LISN
LABORATOIRE INTERDISCIPLINAIRE DES SCIENCES DU NUMÉRIQUE

université PARIS-SACLAY

cnrs

Artificial Intelligence

Cours6 - Constraint Programming

L3 - Informatique

Nadjib Lazaar
Ing - Phd - HDR - Professor - Paris-Saclay University - LISN - LaHDAK
lazaar@lisn.fr https://perso.lisn.upsaclay.fr/lazaar/

14/02/2025

Constraint Programming

« The user states the problem,
the computer solve it! »

Constraint Programming

Programmation Par Contraintes

Objectifs du cours

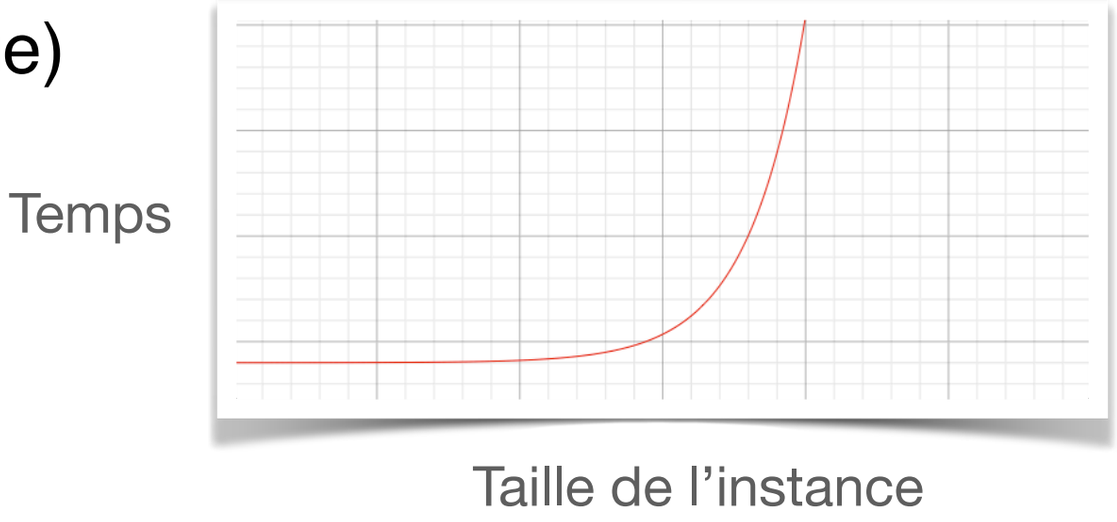
- Comprendre les fondements de la programmation par contraintes
- Explorer les techniques de modélisation
- Introduire les méthodes de résolution

Motivation et Applications

Pourquoi la programmation par contraintes ?

- Formalisme expressif pour de nombreux problèmes
- Méthodes systématiques de recherche de solutions

- **Applications :**
  - Planification (emplois du temps, logistique)
  - Intelligence artificielle (jeu, robotique)
  - Optimisation (affectation, routage)



La PPC

Historique de la Programmation par Contraintes

- **1960-1970 :** Origines en intelligence artificielle, programmation logique et recherche opérationnelle
- **Années 1980 :** Développement des premiers langages de contraintes (CLP, Prolog)
- **Années 1990-2000 :** Apparition des systèmes génériques (Choco, Gecode, MiniZinc)
- **Aujourd'hui :** Large utilisation industrielle et avancées en hybridation avec d'autres méthodes

Programmation Déclarative

Comparaison des Paradigmes de Programmation

Paradigme	Description	Exemples
Impératif	Décrit *comment* résoudre un problème via des instructions explicites	C, Java, Python
Orienté Objet	Structure le code en objets interagissant entre eux	Java, C++, Python
Fonctionnel	Basé sur des fonctions et l'absence d'état mutable	Haskell, Lisp, Scala
Déclaratif	Décrit *quoi* résoudre sans détailler *comment*	Prolog, SQL, MiniZinc

Programmation Déclarative

Pourquoi la programmation déclarative ?

- Facilite la modélisation de problèmes complexes
- Moins de dépendance aux algorithmes sous-jacents
- Plus proche des spécifications naturelles

PPC

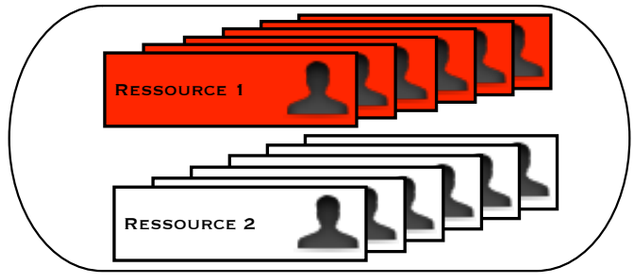
Solveurs et plateformes

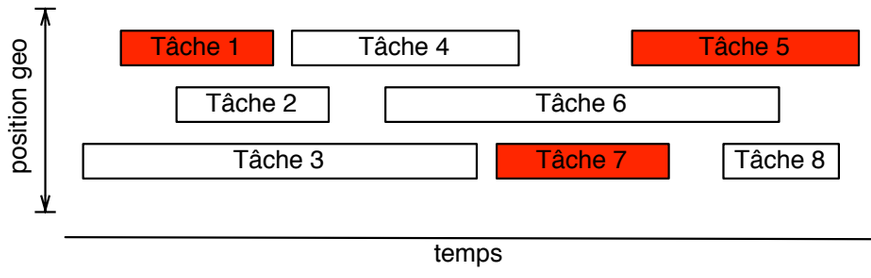
IBM ILOG CP Optimizer (Java, C++, Python, .NET)	IBM
Google OR-Tools (C++, Java, C#, Python)	Google OR-Tools
Artelys Kalis (Java, C++, Python)	ARTELYS KALIS™ Constraint Programming Library
SICStus Prolog (CLPFD bib in prolog)	SICStus 4
Gecode (C++)	G
Choco (Java)	CHOCO
Minizinc (high-level, solver-independent)	Mini Zn

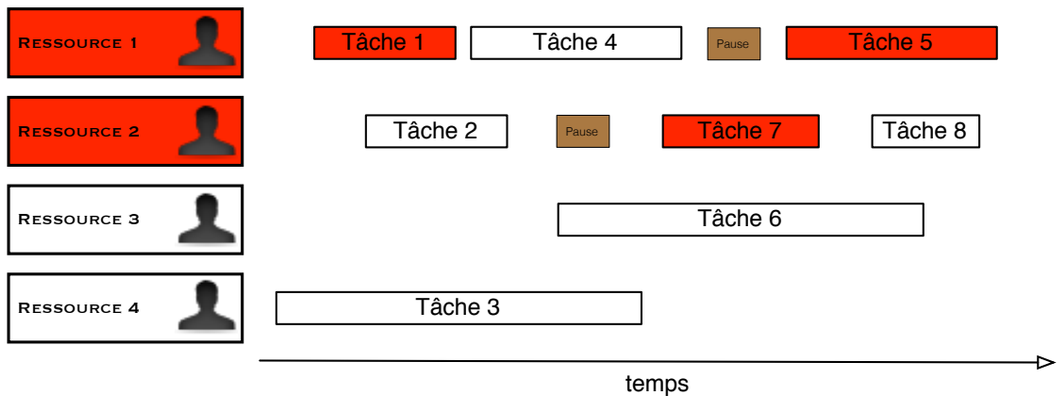
Constraint Programming

SNCF train driver planning using CP

- Legal rules constraints:
  - daily working time
  - daily rest periods
  - …
- Preferences:
  - Type of lines
  - Day of rest
  - Place of rest
  - …
- Solution:
  - Best in terms of ressources
  - Robust one
  - Cheapest one
  - …



