

Proiectarea Algoritmilor

Curs 3 – Backtracking și propagarea restricțiilor

Problema SUDOKU

fiecare rând,
coloană sau
pătrat de
3x3 nu
trebuie să
conțină
decât o dată
cifrele de la
unu la nouă

	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	

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

soluția 2 – backtracking cronologic (orb) (I)

- construiește soluțiile iterativ
- menține 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

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	

soluția 2 – backtracking cronologic (orb)

(III)

					1			
					2			
					3			
		1			4			
		2			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	

nu se găsește nici o soluție
pe această cale
deci trebuie să revenim

soluția 2 – backtracking cronologic (orb)

(IV)

					1			
		1			2			
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	

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

					1			
		1			2			
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
	3							
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~~ găsim o nouă soluție
~~2~~ posibilă și reluăm
~~3~~ avansul

soluția 2 – backtracking cronologic (orb)

(IV)

- **Schema Backtracking**
- Solutie-parțială \leftarrow INIT
- ESEC-DEFINITIV \leftarrow fals
- **Cât timp** Solutie-parțială nu este soluție finală **si not** ESEC-DEFINITIV **repetă**
 - Solutie-parțială \leftarrow AVANS (Solutie-parțială)
 - **Dacă** ESEC(Solutie-parțială)
 - **atunci** REVENIRE(Solutie-parțială)
 - **Dacă** ESEC-DEFINITIV
 - **atunci** Întoarce ESEC
 - **altfel** Întoarce SUCCES
- **Sfârșit.**

- **Procedura** AVANS(Solutie-parțială)
 - **Dacă** există alternativă de extindere
 - **atunci** Solutie-parțială \leftarrow Solutie-parțială \cup alternativă de extindere
 - **altfel Dacă** Solutie-parțială este INIT
 - **atunci** ESEC-DEFINITIV \leftarrow adevărat
 - **altfel** ESEC(Solutie-parțială)

backtracking – optimizări posibile (I)

- alegerea variabilelor în altă ordine
- îmbunătățirea revenirilor
 - necesită detectarea cauzei producerii erorii
- evitarea redundanțelor în spațiul de căutare
 - evitarea repetării unei căutări care știm că va duce la un rezultat greșit

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	...
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	

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	...
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	

Restricții, rețele de restricții, probleme de prelucrarea restricțiilor

Def. 1: O **restricție** c este o relație între 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).

Restricții, rețele de restricții, probleme de prelucrarea restricțiilor

Def. 2 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 $(x_1, \dots, x_m) \in c(v_1, \dots, v_m)$.

Conform Def2.,

o restricție $c_i(v_{i1}, \dots, v_{ij})$, este o submulțime a produsului cartezian $D_{i1} \times D_{i2} \times \dots \times D_{ij}$, constând din toate tuplele de valori considerate că satisfac restricția pentru (v_{i1}, \dots, v_{ij}) .

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
 - $0 < V_{ij} < 10$; $V_{ij} \neq V_{ik}$; $V_{ji} \neq V_{ki}$;
 - ...
 - $\forall j \neq k, 0 < i, j, k < 10$
- 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	

Tipuri de restricții

- unare
 - specificarea domeniului variabilei
 - $\forall i \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- binare
 - între 2 variabile
 - $\forall i \neq j \forall k \neq l, 0 < i, j, k, l < 10$

Problemă de satisfacerea restricțiilor

Def 3: 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 ($c_i(v_{i_1}, \dots, v_{i_j}) \in D_{i_1} \times D_{i_2} \times \dots \times D_{i_j}$).

