



# Inteligența Artificială

---

**Universitatea Politehnica București**  
**Anul universitar 2013-2014**

Adina Magda Florea



# Curs 2 - supliment

---

## Strategii de cautare

- Problema satisfacerii restrictiilor



## 4. Problema satisfacerii restrictiilor

---

$$\begin{array}{lll} \{X_1, \dots, X_n\} & R_j \subset D_{i1} \times \dots \times D_{ij} & X_{i1}, \dots, X_{ij} \\ \{D_1, \dots, D_n\} & & \\ \{R_1, \dots, R_k\} & & \{(X_1, x_{j1}), \dots, (X_n, x_{jn})\} \end{array}$$

- Gradul unei variabile
- Aritatea unei restrictii
- Gradul problemei
- Aritatea problemei



## 4.1 Instante ale CSP

---

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

CSP – problema de cautare, in NP

- Sub-clase de probleme cu timp polinomial
- Reducerea timpului (reducerea sp. de cautare)

**Algoritm:**                    **Backtracking nerecursiv**

1.     Initializeaza FRONTIERA cu  $\{S_i\}$                     /\*  $S_i$  este starea initiala \*/
  2.     **daca** FRONTIERA = { }  
      **atunci intoarce** INSUCCES                    /\* nu exista solutie \*/
  3.     Fie S prima stare din FRONTIERA
  4.     **daca** toate starile succesoare ale lui S au fost deja generate  
      **atunci**
    - 4.1. Elimina S din FRONTIERA
    - 4.2. **repetă de la 2**
  5.     **altfel**
    - 5.1. Genereaza S', noua stare succesoare a lui S
    - 5.2. Introduce S' la începutul listei FRONTIERA
    - 5.3. Stabileste legatura  $S' \rightarrow S$
    - 5.4. Marcheaza în S faptul ca starea succesoare S' a fost generata
    - 5.5. **daca** S' este stare finala  
      **atunci**
      - 5.5.1.     Afiseaza calea spre solutie urmarind legaturile  $S' \rightarrow S ..$
      - 5.5.2.     **întoarce** SUCCES                    /\* s-a gasit o solutie \*/
    - 5.6. **repetă de la 2**
- sfarsit.**



## 4.2 Notatii

---

- $X_1, \dots, X_N$  variabilele problemei,  $N$  fiind numarul de variabile ale problemei,
- $U$  - intreg care reprezinta indicele variabilei curent selectate pentru a i se atribui o valoare
- $F$  - vector indexat dupa indicii variabilelor, in care sunt memorate selectiile de valori facute de la prima variabila si pana la variabila curenta

$$\text{Relatie}(U_1, F[U_1], U_2, F[U_2]) = \begin{cases} \text{adevarat} & \text{daca exista restrictia } R_{U_1 U_2}(F[U_1], F[U_2]) \\ & R_{U_1 U_2} \subset D_{U_1} \times D_{U_2} \\ \text{fals} & \text{in caz contrar} \end{cases}$$

**Algorithm:**      **Backtracking recursive**

## BKT (U, F)

**pentru** fiecare valoare  $V$  a lui  $X_U$  **executa**

1.  $F[U] \leftarrow V$
2. **daca** Verifica (U,F) = adevarat  
**atunci**
  - 2.1. **daca**  $U < N$   
**atunci** BKT(U+1, F)
  - 2.2. **altfel**
    - 2.2.1. Afiseaza valorile din vectorul F  
/\* F reprezinta solutia problemei \*/
    - 2.2.2. **intrerupe ciclul**

**sfarsit.**

## Verifica (U,F)

1. test  $\leftarrow$  adevarat
  2. I  $\leftarrow$  U - 1
  3. **cat timp** I > 0 **executa**
    - 3.1. test  $\leftarrow$  Relatie(I, F[I], U, F[U])
    - 3.2. I  $\leftarrow$  I - 1
    - 3.3. **daca** test = fals  
**atunci intrerupe ciclul**
  4. **intoarce** test
- sfarsit.**





