

Proiectarea Algoritmilor

Curs 3 – Programare dinamica
(continuare)
Backtracking și propagarea
restricțiilor



Bibliografie

[http://ktiml.mff.cuni.cz/~bartak/constraints/
intro.html](http://ktiml.mff.cuni.cz/~bartak/constraints/intro.html)



Algoritm generic

- Programare dinamică (crit_optim, problema){
 - // fie problema₀ problema₁ ... problema_n astfel încât
 - // problema_n = problema_i; problema_i mai simplă decât problema_{i+1}
 - 1. Sol = soluții_initiale(crit_optim, problema₀);
 - 2. Pentru i = 1 la n repetă { // construcție soluții pentru problema_i
// folosind soluțiile problemelor precedente
 - 3. Sol_i = calcul_soluții(Sol, Crit_optim, Problema_i);
// determin soluția problemei_i
 - 4. Sol = Sol U Sol_i;
// noua soluție se adaugă pentru a fi refolosită pe viitor
 - 5. s = soluție_pentru_problema_n(Sol);
// selecție / construcție soluție finală
 - 6. Întoarce s;



Proiectarea Algoritmilor 2010

Caracteristici

- O soluție optimă a unei probleme conține soluții optime ale subproblemelor.
- Decompozabilitatea recursivă a problemei P în subprobleme similare $P = P_n, P_{n-1}, \dots, P_0$ care acceptă soluții din ce în ce mai simple.
- Suprapunerea problemelor (soluția unei probleme P_i participă în procesul de construcție a soluțiilor mai multor probleme P_k de talie mai mare $k > i$) – memoizare (se folosește un tablou pentru salvarea soluțiilor subproblemelor pentru a nu le recalcula)
- În general se folosește o abordare bottom-up, de la subprobleme la probleme.



Proiectarea Algoritmilor 2010

Definiție AOC

- **Definiție:** Fie A un arbore binar de căutare cu chei într-o mulțime K, fie $\{x_1, x_2, \dots, x_n\}$ cheile conținute în A, iar $\{y_0, y_1, \dots, y_n\}$ chei reprezentante ale cheilor din K ce nu sunt în A astfel încât: $y_{i-1} < x_i < y_i, i = \overline{1, n}$. Fie $p_i, i = \overline{1, n}$ probabilitatea de a căuta cheia x_i și $q_j, j = \overline{0, n}$ probabilitatea de a căuta o cheie reprezentată de y_j . Vom avea relația: $\sum_{i=1}^n p_i + \sum_{j=0}^n q_j = 1$. Se numește arbore de căutare probabilistică, un arbore cu costul:

$$\text{Cost}(A) = \sum_{i=1}^n (\text{nivel}(x_i, A) + 1) * p_i + \sum_{j=0}^n \text{nivel}(y_j, A) * q_j$$

- **Definiție:** Un arbore de căutare probabilistică având cost minim este un arbore optim la căutare (AOC).



Proiectarea Algoritmilor 2010

Construcția AOC – Notății

- $A_{i,j}$ desemnează un AOC cu cheile $\{x_{i+1}, x_{i+2}, \dots, x_j\}$ în noduri și cu cheile $\{y_i, y_{i+1}, \dots, y_j\}$ în frunzele fictive.
- $C_{i,j} = \text{Cost}(A_{i,j})$. $\text{Cost}(A_{i,j}) = \sum_{k=i+1}^j (\text{nivel}(x_k, A_{i,j}) + 1) * p_k + \sum_{k=i}^j \text{nivel}(y_k, A_{i,j}) * q_k$
- $R_{i,j}$ este indicele α al cheii x_α din radacina arborelui $A_{i,j}$.
- $w_{i,j} = \sum_{k=i+1}^j p_k + \sum_{k=i}^j q_k$
- **Observatie:** $A_{0,n}$ este chiar arborele A, $C_{0,n} = \text{Cost}(A)$ iar $w_{0,n} = 1$.



