Conținut și arhitectură

- ❑ FC măsoară încrederea acordată unor fapte sau reguli
- ❑ Utilizarea FC → alternativă la actualizarea de tip Bayes

- ❑ FC pot fi aplicați
 - faptelor
 - regulilor (concluziei/concluziilor unei reguli)
 - fapte + reguli

- ❑ Într-un SBR (sistem expert) cu factori de certitudine
 - regulile sunt de forma:
 - ❑ dacă dovada atunci ipoteza [FC]
 - ❑ dacă dovada_[FC] atunci ipoteza
 - ❑ dacă dovada_[FC] atunci ipoteza [FC]
 - ipotezele susținute de probe sunt independente

Sisteme inteligente – SBC – factori de certitudine

Conținut și arhitectură

□ FC – mod de calcul

■ Măsura încrederei (measure of belief – MB)

- măsura creșterii încrederei în ipoteza I pe baza dovezii D

$$MB(I, D) = \begin{cases} 1, & \text{dacă } p(I) = 1 \\ \frac{p(I|D) - p(I)}{1 - p(I)} & \text{dacă } p(I) < 1 \end{cases}$$

■ Măsura neîncrederei (measure of disbelief – MD)

- măsura creșterii neîncrederei în ipoteza I pe baza dovezii D

$$MD(I, D) = \begin{cases} 1, & \text{dacă } p(I) = 0 \\ \frac{p(I) - p(I|D)}{p(I)} & \text{dacă } p(I) > 0 \end{cases}$$

■ Pentru evitarea valorilor negative ale MB și MD:

$$MB(I, D) = \begin{cases} \frac{\max\{p(I|D), p(I)\} - p(I)}{1 - p(I)} & \text{dacă } p(I) = 1 \\ \frac{1}{1 - p(I)} & \text{dacă } p(I) < 1 \end{cases}$$

$$MD(I, D) = \begin{cases} \frac{\min\{p(I|D), p(I)\} - p(I)}{0 - p(I)} & \text{dacă } p(I) = 0 \\ \frac{1}{0 - p(I)} & \text{dacă } p(I) > 0 \end{cases}$$

■ FC – încrederea în ipoteza I dată fiind dovada D

- Număr din $[-1, 1]$
- $FC = -1$ dacă se știe că ipoteza I este falsă
- $FC = 0$ dacă nu se știe nimic despre ipoteza I
- $FC = 1$ dacă se știe că ipoteza I este adevărată

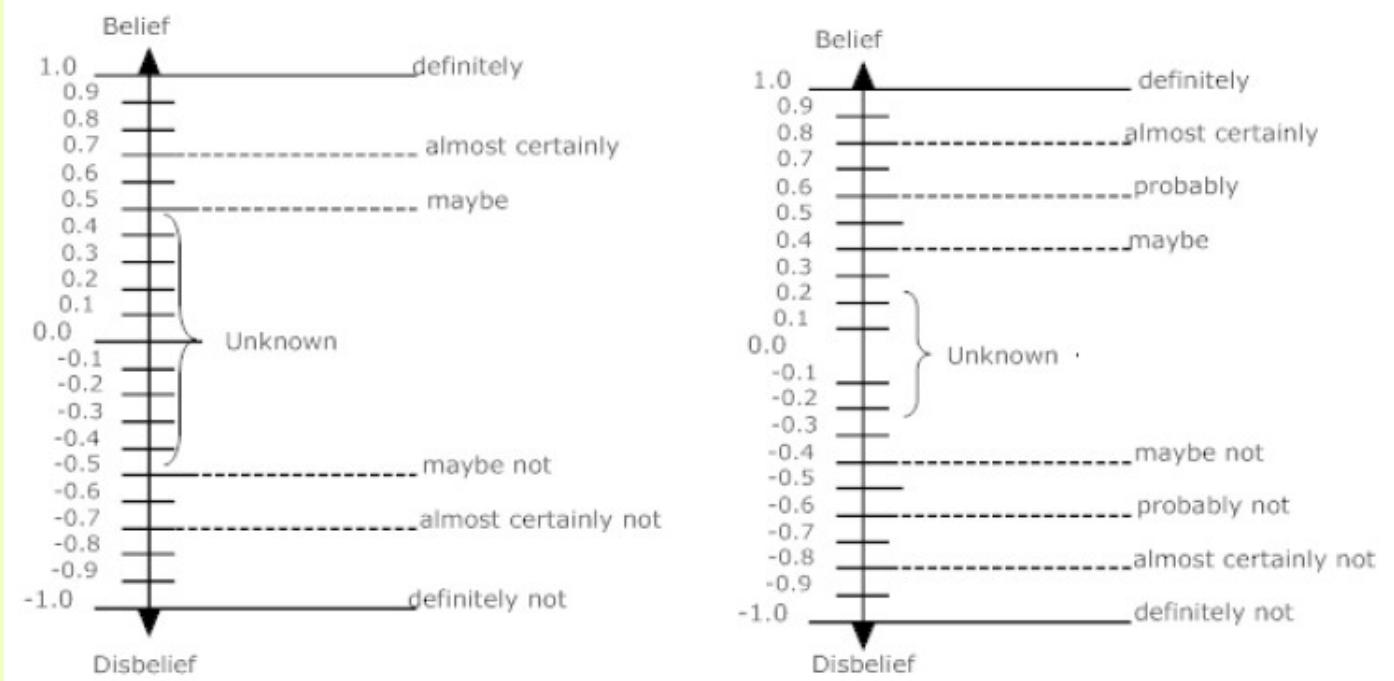
$$FC(I, D) = \frac{MB(I, D) - MD(I, D)}{1 - \min\{MB(I, D), MD(I, D)\}}$$

Sisteme inteligente – SBC – factori de certitudine

Conținut și arhitectură

□ FC – mod de calcul

- Încrederea în ipoteza I dată fiind dovada D
 - $FC=-1$ dacă se știe că ipoteza este falsă
 - $FC=0$ dacă nu se știe nimic despre ipoteză
 - $FC=1$ dacă se știe că ipoteza este adevărată



Sisteme inteligente – SBC – factori de certitudine

Conținut și arhitectură

□ FC – mod de calcul

- Încrederea în ipoteza I dată fiind dovada D
- ipoteza I poate fi:
 - simplă (ex. Dacă D atunci I)
 - compusă (ex. Dacă D atunci I_1 și I_2 și ... I_n)
- dovada D poate fi
 - dpdv al compoziției:
 - simplă (ex. Dacă D atunci I)
 - compusă (ex. Dacă D_1 și D_2 și ... D_n atunci I)
 - dpdv al incertitudinii (încrederii în dovedă):
 - sigură (ex. Dacă D atunci I)
 - nesigură (ex. Dacă $D[FC]$ atunci I)

Sisteme inteligente – SBC – factori de certitudine

Conținut și arhitectură

- FC – mod de calcul pentru combinarea încrederii
 - o doavadă incertă care susține sigur o ipoteză
 - mai multe dovezi incerte care susțin sigur o singură ipoteză
 - o doavadă incertă care susține incert o ipoteză
 - mai multe dovezi incerte care susțin incert o ipoteză

Sisteme inteligente – SBC – factori de certitudine

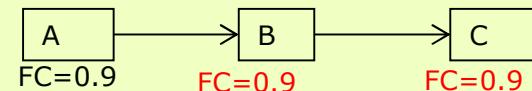
Conținut și arhitectură

- FC – mod de calcul pentru combinarea încrederii
 - O doavadă incertă care susține sigur o ipoteză

$$FC(I) = \begin{cases} FC(D), & \text{dacă } FC(D) > 0 \\ 0, & \text{altfel} \end{cases}$$

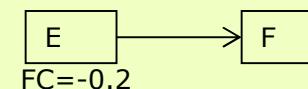
□ Exemplul 1

- $R_1: Dacă A_{[FC=0.9]} atunci B$
- $R_2: Dacă B atunci C$
- $FC(B)=FC(A)=0.9$
- $FC(C)=FC(B)=0.9$



■ Exemplul 2

- $R_1: Dacă E_{[FC=-0.2]} atunci F$
- $FC(E \text{ este adevărat}) = -0.2 \rightarrow$ doavadă negativă \rightarrow nu putem spune nimic despre faptul că E este adevărat \rightarrow nu se poate spune nimic despre F



Sisteme inteligente – SBC – factori de certitudine

Conținut și arhitectură

□ FC – mod de calcul pentru combinarea încrederei

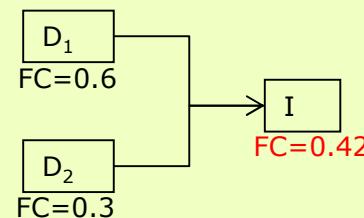
□ Mai multe dovezi incerte care susțin sigur o singură ipoteză

- Dovezi (probe) adunate incremental
- Mai multe reguli care, pe baza unor dovezi diferite, furnizează aceeași concluzie
- Aceeași ipoteză (valoare de atribut) I este obținută pe două căi de deducție distincte, cu două perechi diferite de valori pentru FC , $FC[I, D_1]$ și $FC[I, D_2]$
- Cele două cai de deducție distincte, corespunzătoare dovezilor (probelor) D_1 și D_2 pot fi:
 - ramuri diferite ale arborelui de căutare generat prin aplicarea regulilor
 - dovezi (probe) indicate explicit sistemului

$$FC(I, D_1 \wedge D_2) = \begin{cases} CF(D_1) + CF(D_2)(1 - CF(D_1)), & \text{dacă } CF(D_1), CF(D_2) > 0 \\ CF(D_1) + CF(D_2)(1 + CF(D_1)), & \text{dacă } CF(D_1), CF(D_2) < 0 \\ \frac{CF(D_1) + CF(D_2)}{1 - \min\{|CF(D_1)|, |CF(D_2)|\}}, & \text{dacă } sign(CF(D_1)) \neq sign(CF(D_2)) \end{cases}$$

□ Exemplu

- R_1 : Dacă $D_1 [FC=0.6]$ atunci I
- R_2 : Dacă $D_2 [FC=-0.3]$ atunci I
- $FC(I, D_1 \wedge D_2) = (0.6 + (-0.3)) / (1 - 0.3) = 0.42$



Sisteme inteligente – SBC – factori de certitudine

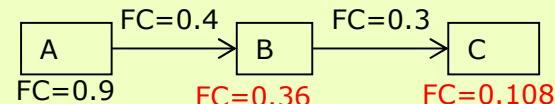
Conținut și arhitectură

- FC – mod de calcul pentru combinarea încrederii
 - O doavadă incertă care susține incert o ipoteză

$$FC(I) = \begin{cases} FC(D) * FC(\text{regulă}), & \text{dacă } FC(D) > 0 \\ 0, & \text{altfel} \end{cases}$$

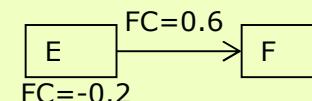
□ Exemplul 1

- $R_1: Dacă A_{[FC=0.9]} atunci B [FC=0.4]$
- $R_2: Dacă B atunci C [FC=0.3]$
- $FC(B) = FC(A) * FC(R_1) = 0.9 * 0.4 = 0.36$
- $FC(C) = FC(B) * FC(R_2) = 0.36 * 0.3 = 0.108$



■ Exemplul 2

- $R_1: Dacă E_{[FC=-0.2]} atunci F [FC=0.6]$
- $FC(E \text{ este adevărat}) = -0.2 \rightarrow$ doavadă negativă \rightarrow nu putem spune nimic despre faptul că E este adevărat \rightarrow nu se poate spune nimic despre F



Sisteme inteligente – SBC – factori de certitudine

Conținut și arhitectură

□ FC – mod de calcul pentru combinarea încrederei

- ▢ Mai multe dovezi incerte care susțin incert o ipoteză
 - Dovezile sunt legate prin SI logic

$$CF(I) = \begin{cases} \min\{CF(D_1), CF(D_2), \dots, CF(D_n)\} * CF(\text{regulă}), & \text{dacă } CF(D_i) > 0, i = 1, 2, \dots, n \\ 0, & \text{altfel} \end{cases}$$

- ▢ Una sau mai multe dintre dovezile incerte care susțin incert o ipoteză
 - Dovezile sunt legate prin SAU logic

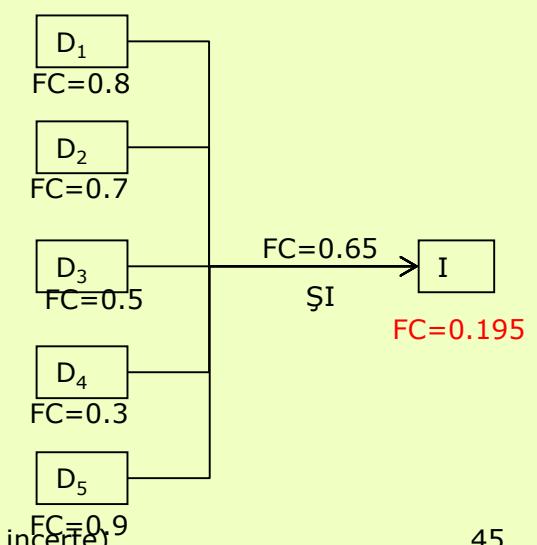
$$CF(I) = \begin{cases} \max\{CF(D_1), CF(D_2), \dots, CF(D_n)\} * CF(\text{regulă}), & \text{dacă } CF(D_i) > 0, i = 1, 2, \dots, n \\ 0, & \text{altfel} \end{cases}$$

▢ Exemplul 1

- $R_1: \text{Dacă } D_1[FC = 0.8] \text{ și } D_2[FC = 0.7] \text{ și } D_3[FC = 0.5] \text{ și }$

$D_4[FC = 0.3] \text{ și } D_5[FC = 0.9]$ atunci $I [FC = 0.65]$

- $FC(I) = 0.3 * 0.65 = 0.195$



Sisteme inteligente – SBC – factori de certitudine

Conținut și arhitectură

- FC – mod de calcul pentru combinarea încrederei
 - Mai multe dovezi incerte care susțin incert o ipoteză
 - Dovezile sunt legate prin SI logic
 - Una sau mai multe dintre dovezile incerte susțin incert o ipoteză
 - Dovezile sunt legate prin SAU logic
- Exemplul 2
 - R₁: Dacă D₁[FC = 0.8] sau D₂[FC = 0.7] sau D₃[FC = 0.5] sau D₄[FC = 0.3] sau D₅[FC = 0.9] atunci I [FC = 0.65]
 - FC(I) = 0.9 * 0.65 = 0.585

$$CF(I) = \begin{cases} \min\{CF(D_1), CF(D_2), \dots, CF(D_n)\} * CF(\text{regulă}), & \text{dacă } CF(D_i) > 0, i = 1, 2, \dots, n \\ 0, & \text{altfel} \end{cases}$$

$$CF(I) = \begin{cases} \max\{CF(D_1), CF(D_2), \dots, CF(D_n)\} * CF(\text{regulă}), & \text{dacă } CF(D_i) > 0, i = 1, 2, \dots, n \\ 0, & \text{altfel} \end{cases}$$



Sisteme inteligente – SBC – factori de certitudine

Exemplu

- Sistem expert pentru diagnosticarea unei răceli

- Fapte în baza de date:

