Proiectarea Algoritmilor

Backtracking şi propagarea restricțiilor

Problema SUDOKU

fiecare rând, coloană sau pătrat de 3x3 nu trebuie să conţină 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 şi testează

- generăm toate soluțiile posibile şi 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)

- construieşte soluțiile iterativ
- menține evidența alegerilor făcute
- în momentul în care se ajunge la o contradicție se revine la ultima decizie luată şi 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)

5	2	~~1~~ ~~2~~ 3	8	1	~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 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 găseşte nici o soluţie
pe această cale
deci trebuie să revenim

soluția 2 – backtracking cronologic (orb) (IV)

5	2	~~1~~ ~~2~~ 3	8	1	~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 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	

se încearcă schimbarea ultimei alegeri (în acest caz nu se poate deci revenim iar la alegerea precedentă)

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

găsim o nouă soluţie posibilă şi reluăm avansul

soluția 2 – backtracking cronologic (orb) (IV)

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

- **Procedura** AVANS(Solutie-parțială)
  - **Dac**ă există alternativă de extindere
  - **atunci** Solutie-parțială <- Solutie-parțială ∪ alternativă de extindere
  - **altfel Dac**ă Solutie-parțială este INIT
    - **atunci** ESEC-DEFINITIV <- 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 ştim că va duce la un rezultat greşit

backtracking – optimizări posibile (II)

- îmbunătățirea revenirilor

revenire la alegerea variabilei care a cauzat eşecul (8 nu poate fi pus decât la poziţia indicată)

5	2	~~1~~ ~~2~~ 3	8	1	~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 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	

backtracking – optimizări posibile (III)

- evitarea redundanțelor în spațiul de căutare

alegerea lui 8 pe această poziţie va produce un eşec în viitor indiferent de celelalte alegeri făcute deci în cazul revenirii în această poziţie nu are sens să facem această alegere

5	2	~~1~~ ~~2~~ 3	8	1	~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 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	

Restricții, rețele de restricții, probleme de prelucrarea restricțiilor

Def. 1:O **restricție** c este o relație între mai multe **variabile** v1,…,vm, (denumite nodurile sau celulele restricției).

Fiecare variabila vi poate lua valori într-o anumita mulțime Di, 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 (x1, ... xm) din domeniile corespunzătoare celor m variabile

satisface restricția c(v1, . . ., vm),

dacă (x1, ... xm) $\in$ c(v1, . . ., vm).

Conform Def2.,

o restricție ci(vi1, . . ., vij), este o submulțime a produsului cartezian Di1 x Di2 x … x Dij, constând din toate tuplele de valori considerate că satisfac restricția pentru (vi1, . . ., vij).

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<Vij<10; Vij!=Vik; Vji!=Vki; ...
    ∀j!=k, 0<l,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
  - $V11 \in \{1,2,3,4,5,6,7,8,9\}$
- binare
  - între 2 variabile
  - $V1j != V1k \ \forall j != k,\ 0 < j,k < 10$
- n-are
  - între n variabile

Problemă de satisfacerea restricțiilor

Def 3: O **problem**ă **de satisfacere a restricțiilor** (PSR) este un triplet <V,D,C>, format din

- o mulțime V formată din n variabile V={v1, ...,vn}
- mulțimea D a domeniilor de valori corespunzătoare acestor variabile: D={D1, ... Dn}
- o mulțime C de restricții C={c1, ..., cp} între submulțimi ale V (ci(vi1, . . ., vij)Í Di1 x Di2 x ... x Dij).

Problemă de satisfacerea restricțiilor

- **Def 4:** O **soluție a unei PSR** <V,D,C> este un tuplu de valori <x1, … , xn> pentru toate variabilele V, din domeniile corespunzătoare D, astfel încât toate restricțiile din C să fie satisfăcute.