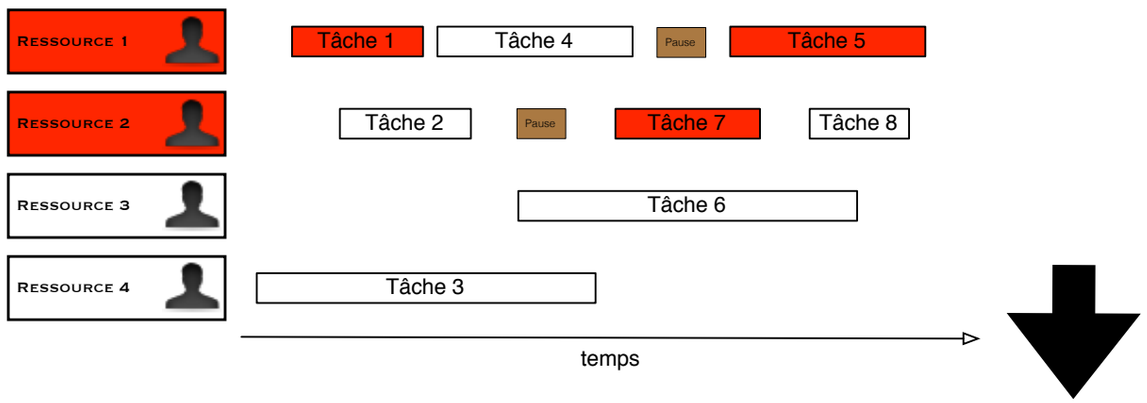


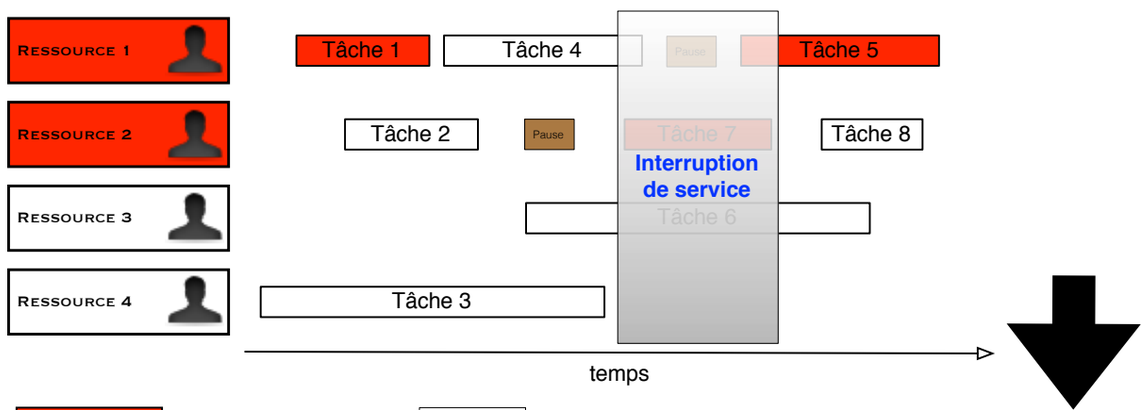


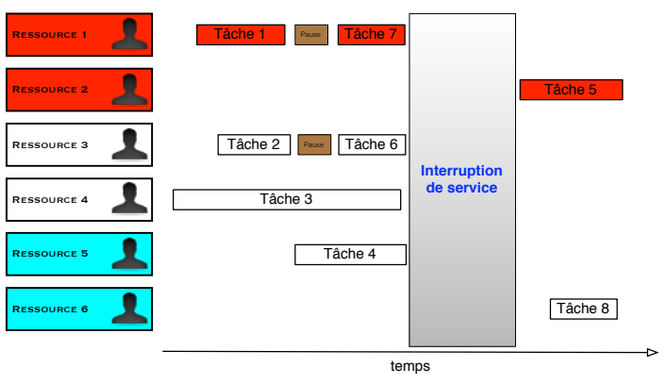
Constraint Programming

SNCF train driver planning using CP

- Maintaining an existing planning





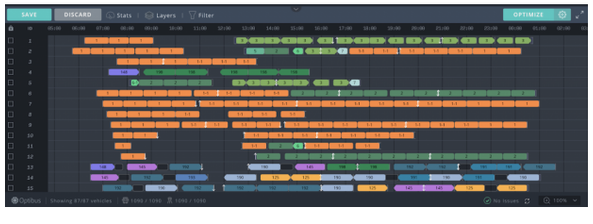


Constraint Programming

ABB Robotics partner projects

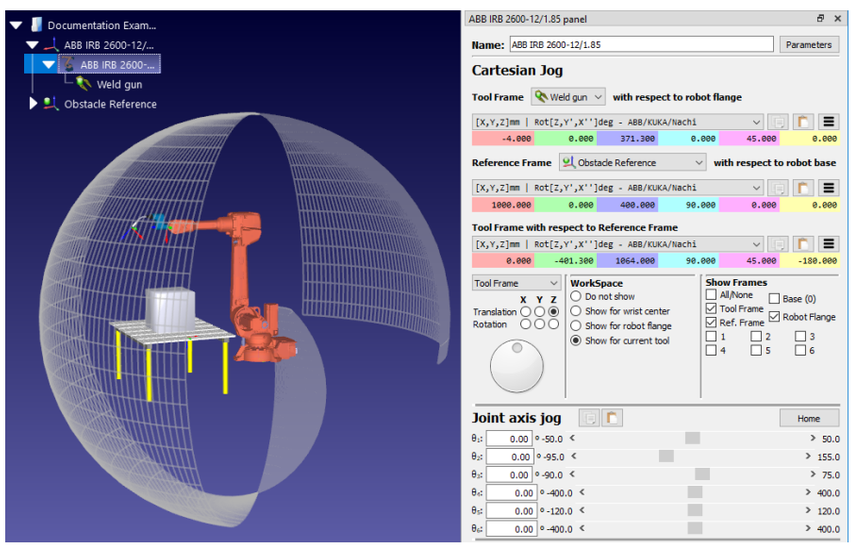
- SWMOD: Test Case Execution Scheduling with CP

"SWMOD deployed at ABB Robotics and used every day to schedule tests throughout several ABB centers in the world (Norway, Sweden, India, China)"



https://github.com/Makouno44/Robtest

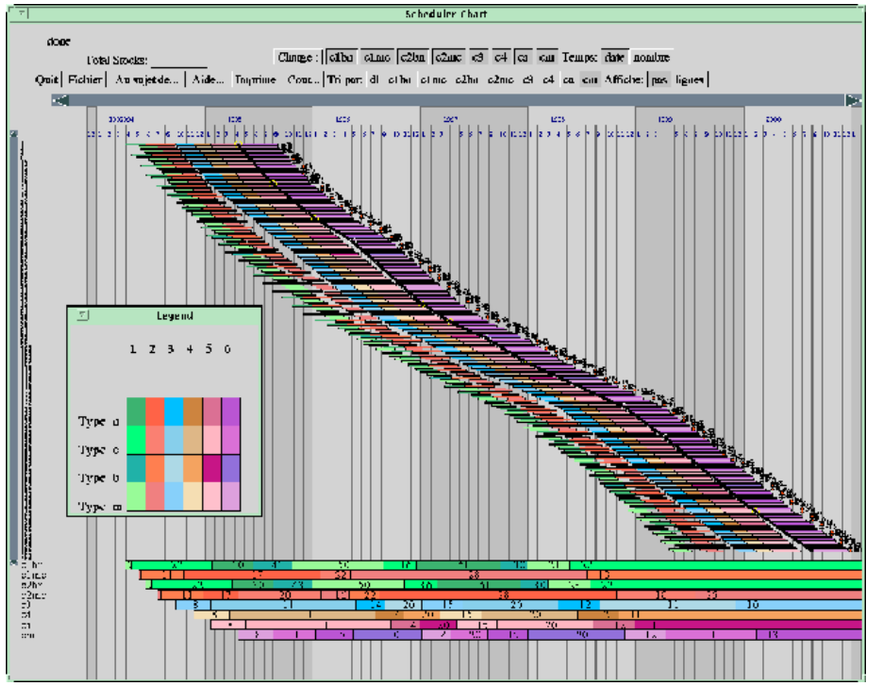
- Robtest: Optimal Stress Test Trajectories for Robots with CP



Constraint Programming

Aircraft Industry - Dassault Aviation

Assembly of Mirage aircraft



DASSAULT AVIATION

Problème de Satisfaction de Contraintes (CSP)

Qu'est-ce qu'un CSP ?

- Un CSP est défini par :
  - **Un Réseau de Contraintes :**
    - **Variables :** Ensemble de variables $X = \{x_1, x_2, \ldots, x_n\}$
    - **Domaines :** $D(x_i)$ pour chaque variable x_i
    - **Contraintes :** Restrictions sur les valeurs possibles des variables

- **Question :** Existe-t-il une affectation des variables respectant toutes les contraintes ?

- **Solution :** Une affectation des variables qui satisfait toutes les contraintes

CSP : Conjonction de sous-Problème

Sous-Problème = Contrainte

- **La Programmation par Contraintes : Conjonction de Sous-Problèmes**
  - Un problème = une conjonction de sous-problèmes
  - Chaque problème peut être décomposé en sous-problèmes plus simples, qui sont eux-mêmes liés par des contraintes.
- **Ces sous-problèmes sont souvent imbriqués, dépendants les uns des autres**
  - Un réseau de contraintes = une conjonction de contraintes
  - Un réseau de contraintes représente l'ensemble des conditions imposées sur les variables
  - Chaque contrainte exprime une relation entre les variables du problème, limitant l'ensemble des solutions possibles
- **Un sous-problème est une contrainte**
  - Chaque sous-problème peut être vu comme une contrainte spécifique qui restreint les solutions possibles
  - Résoudre un sous-problème revient à appliquer une contrainte sur les variables du problème global

CSP : Conjonction de sous-Problème

Sous-Problème = Contrainte

CSP : Conjonction de sous-Problème

Sous-Problème = Contrainte

Problème

CSP : Conjonction de sous-Problème

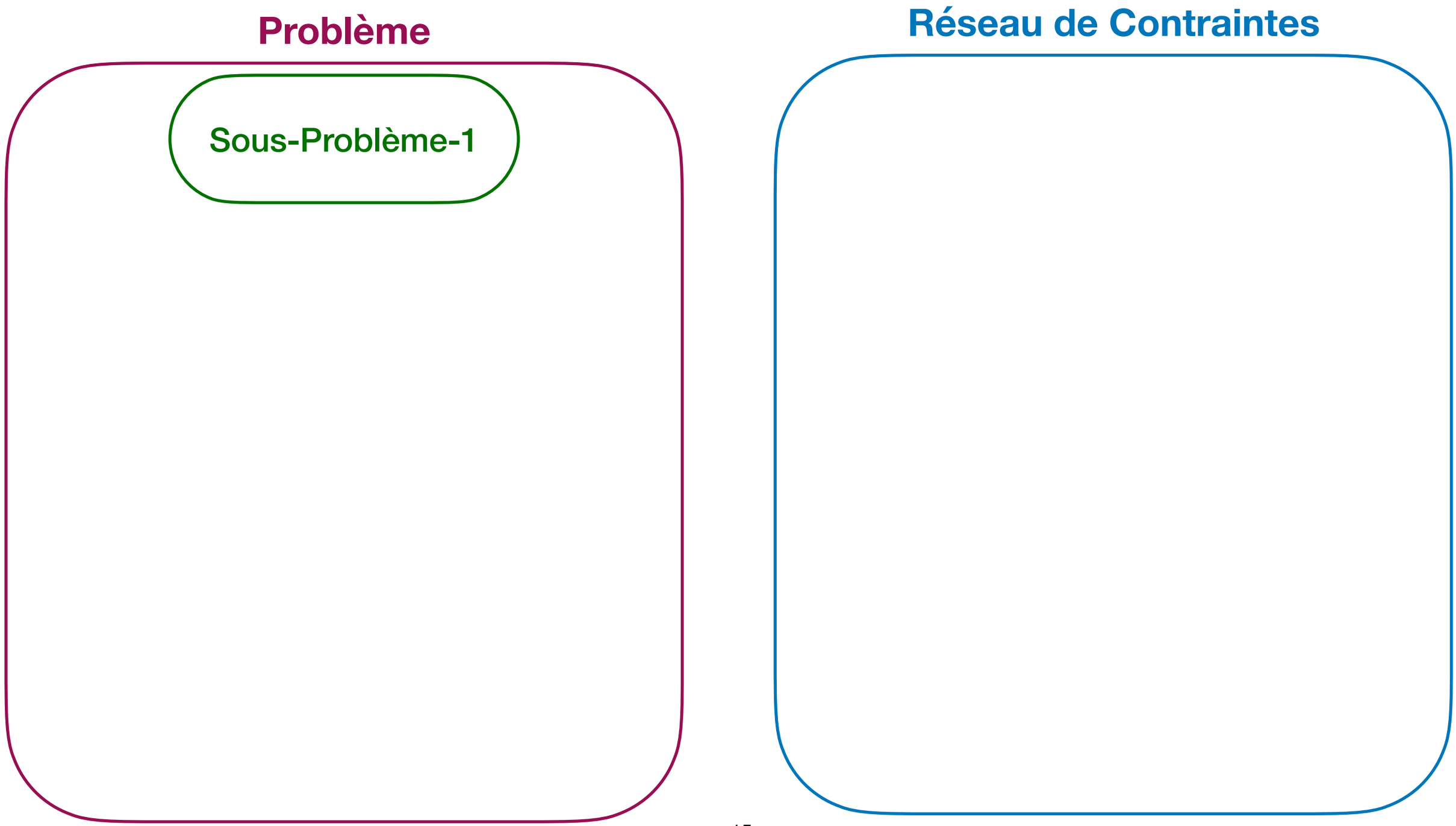
Sous-Problème = Contrainte

Problème

Réseau de Contraintes

CSP : Conjonction de sous-Problème

Sous-Problème = Contrainte



CSP : Conjonction de sous-Problème

Sous-Problème = Contrainte

Problème

Sous-Problème-1

Réseau de Contraintes

$x_1 + x_2 + x_3 + x_4 = 5$

CSP : Conjonction de sous-Problème

Sous-Problème = Contrainte

Problème

Sous-Problème-1

Sous-Problème-2

Réseau de Contraintes

$x_1 + x_2 + x_3 + x_4 = 5$

CSP : Conjonction de sous-Problème

Sous-Problème = Contrainte

Problème

Sous-Problème-1

Sous-Problème-2

Réseau de Contraintes

$x_1 + x_2 + x_3 + x_4 = 5$

$x_1 \wedge x_2 \implies x_3$

CSP : Conjonction de sous-Problème

Sous-Problème = Contrainte

Problème

Sous-Problème-1

Sous-Problème-2

Sous-Problème-3

Réseau de Contraintes

$x_1 + x_2 + x_3 + x_4 = 5$

$x_1 \wedge x_2 \implies x_3$

CSP : Conjonction de sous-Problème

Sous-Problème = Contrainte

Problème

- Sous-Problème-1
- Sous-Problème-2
- Sous-Problème-3

Réseau de Contraintes

- $x_1 + x_2 + x_3 + x_4 = 5$
- $x_1 \wedge x_2 \implies x_3$
- $allDifferent(X)$