Proiectarea Algoritmilor 2010

Construcția AOC

- 1. In etapa d , $d = 1, 2, \dots, n$ se calculează costurile și indicele cheilor din rădăcina arborilor AOC $A_{i, i+d}$, $i = 0, n-d$ cu d noduri și $d + 1$ frunze fictive
- Arborele $A_{i, i+d}$ conține în noduri cheile $\{x_{i+1}, x_{i+2}, \dots, x_{i+d}\}$, iar în frunzele fictive sunt cheile $\{y_i, y_{i+1}, \dots, y_{i+d}\}$. Calculul este efectuat pe baza rezultatelor obținute în etapele anterioare
- Conform lemei avem
$$C_{i, i+d} = \min_{i \leq \alpha \leq i+d} \{C_{i, \alpha-1} + C_{\alpha, i+d}\} + w_{i, i+d}$$
- rădăcina $A_{i, i+d}$ are indicele $R_{i, j} = \alpha$ care minimizează $C_{i, i+d}$.
- 2. Pentru $d = n$, $C_{0, n}$ corespunde arborelui AOC $A_{0, n}$ cu cheile $\{x_1, x_2, \dots, x_n\}$ în noduri și cheile $\{y_0, y_1, \dots, y_n\}$ în frunzele fictive



Proiectarea Algoritmilor 2010

Algoritm AOC

```

AOC(x, p, q, n){
  for( i = 0; i ≤ n ; i++)
    {Ci, i = 0, Ri, i = 0, wi, i = qi} // initializare costuri AOC vid Ai, i
  for( d = 1; d ≤ n ; d++){
    for( i = 0; i ≤ n-d ; i++){ // calcul indice radacina si cost pentru Ai, i+d
      j = i + d, Ci, j = ∞, wi, j = wi, j-1 + pj + qj
      for( α = i + 1; α ≤ j; α++){ // ciclul critic – operatii intensive
        if (Ci, α-1 + Cα, j < Ci, j) // am gasit un cost mai mic?
          { Ci, j = Ci, α-1 + Cα, j; Ri, j = α } // update
        Ci, j = Ci, j + wi, j // update
      }
    }
  }
  return gen_AOC(C, R, x, 0, n) // constructie efectiva arbore A0, n
                                // cunoscand indicii
}

```

Complexitate???



Proiectarea Algoritmilor 2010

Exemplu constructie AOC (I)

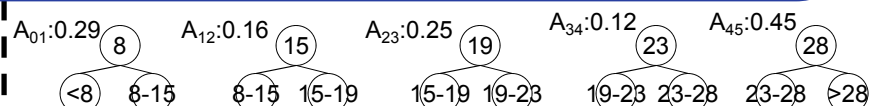
- 8: 0.2; 15: 0.01; 19: 0.1; 23: 0.02; 28: 0.25; (58%)
- [0:8): 0.02; (8:15): 0.07; (15:19): 0.08; (19:23): 0.05; (23:28): 0.05; (28,∞): 0.15 (42%)
- $C_{01}=p_1+q_0+q_1=0.29$
- $C_{12}=p_2+q_1+q_2=0.16$
- $C_{23}=p_3+q_2+q_3=0.25$
- $C_{34}=p_4+q_3+q_4=0.02+0.05+0.05=0.12$
- $C_{45}=p_5+q_4+q_5=0.25+0.05+0.15=0.45$

$$w_{i,j} = \sum_{k=i+1}^j p_k + \sum_{k=i}^j q_k \quad C_{i,i+d} = \min_{i \leq \alpha \leq i+d} \{C_{i,\alpha-1} + C_{\alpha,i+d}\} + w_{i,i+d}$$



Proiectarea Algoritmilor 2010

Exemplu constructie AOC (II)

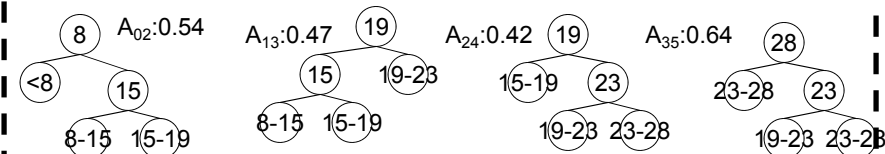


$$C_{02} = \min \{(C_{00}+C_{12}), (C_{01}+C_{22})\} + w_{02} = \min(0.16, 0.29) + 0.38 = 0.54 \quad R_{02} = 1 \quad (\alpha=1)$$

$$C_{13} = \min\{C_{23}, C_{12}\} + w_{13} = \min(0.25, 0.16) + 0.31 = 0.47 \quad R_{13} = 3$$

$$C_{24} = \min\{C_{34}, C_{23}\} + w_{24} = \min(0.12, 0.25) + 0.3 = 0.42 \quad R_{24} = 3$$

$$C_{35} = \min\{C_{45}, C_{34}\} + w_{35} = \min(0.45, 0.12) + 0.52 = 0.64 \quad R_{35} = 5$$