## 3.3 Imbunatatirea performantelor BKT

---

### Algoritmi de imbunatatire a consistentei reprezentarii

- Consistenta locala a arcelor sau a cailor in graful de restrictii

### Algoritmi hibrizi

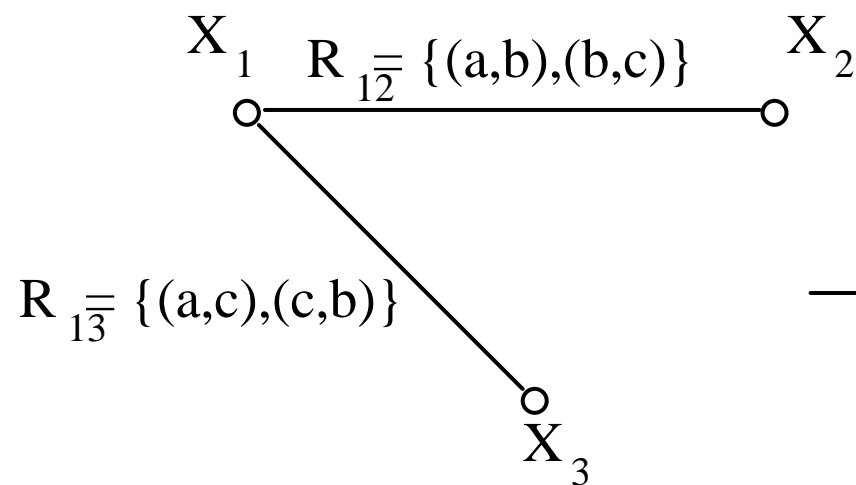
- Imbunatatesc performantele rezolvarii prin reducerea numarului de teste.
  - *Tehnici prospective:*
    - Algoritmul de cautare cu predictie completa
    - Algoritmul de cautare cu predictie partiala
    - Algoritmul de cautare cu verificare predictiva
  - *Tehnici retrospective:*
    - Algoritmul de backtracking cu salt
    - Algoritmul de backtracking cu marcarea
    - Algoritmul de backtracking cu multime conflictuala

### Utilizarea euristiciilor

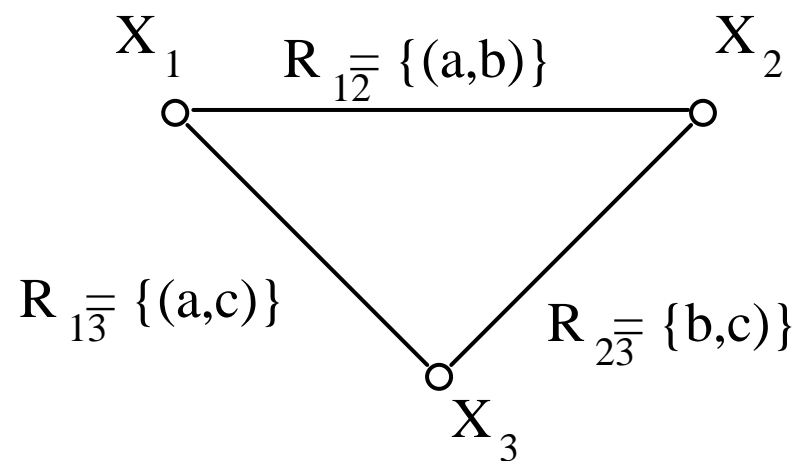
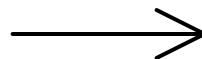
# Algoritmi de imbunatatire a consistentei reprezentarii

## ■ Propagarea restrictiilor

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



(a)



(b)



## 4.4 Propagarea locala a restrictiilor

---

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

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

1. Creeaza o coada  $Q \leftarrow \{ (X_i, X_j) \mid (X_i, X_j) \in \text{Multime arce}, i \neq j \}$
  2. **cat timp**  $Q$  nu este vida **executa**
    - 2.1. Elimina din  $Q$  un arc  $(X_k, X_m)$
    - 2.2. Verifica( $X_k, X_m$ )
    - 2.3. **daca** subprogramul Verifica a facut schimbari in domeniul variabilei  $X_k$   
**atunci**  
 $Q \leftarrow Q \cup \{ (X_i, X_k) \mid (X_i, X_k) \in \text{Multime arce}, i \neq k, m \}$
- sfarsit.**

**Verifica ( $X_k, X_m$ )**

**pentru** fiecare  $x \in D_k$  **executa**

1. **daca** nu exista nici o valoare  $y \in D_m$  astfel incat  $R_{km}(x, y)$   
**atunci** elimina  $x$  din  $D_k$
- sfarsit.**