CSP : Conjonction de sous-Problème

Sous-Problème = Contrainte

Problème

- Sous-Problème-1
- Sous-Problème-2
- Sous-Problème-3
- Sous-Problème-4

Réseau de Contraintes

$x_1 + x_2 + x_3 + x_4 = 5$

$x_1 \wedge x_2 \implies x_3$

$allDifferent(X)$

CSP : Conjonction de sous-Problème

Sous-Problème = Contrainte

Problème

- Sous-Problème-1
- Sous-Problème-2
- Sous-Problème-3
- Sous-Problème-4

Réseau de Contraintes

$x_1 + x_2 + x_3 + x_4 = 5$

$x_1 \wedge x_2 \implies x_3$

$allDifferent(X)$

x_1	x_2	x_3	x_4
1	0	7	0
1	0	7	0
2	1	8	1
4	3	8	1
4	6	8	2
5	9	9	2
5	9	9	3
5	10	9	3

CSP : Conjonction de sous-Problème

Sous-Problème = Contrainte

Problème

- Sous-Problème-1
- Sous-Problème-2
- Sous-Problème-3
- Sous-Problème-4
- Sous-Problème-5

Réseau de Contraintes

$x_1 + x_2 + x_3 + x_4 = 5$

$x_1 \wedge x_2 \implies x_3$

$allDifferent(X)$

x_1	x_2	x_3	x_4
1	0	7	0
1	0	7	0
2	1	8	1
4	3	8	1
4	6	8	2
5	9	9	2
5	9	9	3
5	10	9	3

CSP : Conjonction de sous-Problème

Sous-Problème = Contrainte

Problème

- Sous-Problème-1
- Sous-Problème-2
- Sous-Problème-3
- Sous-Problème-4
- Sous-Problème-5

Réseau de Contraintes

$x_1 + x_2 + x_3 + x_4 = 5$

$x_1 \wedge x_2 \implies x_3$

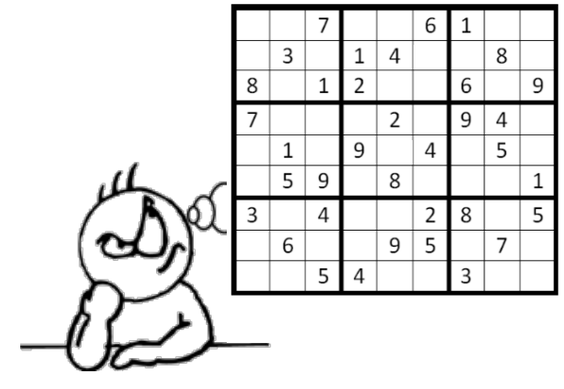
x_1	x_2	x_3	x_4
1	0	7	0
1	0	7	0
2	1	8	1
4	3	8	1
4	6	8	2
5	9	9	2
5	9	9	3
5	10	9	3

$allDifferent(X)$

x_1	x_2	x_3
1	2	3
2	3	4
3	4	5
4	5	6

CSP

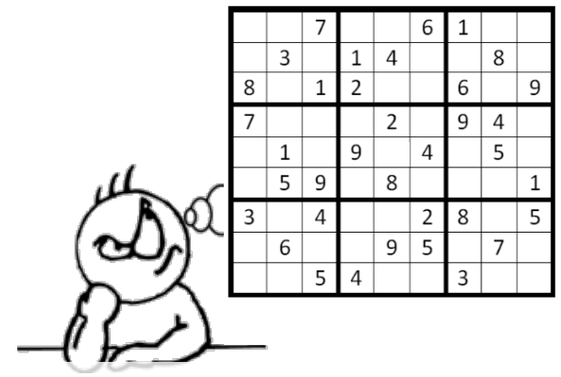
Exemple - Sudoku



- **Réseau de Contraintes :**
  - **Variables :** Les 81 cases de la grille (9×9)
  - **Domaines :** $\{1,2,\ldots,9\}$ pour chaque case
  - **Contraintes :**
    - Chaque ligne contient les chiffres de 1 à 9 sans répétition
    - Chaque colonne contient les chiffres de 1 à 9 sans répétition
    - Chaque sous-grille 3x3 contient les chiffres de 1 à 9 sans répétition
- **Question :** Existe-t-il une affectation des chiffres respectant ces contraintes ?
- **Solution :** Une grille entièrement remplie respectant les contraintes

CSP

Exemple - Sudoku



- **Réseau de Contraintes :**
  - **Variables :** Les 81 cases de la grille (9×9)
  - **Espace de Recherche = Produit cartésien du domaine des 81 variables**

$$D(x_1) \times D(x_2) \ldots D(x_{81})$$

  - **Taille de l'espace de recherche =** D^X
- **Question :** Existe-t-il une affectation des chiffres respectant ces contraintes ?
- **Solution :** Une grille entièrement remplie respectant les contraintes

CSP

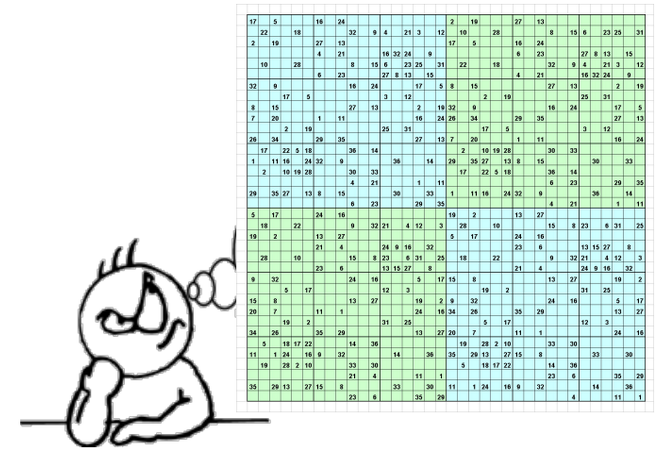
Exemple - Sudoku

Sudoku(36×36)

17		5				16		24										2		19				27		13								
	22			18					32		9	4		21	3		12		10			28					8		15	6		23	25		31
2		19				27		13										17		5				16		24								
						4		21				16	32	24		9								6		23				27	8	13		15	
	10			28					8		15	6		23	25		31		22			18					32		9	4		21	3		12
						6		23				27	8	13		15								4		21				16	32	24		9	
32		9							16		24				17		5	8		15							27		13				2		19
			17		5							3		12							2		19							25		31			
8		15							27		13				2		19	32		9							16		24				17		5
7		20				1		11							16		24	26		34				29		35							27		13
			2		19							25		31							17		5							3		12			
26		34				29		35							27		13	7		20				1		11							16		24
	17		22	5	18				36		14								2		10	19	28				30		33						
1		11	16		24	32		9					36			14		29		35	27		13	8		15					30			33	
	2		10	19	28				30		33								17		22	5	18				36		14						
									4		21				1		11										6		23				29		35
29		35	27		13	8		15					30			33		1		11	16		24	32		9					36			14	
									6		23				29		35										4		21				1		11
5		17				24		16										19		2				13		27								
	18			22					9		32	21		4	12		3		28			10					15		8	23		6	31		25
19		2				13		27										5		17				24		16								
						21		4				24	9	16		32								23		6				13	15	27		8	
	28			10					15		8	23		6	31		25		18			22					9		32	21		4	12		3
						23		6				13	15	27		8								21		4				24	9	16		32	
9		32							24		16				5		17	15		8							13		27				19		2
			5		17							12		3							19		2							31		25			
15		8							13		27				19		2	9		32							24		16				5		17
20		7				11		1							24		16	34		26				35		29							13		27
			19		2							31		25							5		17							12		3			
34		26				35		29							13		27	20		7				11		1							24		16
	5		18	17	22				14		36								19		28	2	10				33		30						
11		1	24		16	9		32					14			36		35		29	13		27	15		8					33			30	
	19		28	2	10				33		30								5		18	17	22				14		36						
									21		4				11		1										23		6				35		29
35		29	13		27	15		8					33			30		11		1	24		16	9		32					14			36	
									23		6				35		29										4						11		1

CSP

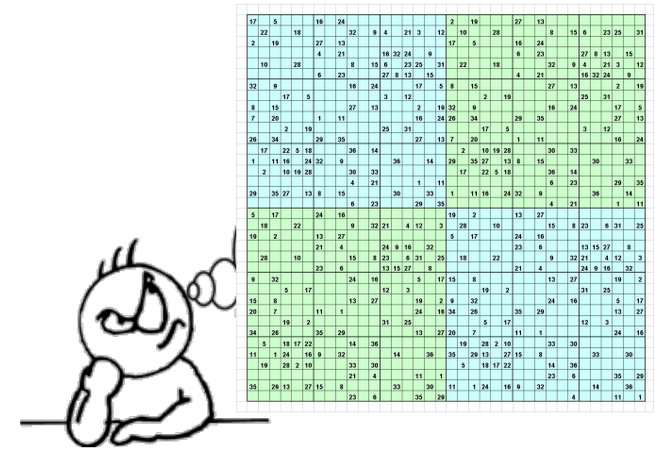
Exemple - Sudoku



- **Réseau de Contraintes :**
  - **Variables :** Les 1296 cases de la grille (36×36)
  - **Domaines :** $\{1,2,\ldots,36\}$ pour chaque case
  - **Contraintes :**
    - Chaque ligne contient les chiffres de 1 à 36 sans répétition
    - Chaque colonne contient les chiffres de 1 à 36 sans répétition
    - Chaque sous-grille 6x6 contient les chiffres de 1 à 36 sans répétition
- **Question :** Existe-t-il une affectation des chiffres respectant ces contraintes ?
- **Solution :** Une grille entièrement remplie respectant les contraintes

CSP

Exemple - Sudoku



- **Réseau de Contraintes :**
  - **Variables :** Les n^2cases de la grille $(n \times n)$
  - **Domaines :** $\{1,2,\ldots,n\}$ pour chaque case
  - **Contraintes :**
    - Chaque ligne contient les chiffres de 1 à n sans répétition
    - Chaque colonne contient les chiffres de 1 à n sans répétition
    - Chaque sous-grille $\sqrt{n}$ contient les chiffres de 1 à n sans répétition
- **Question :** Existe-t-il une affectation des chiffres respectant ces contraintes ?
- **Solution :** Une grille entièrement remplie respectant les contraintes

Modélisation PPC

Stratégies de Modélisation

1. **Identification des variables et domaines**
   - Quels sont les éléments clés de la solution ?
   - Quelles valeurs ces éléments peuvent-ils prendre ?
2. **Définition des contraintes**
   - Quelles sont les relations entre les variables ?
   - Exprimer les contraintes sous une forme mathématique ou logique
3. **Choix du niveau de granularité**
   - Une variable représente-t-elle un sous-problème ou une valeur individuelle ?
   - Faut-il regrouper certaines variables pour simplifier le modèle ?
4. **Optimisation de la modélisation**
   - Éviter les redondances
   - Réduire la complexité en limitant le nombre de contraintes
   - Penser à la propagation des contraintes pour guider la recherche