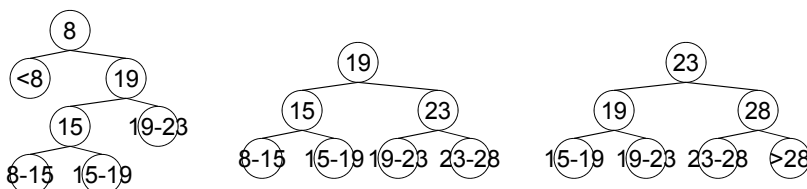
$$w_{i,j} = \sum_{k=i+1}^j p_k + \sum_{k=i}^j q_k \quad C_{i,i+d} = \min_{i \leq \alpha \leq i+d} \{C_{i,\alpha-1} + C_{\alpha,i+d}\} + w_{i,i+d}$$



Proiectarea Algoritmilor 2010

Exemplu constructie AOC (III)

- $C_{03} = \min(C_{00}+C_{13}, C_{01}+C_{23}, C_{02}+C_{33}) + w_{03} = \min(0.47, 0.54, 0.54) + w_{03} \Rightarrow R_{03} = 1$
- $C_{14} = \min(C_{11}+C_{24}, C_{12}+C_{34}, C_{13}+C_{44}) + w_{14} = \min(0.42, 0.28, 0.47) + w_{14} \Rightarrow R_{14} = 3$
- $C_{25} = \min(C_{22}+C_{35}, C_{23}+C_{34}, C_{24}+C_{55}) + w_{25} = \min(0.64, 0.37, 0.42) + w_{25} \Rightarrow R_{25} = 4$



$$w_{i,j} = \sum_{k=i+1}^j p_k + \sum_{k=i}^j q_k \quad C_{i,i+d} = \min_{i \leq \alpha \leq i+d} \{C_{i,\alpha-1} + C_{\alpha,i+d}\} + w_{i,i+d}$$



Proiectarea Algoritmilor 2010

AOC – Corectitudine (I)

- **Teoremă:** Algoritm AOC construiește un arbore AOC A cu cheile $x = \{x_1, x_2, \dots, x_n\}$ conform probabilităților de căutare $p_i, i = 1, n$ și $q_j, j = 0, n$.
- **Demonstrație:** prin inducție după etapa de calcul al costurilor arborilor cu d noduri.
- **Caz de baza:** $d = 0$. Costurile $C_{i,i}$ ale arborilor vizi $A_{i,i}$, $i = 0, n$ sunt 0, așa cum sunt inițializate de algoritm.
- **Pas de inducție:** $d \geq 1$. **Ip. ind.:** pentru orice $d' < d$, algoritmul AOC calculează costurile $C_{i,i+d'}$ și indicii $R_{i,i+d'}$, ai rădăcinilor unor AOC $A_{i,i+d'}$, $i = 0, n-d'$ cu cheile $\{x_{i+1}, x_{i+2}, \dots, x_{i+d'}\}$. Trebuie să arătăm că valorile $C_{i,i+d}$ și $R_{i,i+d}$ corespund unor AOC $A_{i,i+d}$, $i = 0, n-d$ cu cheile $\{x_{i+1}, x_{i+2}, \dots, x_{i+d}\}$.