# Cale-consistenta

---

- O cale de lungime  $m$  prin nodurile  $i_0, \dots, i_m$  ale unui graf de restrictii orientat se numeste ***m-cale-consistenta*** daca si numai daca pentru orice valoare  $x \in D_{i_0}$ , domeniul variabilei  $i_0$  si o valoare  $y \in D_{i_m}$ , domeniul variabilei  $i_m$ , pentru care  $R_{i_0 i_m}(x, y)$ , exista o secventa de valori  $z_1 \in D_{i_1} \dots z_{m-1} \in D_{i_{m-1}}$  astfel incat  $R_{i_0 i_1}(x, z_1), \dots, R_{i_{m-1} i_m}(z_{m-1}, y)$
- **Graf de restrictii orientat *m-arc-consistent***
- Graf minim de restrictii
- $n$ -cale-consistenta
- Comentarii



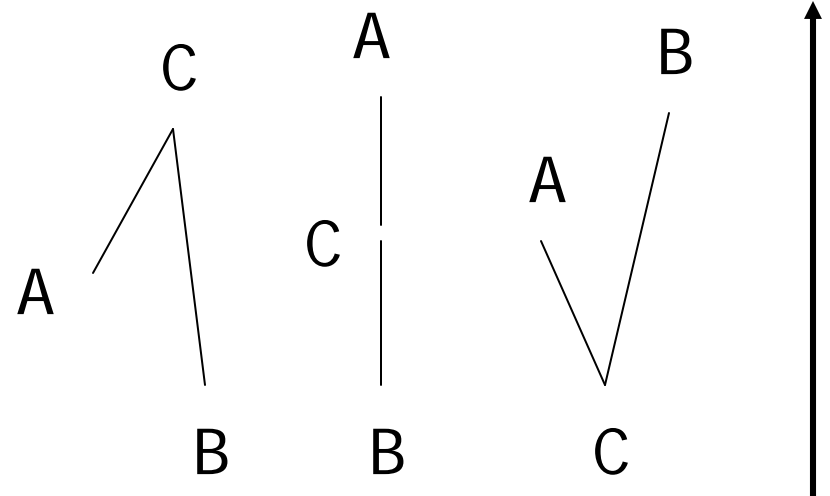
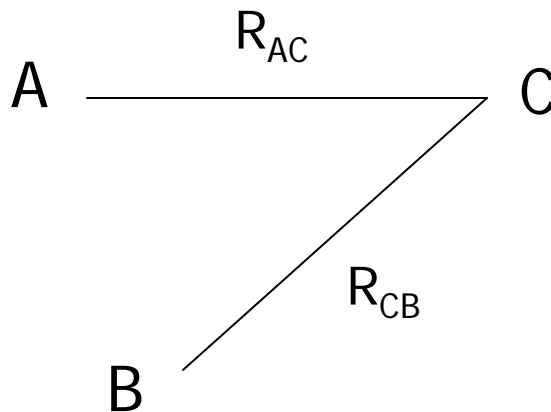
# Complexitate

---

- **N** - numarul de variabile
- **a** - cardinalitatea maxima a domeniilor de valori ale variabilelor
- **e** - numarul de restrictii.
- Algoritmului de realizare a arc-consistentei - *AC-3*: complexitate timp este  $O(e * a^3)$ ; complexitate spatiu:  $O(e + N * a)$
- S-a gasit si un algoritm de complexitate timp  $O(e * a^2)$  – *AC-4*
- Algoritmul de realizare a 2-cale-consistentei - *PC-4*: complexitatea timp  $O(N^3 * a^3)$

## 4.5 CSP fara bkt - conditii

- Graf de restrictii ordonat
- Latimea unui nod
- Latimea unei ordonari a nodurilor
- Latimea unui graf de restrictii





## Teoreme

---

- Daca un graf de restrictii **arc-consistent** are *latimea egala cu unu* (i.e. este un arbore), atunci problema asociata grafului admite o solutie fara backtracking.
- Daca un graf de restrictii **2-cale-consistent** are *latimea egala cu doi*, atunci problema asociata grafului admite o solutie fara backtracking.





## d-consistenta

---

- Fiind data o ordonare  $d$  a variabilelor unui graf de restrictii  $R$ , **graful  $R$  este  $d$ -arc-consistent** daca toate arcele avand directia  $d$  sint arc-consistente.
- Fie un graf de restrictii  $R$ , avind ordonarea variabilelor  $d$  cu latimea egala cu unu. Daca  $R$  este  $d$ -arc-consistent atunci cautarea dupa directia  $d$  este fara backtracking.