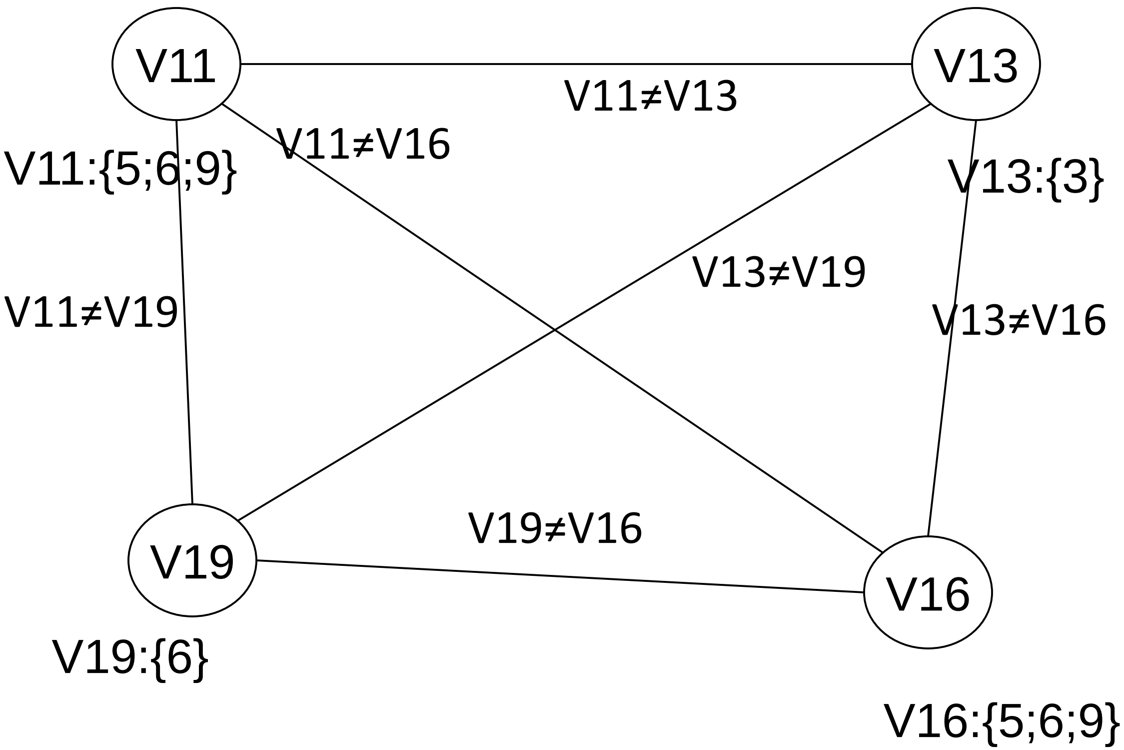
- **Def 5: PSR binară** este o PSR ce conține doar restricții unare şi binare

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!=V1k
  ∀j!=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 opreşte 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 si restricții drept arce
  - Di = domeniul variabilei i
  - Qi = predicat care verifică restricția unară pe variabila i
  - Pij = predicatul care reprezintă restricția binara pe variabilele i si j (O muchie între i si j se înlocuieste cu arcele orientate de la i la j si de la j la i)
  - a = max |Di|

NC-1

- algoritm de consistența nodurilor (pentru restricții unare)
- **procedura NC(i)**este:
  - pentru fiecare x $\in$ Di
  - repetă
    - dacă not Qi (x)
      - atunci sterge x din Di
  - Sfârsit.
- **Algoritm NC-1** este:
  - pentru i <- 1 până la n executa NC(i)
  - Sfârsit.

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 <-fals
  - **pentru fiecare** $x \in Di$ **execută**
    - **dacă** nu exista $y \in Dj$ a.i. Pij(x,y)
    - **atunci**
      - sterge x din Di;
      - STERS <- 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;	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	

REVISE - Concluzie

- Funcția REVISE este apelată pentru un arc al grafului de restricții (binare) si sterge 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 <- {(i,j) | (i,j)∈arce(G), i≠j}
  - **repeta**
    - **executa**
      - SCHIMBAT <- fals
      - **pentru fiecare** (i,j)∈Q
        - **executa** SCHIMBAT <- (REVISE ((i,j)) sau SCHIMBAT)
      - **sfârsit**
    - **până non** SCHIMBAT
  - **Sfârsit.**

AC-1 Caracteristici & Complexitate

- se aplică algoritmul de consistența nodurilor şi apoi se aplică REVISE până nu se mai realizează nici o schimbare
- complexitate: $O(na*2r*a^2)=O(n*r*a^3)$

la fiecare iteraţie eliminăm o singură valoare (şi 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 <- $\{(i,j) \mid (i,j) \in arce(G), i \neq j\}$
  - **cât timp** Q nevid **executǎ**
    - Selectează si sterge un arc (k,m) din Q;
    - **dac**ă REVISE ((k,m))
      - **atunci** Q <- Q $\cup$ $\{(i,k) \mid (i,k) \in arce(G), i \neq k, i \neq m\}$
    - **sfâr**sit
  - **Sfâr**sit**.**

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: $O(r*a^3)$

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 foloseşte
  - backtracking pentru a genera soluții parțiale
  - propagarea restricțiilor după fiecare pas de backtracking pentru a limita spațiul de căutare (şi eventual găsi că soluția nu este validă)