Proiectarea Algoritmilor 2010

AOC – Corectitudine (II)

- Pentru d și i fixate, algoritmul calculează:

$$C_{i,i+d} = \min_{i \leq \alpha \leq i+d} \{C_{i,\alpha-1} + C_{\alpha,i+d}\} + W_{i,i+d}$$

- unde costurile $C_{i,\alpha-1}$ și $C_{\alpha,i+d}$ corespund unor arbori cu un număr de noduri $d' = \alpha - 1 - i$ în cazul $C_{i,\alpha-1}$ și $d' = 1 + d - \alpha$ în cazul $C_{\alpha,i+d}$.
- $0 \leq d' \leq d - 1 \rightarrow$ aceste valori au fost deja calculate în etapele $d' < d$ și conform ipotezei inductive \rightarrow sunt costuri și indici ai rădăcinilor unor AOC.
- Conform Lemei anterioare, $C_{i,j}$ este costul unui AOC. Conform algoritmului \rightarrow rădăcina acestui arbore are indicele $r = R_{i,j}$, iar cheile sunt $\{x_{i+1}, x_{i+2}, \dots, x_{r-1}\}$ și $\{x_r, x_{r+1}, x_{r+2}, \dots, x_j\} = \{x_{i+1}, x_{i+2}, \dots, x_j\}$
- Pentru $d = n$, costul $C_{0,n}$ corespunde unui AOC $A_{0,n}$ cu cheile x și cu rădăcina de indice $R_{0,n}$.



Proiectarea Algoritmilor 2010

AOC – Concluzii

- Câte subprobleme sunt folosite în soluția optimală într-un anumit pas?
 - AOC: 2 subprobleme
- Câte variante de ales avem de făcut pentru determinarea alegerii optimale într-un anumit pas?
 - AOC: $j - i + 1$ candidați pentru rădăcină
- Informal, complexitatea = $N_s * N_a$ (N_s = număr subprobleme; N_a = număr alegeri)
 - Complexitate AOC: $O(n^2) * O(n) = O(n^3)$



Proiectarea Algoritmilor 2010

Proiectarea Algoritmilor

Curs 3 – Backtracking și
propagarea restricțiilor



Bibliografie

[http://ktiml.mff.cuni.cz/~bartak/constraints/
intro.html](http://ktiml.mff.cuni.cz/~bartak/constraints/intro.html)



Proiectarea Algoritmilor 2010

Problema

	2		8	1		7	4	
7					3	1		
	9				2	8		5
		9		4			8	7
4			2		8			3
1	6			3		2		
3		2	7				6	
		5	6					8
	7	6		5	1		9	



Proiectarea Algoritmilor 2010

SUDOKU

- Joc foarte la modă cu reguli foarte simple.
- Fiecare rând, coloană sau regiune nu trebuie să conțină decât o dată cifrele de la unu la nouă (Wikipedia).
- Prin trecerea în revistă a soluțiilor posibile pentru acest joc vom explora tehnicile de rezolvare backtracking și propagarea restricțiilor.



Proiectarea Algoritmilor 2010

soluția 1 – generează și testează

- Generăm toate soluțiile posibile și le testăm.
- 43 spații de completat, 9 posibilități de completare pentru fiecare căsuță => 9^{43} soluții de testat.

1-9	2	1-9	8	1	1-9	7	4	1-9
7	1-9	1-9	1-9	1-9	3	1	1-9	1-9
1-9	9	1-9	1-9	1-9	2	8	1-9	5
1-9	1-9	9	1-9	4	1-9	1-9	8	7
4	1-9	1-9	2	1-9	8	1-9	1-9	3
1	6	1-9	1-9	3	1-9	2	1-9	1-9
3	1-9	2	7	1-9	1-9	1-9	6	1-9
1-9	1-9	5	6	1-9	1-9	1-9	1-9	8
1-9	7	6	1-9	5	1	1-9	9	1-9



Proiectarea Algoritmilor 2010

soluția 2 – backtracking cronologic (orb) (I)

- **Construiește soluțiile iterativ.**
- **Mentține evidența alegerilor făcute.**
- În momentul în care se ajunge la o **contradicție** se revine **la ultima decizie** luată și se încearcă alegerea unei alte variante.