# d-consistenta

---

**Algoritm: Realizarea d-arc-consistentei unui graf de restrictii cu ordonarea variabilelor  $(X_1, \dots, X_N)$**

**pentru  $I \leftarrow N$  la 1 executa**

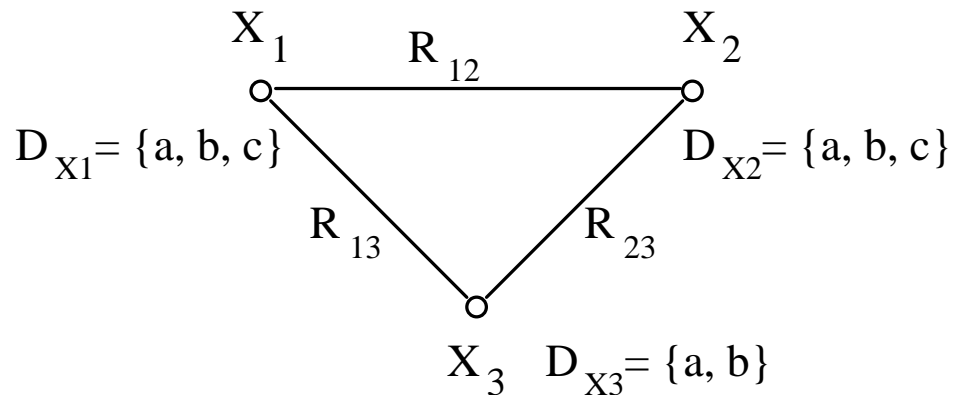
**1. pentru fiecare arc  $(X_j, X_i)$  cu  $j < i$  executa**

**1.1. Verifica( $X_j, X_i$ )**

**sfarsit.**

- Complexitatea timp:  $O(e \cdot a^2)$

# Cale-consistenta – reprezentare matriceala



$$\begin{aligned}
 R_{11} &= \begin{matrix} & \begin{matrix} a & b & c \end{matrix} \\ \begin{matrix} a \\ b \\ c \end{matrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{matrix} & R_{12} = \begin{matrix} & \begin{matrix} a & b & c \end{matrix} \\ \begin{matrix} a \\ b \\ c \end{matrix} & \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} & R_{13} = \begin{matrix} & \begin{matrix} a & b & c \end{matrix} \\ \begin{matrix} a \\ b \\ c \end{matrix} & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\
 R_{22} &= \begin{matrix} & \begin{matrix} a & b & c \end{matrix} \\ \begin{matrix} a \\ b \\ c \end{matrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & R_{23} = \begin{matrix} & \begin{matrix} a & b & c \end{matrix} \\ \begin{matrix} a \\ b \\ c \end{matrix} & \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & R_{33} = \begin{matrix} & \begin{matrix} a & b & c \end{matrix} \\ \begin{matrix} a \\ b \\ c \end{matrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
 \end{aligned}$$



# Operatii asupra matricilor de restrictii

- **Intersectia** a doua restrictii - o restrictie care permite numai perechile de valori ce apar în ambele restrictii.
- Intersectia a doua restrictii se notează cu  $\&$ .
- **Compunerea** a doua restrictii,  $R_{12}$  si  $R_{23}$ , între variabilele  $X_1$  si  $X_2$  si respectiv  $X_2$  si  $X_3$ , determina o **noua restrictie  $R_{13}$**  definita astfel:  $(X_1, X_3) \in R_{13}$  daca exista  $x_2 \in D_{X_2}$  astfel încat  $(X_1, X_2) \in R_{12}$  si  $(X_2, X_3) \in R_{23}$
- În notatia cu matrice, compunerea a doua restrictii se poate obtine prin produsul de matrice  $R_{13}=R_{12}*R_{23}$  si înlocuind în matricea rezultat valorile pozitive cu 1.