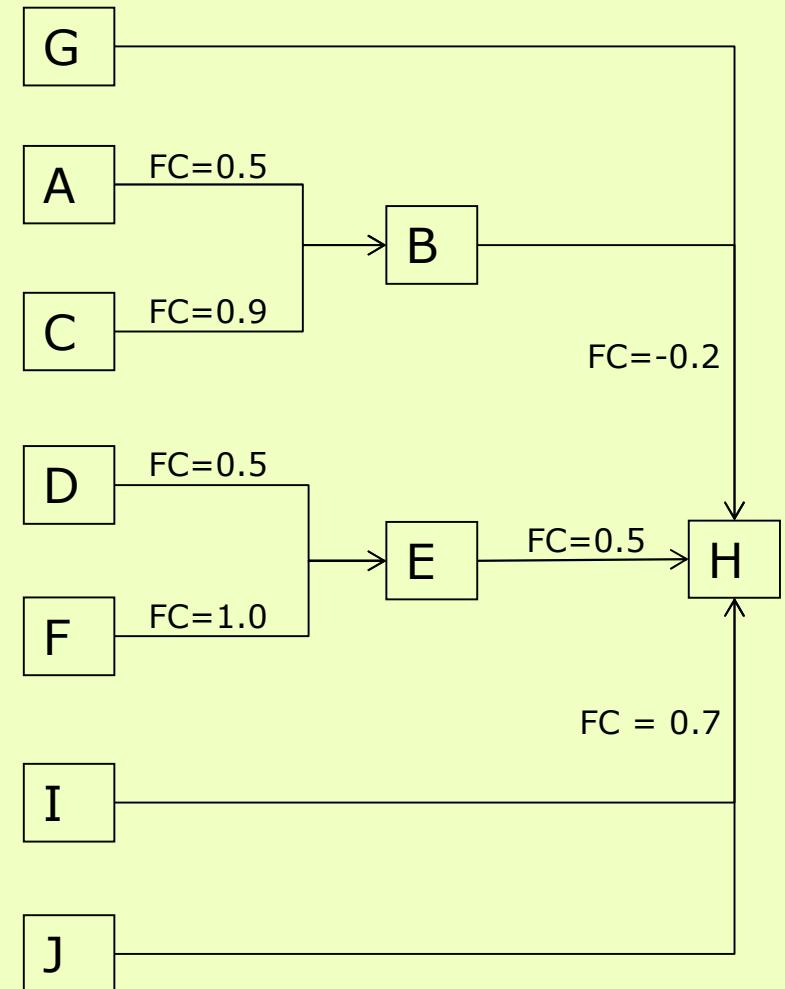
- Febra pacientului 37.4
 - Pacientul tușește de mai puțin de 24 ore
 - Pacientul nu are expectorații
 - Pacientul are o durere de cap cu FC = 0.4
 - Pacientul are nasul înfundat cu FC = 0.5

- Reguli:

- R₁: Dacă A: febra < 37.5 atunci
B: simptomele de răceală sunt prezente [FC=0.5]
 - R₂: Dacă C: febra > 37.5 atunci
B: simptomele de răceală sunt prezente [FC=0.9]
 - R₃: Dacă D: tușește > 24 ore atunci
E: durerea de gât e prezentă [FC=0.5]
 - R₄: Dacă F: tușește > 48 ore atunci
E: durerea de gât e prezentă [FC=1.0]
 - R₅: Dacă B: are simptome de răceală și
G: nu expectorează atunci H: a răcit [FC=-0.2]
 - R₆: Dacă E: îl doare gâtul atunci
H: a răcit [FC=0.5]
 - R₇: Dacă I: îl doare capul și
J: are nasul înfundat atunci H: a răcit [FC=0.7]

- Concluzia:

- Pacientul este sau nu răcit?



Sisteme inteligente – SBC – factori de certitudine

Exemplu

- Sistem expert pentru diagnosticarea unei răceli

- Fapte în baza de date:

- Febra pacientului 37.4
 - Pacientul tușește de mai puțin de 24 ore
 - Pacientul nu are expectorații
 - Pacientul are o durere de cap cu FC = 0.4
 - Pacientul are nasul înfundat cu FC = 0.5

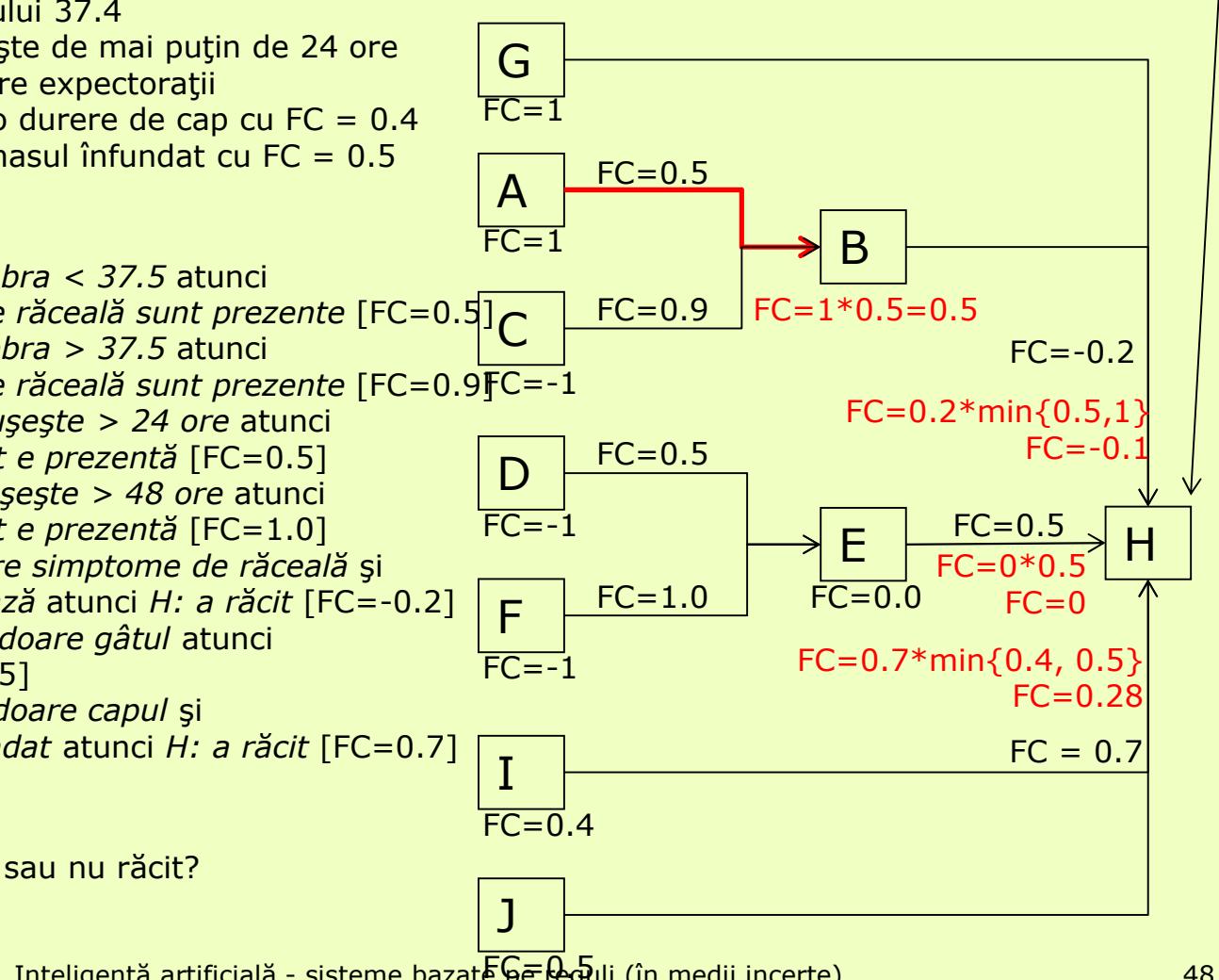
- Reguli:

- R₁: Dacă A: febra < 37.5 atunci B: simptomele de răceală sunt prezente [FC=0.5]
 - R₂: Dacă C: febra > 37.5 atunci B: simptomele de răceală sunt prezente [FC=0.9]
 - R₃: Dacă D: tușește > 24 ore atunci E: durerea de gât e prezentă [FC=0.5]
 - R₄: Dacă F: tușește > 48 ore atunci E: durerea de gât e prezentă [FC=1.0]
 - R₅: Dacă B: are simptome de răceală și G: nu expectorează atunci H: a răcit [FC=-0.2]
 - R₆: Dacă E: îl doare gâtul atunci H: a răcit [FC=0.5]
 - R₇: Dacă I: îl doare capul și J: are nasul înfundat atunci H: a răcit [FC=0.7]

- Concluzia:

- Pacientul este sau nu răcit?

$$FC=(-0.1+0.28)/(1-\min\{|-0.1|, |0.28|\}) \\ FC=0.2$$



Sisteme inteligente – SBC – factori de certitudine

□ Avantaje

- Nu este necesar calculul apriori a probabilităților

□ Limite

- ipotezele sustinute de probe sunt independente.

- exemplu:

- Fie următoarele fapte:

- A: Aspersorul a funcționat noaptea trecută
 - U: Iarba este udă dimineață
 - P: Noaptea trecută a plouat.

- și următoarele două reguli care leagă între ele aceste fapte:

- R_1 : dacă aspersorul a funcționat noaptea trecută atunci există o încredere puternică (0.9) că iarba este udă dimineață
 - R_2 : dacă iarba este udă dimineață atunci există o încredere puternică (0.8) că noaptea trecută a plouat

- Deci:

- $FC[U,A] = 0.9$ - deci proba aspersor sustine iarba uda cu 0.9
 - $FC[P,U] = 0.8$ - deci iarba uda sustine ploaie cu 0.8
 - $FC[P,A] = 0.8 * 0.9 = 0.72$ - deci aspersorul sustine ploaia cu 0.72

Sisteme inteligente – SBC – factori de certitudine

□ SBR de tip Bayes vs. SBR cu FC

Bayes	FC
Teorie probabilităților este veche și fundamentată matematic	Teoria FC este nouă și fără demonstrații matematice
Necesită existența unor informații statistice	Nu necesită existența unor date statistice
Propagarea încrederei crește în timp exponențial	Informația circulă repede și eficient în SBR

- Tehnici de raționare în medii nesigure
 - Teoria Bayesiana – metodă probabilistică
 - Teoria certitudinii
 - **Teoria posibilității (logica fuzzy)**
- 
- Metode euristice

Sisteme inteligente – SBC – sisteme fuzzy

- Teoria posibilității
- Conținut și arhitectură
- Tipologie
- Tool-uri
- Avantaje și limite

Sisteme inteligente – SBC – sisteme fuzzy

Teoria posibilității (logica fuzzy)

❑ De ce fuzzy?

- Problemă: transpuneți în cod (C++) următoarele propoziții:
 - Georgel este înalt.
 - Afara este frig.

❑ Când este important fuzzy?

- Interogări în limbaj natural
- Reprezentarea cunoștințelor în sisteme expert
- Controlul fuzzy – când se lucrează cu fenomene imprecise (perturbate de zgomot)

Sisteme inteligente – SBC – sisteme fuzzy

Amintim componența unui SBC

- Baza de cunoștințe (BC) → Modalități de reprezentare a cunoștințelor
 - Logica formală (limbaje formale)
 - Definiție
 - Știința principiilor formale de raționament
 - Componente
 - Sintaxă
 - Semantică
 - Metodă de inferență sintactică
 - Tipologie
 - În funcție de numărul valorilor de adevăr:
 - logică duală
 - **logică polivalentă**
 - În funcție de tipul elementelor de bază:
 - clasică → primitivele = propoziții (predicate)
 - probabilistică → primitivele = variabile aleatoare
 - În funcție de obiectul de lucru:
 - logica propozițională → se lucrează doar cu propoziții declarative, iar obiectele descrise sunt fixe sau unice (Ionică este student)
 - logica predicatelor de ordin I → se lucrează cu propoziții declarative, cu predicate și cuantificări , iar obiectele descrise pot fi unice sau variabile asociate unui obiect unic (Toți studenții sunt prezenți)
 - Reguli
 - Rețele semantice
 - Modulul de control (MC – pentru inferență)

Sisteme inteligente – SBC – sisteme fuzzy

Teoria posibilității - Un pic de istorie

- ❑ Parmenedes (400 B.C.)
- ❑ Aristotle
 - "Law of the Excluded Middle" – fiecare propoziție trebuie să fie Adevărată sau Falsă
- ❑ Plato
 - O a treia regiune între Adevărat și Fals
 - Pune bazele logicii fuzzy
- ❑ Lukasiewicz (1900)
 - Propune o alternativă sistematică la logica bivalentă a lui Aristotle – logica trivalentă: Adevărat, Fals, Posibil
- ❑ Lotfi A. Zadeh (1965)
 - Descrie matematic teoria mulțimilor fuzzy și logica fuzzy: funcția de apartenență (valorile Adevărat și Fals) operează pe intervalul $[0,1]$
 - A propus noi operații de calcul pt logica fuzzy
 - A considerat logica fuzzy o generalizare a logicii clasice
 - A publicat primul articol despre mulțimile fuzzy

Sisteme inteligente – SBC – sisteme fuzzy

Teoria posibilității

- Logica fuzzy
 - Generalizare a logicii Boolene
 - Manipulează conceptul de adevăr parțial
 - Logica clasică – totul este exprimat în termeni binari
 - 0 sau 1, alb sau negru, da sau nu
 - Logica fuzzy – exprimarea graduală a unui adevăr
 - Valori între 0 și 1
- Analogia *logică vs. algebră*
 - Operatorii logici exprimați în termeni matematici (George Boole):
 - Conjunction = minimum $\rightarrow a \wedge b = \min(a, b)$
 - Disjunction = maximum $\rightarrow a \vee b = \max(a, b)$
 - Negation = scădere $\rightarrow \neg a = 1 - a$

Sisteme inteligente – SBC – sisteme fuzzy

Reamintim: SBR – arhitectură

- baza de cunoștințe
 - Conține
 - Informațiile specifice despre un domeniu sub forma unor
 - fapte – afirmații corecte
 - reguli - euristici speciale care generează informații (cunoștințe)
 - Rol
 - stocarea tuturor elementelor cunoașterii (fapte, reguli, metode de rezolvare, euristici) specifice domeniului de aplicație, preluate de la experții umani sau din alte surse
- modulul de control
 - Conținut
 - regulile prin care se pot obține informații noi
 - algoritmi independenți de domeniu
 - creierul SBR – un algoritm de deducere bazat pe BC și specific metodei de raționare
 - un program în care s-a implementat cunoașterea de control, procedurală sau operatorie, cu ajutorul căruia se exploatează baza de cunoștințe pentru efectuarea de raționamente în vederea obținerii de soluții, recomandări sau concluzii.
 - depinde de complexitate și tipul cunoștințelor cu care are de-a face
 - Rol
 - cu ajutorul lui se exploatează baza de cunoștințe pentru efectuarea de raționamente în vederea obținerii de soluții, recomandări sau concluzii

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură

□ Ideea de bază

- Cf. teoriei informațiilor certe
 - *Popescu este Tânăr*
- Cf. teoriei informațiilor incerte
 - Cf. teoriei probabilităților:
 - *Există 80% șanse ca Popescu să fie Tânăr*
 - Cf. logicii fuzzy:
 - *Gradul de apartenență al lui Popescu la grupul de oameni tineri este 0.80*

□ Necesitate

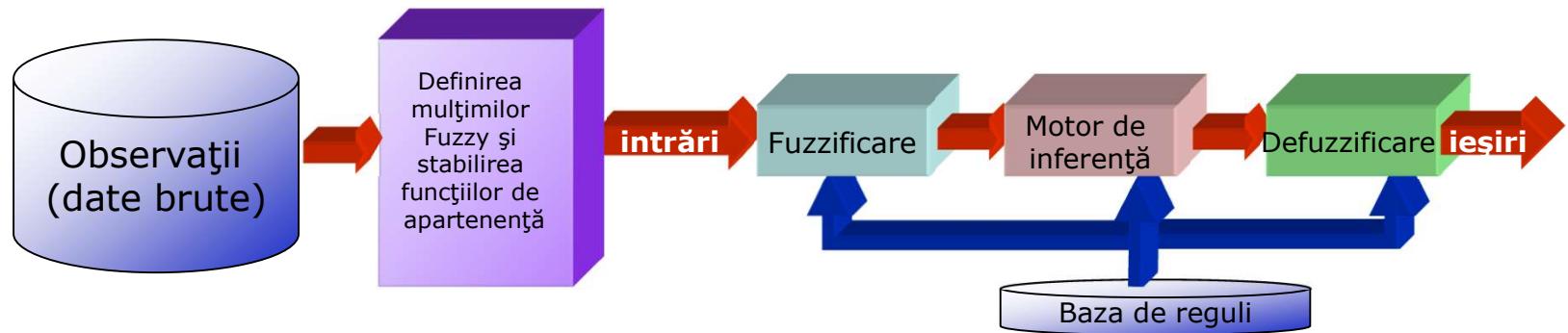
- Abordarea fenomenelor reale implică utilizarea mulțimilor fuzzy
- Exemplu
 - *Temperatura unei camere poate fi:*
 - *joasă,*
 - *medie sau*
 - *ridicată*
 - Aceste mulțimi de temperaturi posibile se pot suprapune
 - o temperatură poate apartine uneia sau mai multor mulțimi în funcție de cel care face evaluarea