Proiectarea Algoritmilor 2010

soluția 2 – backtracking cronologic (orb) (II)

1									
2									
3									
4									
5	2		8	1		7	4		
7				3	1				
	9			2	8				5
		9		4			8	7	
4			2		8				3
1	6			3		2			
3		2	7					6	
		5	6						8
7	6		5	1		9			

5	2	3	8	1	6	7	4	9	
7	4	8	5	9	3	1	2	6	
6	9	1	4	7	2	8	3	5	
2	3	9	1	4	5	6	8	7	
4	5	7	2	6	8	9	1	3	
1	6	∅		3		2			
3		2	7				6		
		5	6					8	
7	6		5	1		9			

1 Nu se găsește nici o soluție
2 pe această cale
3 deci trebuie să revenim.
4
5
...

PA Proiectarea Algoritmilor 2010

soluția 2 – backtracking cronologic (orb) (III)

5	2	3	8	1	6	7	4	9	
7	4	8	5	9	3	1	2	6	
6	9	1	4	7	2	8	3	5	
2	3	9	1	4	5	6	8	7	
4	5	7	2	6	8	9	∅	3	
1	6	∅		3		2			
3		2	7				6		
		5	6					8	
7	6		5	1		9			

5	2	3	8	1	6	7	4	9	
7	4	8	5	9	3	1	2	6	
6	9	1	4	7	2	8	3	5	
2	5	9	1	4	5	6	8	7	
4	5	7	2	6	8	9	∅	3	
1	6	∅		3		2			
3		2	7				6		
		5	6					8	
7	6		5	1		9			

1 Se încercă schimbarea
2 ultimei alegeri (în acest
3 caz nu se poate deci
4 revenim iar la alegerea
5 precedentă).
...

1 Găsim o nouă soluție
2 posibilă și reluăm
3 avansul.
4
5

PA Proiectarea Algoritmilor 2010

soluția 2 – backtracking cronologic (orb) (IV)

- **Schema Backtracking**
- Soluție-parțială ← INIT // inițializez
- EȘEC-DEFINITIV ← fals // nu am ajuns la eșec
- **Cât timp** Soluție-parțială nu este soluție finală și **not** EȘEC-DEFINITIV **repetă**
 - Soluție-parțială ← AVANS (Soluție-parțială) // avansez
 - **Dacă** EȘEC (Soluție-parțială) // nu mai pot avansa
 - **atunci** REVENIRE (Soluție-parțială) // mă întorc
- **Dacă** EȘEC-DEFINITIV // nu s-a găsit nicio soluție
 - **atunci** Întoarce EȘEC
- **altfel** Întoarce SUCCES // am ajuns la soluția problemei
- **Sfârșit.**
- **Procedura AVANS** (Soluție-parțială)
 - **Dacă** există alternativă de extindere // pot avansa?
 - **atunci** Soluție-parțială ← Soluție-parțială ∪ alternativă de extindere // avansez
 - **altfel** **Dacă** Soluție-parțială este INIT
 - **atunci** EȘEC-DEFINITIV ← adevărat // nu s-au găsit soluții pentru problemă
 - **altfel** EȘEC (Soluție-parțială) // ramura curentă a dus la eșec



Proiectarea Algoritmilor 2010

backtracking – optimizări posibile (I)

- **Alegerea variabilelor în altă ordine.**
- **Îmbunătățirea revenirilor.**
 - Necesită detectarea **cauzei producerii erorii.**
- **Evitarea redundanțelor în spațiul de căutare (îmbunătățirea avansului).**
 - Evitarea repetării unei căutări care știm că va duce la un rezultat greșit.



Proiectarea Algoritmilor 2010

backtracking – optimizări posibile (II)

Îmbunătățirea revenirilor

Revenire la alegerea variabilei care a cauzat eșecul (8 nu poate fi pus decât la poziția indicată).

					1				
					2				
					3				
					4				
					5				
5	2	3	8	1	6	7	4	9	...
7	4	8	5	9	3	1	2	6	
6	9	1	4	7	2	8	3	5	
2	3	9	1	4	5	6	8	7	
4	5	7	2	6	8	9	1	3	
1	6	∅		3		2			
3		2	7				6		
		5	6					8	
	7	6		5	1		9		



Proiectarea Algoritmilor 2010

backtracking – optimizări posibile (III)

Evitarea redundanțelor în spațiul de căutare

Alegerea lui 8 pe această poziție va produce un eșec în viitor indiferent de celelalte alegeri făcute deci în cazul revenirii în această poziție nu are sens să facem această alegere.