**Algoritm:**      PC4- **Realizarea d-cale-consistentei unui graf de restrictii cu variabile ordonate**

$d = (X_1, X_2, \dots, X_n) \quad (X_1, X_2, \dots, X_n)$

$V$  – noduri,  $E$  - arce

---

1.  $Y^0 \leftarrow \{R_{ij} \mid 1 \leq i, j \leq N\}$

2. **pentru**  $k \leftarrow N$  **pana la 1** **execută**

2.1. **pentru** fiecare pereche  $(i, k)$  cu  $1 \leq i \leq k$  **si**  $(X_i, X_k) \in E$   
**executa**

*/\* $X_i$  legat la  $X_k$  \*/*

2.1.1.  $Y'_{ii} \leftarrow Y_{ii}^0 \& (Y_{ik} \cdot Y_{kk} \cdot Y_{ki})$

2.2. **pentru** fiecare triplet  $(i, j, k)$  cu  $1 \leq i, j \leq k$  **si**  $(X_i, X_k) \in E$  **si**  
 $(X_j, X_k) \in E$  **executa**

2.2.1.  $Y_{ij} \leftarrow Y_{ij} \& (Y_{ik} \cdot Y_{kk} \cdot Y_{kj})$

2.2.2.  $E \leftarrow E \cup (X_i, X_j)$

**sfarsit.**

## 3.6 Tehnici prospective

- Conventii
- Notatii: U, N, F (F[U]), T (T[U] ... X<sub>U</sub>), TNOU, LINIE\_VIDA

$$\text{Relatie}(U1, L1, U2, L2) = \begin{cases} \text{adevarat} & \text{daca } R_{U_1 U_2}(L_1, L_2) \\ \text{fals} & \text{în caz contrar} \end{cases}$$

- *Verifica\_Inainte*
- *Verifica\_Viitoare*
- **Predictie completa**
- **Predictie partiala**
- **Verificare predictiva**

**Algoritm:**        **Backtracking cu predictie completa**

**Predictie(U, F, T)**

**pentru** fiecare element L din T[U] **executa**

1.  $F[U] \leftarrow L$
2. **daca**  $U < N$  **atunci**        *//verifica consistenta atribuirii*
  - 2.1  $TNOU \leftarrow \text{Verifica\_Inainte}(U, F[U], T)$
  - 2.2 **daca**  $TNOU \neq \text{LINIE\_VIDA}$   
      **atunci**  $TNOU \leftarrow \text{Verifica\_Viitoare}(U, TNOU)$
  - 2.3 **daca**  $TNOU \neq \text{LINIE\_VIDA}$   
      **atunci** **Predictie** (U+1, F, TNOU)
3. **altfel** afiseaza atribuirile din F

**sfarsit**

## Verifica\_Inainte (U, L, T)

1. TNOU  $\leftarrow$  tabela vida
  2. **pentru** U2  $\leftarrow$  U+1 pana la N **executa**
    - 2.1 **pentru** fiecare element L2 din T[U2] **executa**
      - 2.1.1 **daca** Relatie(U, L, U2, L2) = adevarat  
atunci introduce L2 in TNOU[U2]
    - 2.2 **daca** TNOU[U2] este vida  
**atunci** intoarce LINIE\_VIDA
  3. **intoarce** TNOU
- sfarsit**



## **Verifica\_Viitoare (U, TNOU)**

**daca**  $U+1 < N$  **atunci**

1. **pentru**  $U1 \leftarrow U+1$  **pana la**  $N$  **executa**

1.1 **pentru** fiecare element  $L1$  din  $TNOU[U1]$  **executa**

1.1.1 **pentru**  $U2 \leftarrow U+1$  **pana la**  $N$ ,  $U2 \neq U1$  **executa**

i. **pentru** fiecare element  $L2$  din  $TNOU[U2]$  **executa**

- **daca**  $Relatie(U1, L1, U2, L2) = \text{adevarat}$

**atunci intrerupe ciclul** //dupa  $L2$

ii. **daca** nu s-a gasit o valoare consistenta pentru  $U2$   
**atunci**

- elimina  $L1$  din  $TNOU[U1]$

- **intrerupe ciclul** // dupa  $U2$

1.2 **daca**  $TNOU[U1]$  este vida

**atunci** intoarce  $LINIE\_VIDA$

2. **intoarce**  $TNOU$

sfarsit



# BKT cu predictie partiala

- Se modifica Verifica\_Viitoare pasii marcati cu rosu

**Verifica\_Viitoare (U, TNOU)**

**daca**  $U+1 < N$  **atunci**

**1. pentru**  $U1 \leftarrow U+1$  **pana la**  $N - 1$  **executa**

**1.1 pentru** fiecare element  $L1$  din  $TNOU[U1]$  **executa**

**1.1.1 pentru**  $U2 \leftarrow U1+1$  **pana la**  $N$  **executa**

**i. pentru** fiecare element  $L2$  din  $TNOU[U2]$  **executa**

- **daca**  $Relatie(U1, L1, U2, L2) = \text{adevarat}$   
**atunci intrerupe ciclul** //dupa  $L2$