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură

□ Pași în construirea unui sistem fuzzy

- Definirea intrărilor și ieșirilor de către expert
 - Datele de intrare și ieșire brute
 - Fuzzificarea datelor de intrare și ieșire
 - Stabilirea variabilelor fuzzy și a mulțimilor fuzzy pe baza funcțiilor de apartenență
- Construirea unei baze de reguli de către expert
 - Matricea de decizie a bazei de cunoștințe
- Evaluarea regulilor
 - Inferență – transformarea intrărilor fuzzy în ieșiri fuzzy prin aplicarea regulilor din baza de cunoștințe
- Agregarea rezultatelor
- Defuzificarea
- Interpretarea rezultatelor

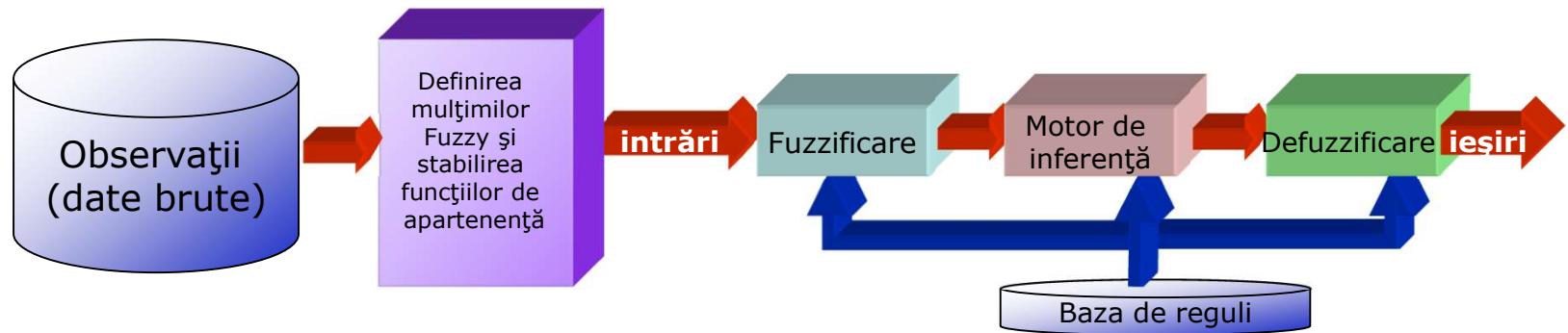


Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură

□ Pași în construirea unui sistem fuzzy

- Definirea intrărilor și ieșirilor de către expert
 - Datele de intrare și ieșire brute
 - Fuzzificarea datelor de intrare și ieșire
 - Stabilirea variabilelor fuzzy și a mulțimilor fuzzy pe baza funcțiilor de apartenență
- **Construirea unei baze de reguli de către expert**
 - Matricea de decizie a bazei de cunoștințe
- Evaluarea regulilor
 - Inferență – transformarea intrărilor fuzzy în ieșiri fuzzy prin aplicarea regulilor din baza de cunoștințe
- Agregarea rezultatelor
- Defuzificarea
- Interpretarea rezultatelor



Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

- ▣ Elemente de teoria posibilității (logica fuzzy)
 - Fapte (multimi) fuzzy
 - ▢ Definire
 - ▢ Reprezentare
 - ▢ Operații – complement, containment, intersecție, reuniune, egalitate, produs algebric, sumă algebrică
 - ▢ Proprietăți – asociativitate, distributivitate, comutativitate, tranzitivitate, idempotență, identitate, involuție, legile lui De Morgan
 - ▢ Hedges
 - Variabile fuzzy
 - ▢ Definiție
 - ▢ Proprietăți
- ▣ Stabilirea variabilelor fuzzy și a multimilor fuzzy pe baza funcțiilor de apartenență

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

- ▣ Elemente de teoria posibilității (logica fuzzy) → Fapte (mulțimi) fuzzy → **definire**

- Definirea oricărei mulțimi – 2 moduri:
 - ▣ Prin enumerarea elementelor
 - Ex. *Mulțimea elevilor = {Ana, Maria, Ioana}*
 - ▣ Prin specificare unei proprietăți a elementelor mulțimii
 - Ex. *Mulțimea numerelor pare = {x | x = 2n, unde n – număr natural}*
- Funcția caracteristică μ a unei mulțimi
 - ▣ Fie X o mulțime universală și x un element al mulțimii ($x \in X$)
 - ▣ Logica clasică
 - Fie R o submulțime a lui X : $R \subset X$, R - mulțime regulară
 - Orice element x aparține sau nu mulțimii R
 - $\mu_R : X \rightarrow \{0, 1\}$, unde $\mu_R(x) = \begin{cases} 1, & x \in R \\ 0, & x \notin R \end{cases}$
 - ▣ Logica fuzzy
 - Fie F o submulțime a lui X (univers de discurs): $F \subset X$, F - mulțime fuzzy
 - orice element x aparține mulțimii F într-un anumit grad $\mu_F(x)$
 - $\mu_F : X \rightarrow [0, 1]$, $\mu_F(x) = g$, unde $g \in [0, 1]$ – gradul de apartenență al lui x la F
 - $g = 0 \Rightarrow$ ne-apartență
 - $g = 1 \Rightarrow$ apartenență
 - O mulțime fuzzy = o pereche (F, μ_F) , unde $\mu_F(x) = \begin{cases} 1, & \text{dacă } x \text{ este total în } F \\ 0, & \text{dacă } x \text{ nu este în } F \\ \in (0,1) & \text{dacă } x \text{ este parte din } F (x \text{ este numar fuzzy}) \end{cases}$

Sisteme inteligente – SBC – sisteme fuzzy

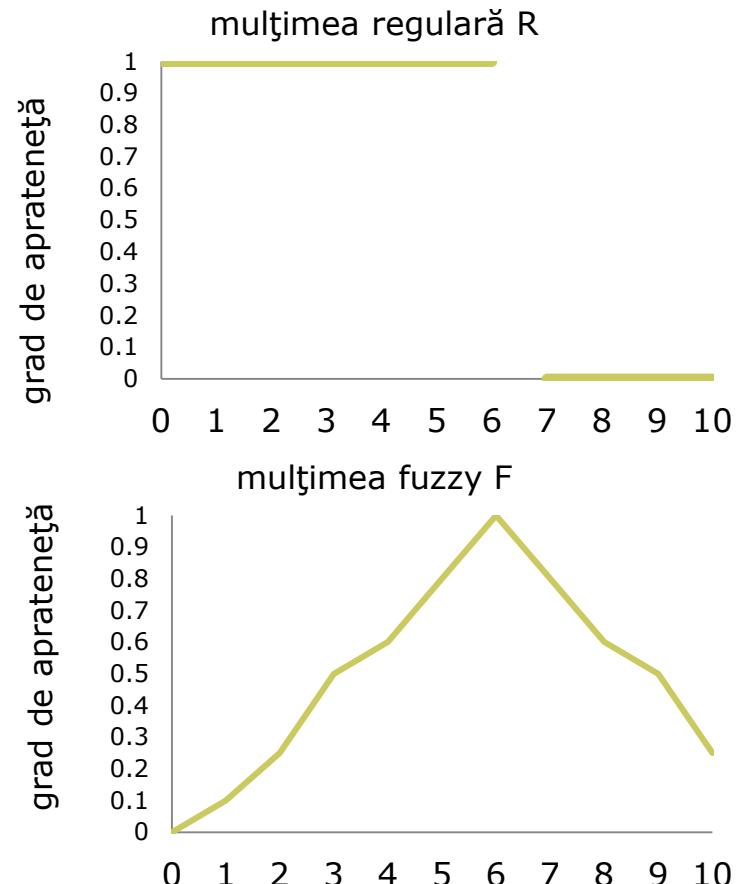
Conținut și arhitectură → fuzzificarea datelor de intrare

- ▣ Elemente de teoria posibilității (logica fuzzy) → Fapte (mulțimi) fuzzy → **definire**

- Exemplul 1

- ▢ X – mulțimea numerelor naturale mai mici decât 11
 - ▢ R – mulțimea numerelor mai mici decât 7
 - ▢ F – mulțimea numerelor apropiate de 6

x	$\mu_R(x)$	$\mu_F(x)$
0	1	0
1	1	0.1
2	1	0.25
3	1	0.5
4	1	0.6
5	1	0.8
6	1	1
7	0	0.8
8	0	0.6
9	0	0.5
10	0	0.25



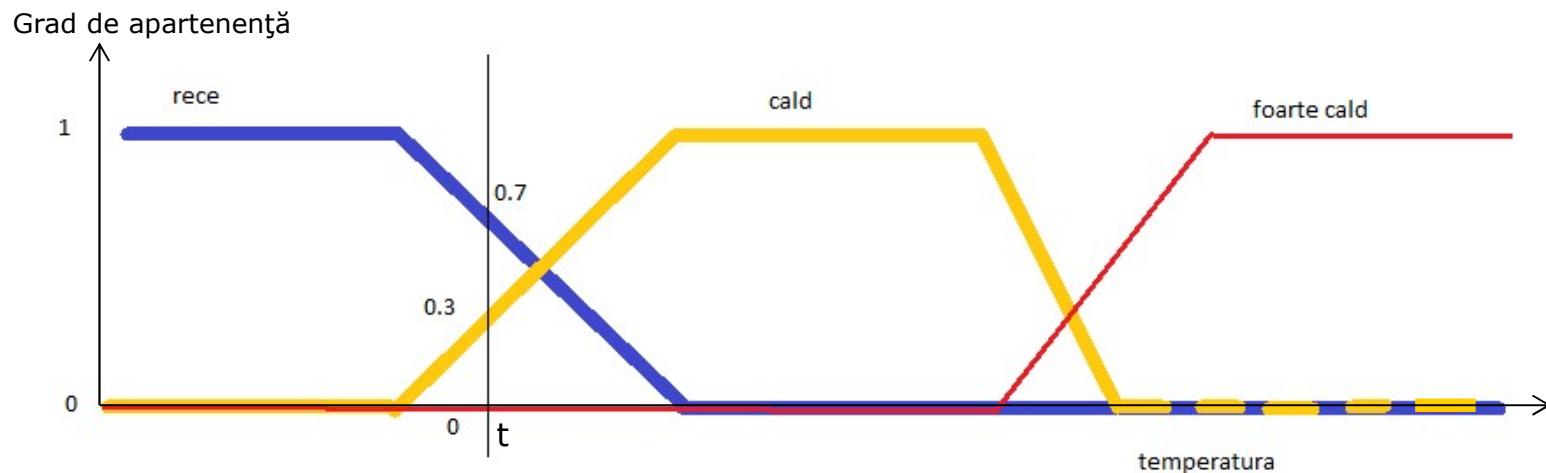
Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

- ▣ Elemente de teoria posibilității (logica fuzzy) → Fapte (multimi) fuzzy → **definire**

- Exemplul 2

- ▢ O anumită temperatură t poate avea 3 valori de adevăr:
 - Roșu (0): nu e fierbinte
 - Portocaliu (0.3): puțin cald
 - Albastru (0.7): aproape rece



Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

- ▣ Elemente de teoria posibilității (logica fuzzy) → Fapte (multimi) fuzzy → **reprezentare**

- Multimile regulare

- Delimitări clare (exacte) → diagrame Venn

- Multimile fuzzy

- Delimitări graduale → reprezentări bazate pe funcția de apartenență

- Singulară

- $\mu(x) = s$, unde s este un scalar

- Triunghiulară

- $\mu(x) = \max \left\{ 0, \min \left\{ \frac{x-a}{b-a}, 1, \frac{c-x}{c-b} \right\} \right\}$

- Trapezoidală

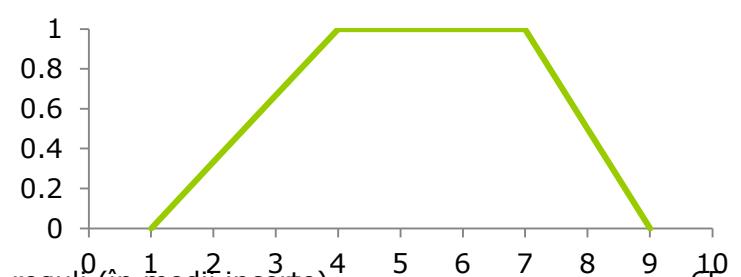
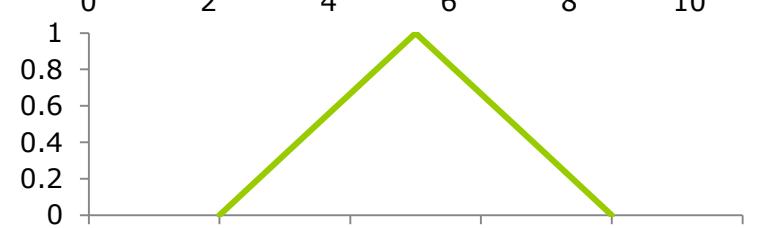
- $\mu(x) = S(x) = \max \left\{ 0, \min \left\{ \frac{x-a}{b-a}, 1, \frac{d-x}{d-c} \right\} \right\}$

- Funcția Z

- $\mu(x) = 1 - S(x)$

- Funcția Π

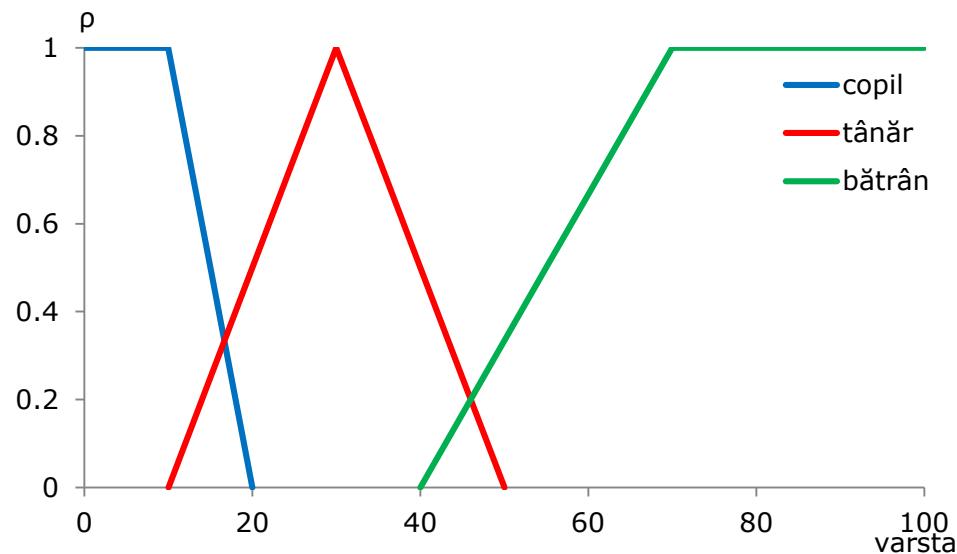
- $\mu(x) = \Pi(x) = \begin{cases} S(x), & \text{dacă } x \leq c \\ Z(x), & \text{dacă } x > c \end{cases}$



Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilității (logica fuzzy) → Fapte (multimi) fuzzy → **rezolvare**
 - Exemplu
 - Vârsta unei persoane



Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

- ▣ Elemente de teoria posibilității (logica fuzzy) → Fapte (multimi) fuzzy → **Operații**
 - ▢ complement
 - ▢ incluziune
 - ▢ intersecție
 - ▢ reunire
 - ▢ egalitate
 - ▢ produs algebric
 - ▢ sumă algebrică