					1				
					2				
					3				
					4				
					5				
5	2	3	8	1	6	7	4	9	...
7	4	8	5	9	3	1	2	6	
6	9	1	4	7	2	8	3	5	
2	3	9	1	4	5	6	8	7	
4	5	7	2	6	8	9	1	3	
1	6	∅		3		2			
3		2	7				6		
		5	6					8	
	7	6		5	1		9		



Proiectarea Algoritmilor 2010

Restricții, rețele de restricții, probleme de prelucrarea restricțiilor

- **Definiție:** O **restricție** c este o relație între una sau mai multe **variabile** v_1, \dots, v_m , (denumite **nodurile** sau **celulele** restricției). Fiecare variabilă v_i poate lua valori într-o anumită mulțime D_i , denumită **domeniul** ei (ce poate fi finit sau nu, numeric sau nu).
- **Definiție:** Se spune că un **tuplu** (o atribuire) de valori (x_1, \dots, x_m) din domeniile corespunzătoare celor m variabile **satisface** restricția $c(v_1, \dots, v_m)$, dacă $(x_1, \dots, x_m) \in c(v_1, \dots, v_m)$.



Proiectarea Algoritmilor 2010

Exprimarea restricțiilor

- **Enumerarea tuplelor restricției.**
 - $(5,2,3,8,1,6,7,4,9)$;
 - $(6,2,3,8,1,5,7,4,9)$; etc.
- **Formule matematice,** cum ar fi ecuațiile sau inecuațiile.
 - $0 < V_{1j} < 10; V_{1j} \neq V_{1k};$
 $\forall j \neq k, 0 < j, k < 10$
- **Precizarea unei mulțimi de reguli.**

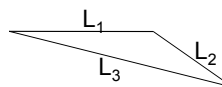
	2		8	1		7	4	
7					3	1		
	9				2	8		5
		9		4			8	7
4			2		8			3
1	6			3		2		
3		2	7				6	
		5	6					8
	7	6		5	1		9	



Proiectarea Algoritmilor 2010

Tipuri de restricții

- **Unare**
 - Specificarea domeniului variabilei.
 - $V_{11} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \setminus \{2, 8, 1, 7, 4\}$
- **Binare**
 - Între 2 variabile.
 - $V_{1j} \neq V_{1k} \forall j \neq k, 0 < j, k < 10$
- **N-are**
 - Între n variabile.
 - Regula triunghiului: $L_1 + L_2 > L_3$.



Proiectarea Algoritmilor 2010

Problemă de satisfacerea restricțiilor

- **Definiție:** O **problemă de satisfacere a restricțiilor (PSR)** este un triplet $\langle V, D, C \rangle$, format din:
 - o mulțime V formată din n variabile $V = \{v_1, \dots, v_n\}$;
 - mulțimea D a domeniilor de valori corespunzătoare acestor variabile: $D = \{D_1, \dots, D_n\}$;
 - o mulțime C de restricții $C = \{c_1, \dots, c_p\}$ între submulțimi ale V ($c_i(v_{i_1}, \dots, v_{i_j}) \subseteq D_{i_1} \times D_{i_2} \times \dots \times D_{i_j}$).
- Conform **Definiției tuplului**, o restricție $c_i(v_{i_1}, \dots, v_{i_j})$, este o **submulțime a produsului cartezian** $D_{i_1} \times D_{i_2} \times \dots \times D_{i_j}$, constând din toate tuplele de valori considerate că **satisfac restricția** pentru $(v_{i_1}, \dots, v_{i_j})$.



Proiectarea Algoritmilor 2010

Problemă de satisfacerea restricțiilor

- **Definiție:** O soluție a unei PSR $\langle V, D, C \rangle$ este un **tuplu de valori** $\langle x_1, \dots, x_n \rangle$ pentru toate variabilele V , din domeniile corespunzătoare D , astfel încât **toate restricțiile din C să fie satisfăcute**.
- **Definiție:** PSR binară este o PSR ce conține **doar restricții unare și binare**.