**ii. daca** nu s-a gasit o valoare consistenta pentru  $U2$   
**atunci**

- elimina  $L1$  din  $TNOU[U1]$   
- **intrerupe ciclul** // dupa  $U2$

**1.2 daca**  $TNOU[U1]$  este vida

**atunci** intoarce  $LINIE\_VIDA$

**2. intoarce**  $TNOU$

sfarsit



# BKT cu verificare predictiva

- Se elimina apelul Verifica\_Viitoare(U, TNOU) in subprogramul Predictie

**Algoritm:**            **Backtracking cu verificare predictiva**

**Predictie(U, F, T)**

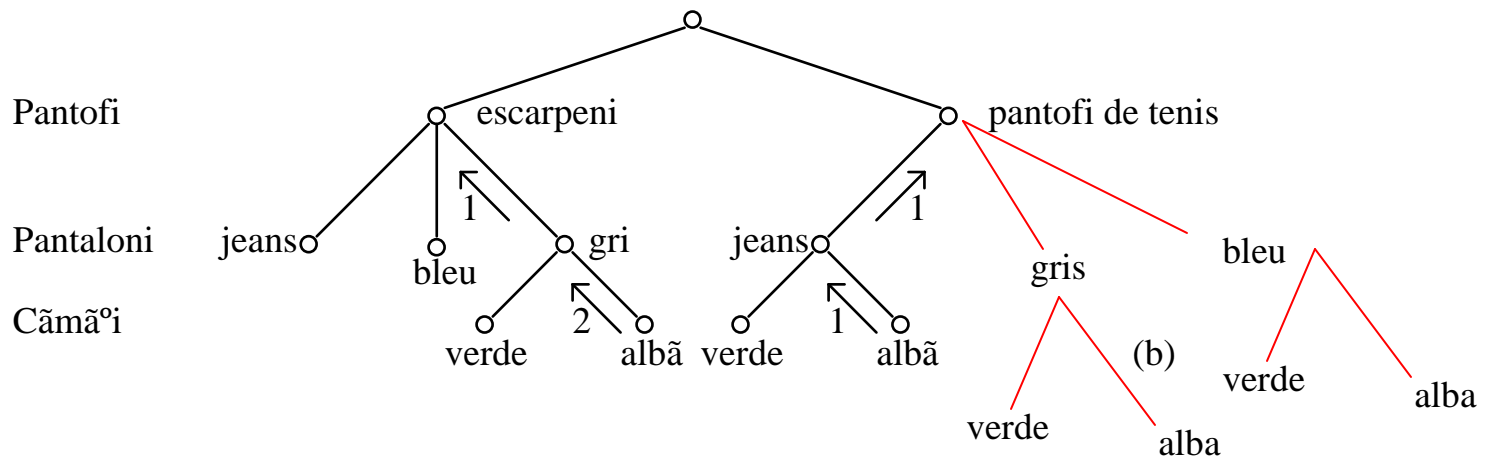
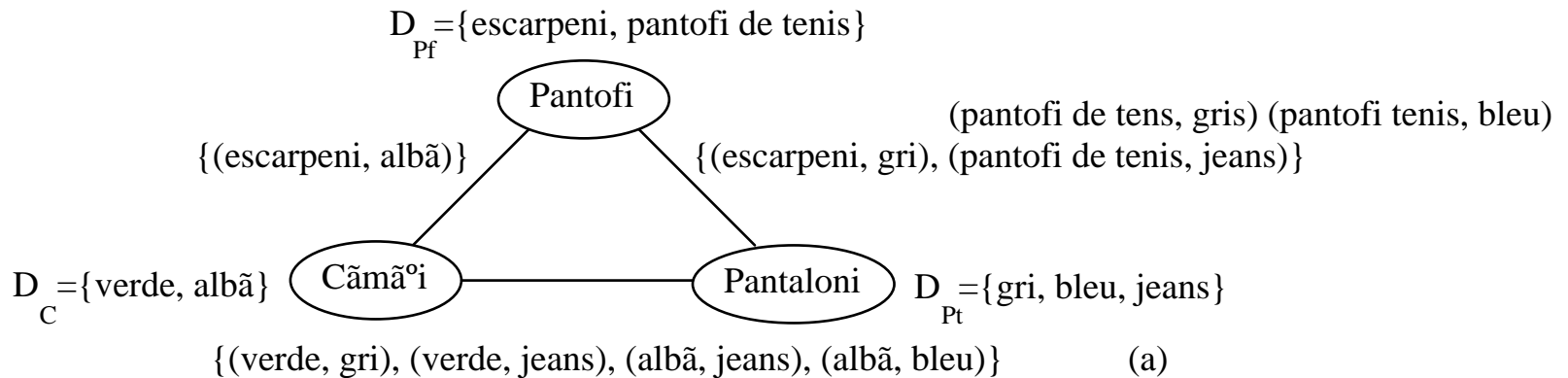
**pentru** fiecare element L din T[U] **executa**