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

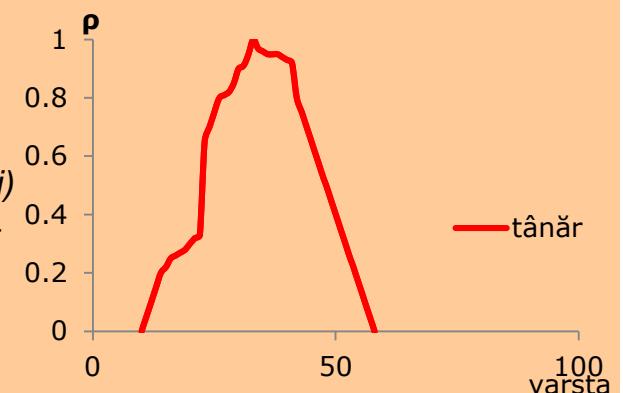
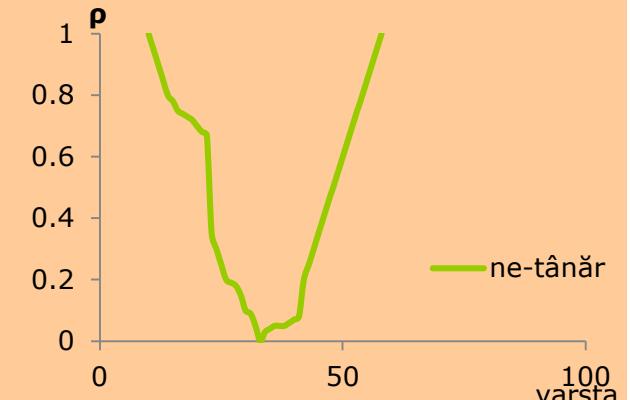
- Elemente de teoria posibilității (logica fuzzy) → Fapte (multimi) fuzzy → **Operații**

■ Complement

- X - un univers
- A – o multimea fuzzy (cu universul X)
- B – o multimea fuzzy (cu universul X)
- B este complementul lui A ($B = \neg A$) dacă:
 - $\mu_B(x) = \mu_{\neg A}(x) = 1 - \mu_A(x)$ pentru orice $x \in X$

□ Exemplu:

- persoane bătrâne (pe baza vîrstei)
 - $A = \{(30, 0), (40, 0.2), (50, 0.4), (60, 0.6), (70, 0.8), (80, 1)\}$
- persoane tinere (care nu sunt bătrâne) (pe baza vîrstei)
 - $\neg A = \{(30, 1), (40, 0.8), (50, 0.6), (60, 0.4), (70, 0.2), (80, 0)\}$



Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

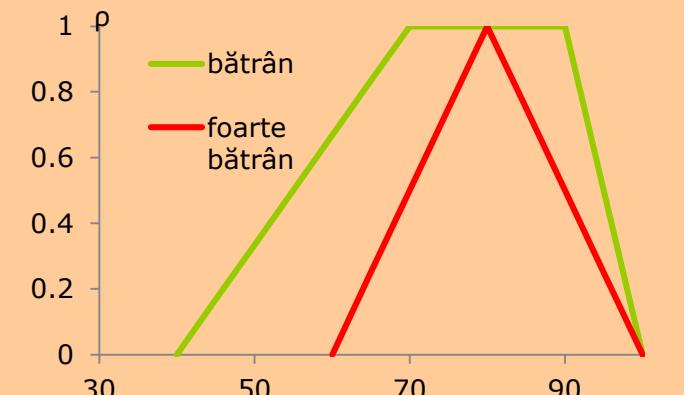
- ▣ Elemente de teoria posibilității (logica fuzzy) → Fapte (mulțimi) fuzzy → **Operații**

- Incluziune

- ▣ X - un univers
 - ▣ A – o mulțimea fuzzy (cu universul X)
 - ▣ B – o mulțimea fuzzy (cu universul X)
- ▣ B este submulțime a lui A ($B \subset A$) dacă:
 - $\mu_B(x) \leq \mu_A(x)$ pentru orice $x \in X$

- ▣ Exemplu

- persoane bătrâne (pe baza vîrstei)
 - $A = \{(60, 0.6), (65, 0.7), (70, 0.8), (75, 0.9), (80, 1)\}$
 - persoane foarte bătrâne (pe baza vîrstei)
 - $B = \{(60, 0.6), (65, 0.67), (70, 0.8), (75, 0.8), (80, 0.95)\}$



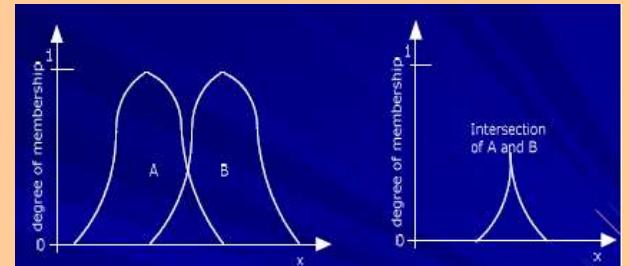
Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

- ▣ Elemente de teoria posibilității (logica fuzzy) → Fapte (multimi) fuzzy → **Operații**

- Intersecție

- ▣ X - un univers
 - ▣ A - o mulțimea fuzzy (cu universul X)
 - ▣ B - o mulțimea fuzzy (cu universul X)
 - ▣ C - o mulțimea fuzzy (cu universul X)
 - ▣ C este intersecția lui A cu B ($C = A \cap B$) dacă:
 - $\mu_C(x) = \mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\} = \mu_A(x) \cap \mu_B(x)$ pentru orice $x \in X$



- ▣ Exemplu

- *persoane bătrâne (pe baza vîrstei)*
 - $A = \{(30, 0) (40, 0.1) (50, 0.2) (60, 0.6), (65, 0.7) (70, 0.8), (75, 0.9), (80, 1)\}$
 - *persoane de vîrstă medie*
 - $B = \{(30, 0.1) (40, 0.2) (50, 0.6) (60, 0.5), (65, 0.2) (70, 0.1), (75, 0), (80, 0)\}$
 - *persoane bătrâne și de vîrstă medie*
 - $C = \{(30, 0) (40, 0.1) (50, 0.2) (60, 0.5), (65, 0.2) (70, 0.1), (75, 0), (80, 0)\}$

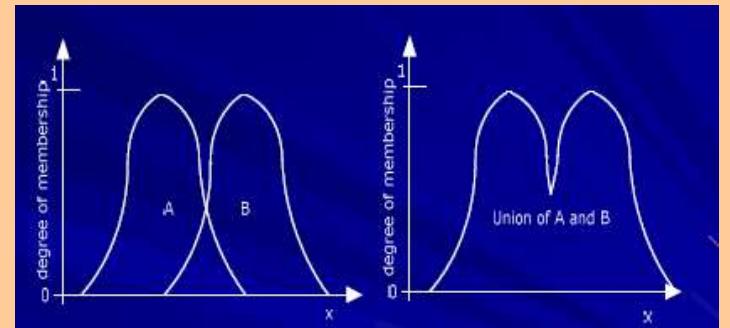
Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

- ▣ Elemente de teoria posibilității (logica fuzzy) → Fapte (multimi) fuzzy → **Operații**

- Reuniune

- ▣ X - un univers
 - ▣ A - o mulțimea fuzzy (cu universul X)
 - ▣ B - o mulțimea fuzzy (cu universul X)
 - ▣ C - o mulțimea fuzzy (cu universul X)
 - ▣ C este reuniunea lui A cu B ($C = A \cup B$) dacă:
 - $\mu_C(x) = \mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\} = \mu_A(x) \cup \mu_B(x)$ pentru orice $x \in X$



- ▣ Exemplu

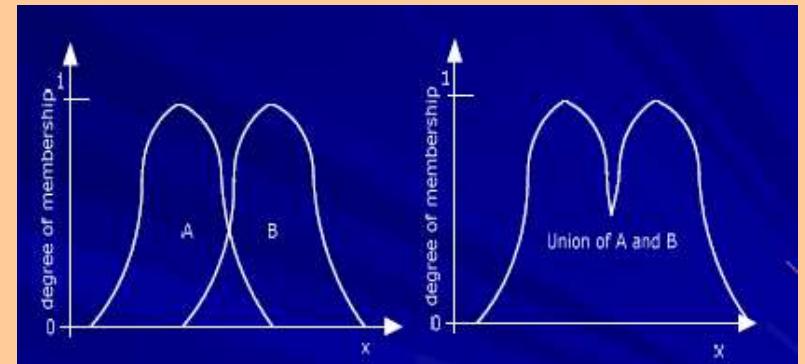
- *persoane bătrâne (pe baza vîrstei)*
 - $A = \{(30, 0) (40, 0.1) (50, 0.2) (60, 0.6), (65, 0.7) (70, 0.8), (75, 0.9), (80, 1)\}$
 - *persoane de vîrstă medie*
 - $B = \{(30, 0.1) (40, 0.2) (50, 0.6) (60, 0.5), (65, 0.2) (70, 0.1), (75, 0), (80, 0)\}$
 - *persoane bătrâne sau de vîrstă medie*
 - $C = \{(30, 0.1) (40, 0.2) (50, 0.6) (60, 0.6), (65, 0.7) (70, 0.8), (75, 0.9), (80, 1)\}$

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

- ▣ Elemente de teoria posibilității (logica fuzzy) → Fapte (multimi) fuzzy → **Operații**

- Egalitate, produs și sumă algebrică
 - ▣ X - un univers
 - ▣ A - o mulțimea fuzzy (cu universul X)
 - ▣ B - o mulțimea fuzzy (cu universul X)
 - ▣ C - o mulțimea fuzzy (cu universul X)
- ▣ B este egal cu A ($B=A$) dacă:
 - $\mu_B(x)=\mu_A(x)$ pentru orice $x \in X$



- ▣ C este produsul dintre A și B ($C=A*B$) dacă:
 - $\mu_C(x)=\mu_A*\mu_B(x)=\mu_A(x)*\mu_B(x)$ pentru orice $x \in X$
- ▣ C este suma lui A cu B ($C=A+B$) dacă:
 - $\mu_C(x)=\mu_A+\mu_B(x)=\mu_A(x)+\mu_B(x)$ pentru orice $x \in X$

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

- ▣ Elemente de teoria posibilității (logica fuzzy) → Fapte (multimi) fuzzy → **proprietăți**
 - Asociativitate
 - Comutativitate
 - Distributivitate
 - Tranzitivitate
 - Idempotență
 - Identitate
 - Involuție

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

- ▣ Elemente de teoria posibilității (logica fuzzy) → Fapte (mulțimi) fuzzy → **Hedges (nuanțatori)**

■ Ideea de bază

- ▢ Modificatori, adjective sau adverbe care schimbă valorile de adevăr ale unor afirmații
 - Ex. *Foarte, mai puțin, mai mult, aproape, etc.*
- ▢ Modifică forma mulțimilor fuzzy
- ▢ Pot aciona la nivel de
 - Numere fuzzy
 - Valori de adevăr
 - Funcții de apartenență
- ▢ Euristici

■ Utilitate

- ▢ Apropierea de limbajul natural → subiectivism
- ▢ Evaluarea variabilelor lingvistice

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

□ Elemente de teoria posibilității (logica fuzzy) →

Fapte (mulțimi) fuzzy → **Hedges (nuanțatori)**

- Tipologie

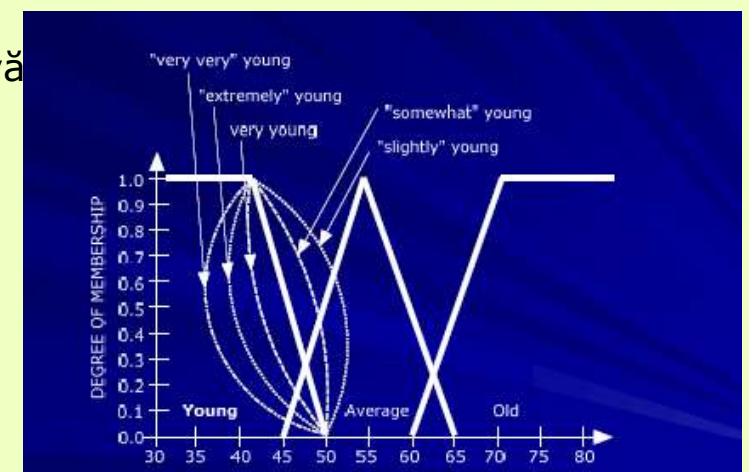
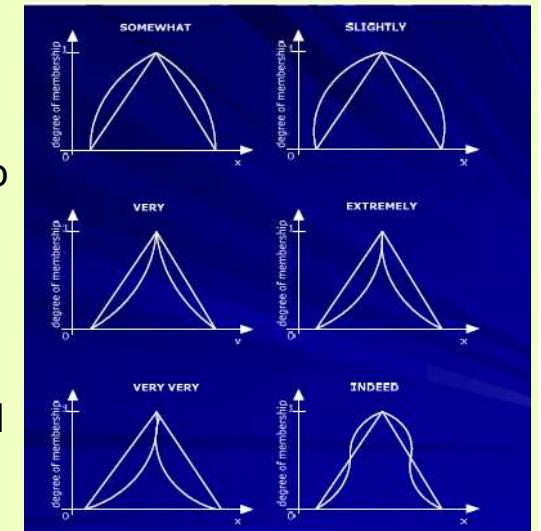
- Hedge-uri care reduc valoarea de adevăr (producând o concentrație)
 - foarte $\mu_{A_foarte}(x) = (\mu_A(x))^2$
 - extrem $\mu_{A_extrem}(x) = (\mu_A(x))^3$
 - foarte foarte $\mu_{A_foarte_foarte}(x) = (\mu_{A_foarte}(x))^2 = (\mu_A(x))^4$

- Hedge-uri care măresc valoarea de adevăr (producând o dilatare)
 - oarecum $\mu_{A_oarecum}(x) = (\mu_A(x))^{1/2}$
 - ușor $\mu_{A_usor}(x) = (\mu_A(x))^{1/3}$

- Hedge-uri care intensifică valoarea de adevăr

- întradevăr

$$\mu_{A_intradevar}(x) = \begin{cases} 2(\mu_A(x))^2, & \text{dacă } 0 \leq \mu_A(x) \leq 0.5 \\ 1 - 2(1 - \mu_A(x))^2, & \text{dacă } 0.5 \leq \mu_A(x) \leq 1 \end{cases}$$



Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

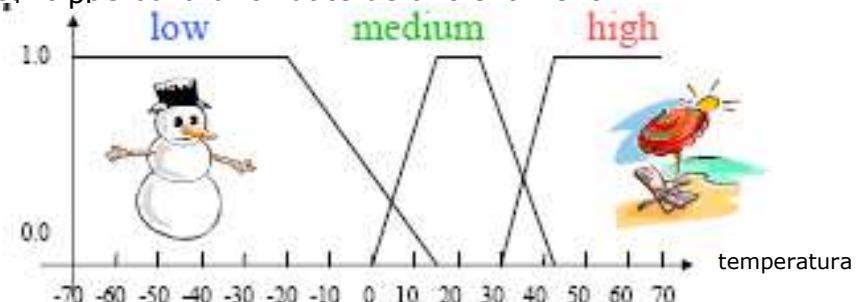
- ▣ Elemente de teoria posibilității (logica fuzzy)
 - Fapte (multimi) fuzzy
 - ▢ Definire
 - ▢ Reprezentare
 - ▢ Operații – complement, containment, intersecție, reuniune, egalitate, produs algebric, sumă algebrică
 - ▢ Proprietăți – asociativitate, distributivitate, comutativitate, tranzitivitate, idempotență, identitate, involuție, legile lui De Morgan
 - ▢ Hedges
 - **Variabile fuzzy**
 - ▢ Definiție
 - ▢ Proprietăți
- ▣ Stabilirea variabilelor fuzzy și a multimilor fuzzy pe baza funcțiilor de apartenență

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

❑ Elemente de teoria posibilității (logica fuzzy) → Variabile fuzzy → Definiție

- O variabilă fuzzy V este definită de quadruplul $V = \{x, l, u, m\}$, unde:
 - ❑ X – numele variabilei simbolice
 - ❑ L – mulțimea etichetelor posibile pentru variabila x
 - ❑ U – universul variabilei
 - ❑ M – regulile semantice care definesc înțelesul fiecărei etichete din L (funcțiile de apartenență)
- Funcția de apartenență
 - ❑ Evaluare subiectivă
 - forma funcțiilor este definită de experti/specialiști
 - ❑ Evaluare ad-hoc
 - funcții simple care să poată rezolva problema
 - ❑ Evaluare bazată pe distribuții și probabilități ale informațiilor extrase din măsurători
 - ❑ Evaluare adaptată
 - prin teste
 - ❑ Evaluare automată
 - algoritmi folosiți pentru definirea funcțiilor pe baza unor date de antrenament
- Exemplu
 - ❑ X = Temperatura
 - ❑ L = {joasă, medie, ridicată}
 - ❑ U = $\{x \in X \mid -70^\circ \leq x \leq +70^\circ\}$
 - ❑ M =