Modélisation PPC

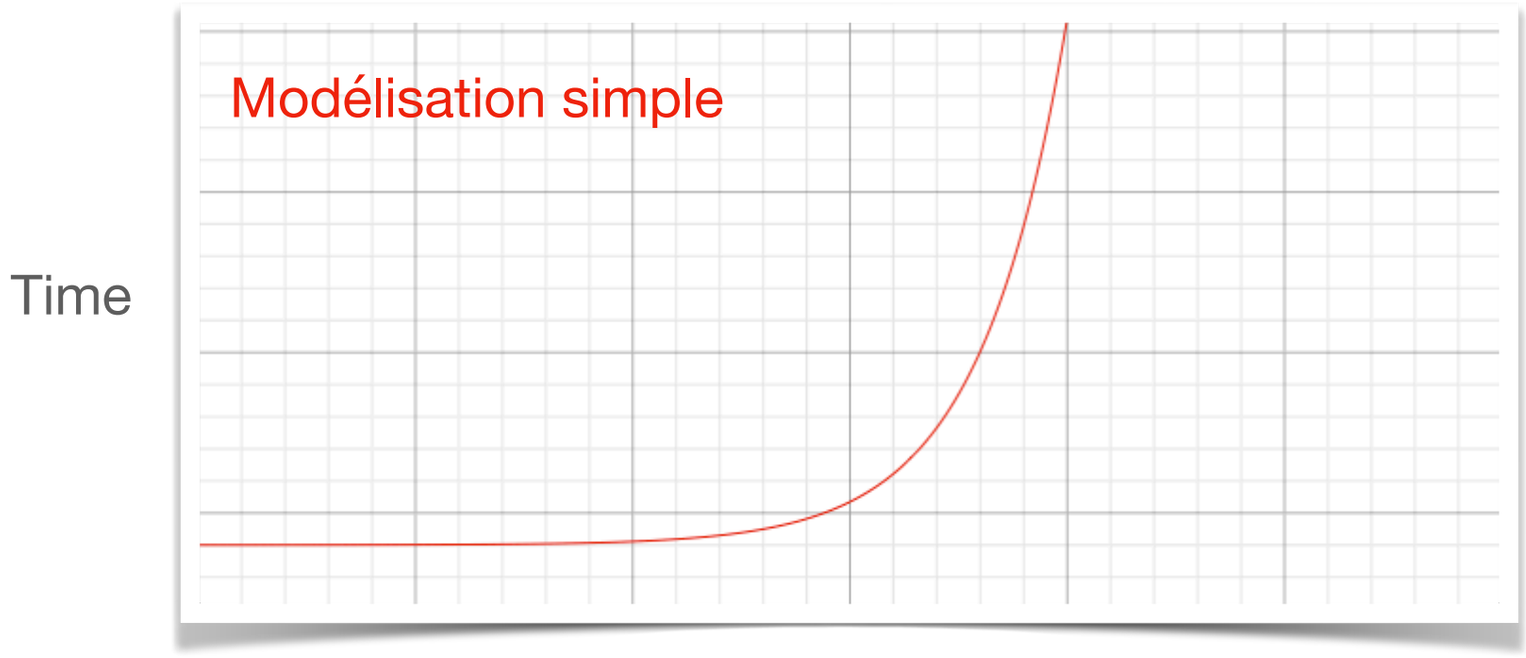
Stratégies de Modélisation

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

1. **Identification des variables et domaines**
   - Quels sont les éléments clés de la solution ?
   - Quelles valeurs ces éléments peuvent-ils prendre ?
2. **Définition des contraintes**
   - Quelles sont les relations entre les variables ?
   - Exprimer les contraintes sous une forme mathématique ou logique
3. **Choix du niveau de granularité**
   - Une variable représente-t-elle un sous-problème ou une valeur individuelle ?
   - Faut-il regrouper certaines variables pour simplifier le modèle ?
4. **Optimisation de la modélisation**
   - Éviter les redondances
   - Réduire la complexité en limitant le nombre de contraintes
   - Penser à la propagation des contraintes pour guider la recherche

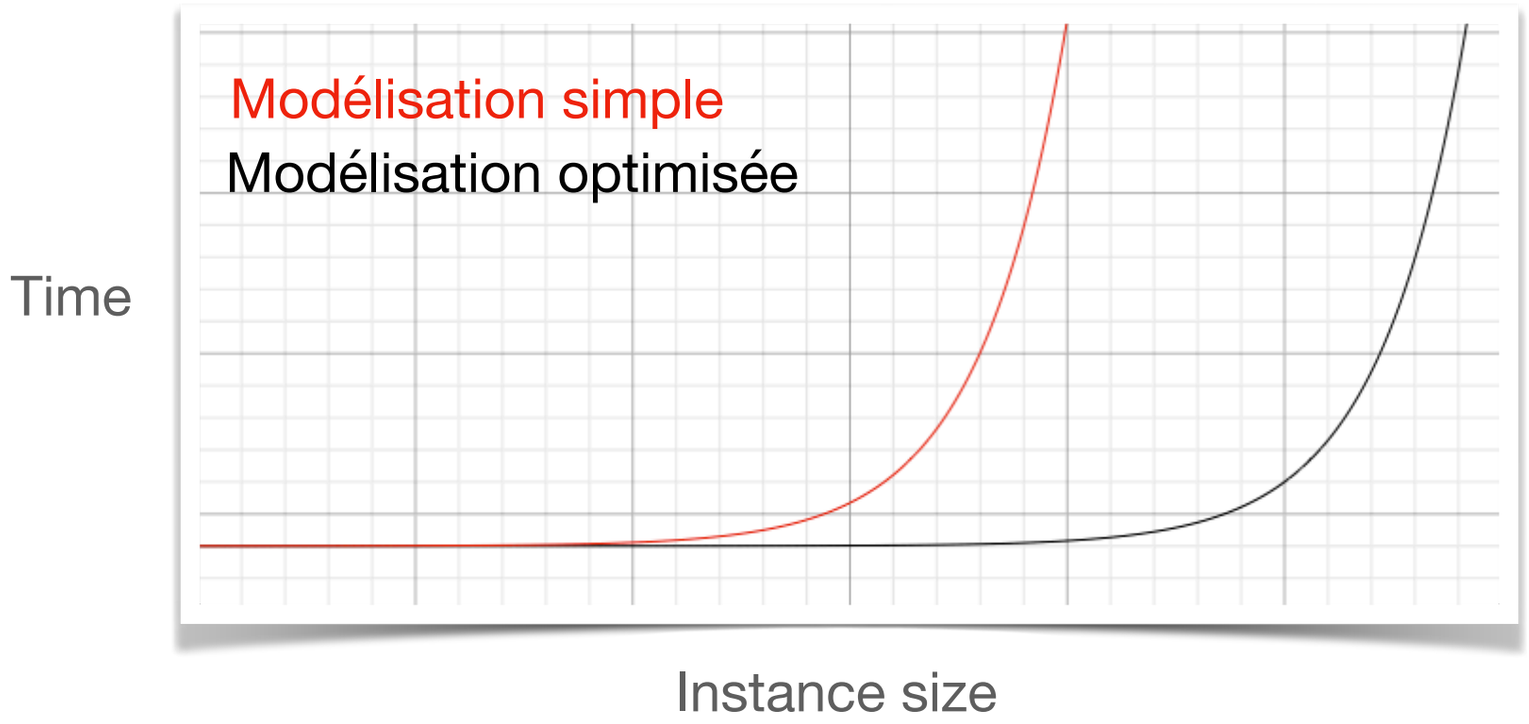
Modélisation PPC

Stratégies de Modélisation



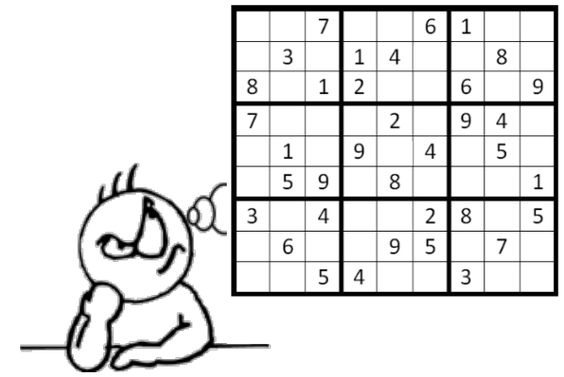
Modélisation PPC

Stratégies de Modélisation



CSP

Exemple - Sudoku - MiniZinc



```
include "globals.mzn";

int: N ; % Taille de la grille
int: S = ceil(sqrt(N));
set of int: PuzzleRange = 1..N;
int: digs = ceil(log(10.0,int2float(N))); % digits for output


array[PuzzleRange, PuzzleRange] of var 1..9: X; % Variables représentant chaq

% Contraintes de ligne (paires distinctes)
constraint forall(i in PuzzleRange, j1 in PuzzleRange, j2 in j1+1..N) (
    X[i, j1] != X[i, j2]
);

% Contraintes de colonne (paires distinctes)
constraint forall(j in PuzzleRange, i1 in PuzzleRange, i2 in i1+1..N) (
    X[i1, j] != X[i2, j]
);

% Contraintes de bloc 3x3 (paires distinctes)
constraint forall(bi in 0..2, bj in 0..2, di1 in 1..3, dj1 in 1..3,
                  di2 in 1..3, dj2 in 1..3 where (di1, dj1) != (di2, dj2)) (
    X[3 * bi + di1, 3 * bj + dj1] != X[3 * bi + di2, 3 * bj + dj2]
);

solve satisfy;

output  [ show_int(digs,X[i,j]) ++ " " ++
         if j mod S == 0 then " " else "" endif ++
         if j == N /\ i != N then
             if i mod S == 0 then "\n\n" else "\n" endif
         else "" endif
         | i,j in PuzzleRange ] ++ ["\n"];

```

Modélisation PPC

Cassure de Symétrie

1. **Pourquoi la symétrie est un problème ?**
   - Les problèmes symétriques ont plusieurs solutions équivalentes qui mènent à la même réponse
   - Sans intervention, l'algorithme peut explorer ces solutions équivalentes de manière redondante, augmentant le temps de calcul
2. **Cassure de symétrie = Réduire l'espace de recherche**
   - La cassure de symétrie consiste à ajouter des contraintes supplémentaires pour éliminer ces solutions symétriques
   - Elle permet de réduire l'espace de recherche en excluant les solutions redondantes
3. **Méthodes courantes pour briser la symétrie**
   - Fixer une valeur pour certaines variables : Par exemple, fixer la première variable à une valeur donnée pour éviter des permutations équivalentes
   - Ajouter des contraintes d'ordre : Exiger que certaines variables soient strictement supérieures ou inférieures à d'autres, créant ainsi un ordre préférentiel
   - Utiliser des heuristiques : Choisir des valeurs ou des affectations selon un critère particulier (par exemple, minimiser les permutations)

Modélisation PPC

Cassure de Symétrie - Exemple

Modélisation PPC

Cassure de Symétrie - Exemple

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

Modélisation PPC

Cassure de Symétrie - Exemple

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

Modélisation PPC

Cassure de Symétrie - Exemple

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

Modélisation PPC

Cassure de Symétrie - Exemple

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

Modélisation PPC

Cassure de Symétrie - Exemple

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

réflexion verticale

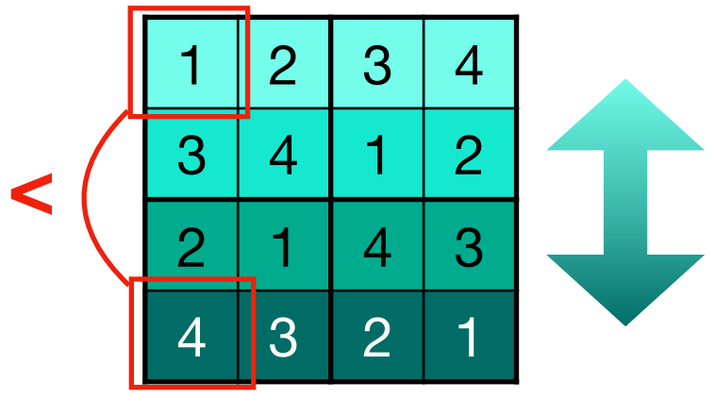
4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

