

# Proiectarea Algoritmilor

Curs 4 – Backtracking și  
propagarea restricțiilor

# Bibliografie

- <http://ktiml.mff.cuni.cz/~bartak/constraints/intro.html>
- <https://algorithm-visualizer.org/>
- <https://gist.github.com/johnthebrave/229f28a66efc71bd9a72bb79e7b66a58>
- <https://gist.github.com/johnthebrave/266ac941d5cba287235ced3ec9d89a7e>
- <https://gist.github.com/johnthebrave/dfc43837aa466dbf70912d5a833f460c>

# 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	

# 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**.

# Soluția 1 – generează și testează

- Generăm toate soluțiile posibile și le testăm.
- 45 spații de completat, 9 posibilități de completare pentru fiecare căsuță =>  $9^{45}$  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)

<del>1</del>								
<del>2</del>								
<del>3</del>								
<del>4</del>								
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	

					<del>1</del>			
		<del>2</del>						
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	Ø		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)

## (III)

		<del>1</del> <del>2</del>			<del>1</del> <del>2</del> <del>3</del> <del>4</del> <del>5</del>			...
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	

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

		<del>1</del> <del>2</del> <del>3</del> <del>4</del> <del>5</del>			<del>1</del> <del>2</del> <del>3</del> <del>4</del> <del>5</del>			...
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	

Găsim o nouă soluție  
posibilă și reluăm  
avansul.



# Soluția 2 – Backtracking cronologic (orb)

## (IV)

### Schema Backtracking

- Soluție-parțială  $\leftarrow$  INIT // inițializez
- EȘEC-DEFINITIV  $\leftarrow$  fals // nu am ajuns (încă) la eșec
- **Cât timp** Soluție-parțială nu este soluție finală **și nu avem** EȘEC-DEFINITIV
  - Soluție-parțială  $\leftarrow$  **AVANS (Soluție-parțială)** // avansez
  - **Dacă** EȘEC (Soluție-parțială) // nu mai pot avansa
    - Atunci REVENIRE (Soluție-parțială) // mă întorc
- **Dacă** EȘEC-DEFINITIV
  - Atunci Întoarce EȘEC // nu s-a găsit nicio soluție
  - Altfel Întoarce SUCCES // am ajuns la soluția problemei
- **Sfârșit.**

### Procedura **AVANS (Soluție-parțială)**

- **Dacă** există alternativă de extindere // pot avansa?
  - Atunci Soluție-parțială  $\leftarrow$  Soluție-parțială  $\cup$  alternativă de extindere // avansez
  - Altfel **Dacă** Soluție-parțială este INIT
    - Atunci EȘEC-DEFINITIV  $\leftarrow$  adevărat // nu s-au găsit soluții pentru problemă
    - Altfel EȘEC (Soluție-parțială) // ramura curentă a dus la eșec

# 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.

# Backtracking – optimizări posibile (II)

## Îmbunătățirea revenirilor

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

		<del>1</del> <del>2</del> <del>3</del> <del>4</del> <del>5</del>			<del>1</del> <del>2</del> <del>3</del> <del>4</del> <del>5</del>			
5	2	3	8	1	6	7	4	...
7	4	8	<del>5</del>	<del>9</del>	3	1	<del>2</del>	<del>6</del>
<del>6</del>	9	<del>1</del>	<del>4</del>	<del>7</del>	2	8	<del>3</del>	5
<del>2</del>	<del>3</del>	9	<del>1</del>	4	<del>5</del>	<del>6</del>	8	7
4	<del>5</del>	<del>7</del>	2	<del>6</del>	8	<del>9</del>	<del>1</del>	3
1	6	<del>8</del>		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 mai facem această alegere.

		<del>1</del> <del>2</del> <del>3</del> <del>4</del> <del>5</del>			<del>1</del> <del>2</del> <del>3</del> <del>4</del> <del>5</del>			
5	2	3	8	1	6	7	4	...
7	4	<del>8</del>	<del>5</del>	<del>9</del>	3	1	<del>2</del>	<del>6</del>
<del>6</del>	9	<del>1</del>	<del>4</del>	<del>7</del>	2	8	<del>3</del>	5
<del>2</del>	<del>3</del>	9	<del>1</del>	4	<del>5</del>	<del>6</del>	8	7
4	<del>5</del>	<del>7</del>	2	<del>6</del>	8	<del>9</del>	<del>1</del>	3
1	6	Ø		3		2		
3		2	7				6	
		5	6					8
	7	6		5	1		9	

# 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 **tuflu** (o atribuire) de valori  $(x_1, \dots, x_m)$  din domeniile corespunzătoare celor  $m$  variabile **satisfacă** restricția  $c(v_1, \dots, v_m)$ , dacă  $(x_1, \dots, x_m) \in c(v_1, \dots, v_m)$ .