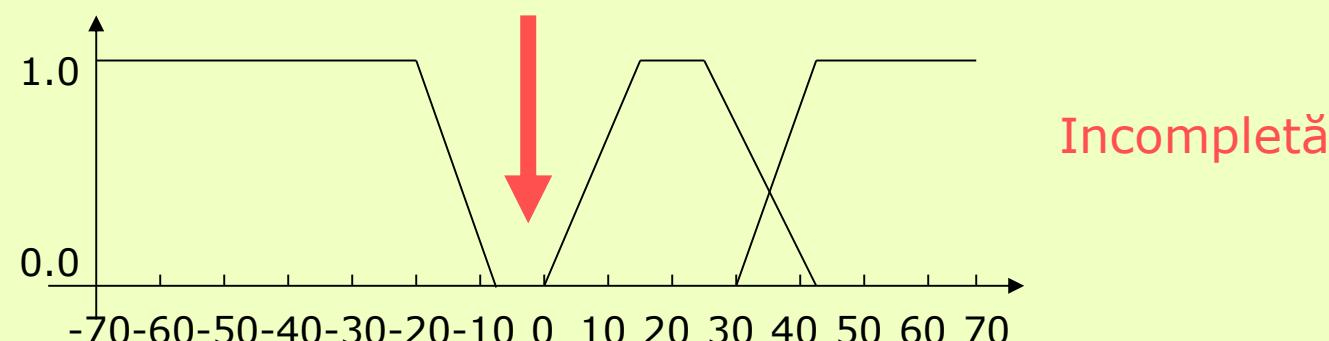
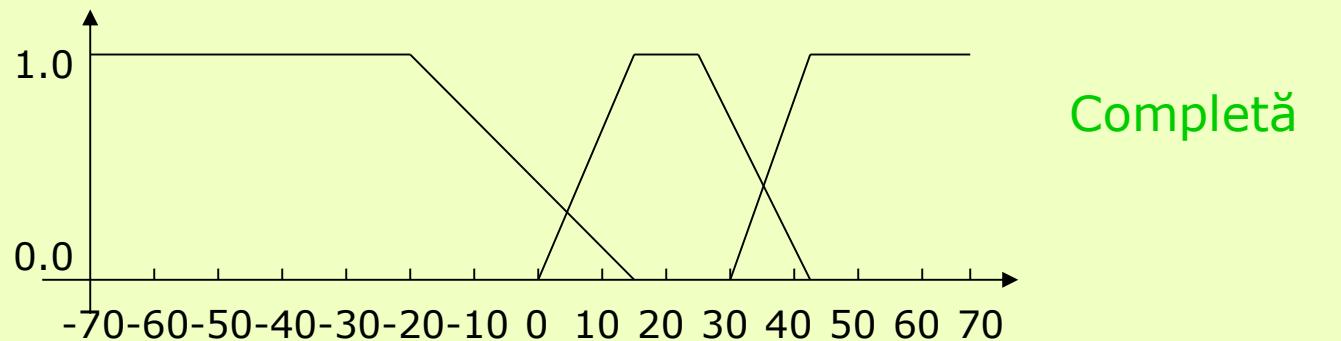
Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

- ▣ Elemente de teoria posibilității (logica fuzzy) → Variabile fuzzy →
Proprietăți

- Completitudine

- ▢ O variabilă fuzzy V este completă dacă pentru orice $x \in X$ există o mulțime fuzzy A astfel încât $\mu_A(x) > 0$



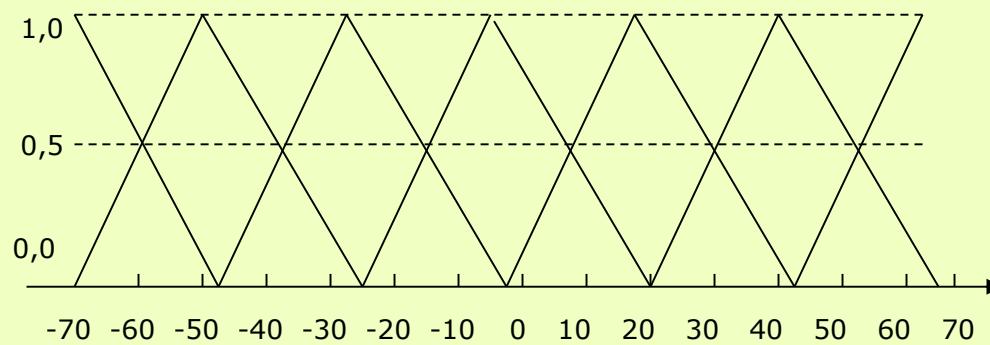
Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

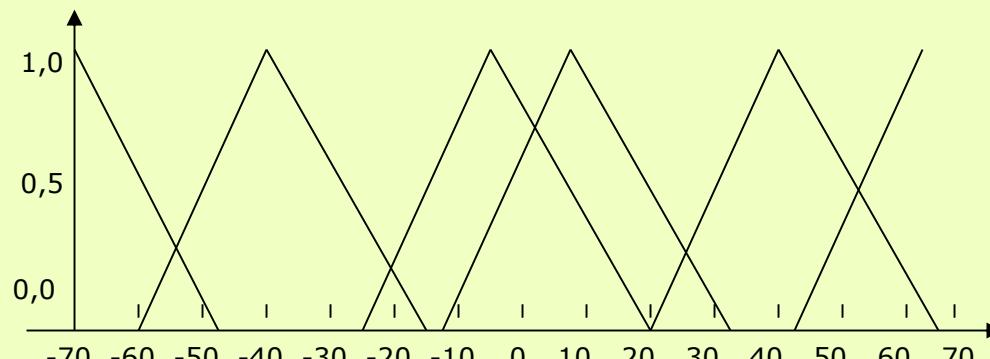
- Elemente de teoria posibilității (logica fuzzy) → Variabile fuzzy → **Proprietăți**

- Partiționare a unității

- O variabilă fuzzy V formează o partiție a unității dacă pentru orice valoare de intrare x avem $\sum_{i=1}^p \mu_{A_i}(x) = 1$
 - Unde p este numărul mulțimilor cărora aparține x
 - Nu există reguli pentru definirea suprapunerii a 2 mulțimi vecine
 - De obicei, suprapunerea trebuie să fie între 25% și 50%



Partiționare a unității



Nepartiționare a unității

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

- ▣ Elemente de teoria posibilității (logica fuzzy) → Variabile fuzzy →
Proprietăți

- Partiționare a unității

- ▣ O variabilă fuzzy completă poate fi transformată într-o partiționare a unității astfel:

$$\mu_{\hat{A}_i}(x) = \frac{\mu_{A_i}(x)}{\sum_{j=1}^p \mu_{A_j}(x)} \text{ for } i = 1, \dots, p$$

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

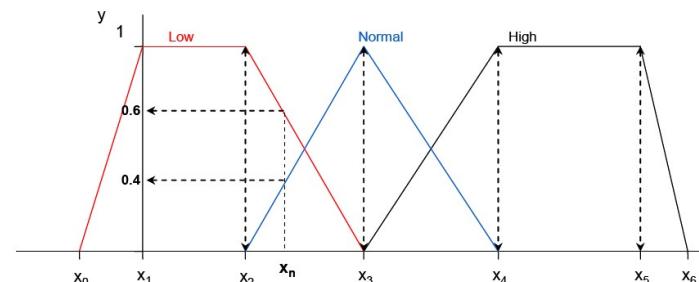
- ▣ Elemente de teoria posibilității (logica fuzzy)
 - Fapte (multimi) fuzzy
 - ▢ Definire
 - ▢ Reprezentare
 - ▢ Operații – complement, containment, intersecție, reuniune, egalitate, produs algebric, sumă algebrică
 - ▢ Proprietăți – asociativitate, distributivitate, comutativitate, tranzitivitate, idempotență, identitate, involuție, legile lui De Morgan
 - ▢ Hedges
 - Variabile fuzzy
 - ▢ Definiție
 - ▢ Proprietăți
- ▣ **Stabilirea variabilelor fuzzy și a mulțimilor fuzzy pe baza funcțiilor de apartenență → mecanism**

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

□ Mecanism

- Se stabilesc datele (de intrare și ieșire) brute ale sistemului
- Pentru fiecare dată de intrare se definesc funcțiile de apartenență (2, 3 sau mai multe)
 - Fiecarei funcții de apartenență i se asociază o etichetă calitativă – variabilă lingvistică
 - unei variabile brute îi pot corespunde una sau mai multe variabile lingvistice
 - Exemplu
 - Variabilă brută: temperatura T
 - Variabile lingvistice asociate: redusă → A1, medie → A2, înaltă → A3
- Se transformă fiecare dată brută de intrare într-o variabilă lingvistică → fuzzificare
 - Se stabilește mulțimea fuzzy din care face parte variabila brută
 - Cum?
 - Pentru o valoare brută dată a unei variabile (care poate apartine uneia sau mai multor mulțimi fuzzy) se calculează valoarea asociată funcției caracteristice corespunzătoare pentru fiecare dintre mulțimile de apartenență (folosind funcția de apartenență)
 - Exemplu
 - $T (=x_n) = 5^\circ$
 - $A_1 \rightarrow \mu_{A1}(T) = 0.6$
 - $A_2 \rightarrow \mu_{A2}(T) = 0.4$



Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → fuzzificarea datelor de intrare

❑ Mecanism

- Exemplu pentru reglarea unui aparat de aer condiționat

- ❑ Intrări:

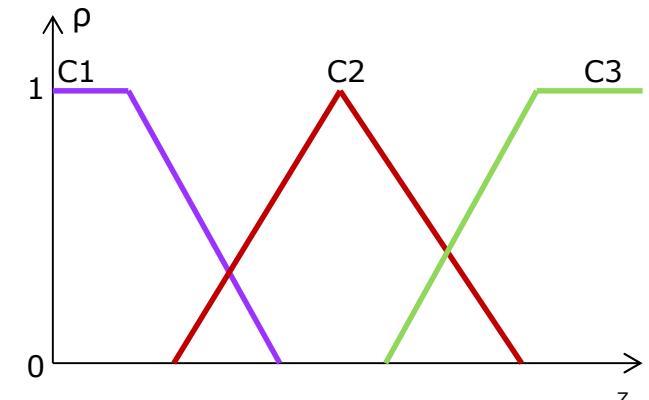
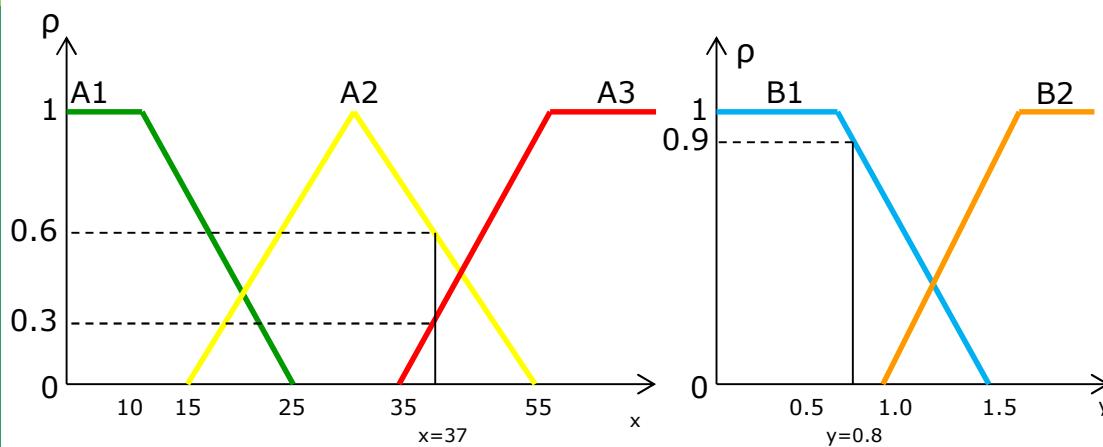
- x (temperatura – rece, normal, fierbinte) și
 - y (umiditatea – mică, mare)

- ❑ Ieșiri:

- z (puterea aparatului – redusă, medie, ridicată)

- ❑ Date de intrare:

- Temperatura x = 37
 - $\mu_{A1}(x)=0$, $\mu_{A2}(x)=0.6$, $\mu_{A3}(x)=0.3$
 - Umiditatea y = 0.8
 - $\mu_{B1}(y)=0.9$, $\mu_{B2}(y)=0$

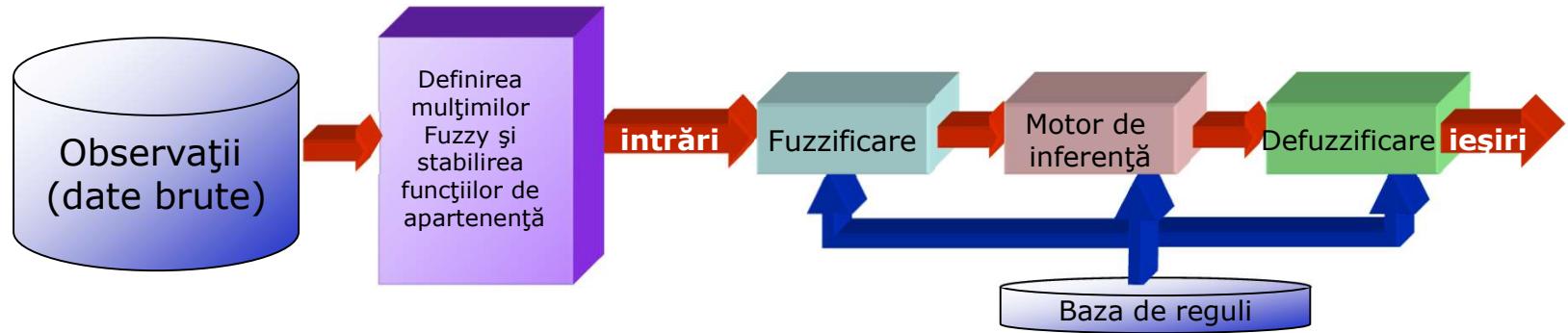


Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură

□ Pași în construirea unui sistem fuzzy

- Definirea intrărilor și ieșirilor de către expert
 - Datele de intrare și ieșire brute
 - Fuzzificarea datelor de intrare și ieșire
 - Stabilirea variabilelor fuzzy și a mulțimilor fuzzy pe baza funcțiilor de apartenență
- **Construirea unei baze de reguli de către expert**
 - Matricea de decizie a bazei de cunoștințe
- Evaluarea regulilor
 - Inferență – transformarea intrărilor fuzzy în ieșiri fuzzy prin aplicarea regulilor din baza de cunoștințe
- Agregarea rezultatelor
- Defuzificarea
- Interpretarea rezultatelor



Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Construirea unei baze de reguli de către expert

□ Reguli

■ Definiție

- Construcții lingvistice de tipul
 - Afirmațiilor: A
 - Condiționărilor: dacă A atunci B
- Unde A și B sunt (colecții de) propoziții care conțin variabile lingvistice
 - A – premisa regulii
 - B – consecința regulii

■ Tipologie

□ Necondiționale

- x este (în) A_i
- ex. *Salvează energia*

□ Condiționale

- Dacă x este (în) A_i atunci y este (în) C_k
- Dacă x este (în) A_i și y este (în) B_j atunci z este (în) C_k
- Dacă x este (în) A_i sau y este (în) B_j atunci z este (în) C_k

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Construirea unei baze de reguli de către expert

- Reguli
 - Exemplu

	Reguli în logica clasică	Reguli în logica fuzzy
R_1	<i>Dacă temperatura este -5 atunci vremea este rece</i>	<i>Dacă temperatura este joasă atunci vremea este rece</i>
R_2	<i>Dacă temperatura este 15 atunci vremea este călduroasă</i>	<i>Dacă temperatura este medie atunci vremea este călduroasă</i>
R_3	<i>Dacă temperatura este 35 atunci vremea este caniculară</i>	<i>Dacă temperatura este ridicată atunci vremea este caniculară</i>