Proiectarea Algoritmilor 2010

Probleme de satisfacere a restricțiilor - reprezentare

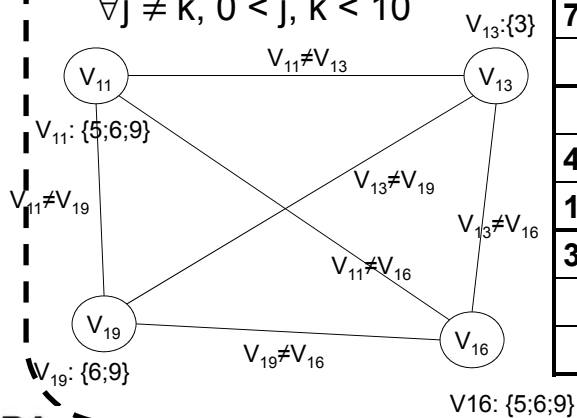
- Reprezentare PSR prin **rețele de restricții**.
- Reprezentarea folosită: **graf**
 - **nodurile:** variabilele restricției
 - **arcele:** restricțiile problemei
 - **valabilă doar pentru PSR binare**
- Altă reprezentare posibilă: **graf**
 - **nodurile:** restricțiile problemei
 - **arcele:** variabilele restricției
 - **Valabilă pentru orice tip de PSR**



Proiectarea Algoritmilor 2010

Exemplu de reprezentare a PSR

- $0 < V_{1j} < 10; V_{1j} \neq V_{1k};$
 $\forall j \neq k, 0 < j, k < 10$



5;6	2	3	8	1	5;6;9	7	4	6;9
7					3	1		
	9				2	8		5
		9		4			8	7
4			2		8			3
1	6			3		2		
3	2	7					6	
		5	6					8
	7	6		5	1		9	



Proiectarea Algoritmilor 2010

Algoritmi de rezolvare a PSR

- **Caracteristici**
 - Rezolvă PSR binare;
 - Variabilele au domenii finite de valori;
 - Prin propagarea restricțiilor se filtrează mulțimile de valori (se elimină elementele din domeniu conform unui criteriu dat);
 - Procesul de propagare se oprește când:
 - O mulțime de valori este vidă → EȘEC;
 - Nu se mai modifică domeniul vreunei mulțimi.



Proiectarea Algoritmilor 2010

Algoritmi de rezolvare a PSR

Notății:

- n = număr variabile = număr restricții unare;
- r = număr restricții binare;
- G = rețeaua de restricții cu variabile drept noduri și restricții drept arce;
- D_i = domeniul variabilei i ;
- Q_i = predicat care verifică restricția unară pe variabila i ;
- P_{ij} = predicatul care reprezintă restricția binară pe variabilele i și j (O muchie între i și j se înlocuiește cu arcele orientate de la i la j și de la j la i);
- $a = \max |D_i|$.



Proiectarea Algoritmilor 2010

NC-1 (Nod Consistency -1)

- Algoritm de consistența nodurilor (pentru restricții unare).
- procedura $NC(i)$ este:
 - pentru fiecare $x \in D_i$
 - repetă
 - dacă not $Q_i(x)$
 - atunci șterge x din D_i
 - Sfârșit.
- Algoritm NC-1 este:
 - pentru $i \leftarrow 1$ până la n execută $NC(i)$
 - Sfârșit.



Proiectarea Algoritmilor 2010

NC-1: Exemplu (I)

- Elimină din domeniul de valori al fiecărui nod valorile care nu satisfac restricțiile care au ca argument variabila din nodul respectiv.
- În cazul Sudoku variabilele inițial iau valori între 1-9.
- Algoritmul NC-1 elimină pentru fiecare variabilă acele valori din domeniu care nu sunt consistente cu valorile fixe (celulele deja fixate).



Proiectarea Algoritmilor 2010

NC-1: Exemplu (II)

5;6	2	3;	8	1	5;6; 9	7	4	6;9;
7	4;5; 8;	4;8;			3	1		
6;	9	1;3; 4;			2	8		5
		9		4			8	7
4			2		8			3
1	6			3		2		
3		2	7				6	
		5	6					8
	7	6		5	1		9	



Proiectarea Algoritmilor 2010

Algoritmi de consistență a arcelor

- Algoritmii de consistență a arcelor înlătură toate inconsistențele submulțimilor de 2 elemente ale rețelei de restricții.
- Funcția REVERSE ((i,j)) este:
 - ȘTERS ← fals
 - pentru fiecare $x \in D_i$ execută
 - dacă nu există $y \in D_j$ a.i. $P_{ij}(x,y)$
 - atunci
 - șterge x din D_i ;
 - ȘTERS ← adevărat;
 - întoarce ȘTERS
 - Sfârșit.