Problemă de satisfacerea restricțiilor

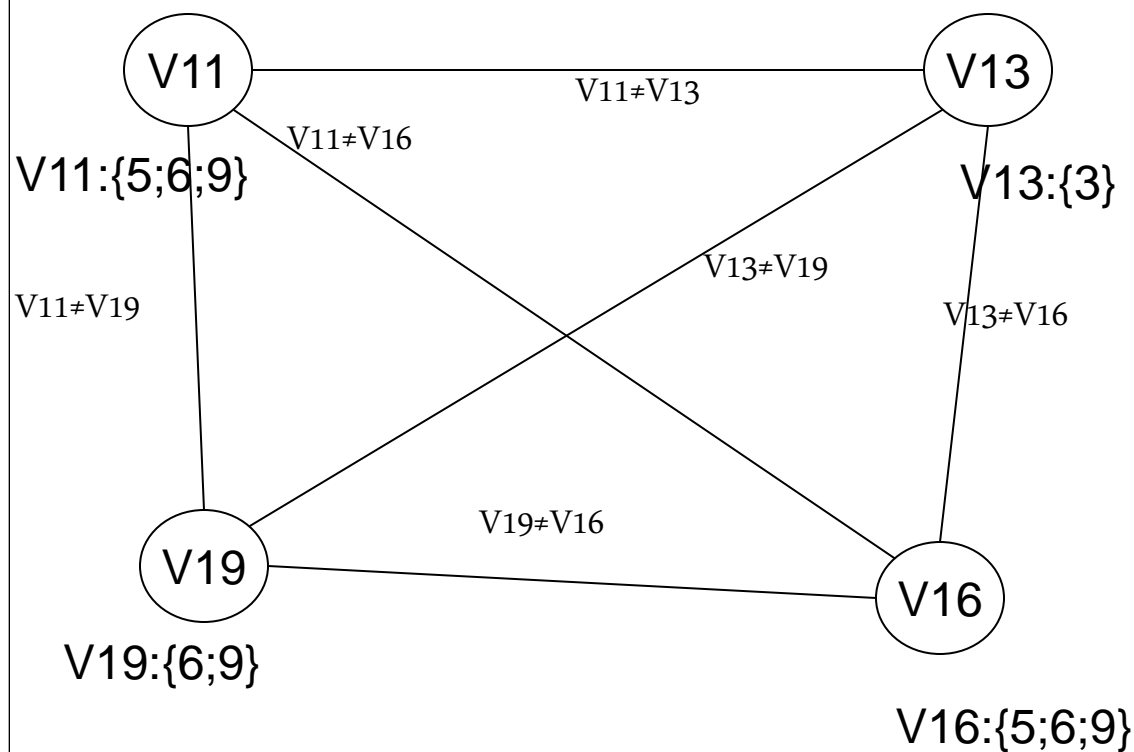
- **Def 4:** O soluție a unei PSR $\langle V, D, C \rangle$ este un tuplu de valori $\langle x_1, \dots, x_n \rangle$ pentru toate variabilele V , din domeniile corespunzătoare D , astfel încât toate restricțiile din C să fie satisfăcute.
- **Def 5:** PSR binară este o PSR ce conține doar restricții unare și binare

Probleme de satisfacere a restricțiilor

- reprezentare PSR prin **rețele de restricții**
- reprezentarea folosită: graf
 - nodurile grafului: variabilele restricției
 - arcele: restricțiile problemei
 - valabilă doar pentru PSR binare

Exemplu de reprezentare a PSR

- $0 < V1j < 10; V1j \neq V1k$
 $\forall j \neq k, 0 < j, k < 10$



5;6; 9;	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	

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 => ESEC
 - nu se mai modifică domeniul vreunei mulțimi

Algoritmi de rezolvare a PSR

- Notații

- 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)
- $a = \max |D_i|$

NC-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 sterge x din D_i
 - Sfârșit.
- **Algoritm NC-1** este:
 - pentru $i \leftarrow 1$ până la n executa NC(i)
 - Sfârșit.

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)

NC-1: Exemplu (II)

5;6; 9;	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	

Algoritmi de consistență a arcelor

- Algoritmii de consistență a arcelor înlătură toate inconsistențele submulțimilor de 2 elemente ale rețelei de restricții
- **funcția** REVISE ((i,j)) **este:**
 - STERS \leftarrow fals
 - **pentru fiecare** $x \in D_i$ **execută**
 - dacă nu exista $y \in D_j$ a.i. $P_{ij}(x,y)$
 - **atunci**
 - sterge x din D_i ;
 - STERS \leftarrow adevărat
 - **sfârsit**;
 - **întoarce** STERS
 - **Sfârsit.**

Exemplu funcționare REVISE

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

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: $O(a^2)$

AC-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.**

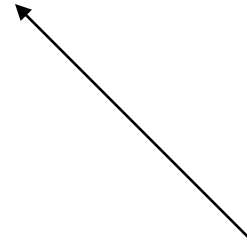
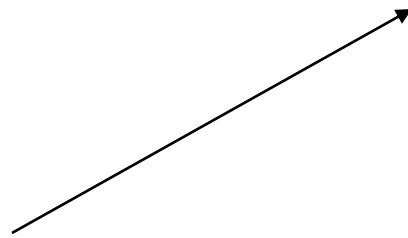
AC-1 Caracteristici & Complexitate

- se aplică algoritmul de consistența nodurilor și apoi se aplică REVISE până nu se mai realizează nici o schimbare
- complexitate: $O(na \cdot 2r \cdot a^2)$

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

numărul de apelări al
revise

complexitate revise



Exemplu AC-1

5,6,9	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	

AC-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.**

AC-3 Caracteristici

- se elimină pe rând arcele (constrângerile)
- dacă o constrângere aduce modificări în rețea adăugăm pentru reverificare nodurile care punctează 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: $O(a^3r)$

Backtracking + propagarea restricțiilor

- Propagarea restricțiilor nu poate rezolva în general 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)