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Construirea unei baze de reguli de către expert

- Reguli

- Baza de reguli fuzzy

- R_{11} : Dacă x este A_1 și y este B_1 atunci z este C_u
 - R_{12} : Dacă x este A_1 și y este B_2 atunci z este C_v
 - ...
 - R_{1n} : Dacă x este A_1 și y este B_n atunci z este C_x
 - R_{21} : Dacă x este A_2 și y este B_1 atunci z este C_y
 - R_{22} : Dacă x este A_2 și y este B_2 atunci z este C_z
 - ...
 - R_{2n} : Dacă x este A_2 și y este B_n atunci z este C_v
 - ...
 - R_{m1} : Dacă x este A_m și y este B_1 atunci z este C_z
 - R_{m2} : Dacă x este A_m și y este B_2 atunci z este C_v
 - ...
 - R_{mn} : Dacă x este A_m și y este B_n atunci z este C_u

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Construirea unei baze de reguli de către expert

□ Reguli

■ Proprietăți ale bazei de reguli

□ Completitudine

- O bază de reguli fuzzy este completă
 - dacă orice valoare de intrare are asociată o valoare între 0 și 1
 - dacă orice variabilă fuzzy este completă
 - dacă mulțimile folosite în bază au suport ne-compact

□ Consistență

- O mulțime de reguli fuzzy este inconsistentă dacă 2 reguli care au aceleași premise conduc la consecințe diferite
 - Dacă x este A_1 și y este B_1 atunci z este C_r
 - Dacă x este A_1 și y este B_1 atunci z este C_v

■ Probleme cu baza de reguli fuzzy

□ Explosia regulilor

- Numărul regulilor crește exponențial odată cu numărul variabilelor de intrare
- Numărul de combinații ale mulțimilor de intrare este
 - unde a i -a variabilă este formată din p_i mulțimi $P = \prod_{i=1}^n p_i$

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Construirea unei baze de reguli de către expert

- Matricea de decizie a bazei de cunoștințe
 - Exemplu pentru reglarea unui aparat de aer condiționat
 - Intrări:
 - x (temperatura – rece, normal, fierbinte) și
 - y (umiditatea – mică, mare)
 - Ieșiri:
 - z (puterea aparatului – redusă, menținută constantă, mărită)
 - Reguli:
 - *Dacă temperatura este normală și umiditatea mică atunci puterea aparatului de aer condiționat trebuie să fie menținută constantă*

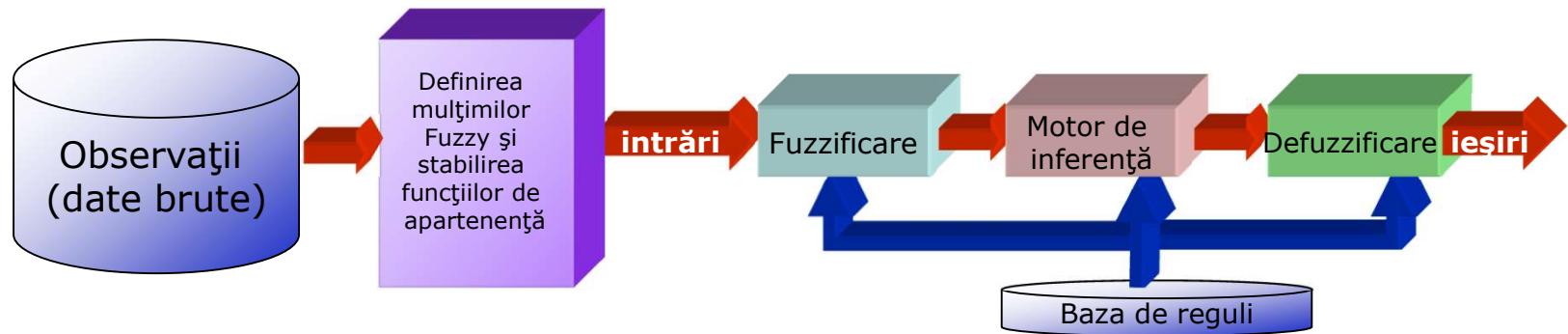
		Data de intrare y	
		Mică	Mare
Data de intrare x	Rece	Reducă	Constantă
	Normal	Constantă	Mărită
	Fierbinte	Mărită	Mărită

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură

□ Pași în construirea unui sistem fuzzy

- Definirea intrărilor și ieșirilor de către expert
 - Datele de intrare și ieșire brute
 - Fuzzificarea datelor de intrare și ieșire
 - Stabilirea variabilelor fuzzy și a mulțimilor fuzzy pe baza funcțiilor de apartenență
- Construirea unei baze de reguli de către expert
 - Matricea de decizie a bazei de cunoștințe
- **Evaluarea regulilor**
 - **Inferență – transformarea intrărilor fuzzy în ieșiri fuzzy prin aplicarea regulilor din baza de cunoștințe**
- Agregarea rezultatelor
- Defuzificarea
- Interpretarea rezultatelor



Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Evaluarea regulilor (inferență fuzzy)

- Care reguli se evaluatează mai întâi?
 - Inferență de tip Fuzzy
 - Regulile se evaluatează în **paralel**, fiecare regulă contribuind la forma rezultatului final.
 - Mulțimile fuzzy rezultate sunt defuzzificate **abia după ce toate regulile** au fost evaluate
 - Amintim
 - Inferență înainte
 - Pentru o stare a problemei se colectează informațiile necesare și se aplică regulile posibile pentru BC construită
 - Inferență înapoi
 - Se identifică regulile care determină apariția faptului obiectiv și (dacă este posibil) se aplică aceste reguli
- Cum se evaluatează regulile?
 - Evaluarea antecedentelor
 - Evaluarea consecințelor

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Evaluarea regulilor (inferență fuzzy)

■ Evaluarea antecedentelor

- Pentru fiecare premisă a unei reguli (dacă x este (\hat{in}) A) se ia în considerare gradul de apartenență al datelor brute la fiecare multime fuzzy din care face parte (anterior calculat în etapa de fuzzificare)
- O regulă poate avea una sau mai multe premise (antecedente) legate prin operatori logici: AND , OR sau NOT → Aplicarea operatorilor fuzzy în evaluarea regulilor
 - Operator AND → intersecția (minimul) a 2 mulțimi
 - $\mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}$
 - Operator OR → reuniunea (maximul) a 2 mulțimi
 - $\mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}$
 - Operator NOT → negația (complementul) unei mulțimi
 - $\mu_{\neg A}(x) = 1 - \mu_A(x)$
- Rezultatul evaluării premisei unei reguli
 - Gradul în care antecedentul unei reguli este satisfăcut
 - Poate fi rezultatul unei operații de tip AND sau OR
 - Alte denumiri:
 - Rule's firing strength
 - Degree of fulfillment

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Evaluarea regulilor (inferență fuzzy)

- ▣ Evaluarea consecințelor (aplicarea metodei de implicare)
 - Determinarea rezultatelor
 - ▣ stabilirea gradului de apartenență la mulțimile fuzzy corespunzătoare a variabilelor existente în consecința regulilor
 - Fiecare regiune de ieșire va trebui defuzzificată pentru obținerea valorilor crisp
 - În funcție de tipul consecințelor unei reguli
 - ▣ Modelul Mamdani – consecința regulii este de forma “variabila de ieșire face parte dintr-o mulțime fuzzy”
 - ▣ Modelul Sugeno – consecința regulii este de forma “variabila de ieșire este o funcție crisp care depinde de intrări”
 - ▣ Modelul Tsukamoto – consecința regulii este de forma “variabila de ieșire face parte dintr-o mulțime fuzzy cu o funcție de apartenență monotonă”

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Evaluarea regulilor (inferență fuzzy) → evaluarea consecințelor

□ Modelul Mamdani

- Ideea de bază:
 - consecința regulii este de forma “variabila de ieșire face parte dintr-o mulțime fuzzy”
 - Rezultatul evaluării premiselor se aplică pentru funcția de apartenență a consecinței
 - Exemplu
 - **Dacă x este în A și y este în B atunci z este în C**
- Tipologie (în funcție de modul de aplicare a rezultatului asupra funcției de apartenență a consecinței)
 - Mulțimi fuzzy rezultat de tip *clipped*
 - Mulțimi fuzzy rezultat de tip *scaled*

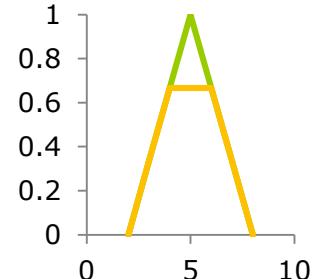
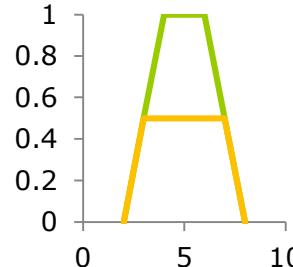
Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Evaluarea regulilor (inferență fuzzy) → evaluarea consecințelor

□ Modelul Mamdani

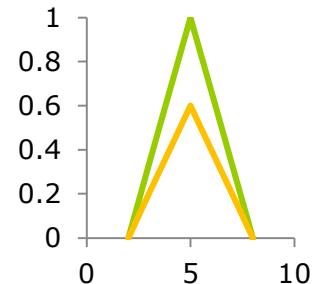
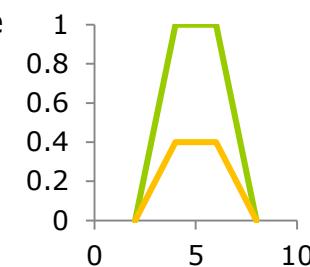
- Tipologie (în funcție de modul de aplicare a rezultatului asupra funcției de apartenență a consecinței)
 - Multimi fuzzy rezultat de tip *clipped*

- Funcția de apartenență a consecinței se taie la nivelul indicat de valoarea de adevăr a rezultatului
 - Avantaj → calcule ușoare
 - Dezavantaj → se pierd informații



□ Multimi fuzzy rezultat de tip *scaled*

- Funcția de apartenență a consecinței se ajustează prin scalare (multiplicare) valoarea de adevăr a rezultatului
 - Avantaj → se pierde mai puțină informație
 - Dezavantaj → calcule mai complicate



Sisteme inteligente – SBC – sisteme fuzzy

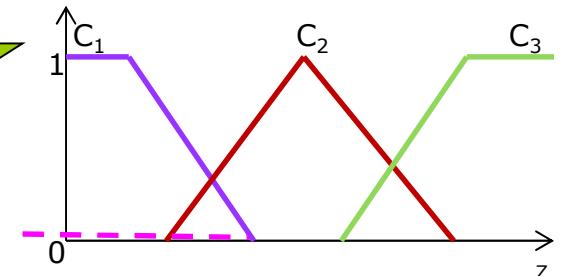
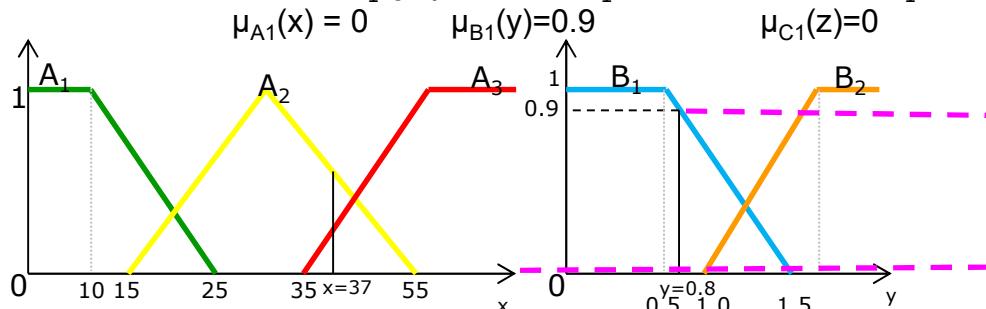
- Conținut și arhitectură → Evaluarea regulilor → evaluarea consecințelor → Modelul Mamdani
 - Exemplu pentru reglarea unui aparat de aer condiționat
 - Intrări:
 - x (temperatura – rece (A_1), normal (A_2), fierbinte (A_3)) și
 - y (umiditatea – mică (B_1), mare (B_2)))
 - Ieșiri:
 - z (puterea aparatului – redusă (C_1), medie (C_2), ridicată (C_3))
 - Date de intrare:
 - Temperatura x = 37
 - $\mu_{A1}(x)=0$, $\mu_{A2}(x)=0.6$, $\mu_{A3}(x)=0.3$
 - Umiditatea y = 0.8
 - $\mu_{B1}(x)=0.9$, $\mu_{B2}(x)=0$

		Data de intrare y	
		Mică	Mare
Data de intrare x	Rece	Redusă	Constantă
	Normal	Constantă	Mărită
	Fierbinte	Mărită	Mărită

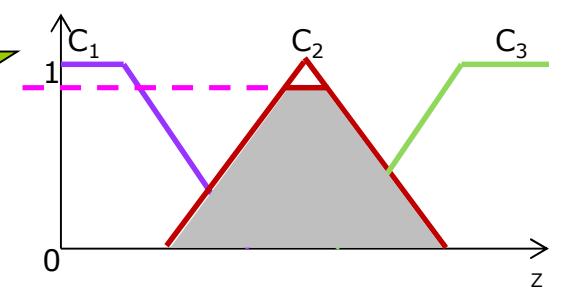
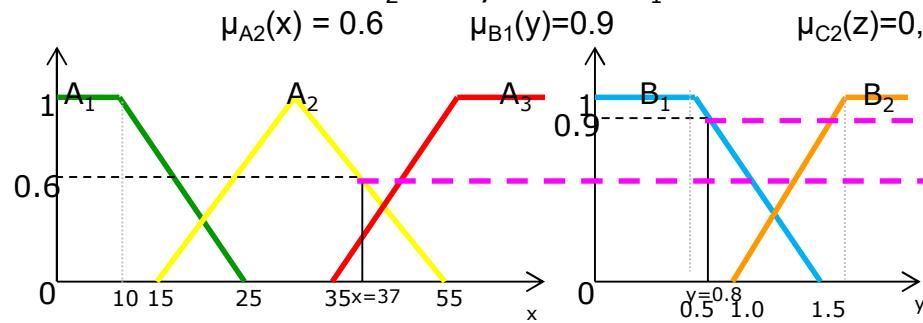
Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Evaluarea regulilor → evaluarea consecințelor → Modelul Mamdani

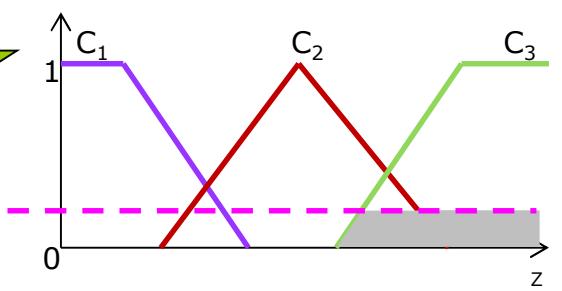
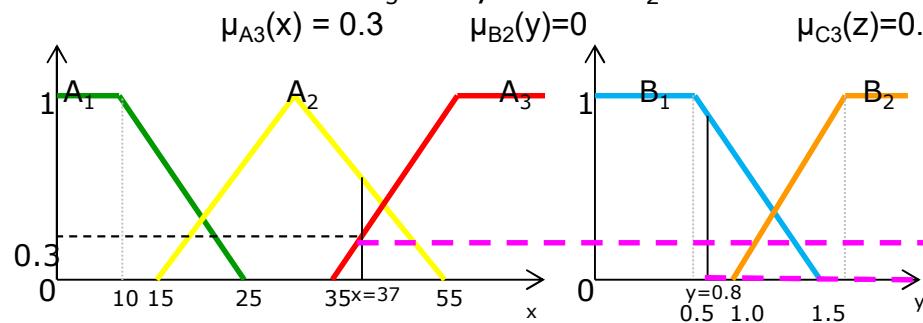
R1: dacă x este în A_1 și y este în B_1 atunci z este în C_1



R2: dacă x este în A_2 sau y este în B_1 atunci z este în C_2



R3: dacă x este în A_3 sau y este în B_2 atunci z este în C_3



Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Evaluarea regulilor (inferență fuzzy) → evaluarea consecințelor