Proiectarea Algoritmilor 2010

Exemplu funcționare REVERSE

	5;6	2	3;	8	1	5;6; 9	7	4	6;9;
x	7	4;5; 8;	4;8;			3	1		
y	6;	9	1;3;4;			2	8		5
	5;6;2;		9		4			8	7
	4			2		8			3
	1	6			3		2		
	3		2	7				6	
	9;		5	6					8
	8;	7	6		5	1		9	

Complexitate Reverse ?



Proiectarea Algoritmilor 2010

REVISE - Concluzie

- Funcția **Revise** este apelată pentru un arc al grafului de restricții (binare) și **șterge acele valori** din **domeniul** de definiție al **unei variabile** pentru care **nu este satisfăcută restricția** pentru **nici o valoare** corespunzătoare celeilalte variabile a restricției.
- **Complexitate Revise: $O(a^2)$**



Proiectarea Algoritmilor 2010

AC-1 (Arc Consistency -1)

- **Algoritm AC-1** este:
 - NC-1;
 - $Q \leftarrow \{(i,j) \mid (i,j) \in \text{arce}(G), i \neq j\}$
 - *repetă*
 - *execută*
 - SCHIMBAT \leftarrow fals
 - *pentru fiecare* $(i,j) \in Q$
 - *execută* SCHIMBAT \leftarrow (REVISE $((i,j))$ sau SCHIMBAT)
 - sfârșit
 - *până non* SCHIMBAT
 - Sfârșit.



Proiectarea Algoritmilor 2010

AC-1 Caracteristici & Complexitate

- Se aplică algoritmul de consistența nodurilor și apoi se aplică REVISE până nu se mai realizează nici o schimbare.
- Complexitate: $O(na * 2r * a^2)$

La fiecare iterație eliminăm o singură valoare (și avem maxim na valori posibile).

Numărul maxim de apelări al Revise.

Complexitate revise.



Proiectarea Algoritmilor 2010

Exemplu AC-1

5,6	2	3;	8	1	5;6; 9	7	4	6;9;
7	4;5; 8;	4;8;			3	1		
6;	9	1,3 ;4;			2	8		5
5,6 ;2;	3,5	9		4			8	7
4	5,7	7	2		8			3
1	6	8		3		2		
3		2	7				6	
9;		5	6					8
8;	7	6		5	1		9	



Proiectarea Algoritmilor 2010

AC-3 (Arc Consistency -3)

- Algoritm AC-3 este:
 - NC-1;
 - $Q \leftarrow \{(i,j) \mid (i,j) \in \text{arce}(G), i \neq j\}$
 - *cât timp* Q nevid *execută*
 - Selectează și șterge un arc (k,m) din Q;
 - *dacă* REVISE ((k,m))
 - *atunci* $Q \leftarrow Q \cup \{(i,k) \mid (i,k) \in \text{arce}(G), i \neq k, i \neq m\}$
 - sfârșit
 - Sfârșit.



Proiectarea Algoritmilor 2010

AC-3 Caracteristici

- Se elimină pe rând arcele (constrângerile).
- Dacă o **constrângere aduce modificări în rețea** adăugăm pentru **reverificare nodurile care puntează către nodul de plecare al restricției** verificate.
 - **Scopul:** Reverificarea nodurilor direct implicate de o constrângere din rețea.
- **Avantaj:** Se fac **mult mai puține apeluri ale funcției REVISE.**
- **Complexitate:** $O(a^3r)$.



Proiectarea Algoritmilor 2010

Backtracking + propagarea restricțiilor

- În general, propagarea restricțiilor **nu poate rezolva complet** problema dată.
- Metoda ajută la **limitarea spațiului de căutare** (foarte importantă în condițiile în care backtracking-ul are complexitate exponențială!).
- În cazul în care propagarea restricțiilor nu rezolvă problema se folosește:
 - Backtracking pentru **a genera soluții parțiale**;
 - Propagarea restricțiilor după fiecare pas de backtracking pentru a **limita spațiul de căutare** (și eventual găsi că soluția nu este validă)



Proiectarea Algoritmilor 2010

Întrebări?



Proiectarea Algoritmilor 2010

48