1.  $F[U] \leftarrow L$
2. **daca**  $U < N$  **atunci**    *//verifica consistenta atribuirii*
  - 2.1  $TNOU \leftarrow \text{Verifica\_Inainte}(U, F[U], T)$
  - ~~2.2 **daca**  $TNOU \neq \text{LINIE\_VIDA}$~~
  - ~~**atunci**  $TNOU \leftarrow \text{Verifica\_Viitoare}(U, TNOU)$~~
  - 2.2 **daca**  $TNOU \neq \text{LINIE\_VIDA}$   
          **atunci** Predictie (U+1, F, TNOU)
3. **altfel** afiseaza atribuirile din F

**sfarsit**

## 3.7 Tehnici retrospective

### ■ Backtracking cu salt



**Algorithm: Backtracking cu salt**

**BacktrackingCuSalt(U, F)** /\* intoarce Nivel – blocare \*/  
/\* NrBlocari, NivelVec, I, test– var locale \*/

1.  $\text{NrBlocari} \leftarrow 0$ ,  $I \leftarrow 0$ ,  $\text{Nivel} \leftarrow U$

2 **pentru** fiecare element V a lui  $X_U$  **executa**

2.1  $F[U] \leftarrow V$

2.2  $\text{test}, \text{NivelVec}[I] \leftarrow \text{Verifica1}(U, F)$

2.3 **daca** test = adevarat **atunci**

2.3.1 **daca**  $U < N$  **atunci**

i.  $\text{Nivel} \leftarrow \text{BacktrackingCuSalt}(U+1, F)$

ii. **daca**  $\text{Nivel} < U$  **atunci** salt la pasul 4

2.3.2 **altfel** afiseaza valorile vectorului F // solutia

2.4 **altfel**  $\text{NrBlocari} \leftarrow \text{NrBlocari} + 1$

2.5  $I \leftarrow I + 1$

3. **daca**  $\text{NrBlocari} = \text{numar valori ale lui } X[U]$  **si**

toate elementele din NivelVec sunt egale

**atunci**  $\text{Nivel} \leftarrow \text{NivelVec}[1]$

4. **intoarce** Nivel

**sfarsit**

## Verifica1 (U, F)

/\* intoarce test si nivelul la care s-a produs blocarea sau 0 \*/

1. test  $\leftarrow$  adevarat

2. I  $\leftarrow$  U-1

3. **cat timp** I>0 **executa**

3.1 test  $\leftarrow$  Relatie(I, F[I], U, F[U])