# Exprimarea restricțiilor

- Enumerarea tuplurilor 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	

# Tipuri de restricții

## Unare

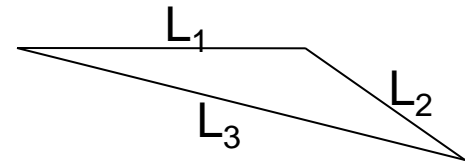
- Specificarea domeniului variabilei.
- $V_{11} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \setminus \{2, 8, 1, 7, 4, 3, 9\}$

## Binare

- Între 2 variabile.
- $V_{1j} \neq V_{1k} \quad \forall j \neq k, 0 < j, k < 10$

## N-are

- Între n variabile.
- Regula triunghiului:  $L_1 + L_2 > L_3$ .



# Probleme 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  $C = \{c_1, \dots, c_p\}$  peste submulțimi ale  $V$  ( $c_i(v_{i1}, \dots, v_{ij}) \subseteq D_{i1} \times D_{i2} \times \dots \times D_{ij}$ ).
- Conform **Definiției tuplului**, 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 tuplurile de valori considerate că **satisfac restricția** pentru  $(v_{i1}, \dots, v_{ij})$ .



# Soluții ale PSR

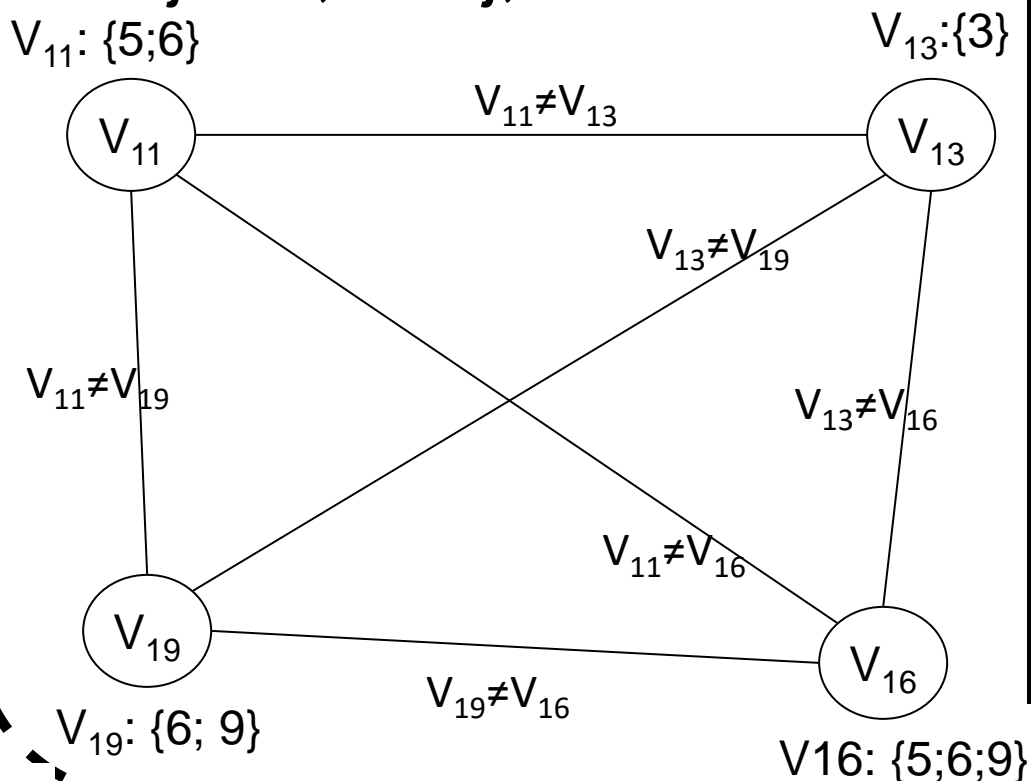
- **Definiție:** O soluție a unei PSR  $\langle V, D, C \rangle$  este un **tuflu de valori**  $\langle x_1, \dots, x_n \rangle$  care conține toate variabilele  $V$ , din domeniile corespunzătoare  $D$ , astfel încât **toate restricțiile din  $C$  să fie satisfăcute**.
- **Definiție:** PSR binară este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