Modélisation PPC

Cassure de Symétrie - Exemple

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1



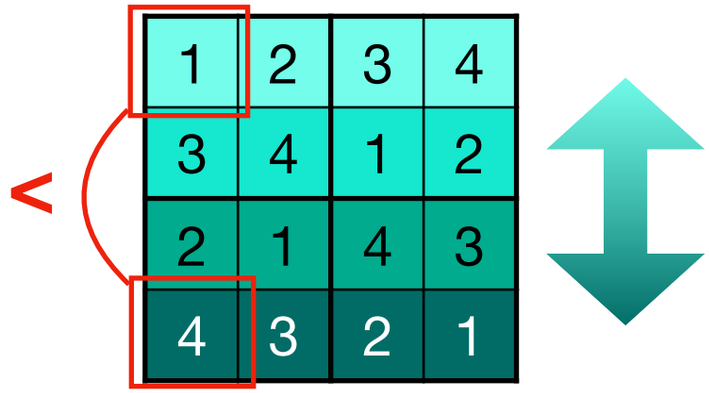
réflexion verticale

4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

Modélisation PPC

Cassure de Symétrie - Exemple

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1



1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

réflexion verticale

4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

Modélisation PPC

Cassure de Symétrie - Exemple

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

<

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

réflexion verticale

4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

Modélisation PPC

Cassure de Symétrie - Exemple

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

<

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

réflexion verticale

réflexion horizontale

4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

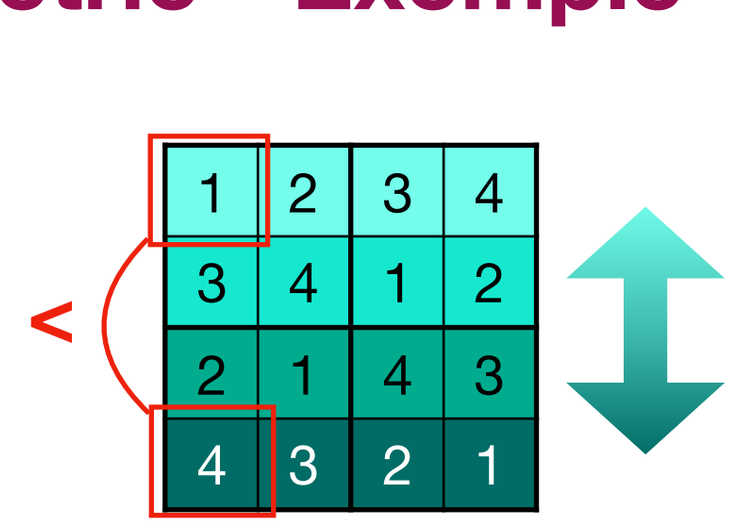
4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

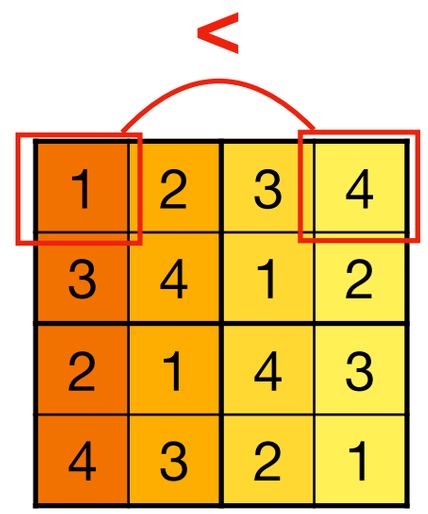
4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

Modélisation PPC

Cassure de Symétrie - Exemple

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1





4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

réflexion verticale

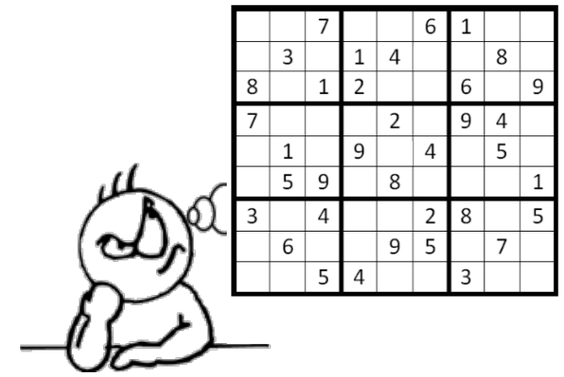
4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

réflexion horizontale

4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

Modélisation PPC

Cassure de Symétrie - Sudoku



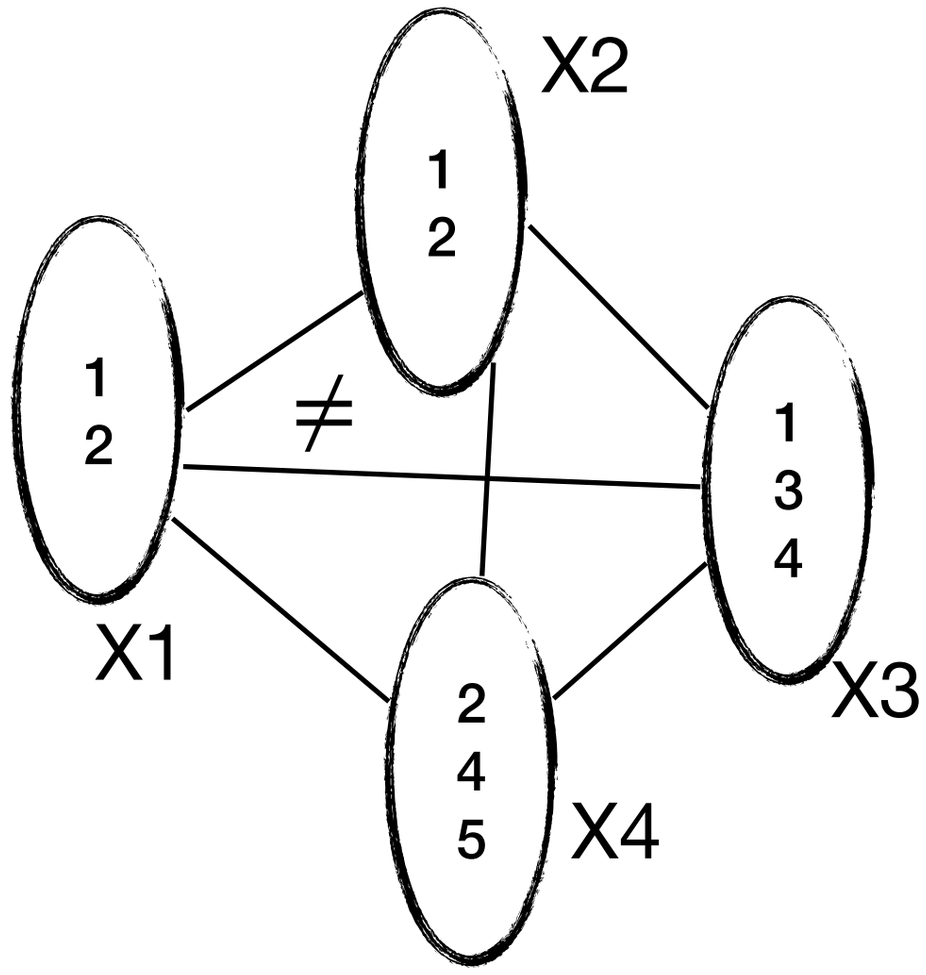
```
int: N;
int: S = ceil(sqrt(N));
int: digs = ceil(log(10.0,int2float(N))); % digits for output

set of int: PuzzleRange = 1..N;


array[PuzzleRange, PuzzleRange] of var 1..N: X; % Variables représentant cha

% Contraintes de ligne (paires distinctes)
constraint forall(i in PuzzleRange, j1 in PuzzleRange, j2 in j1+1..N) (
    X[i, j1] != X[i, j2]
);

% Cassure de symétrie : empêcher la réflexion horizontale
constraint X[1,1] < X[1,N];

% Contraintes de colonne (paires distinctes)
constraint forall(j in PuzzleRange, i1 in PuzzleRange, i2 in i1+1..N) (
    X[i1, j] != X[i2, j]
);

% Contraintes de bloc (paires distinctes)
constraint forall(bi in 0..S-1, bj in 0..S-1, di1 in 1..S, dj1 in 1..S,
                  di2 in 1..S, dj2 in 1..S where (di1, dj1) != (di2, dj2)) (
    X[S * bi + di1, S * bj + dj1] != X[S * bi + di2, S * bj + dj2]
);

solve satisfy;

output  [ show_int(digs,X[i,j]) ++ " " ++
          if j mod S == 0 then " " else "" endif ++
          if j == N /\ i != N then
              if i mod S == 0 then "\n\n" else "\n" endif
          else "" endif
          | i,j in PuzzleRange ] ++ ["\n"];
```