3.2 **daca** test = fals

**atunci** intrerupe ciclul

3.3 I  $\leftarrow$  I - 1

4. NivelAflat  $\leftarrow$  I

5. **intoarce** test, NivelAflat

**sfarsit**



## 4.8 Euristici

---

- **Ordonarea variabilelor**
- **Ordonarea valorilor**
- **Ordonarea testelor**



## 4.9 CSP partiala

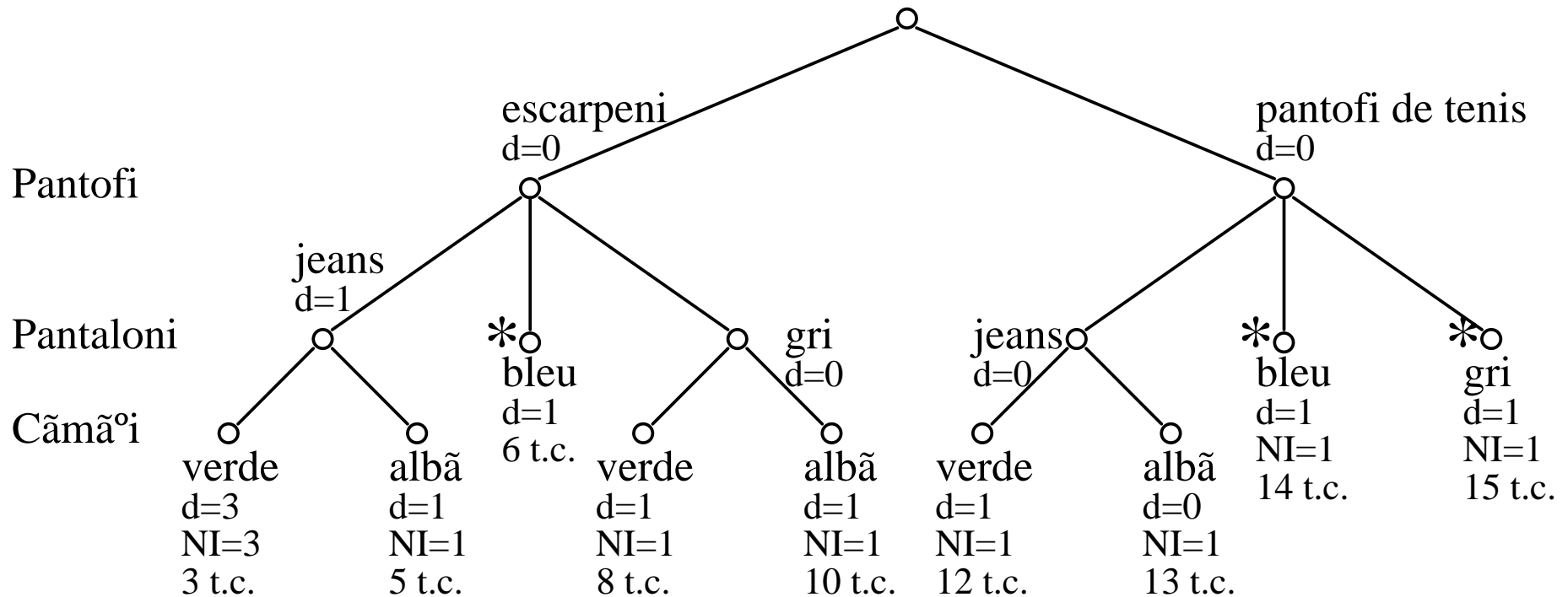
---

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





# CSP partiala





# CSP partiala

---

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

**Algoritm: CSP Partiala**  
**PBKT(Cale, Distanta, Variabile, Valori)**

*/\* intoarce GATA sau CONTINUA \*/*

1. **daca** Variabile = { }  
**atunci**

1.1 CeaMaiBuna  $\leftarrow$  Cale

1.2 NI  $\leftarrow$  Distanta

1.3 **daca** NI  $\leq$  S **atunci** intoarce GATA  
**altfel** intoarce CONTINUA

2. **altfel**

2.1 **daca** Valori = { } **atunci** CONTINUA

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

2.2 **altfel**

2.2.1 **daca** Distanta  $\geq$  NI

**atunci** intoarce CONTINUA

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

### 2.2.2 altfel

- i.  $\text{Var} \leftarrow \text{first}(\text{Variabile})$
- ii.  $\text{Val} \leftarrow \text{first}(\text{Valori})$
- iii.  $\text{DistNoua} \leftarrow \text{Distanta}$
- iv.  $\text{Cale1} \leftarrow \text{Cale}$
- v. **cat timp**  $\text{Cale1} \neq \{\}$  **si**  $\text{DistNoua} < \text{NI}$  **executa**
  - $(\text{VarC}, \text{ValC}) \leftarrow \text{first}(\text{Cale1})$
  - **daca**  $\text{Rel}(\text{Var}, \text{Val}, \text{VarC}, \text{ValC}) = \text{fals}$
  - **atunci**  $\text{DistNoua} \leftarrow \text{DistNoua} + 1$
  - $\text{Cale1} \leftarrow \text{Rest}(\text{Cale1})$
- vi. **daca**  $\text{DistNoua} < \text{NI}$  **si**  
 $\text{PBKT}(\text{Cale} + (\text{Val}, \text{Var}), \text{DistNoua}, \text{Rest}(\text{Variabile}), \text{ValoriNoi})$   
 $= \text{GATA}$

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

**atunci intoarce** GATA

**altfel intoarce**

$\text{PBKT}(\text{Cale}, \text{Distanta}, \text{Variabile}, \text{Rest}(\text{Valori}))$

**sfarsit**