□ Modelul Sugeno

■ Ideea de bază

- consecința regulii este de forma “variabila de ieșire este o funcție crisp care depinde de intrări”

□ Exemplu

- **Dacă x este în A și y este în B atunci z este $f(x,y)$**

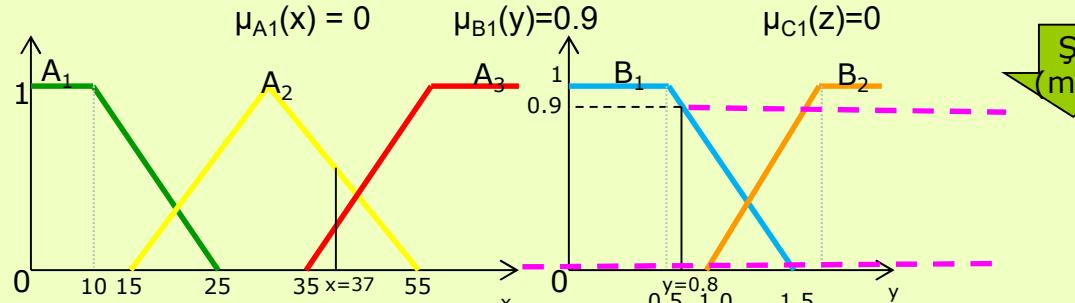
□ Tipologie (în funcție de caracteristicile lui $f(x,y)$)

- Model Sugeno de grad 0 → dacă $f(x,y) = k$ – constantă (funcțiile de apartenență ale consecințelor sunt de forma *singleton* – mulțime fuzzy a cărei funcție de apartenență ia valoarea 1 pentru un singur punct din universul de discurs și 0 în toate celelalte puncte)
 - Model Sugeno de grad 1 → dacă $f(x,y) = ax + by + c$

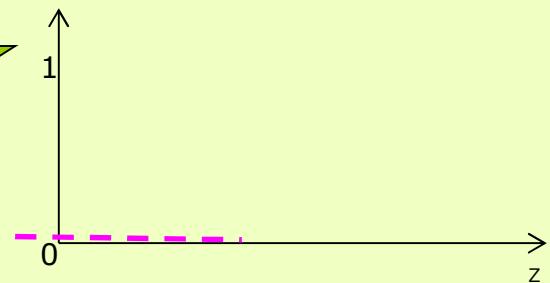
Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Evaluarea regulilor → evaluarea consecințelor → Modelul Sugeno

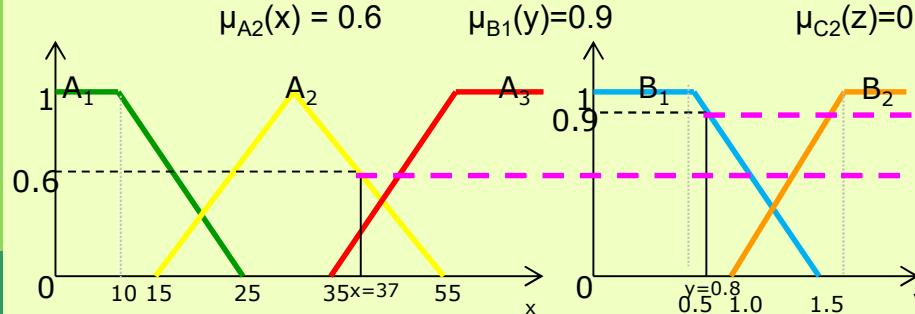
R1: dacă x este în A_1 și y este în B_1 atunci z este în C_1



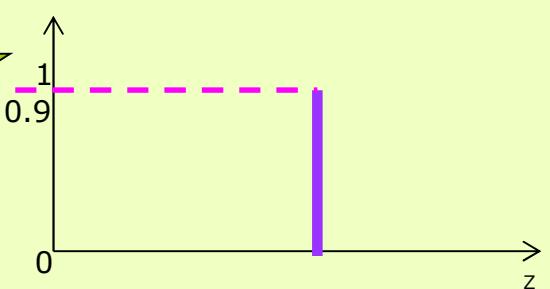
SI
(min)



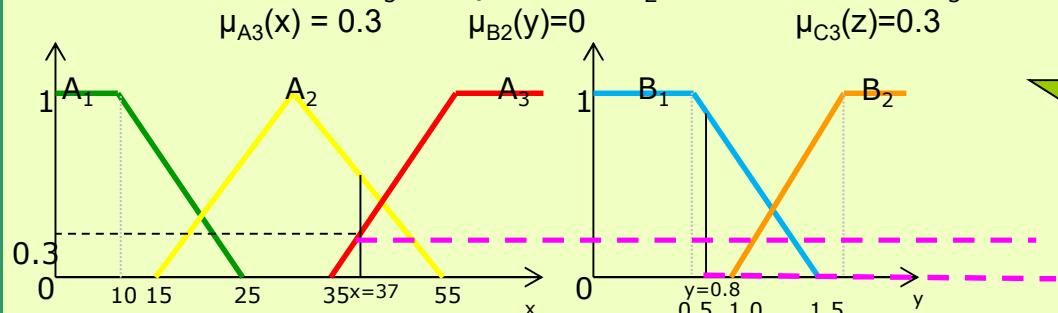
R2: dacă x este în A_2 sau y este în B_1 atunci z este în C_2



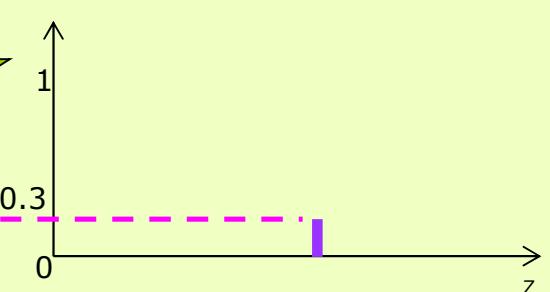
SAU
(max)



R3: dacă x este în A_3 sau y este în B_2 atunci z este în C_3



SAU
(max)



Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Evaluarea regulilor (inferență fuzzy) → evaluarea consecințelor

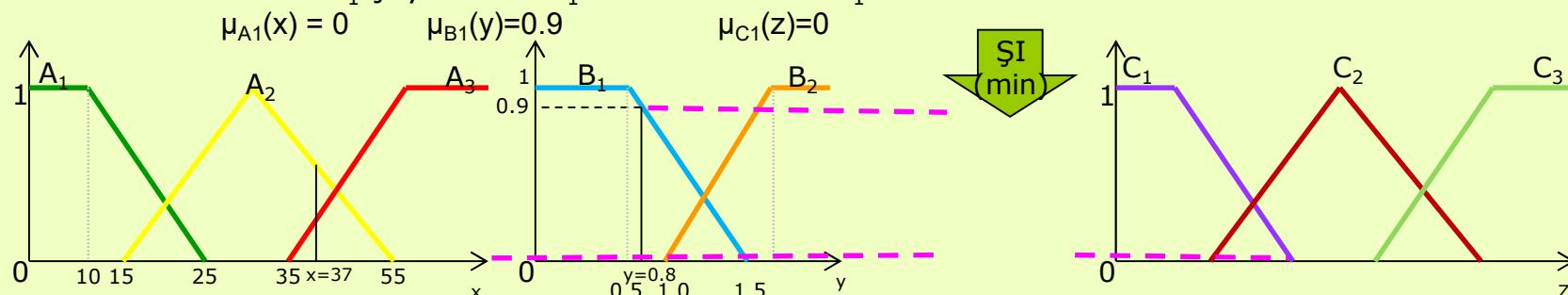
- Modelul Tsukamoto

- Ideea de bază
 - consecința regulii este de forma “variabila de ieșire face parte dintr-o mulțime fuzzy cu o funcție de apartenență monotonă”
 - Prin inferență se obține o ieșire crisp indusă gradul de satisfacere a regulii (*rule's firing strength*)

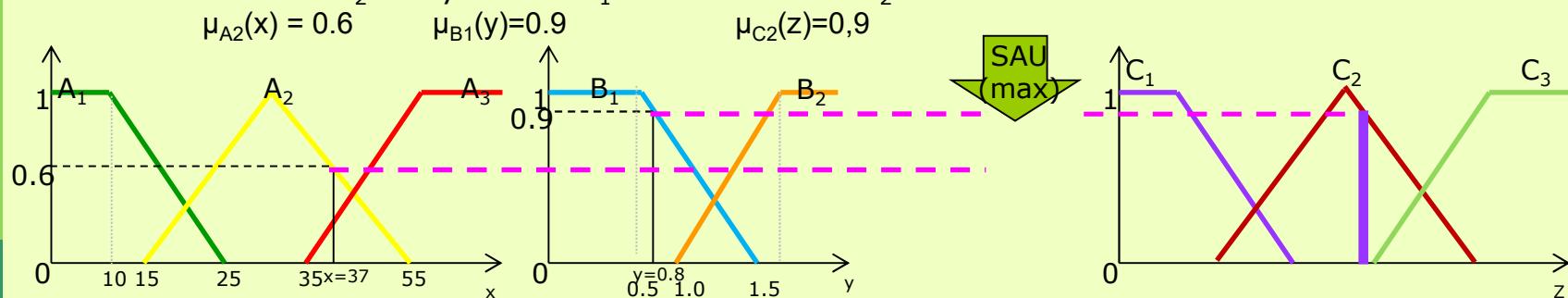
Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Evaluarea regulilor → evaluarea consecințelor → Modelul Tsukamoto

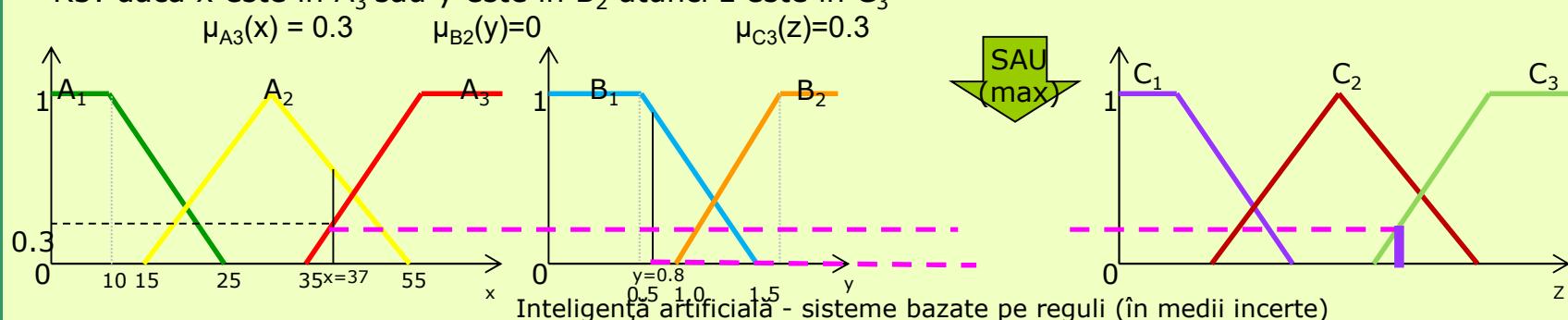
R1: dacă x este în A_1 și y este în B_1 atunci z este în C_1



R2: dacă x este în A_2 sau y este în B_1 atunci z este în C_2



R3: dacă x este în A_3 sau y este în B_2 atunci z este în C_3

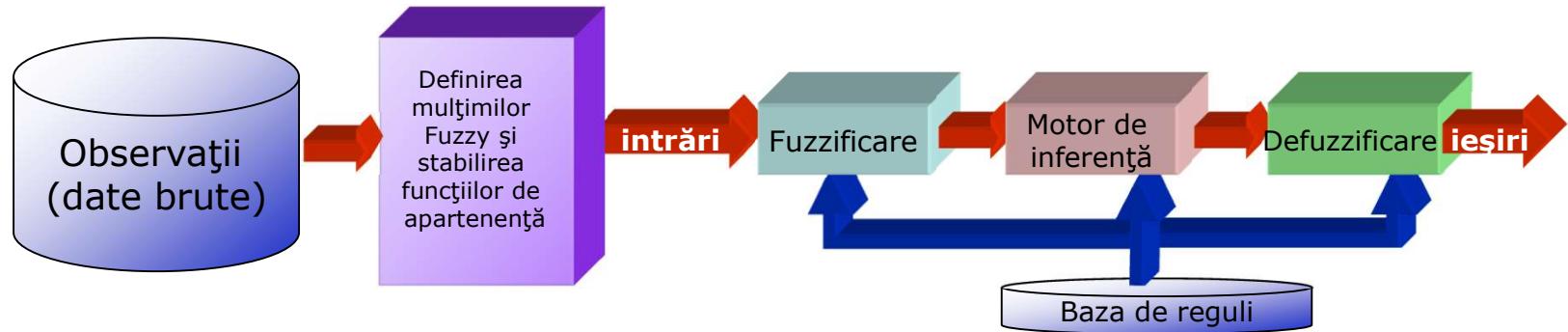


Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură

□ Pași în construirea unui sistem fuzzy

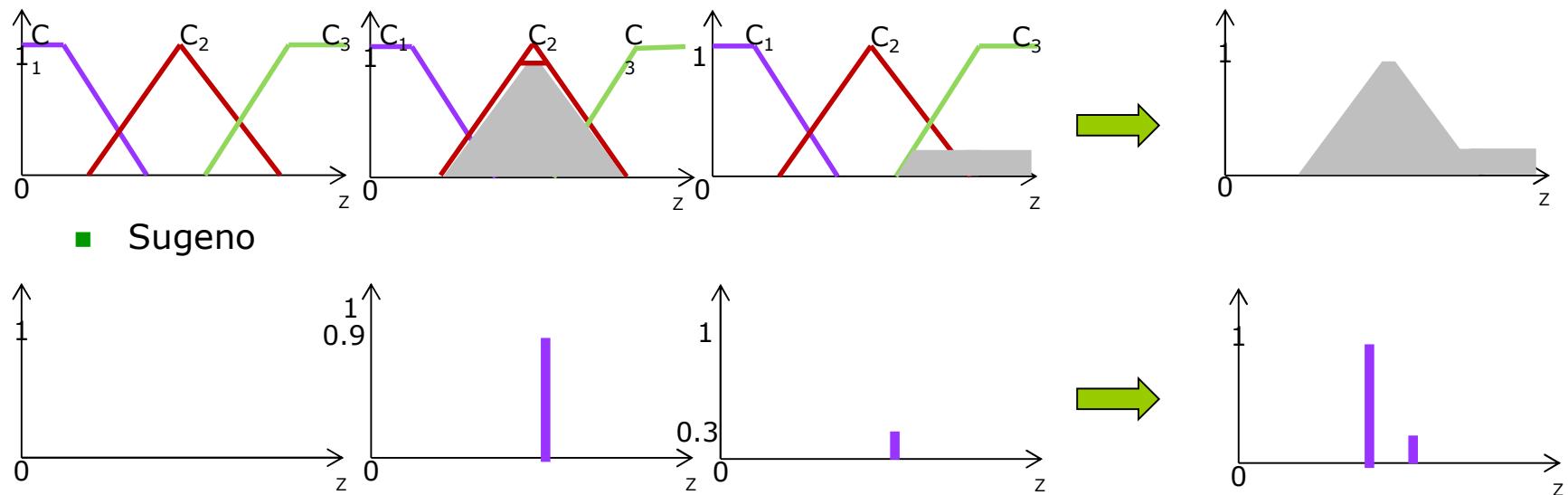
- Definirea intrărilor și ieșirilor de către expert
 - Datele de intrare și ieșire brute
 - Fuzzificarea datelor de intrare și ieșire
 - Stabilirea variabilelor fuzzy și a mulțimilor fuzzy pe baza funcțiilor de apartenență
- Construirea unei baze de reguli de către expert
 - Matricea de decizie a bazei de cunoștințe
- Evaluarea regulilor
 - Inferență – transformarea intrărilor fuzzy în ieșiri fuzzy prin aplicarea regulilor din baza de cunoștințe
- **Agregarea rezultatelor**
- Defuzificarea
- Interpretarea rezultatelor



Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Agregarea rezultatelor

- ❑ Unificarea ieșirilor tuturor regulilor aplicate
- ❑ Se iau funcțiile de apartenență corespunzătoare tuturor consecințelor și se combină într-o singură mulțime fuzzy, respectiv într-un singur rezultat
- ❑ Procesul de agregare are ca
 - intrări → funcțiile de apartenență (tăiate sau scalate) ale consecințelor
 - ieșire → o mulțime fuzzy pentru variabila de ieșire
- ❑ Exemplu
 - Mamdani

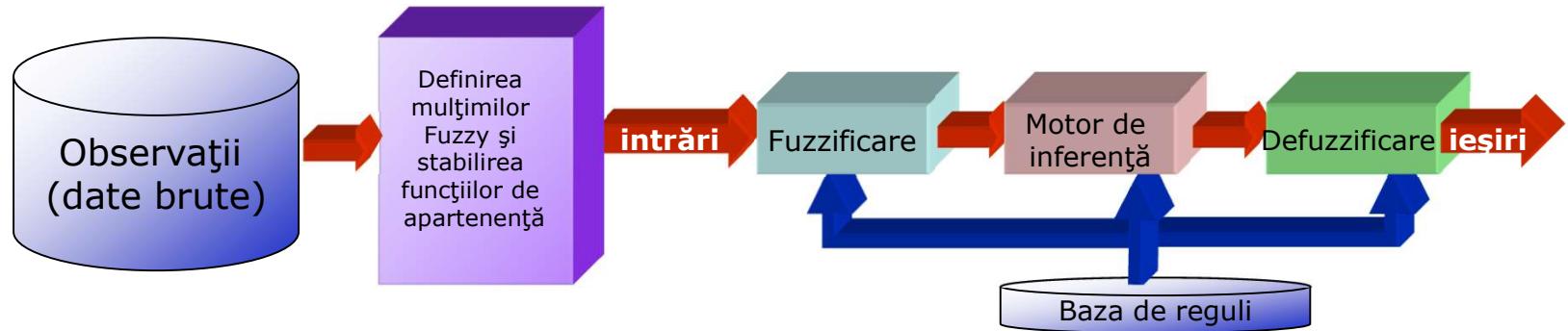


Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură

□ Pași în construirea unui sistem fuzzy

- Definirea intrărilor și ieșirilor de către expert
 - Datele de intrare și ieșire brute
 - Fuzzificarea datelor de intrare și ieșire
 - Stabilirea variabilelor fuzzy și a mulțimilor fuzzy pe baza funcțiilor de apartenență
- Construirea unei baze de reguli de către expert
 - Matricea de decizie a bazei de cunoștințe
- Evaluarea regulilor
 - Inferență – transformarea intrărilor fuzzy în ieșiri fuzzy prin aplicarea regulilor din baza de cunoștințe
- Agregarea rezultatelor
- **Defuzificarea**
- Interpretarea rezultatelor



Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Defuzificarea

- Ideea de bază
 - Transformarea rezultatului fuzzy al agregării într-o valoare crisp
 - Inferență → obținerea unor regiuni fuzzy pentru fiecare variabilă de ieșire
 - Defuzzificarea → Fiecare regiune de ieșire trebuie defuzzificată pentru a produce valori crisp
 - Se poate pierde informație
- Metode
 - Bazate pe centrul de greutate
 - Centrul de greutate (COG sau COA – *Centroid Area*)
 - Bisectoarea (BOA – *Bisector of area*)
 - Bazate pe maximul funcției de apartenență
 - Media maximelor (MOM - *Mean of maximum*)
 - Cel mai mic maxim (SOM - *Smallest of maximum*)
 - Cel mai mare maxim (LOM - *Largest of maximum*)

Sisteme inteligente – SBC – sisteme fuzzy

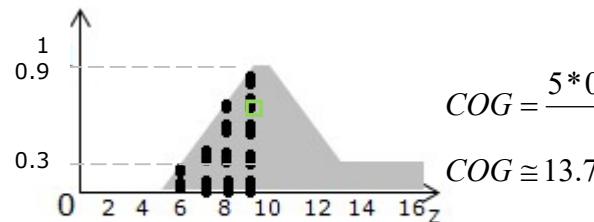
Conținut și arhitectură → Defuzificarea → Metode

- ❑ Centrul de greutate (COG sau COA – Centroid Area)
 - Găsește punctul z din mijlocul mulțimii aggregate

$$COG = \frac{\sum_{i=0}^n x_i \mu_A(x_i)}{\sum_{i=0}^n \mu_A(x_i)} \text{ sau } COG = \frac{\int x_i \mu_A(x_i) dx}{\int \mu_A(x_i) dx}$$

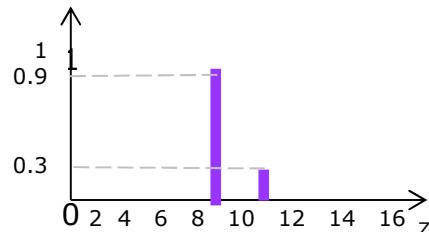
- Exemplu

- ❑ Modelul Mamdani → estimarea centrului prin calcularea COG pe baza unui eșantion de n puncte (x_i , $i = 1, 2, \dots, n$) din mulțimea fuzzy rezultat



$$COG = \frac{5*0 + 6*0.3 + 7*0.5 + 8*0.7 + 9*0.9 + 10*0.9 + 11*0.7 + 12*0.5 + 13*0.3 + 14*0.3 + 15*0.3 + 16*0.3}{0 + 0.3 + 0.5 + 0.7 + 0.9 + 0.9 + 0.7 + 0.5 + 0.3 + 0.3 + 0.3 + 0.3}$$
$$COG \approx 13.7$$

- ❑ Modelul Sugeno sau Tsukamoto, COG devine media ponderată a celor m valori crisp rezultate prin aplicarea tuturor regulilor (în număr de m)



$$COG = \frac{9*0.9 + 11*0.3}{0.9 + 0.3}$$
$$COG \approx 9.5$$

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Defuzificarea → Metode

□ Bisectoarea (BOA – Bisector of area)

- Găsește punctul z în care o linie verticală va separa mulțimea agregată în 2 părți de arii egale

$$BOA = \int_{\alpha}^z \mu_A(x) dx = \int_z^{\beta} \mu_A(x) dx,$$

unde $\alpha = \min\{x \mid x \in A\}$ și $\beta = \max\{x \mid x \in A\}$

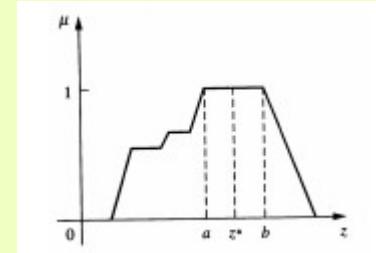
Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Defuzificarea → Metode

- Media maximelor (MOM - *Mean of maximum*)

- Găsește punctul z care reprezintă media punctelor din mulțimea agregată care au valoarea funcției de apartenență (μ) maximă

$$MOM = \frac{\sum_{x_i \in \max \mu} x_i}{|\max \mu|}, \text{ unde } \max \mu = \mu^* = \{x \mid x \in A, \mu(x) = \text{maxim}\}$$



- Cel mai mic maxim (SOM - *Smallest of maximum*)

- Găsește cel mai mic punct z din mulțimea agregată care are valoarea funcției de apartenență (μ) maximă

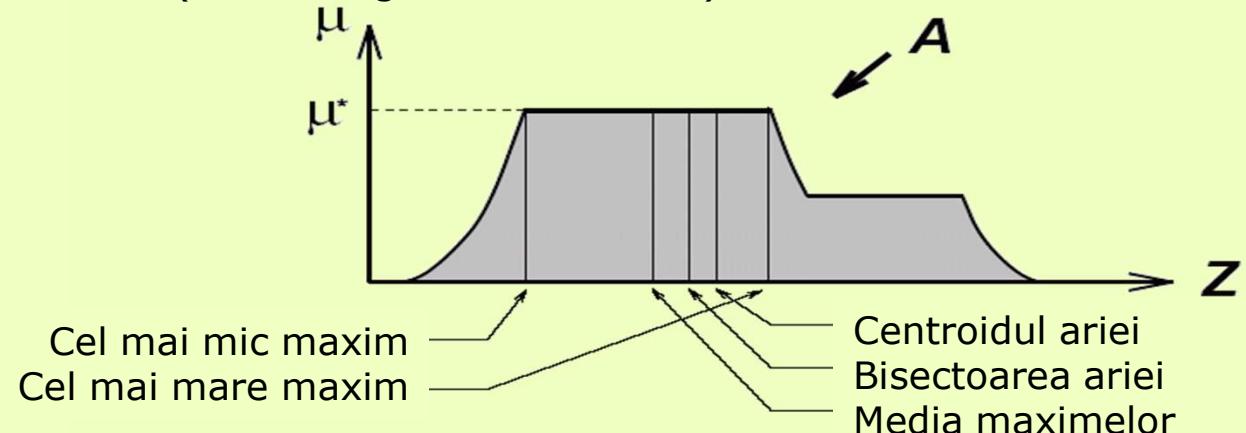
- Cel mai mare maxim (LOM - *Largest of maximum*)

- Găsește cel mai mare punct z din mulțimea agregată care are valoarea funcției de apartenență (μ) maximă

Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură → Defuzificarea

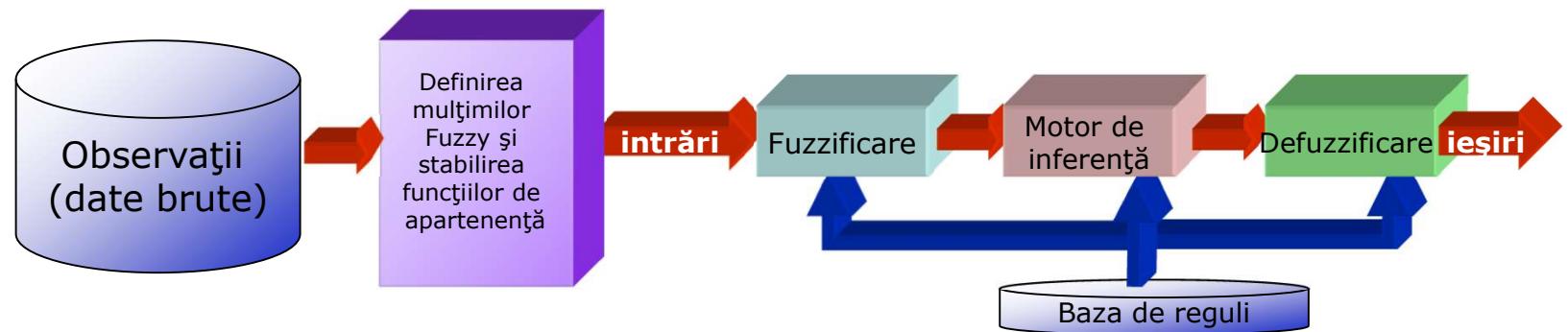
- Ideea de bază
 - Transformarea rezultatului fuzzy al agregării într-o valoare crisp
- Metode
 - Bazate pe centrul de greutate
 - Centrul de greutate (COG sau COA – Centroid Area)
 - Bisectoarea (BOA – Bisector of area)
 - Bazate pe maximul funcției de apartenență
 - Media maximelor (MOM - Mean of maximum)
 - Cel mai mic maxim (SOM - Smallest of maximum)
 - Cel mai mare maxim (LOM - Largest of maximum)



Sisteme inteligente – SBC – sisteme fuzzy

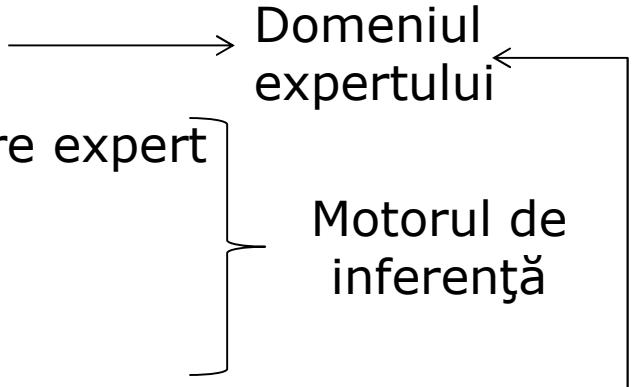
Conținut și arhitectură

- Pași în construirea unui sistem fuzzy
 - Definirea intrărilor și ieșirilor de către expert
 - Datele de intrare și ieșire brute
 - Fuzzificarea datelor de intrare și ieșire
 - Stabilirea variabilelor fuzzy și a mulțimilor fuzzy pe baza funcțiilor de apartenență
 - Construirea unei baze de reguli de către expert
 - Matricea de decizie a bazei de cunoștințe
 - Evaluarea regulilor
 - Inferență – transformarea intrărilor fuzzy în ieșiri fuzzy prin aplicarea regulilor din baza de cunoștințe
 - Agregarea rezultatelor
 - Defuzificarea
 - **Interpretarea rezultatelor**



Sisteme inteligente – SBC – sisteme fuzzy

Conținut și arhitectură

- Pași în construirea unui sistem fuzzy
 - Definirea intrărilor și ieșirilor de către expert
 - Datele de intrare și ieșire brute
 - Fuzzificarea datelor de intrare și ieșire
 - Construirea unei baze de reguli de către expert
 - Evaluarea regulilor
 - Agregarea rezultatelor
 - Defuzificarea
 - Interpretarea rezultatelor
- 

Sisteme inteligente – SBC – sisteme fuzzy

□ Avantaje

- Se pot folosi reguli pentru exprimarea impreciziei și aproximărilor din lumea reală
- Ușor de înțeles, testat și întreținut
- Robuste → pot opera și când regulile nu sunt foarte clare
- Necesită mai puține reguli decât alte SBC
- Regulile se evaluatează în paralel

□ Dezavantaje

- Necesită multe simulări și testări
- Nu învață singure
- Este dificilă stabilirea celor mai corecte reguli
- Lipsa unui model matematic exact

Sisteme inteligente – SBC – sisteme fuzzy

Aplicații

- ❑ Control în spațiu
 - Altitudinea sateliștilor
 - Reglaje în avioane
- ❑ Control auto
 - Transmisie automată, controlul traficului, Sisteme împotriva blocării roților în timpul frânării
 - *Dacă frâna este caldă și viteza nu este foarte mare atunci presiunea de frânare se reduce ușor*
- ❑ Business
 - Sisteme de decizie, evaluarea personalului, managementul fondurilor, predicții de piață, sisteme de asigurarea a securității tranzacțiilor
- ❑ Industrie
 - Controlul schimbului de energie, controlul purificării apei,
 - Controlul pH-ului, distilarea chimică, producerea de polimeri, formarea metalelor
- ❑ Electronică
 - Controlul expunerii automate a camerelor foto/video, controlul umidității, sisteme de aer condiționat, Sisteme pentru reglajul dușului
 - Sisteme pentru reglarea congelatoarelor
 - Sisteme pentru reglarea mașinilor de spălat (Încărcătura, concentrația de detergent, etc.)

Sisteme inteligente – SBC – sisteme fuzzy

Aplicații

- ❑ Alimentație
 - Procese de producere a brânzei
- ❑ Militar
 - Recunoașterea subacvatică, recunoașterea imaginilor în infraroșu, sisteme de decizie în traficul naval
- ❑ Marină
 - Piloti automați pentru nave, și submarine selecția automată a rutelor
- ❑ Medical
 - Sisteme de diagnosticare (diabet, cancer), controlul presiunii arteriale în timpul anesteziei, modelarea rezultatelor neuropatologice pentru pacienții cu Alzheimer
- ❑ Robotică
 - Controlul manipulatorilor flexibili și a brațelor robotice



Recapitulare

□ SBC

- Sisteme computaționale în care baza de cunoștințe și modulul de control se suprapun

□ SBC pot funcționa

- În medii certe
 - SBL
 - SBR
- În medii incerte
 - Sisteme de tip Bayes
 - Regulile au asociate probabilități de realizare
 - Sisteme bazate pe factori de certitudine
 - Faptele și regulile au asociați factori de certitudine
 - Sisteme fuzzy
 - Faptele au asociate grade de apartenență la anumite mulțimi

-
- Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean –
www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan -
www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop -
www.cs.ubbcluj.ro/~hfpop