# 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**

# 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	

# Algoritmi de rezolvare a PSR – Propagarea restricțiilor

## ● 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.

# Algoritmi de rezolvare a PSR – Propagarea restricțiilor

## Notatii:

- $n$  = 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 (Node Consistency -1)

- Algoritm de consistența nodurilor (pentru restricții unare).
- Procedura NC(i) este:
  - Pentru fiecare  $x \in D_i$ 
    - Dacă not  $Q_i(x)$  // nu este satisfăcută restricția unară
      - Atunci șterge  $x$  din  $D_i$
  - Sfârșit.
- Algoritm NC-1 este:
  - Pentru  $i$  de la 1 la  $n$  Execută NC(i) // pentru fiecare var
  - Sfârșit.

Complexitate NC-1? na

# 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	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:
  - $\text{ȘTERS} \leftarrow \text{fals}$  // nu am modificat domeniul de valori
  - Pentru fiecare  $x \in D_i$ 
    - Dacă nu există  $y \in D_j$  a.î.  $P_{ij}(x,y)$  // nu se respectă restricția
      - Atunci
        - Șterge  $x$  din  $D_i$ ;
        - $\text{ȘTERS} \leftarrow \text{adevărat}$ ; // am făcut modificări
  - Întoarce **ȘTERS**.

# Exemplu funcționare REVISE

<del>5;6</del>	2	<del>3</del>	8	1	5;6; 9	7	4	6;9;
7	4;5; 8;	4;8;			3	1		
6;	9	1; <del>3</del> ,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 Revise ?  $a^2$

# 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 (Arc Consistency -1)

● Algoritm AC-1 este:

- NC-1; // reduc domeniul de valori
- $Q \leftarrow \{(i,j) \mid (i,j) \in \text{arce}(G), i \neq j\}$  // adaug restricțiile
- Repetă
  - SCHIMBAT  $\leftarrow$  fals // nu am modificat niciun  
// domeniu
  - Pentru fiecare  $(i,j) \in Q$  // pentru fiecare restricție
    - SCHIMBAT  $\leftarrow$  (REVISE  $((i,j))$  sau SCHIMBAT)
  - Până când **non** SCHIMBAT // nu am mai făcut  
// modificări
- 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:  $O(na * 2r * a^2)$

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

Numărul de apelări al Revise într-o iterație (din Repetă).

Complexitate Revise.

# Exemplu AC-1

<del>5;6</del>	2	3;	8	1	5;6; 9	7	4	6;9;
7	4;5; 8;	4;8;			3	1		
6;	9	<del>1;3</del> ;4;			2	8		5
<del>5;6;2;</del>	<del>3;5</del>	9		4			8	7
4	5	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 (Arc Consistency -3)

- **Algoritm AC-3** este:
  - NC-1; // **reduc domeniul de valori**
  - $Q \leftarrow \{(i,j) \mid (i,j) \in \text{arce}(G), i \neq j\}$  // **adaug**  
// **restricțiile**
  - **Cât timp**  $Q$  nevid
    - Selectează și șterge un arc  $(k,m)$  din  $Q$ ;
    - **Dacă** REVISE  $((k,m))$  // **am modificat domeniul**
      - **Atunci**  $Q \leftarrow Q \cup \{(i,k) \mid (i,k) \in \text{arce}(G), i \neq k, i \neq m\}$  // **verific dacă nu se modifică și alte domenii**

# 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

- 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 a găsi că soluția nu este validă).

# ÎNTREBĂRI?

# Bibliografie Curs 5

- Giumale – Introducere in Analiza Algoritmilor cap 7.6
- <http://www.dwheeler.com/chess-openings/#Sicilian%20Defense>
- [http://mouserunner.com/MozillaTicTacToe/Mozilla\\_Tic\\_Tac\\_Toe.htm](http://mouserunner.com/MozillaTicTacToe/Mozilla_Tic_Tac_Toe.htm)
- [http://www.emunix.emich.edu/~evett/AI/AlphaBeta\\_movie/index\\_movie.htm](http://www.emunix.emich.edu/~evett/AI/AlphaBeta_movie/index_movie.htm)