Contraintes Globales

Exemple : Alldifferent

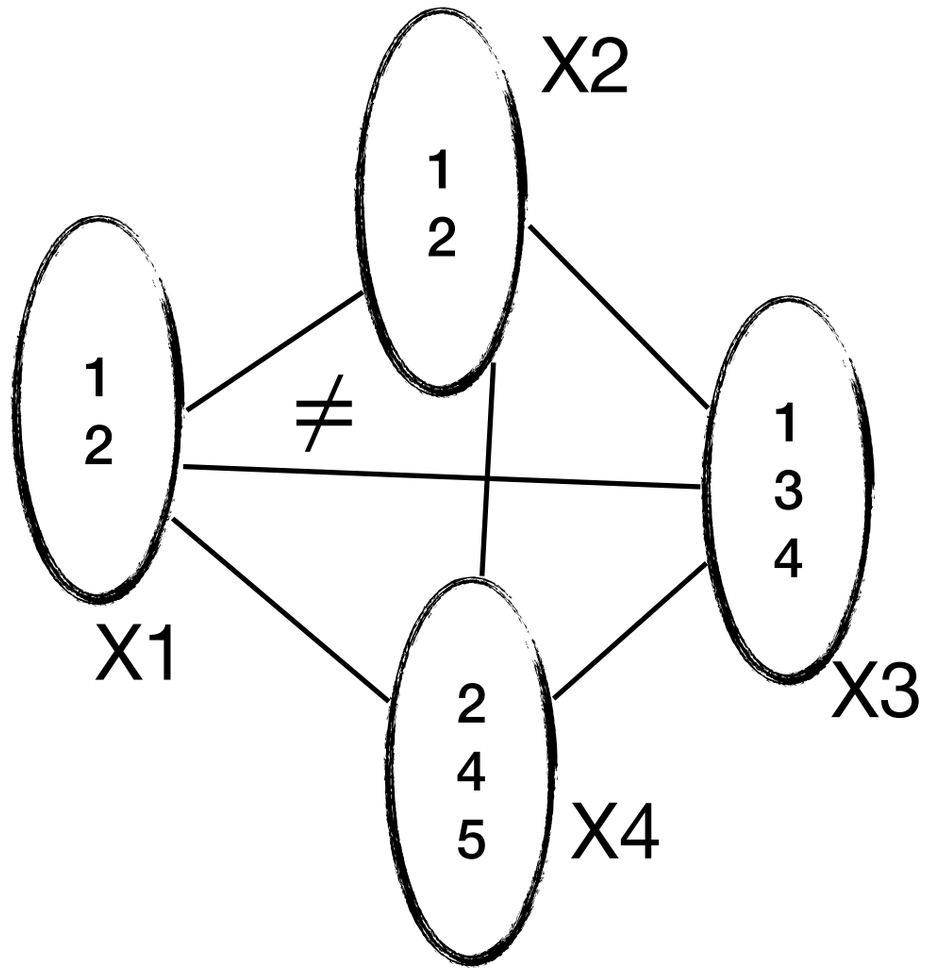
Contraintes Globales

Exemple : Alldifferent



Contraintes Globales

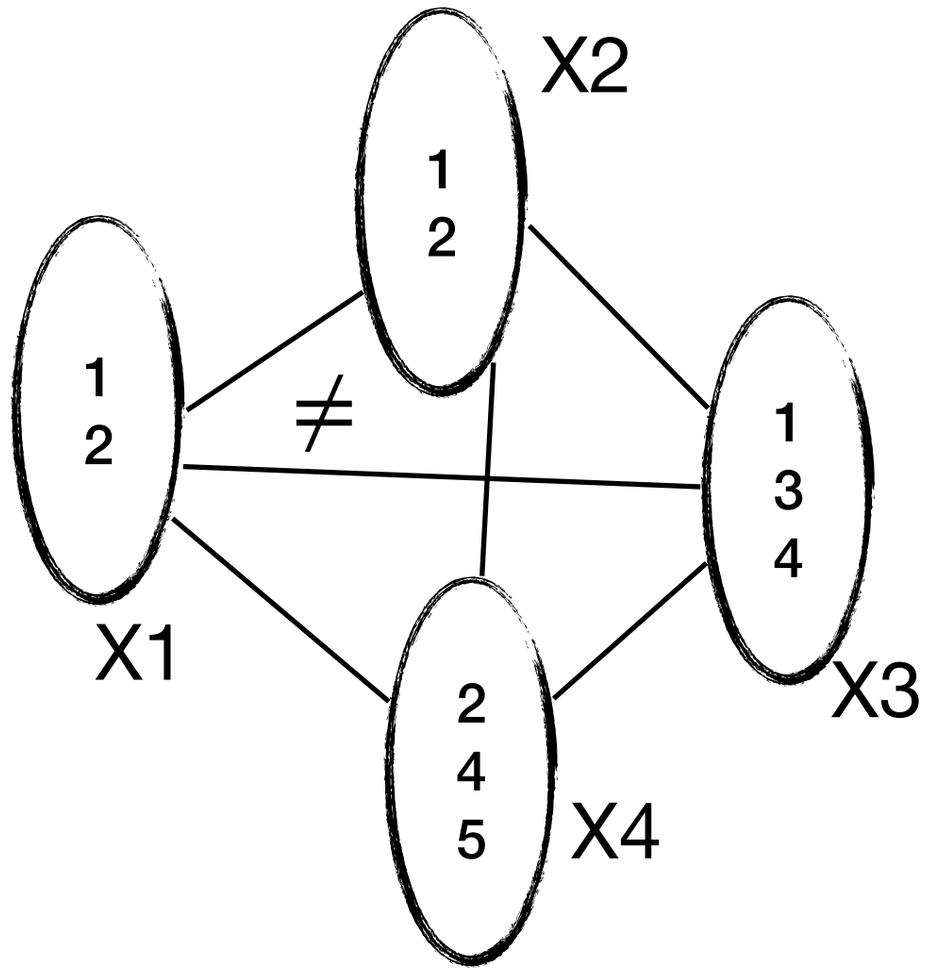
Exemple : Alldifferent



$$\forall x_i, x_j \in X : \; i < j$$
$$x_i \neq x_j$$

Contraintes Globales

Exemple : Alldifferent

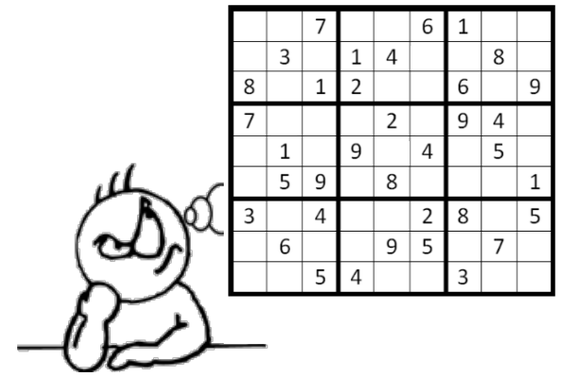


$$\forall x_i, x_j \in X : \; i < j$$
$$x_i \neq x_j$$

$$allDifferent(X)$$

Modélisation PPC

Cassure de Symétrie - Sudoku

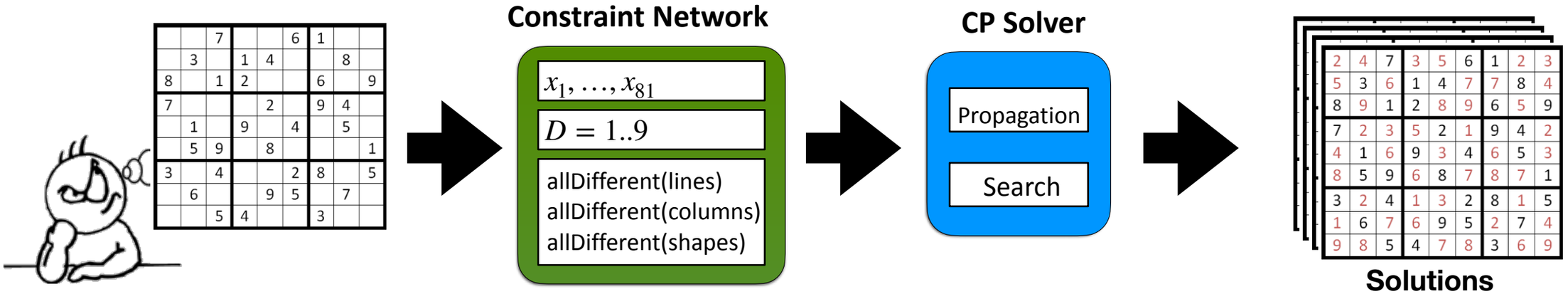


```
int: N;
int: S = ceil(sqrt(N));
int: digs = ceil(log(10.0,int2float(N))); % digits for output

set of int: PuzzleRange = 1..N;


array[PuzzleRange, PuzzleRange] of var 1..N: X; % Variables représentant cha

% Contraintes de ligne (paires distinctes)
constraint forall(i in PuzzleRange, j1 in PuzzleRange, j2 in j1+1..N) (
    X[i, j1] != X[i, j2]
);

% Cassure de symétrie : empêcher la réflexion horizontale
constraint X[1,1] < X[1,N];

% Contraintes de colonne (paires distinctes)
constraint forall(j in PuzzleRange, i1 in PuzzleRange, i2 in i1+1..N) (
    X[i1, j] != X[i2, j]
);

% Contraintes de bloc (paires distinctes)
constraint forall(bi in 0..S-1, bj in 0..S-1, di1 in 1..S, dj1 in 1..S,
                  di2 in 1..S, dj2 in 1..S where (di1, dj1) != (di2, dj2)) (
    X[S * bi + di1, S * bj + dj1] != X[S * bi + di2, S * bj + dj2]
);

solve satisfy;

output  [ show_int(digs,X[i,j]) ++ " " ++
         if j mod S == 0 then " " else "" endif ++
         if j == N /\ i != N then
             if i mod S == 0 then "\n\n" else "\n" endif
         else "" endif
         | i,j in PuzzleRange ] ++ ["\n"];
```

```
include "alldifferent.mzn";

int: N;
int: S = ceil(sqrt(N));
int: digs = ceil(log(10.0,int2float(N))); % digits for output

set of int: PuzzleRange = 1..N;
set of int: SubSquareRange = 1..S;

array[1..N,1..N] of var PuzzleRange: X;


% Cassure de symétrie : empêcher la réflexion horizontale
constraint X[1,1] < X[1,N];

% All different in rows
constraint forall (i in PuzzleRange) (
                   alldifferent( [ X[i,j] | j in PuzzleRange ]) );

% All different in columns.
constraint forall (j in PuzzleRange) (
                   alldifferent( [ X[i,j] | i in PuzzleRange ]) );

% All different in sub-squares:
constraint
      forall (a, o in SubSquareRange)(
                alldifferent( [ X[(a-1) *S + a1, (o-1)*S + o1] |
                                a1, o1 in SubSquareRange ] ) );

solve satisfy;

output  [ show_int(digs,X[i,j]) ++ " " ++
         if j mod S == 0 then " " else "" endif ++
         if j == N /\ i != N then
             if i mod S == 0 then "\n\n" else "\n" endif
         else "" endif
         | i,j in PuzzleRange ] ++ ["\n"];
```

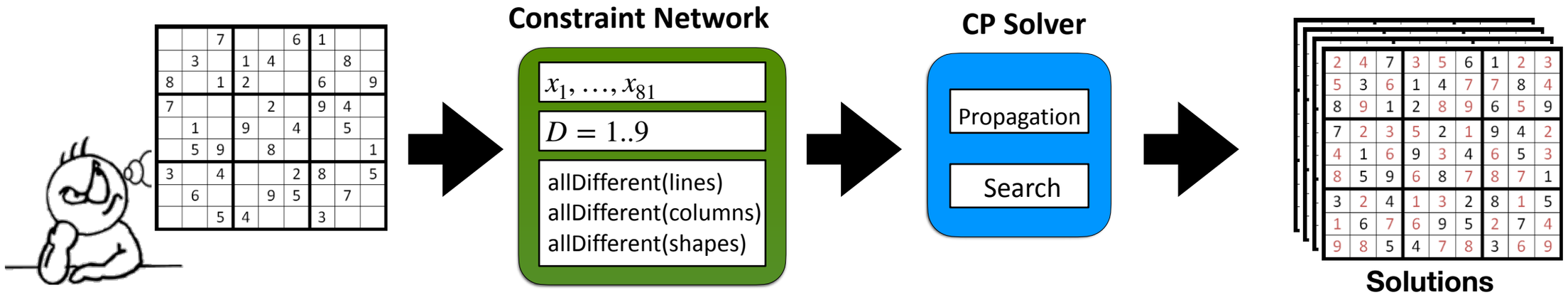
Résolution en PPC

Sudoku



Résolution en PPC

Sudoku



Propagation:

x_1={1,2,3,4,5,6,7,8,9}

…

Search:

$x_1 \leftarrow 2$

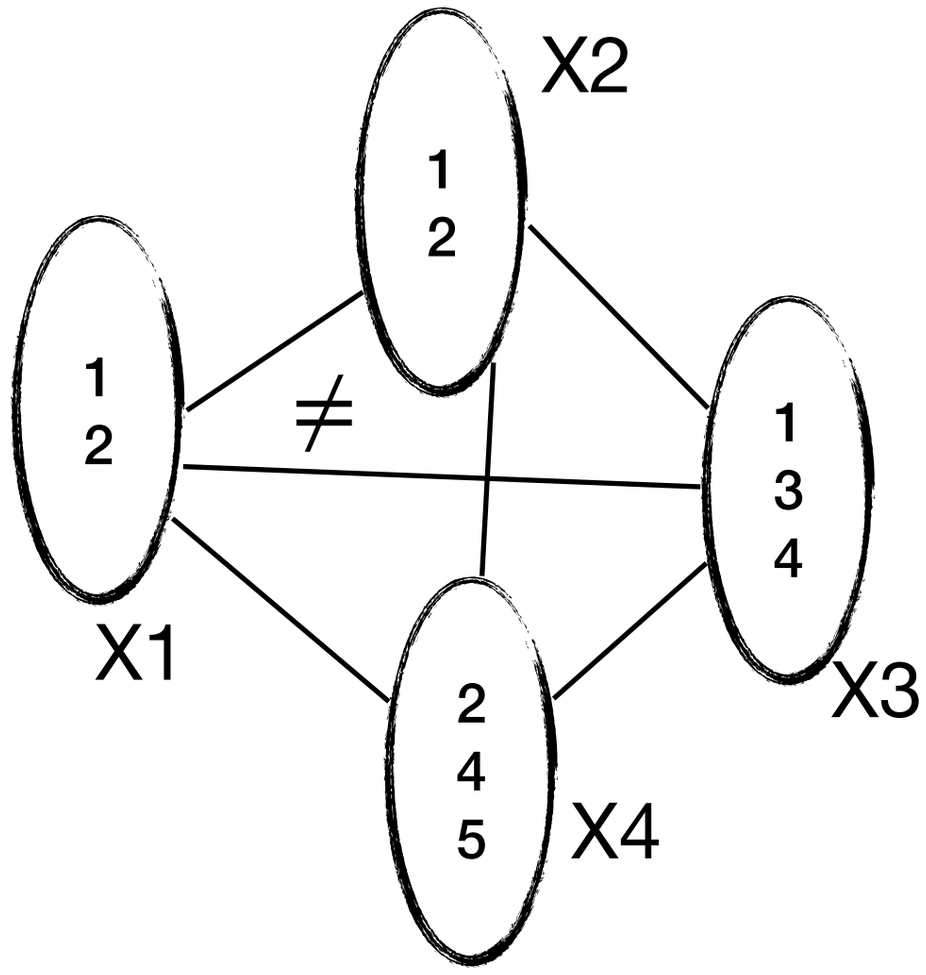
		7			6	1		
	3		1	4			8	
8		1	2			6		9
7				2		9	4	
	1		9		4		5	
	5	9		8				1
3		4			2	8		5
	6			9	5		7	
		5	4			3		

Solution:

X1 = 2
x2 = 4
X3 = 7
…
X81 = 9

Contraintes Globales

Exemple : Alldifferent

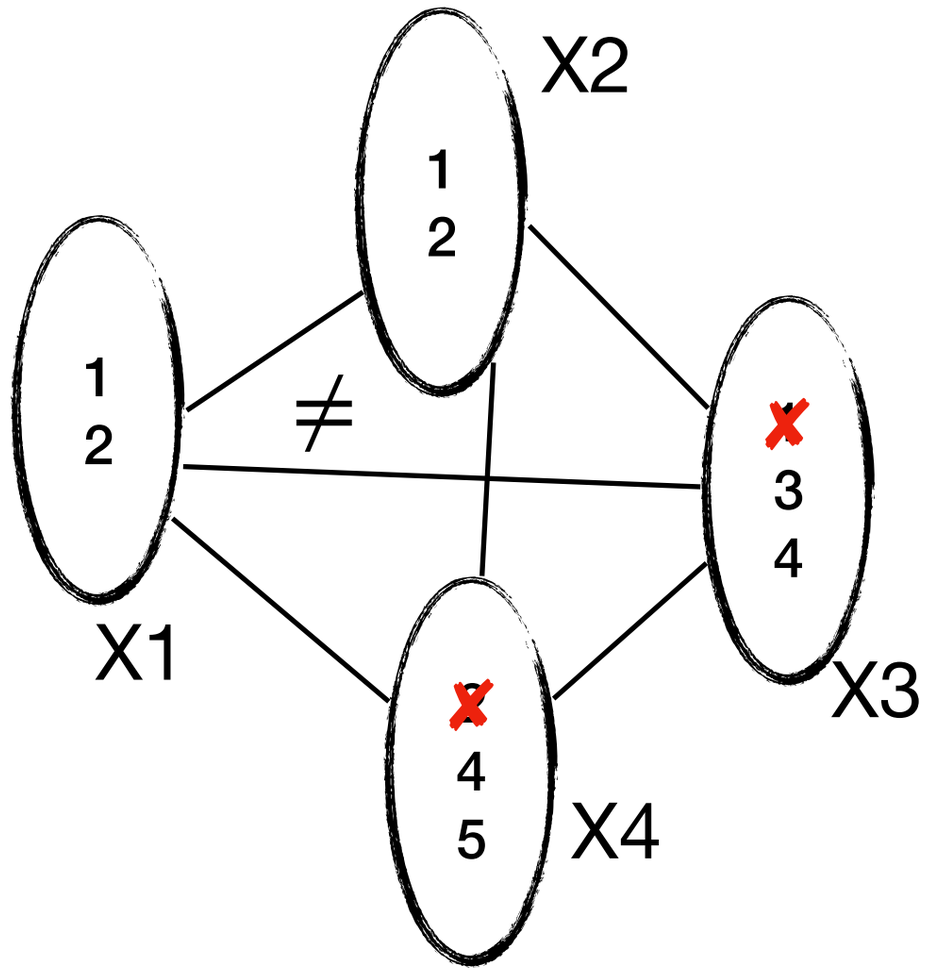


$$\forall x_i, x_j \in X : \; i < j$$
$$x_i \neq x_j$$

$$allDifferent(X)$$

Contraintes Globales

Exemple : Alldifferent



$$\forall x_i, x_j \in X : \; i < j$$
$$x_i \neq x_j$$

$$allDifferent(X)$$

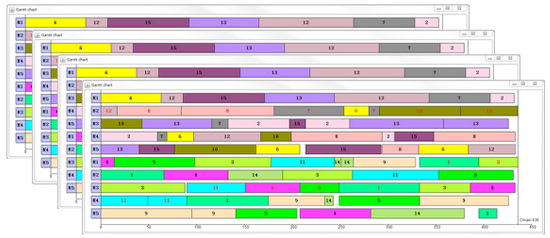
Constraint Programming

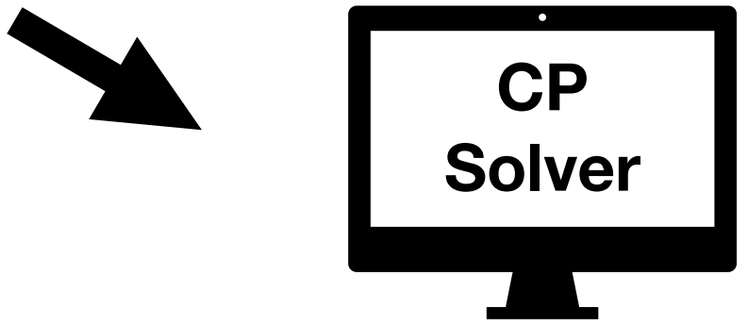
Global constraints

Constraint Programming

Global constraints

Scheduling instances

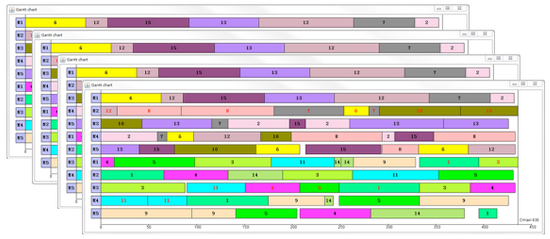


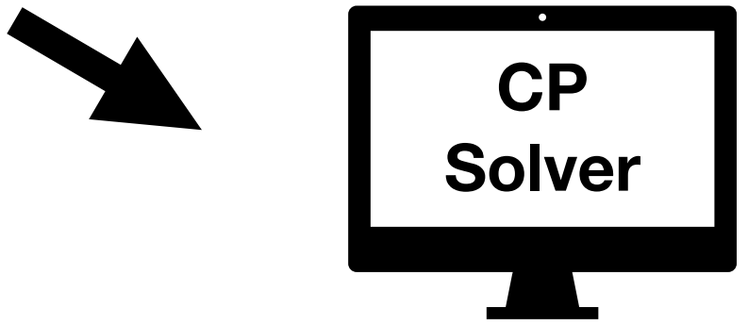


Constraint Programming

Global constraints

Scheduling instances



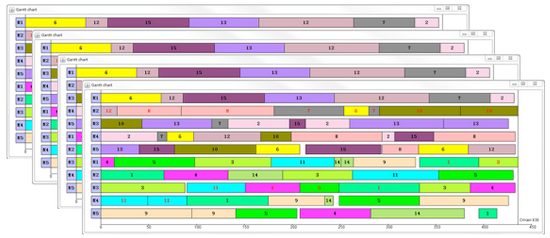


`Cumulative`

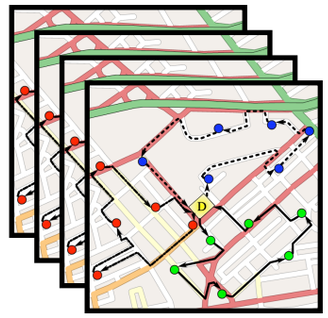
Constraint Programming

Global constraints

Scheduling instances



Vehicule routing instances

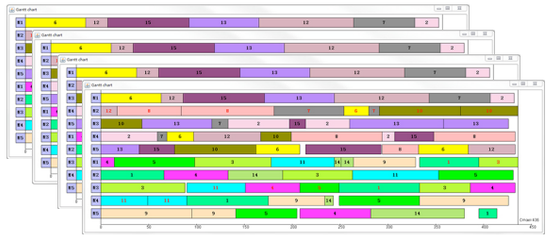


`Cumulative`

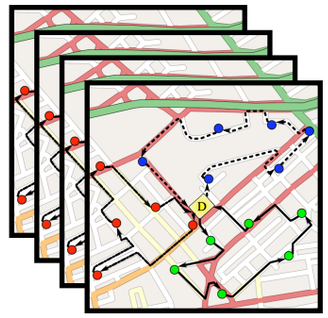
Constraint Programming

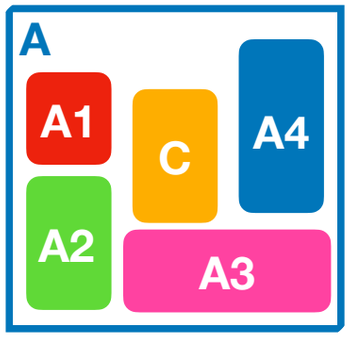
Global constraints

Scheduling instances



Vehicule routing instances



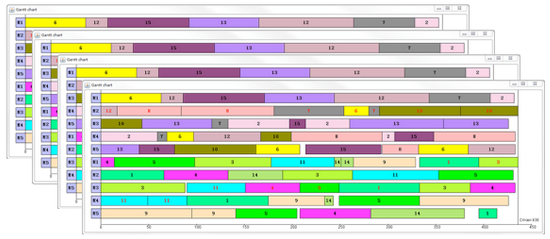


`Cumulative`

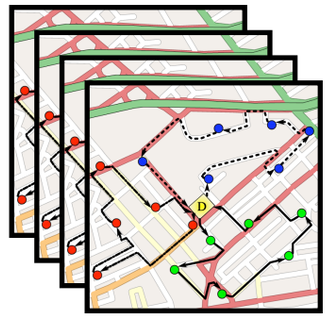
Constraint Programming

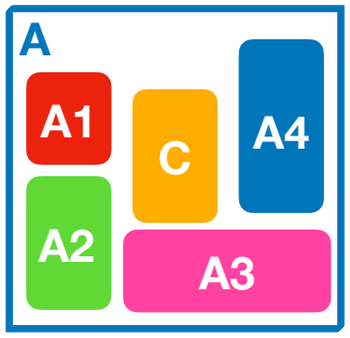
Global constraints

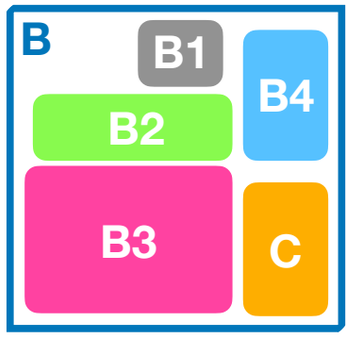
Scheduling instances



Vehicule routing instances





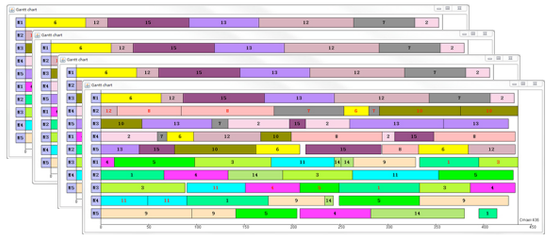


`Cumulative`

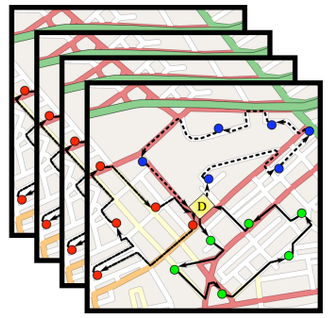
Constraint Programming

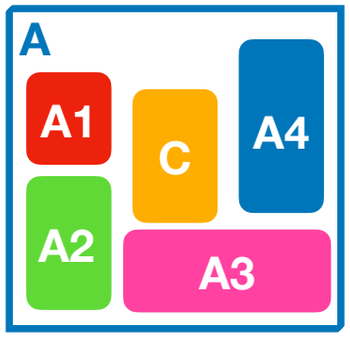
Global constraints

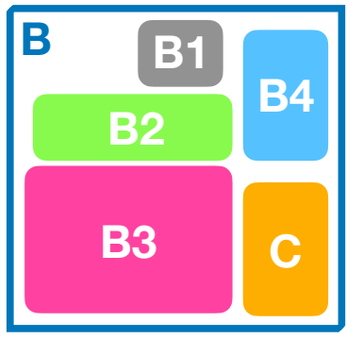
Scheduling instances



Vehicule routing instances





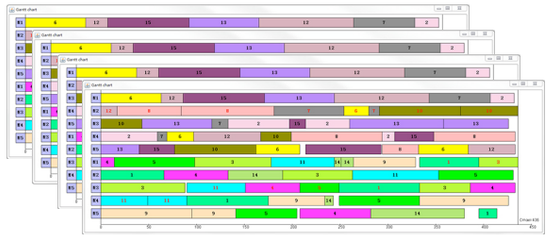



```
Cumulative
Alldifferent
```

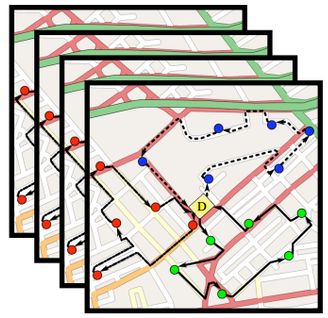
Constraint Programming

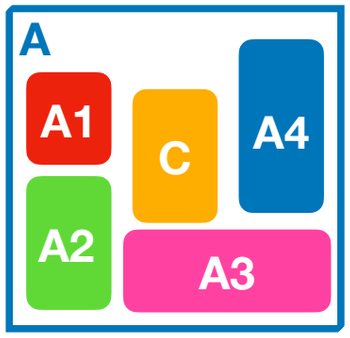
Global constraints

Scheduling instances

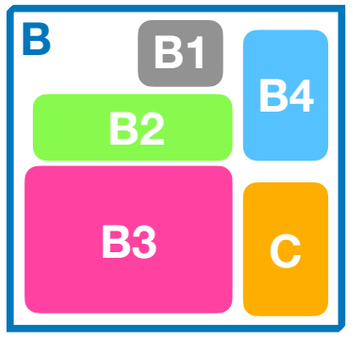


Vehicule routing instances





CP Solver



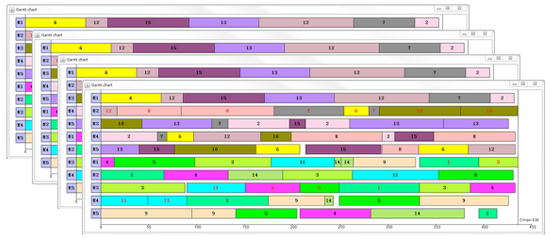

```
Cumulative
Alldifferent
```

•
•
•

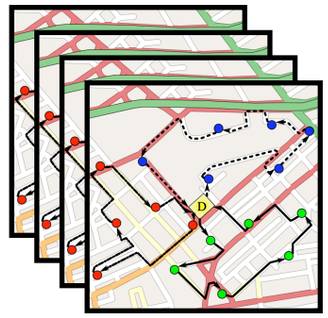
Constraint Programming

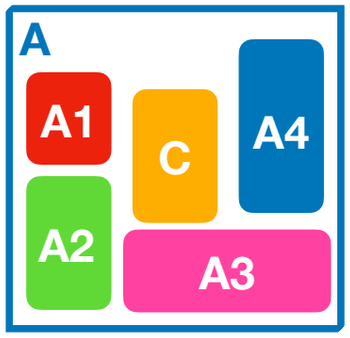
Global constraints

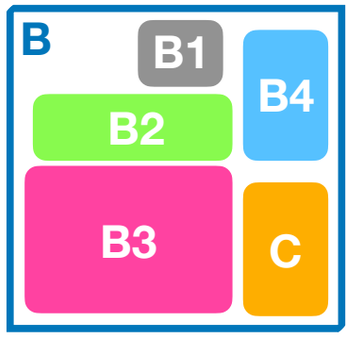
Scheduling instances



Vehicule routing instances







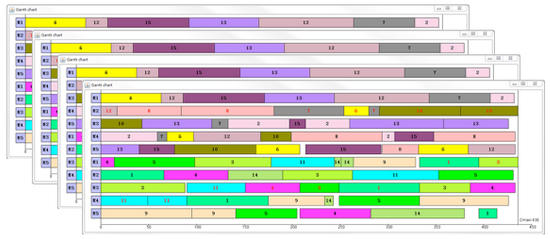

```
Cumulative
Alldifferent
Sum
knapsack
Element
GlobalCardinality
Regular
…
```

-
-
-

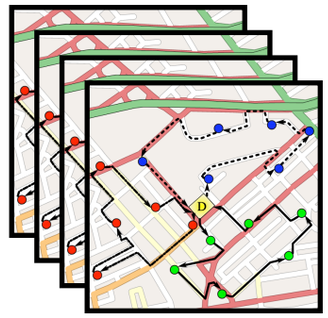
Constraint Programming

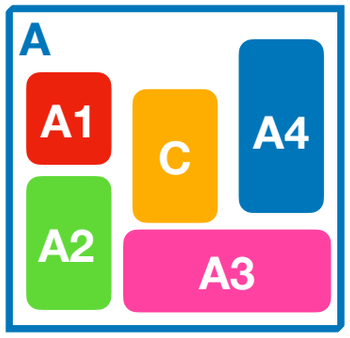
Global constraints

Scheduling instances



Vehicule routing instances

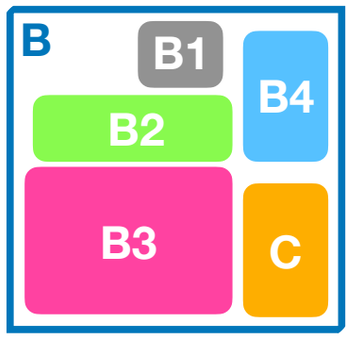




CP Solver

toolbox

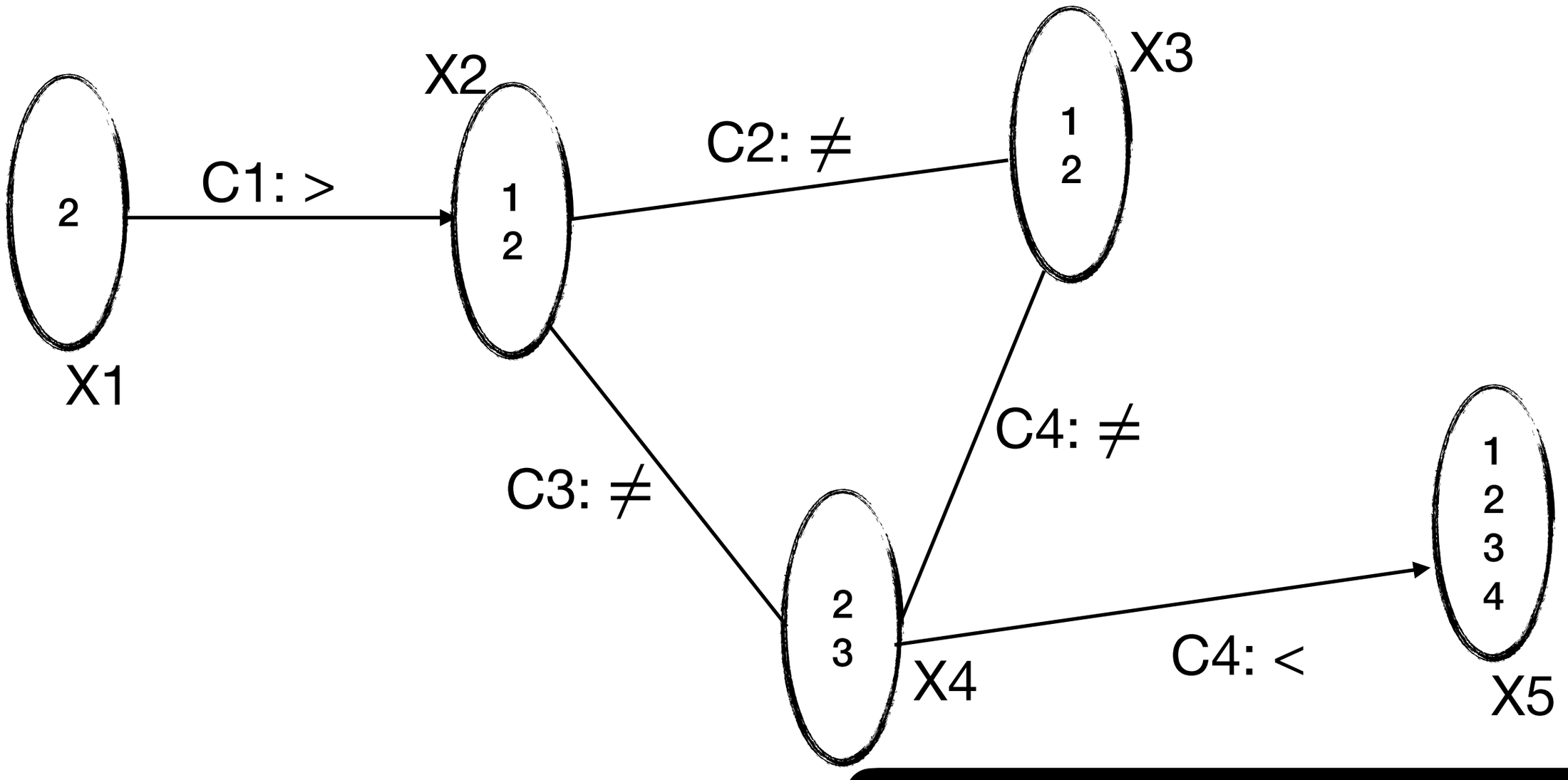
```
Cumulative
Alldifferent
Sum
knapsack
Element
GlobalCardinality
Regular
…
```



•
•
•

Résolution en PPC

Propagation (Consistance Locale)



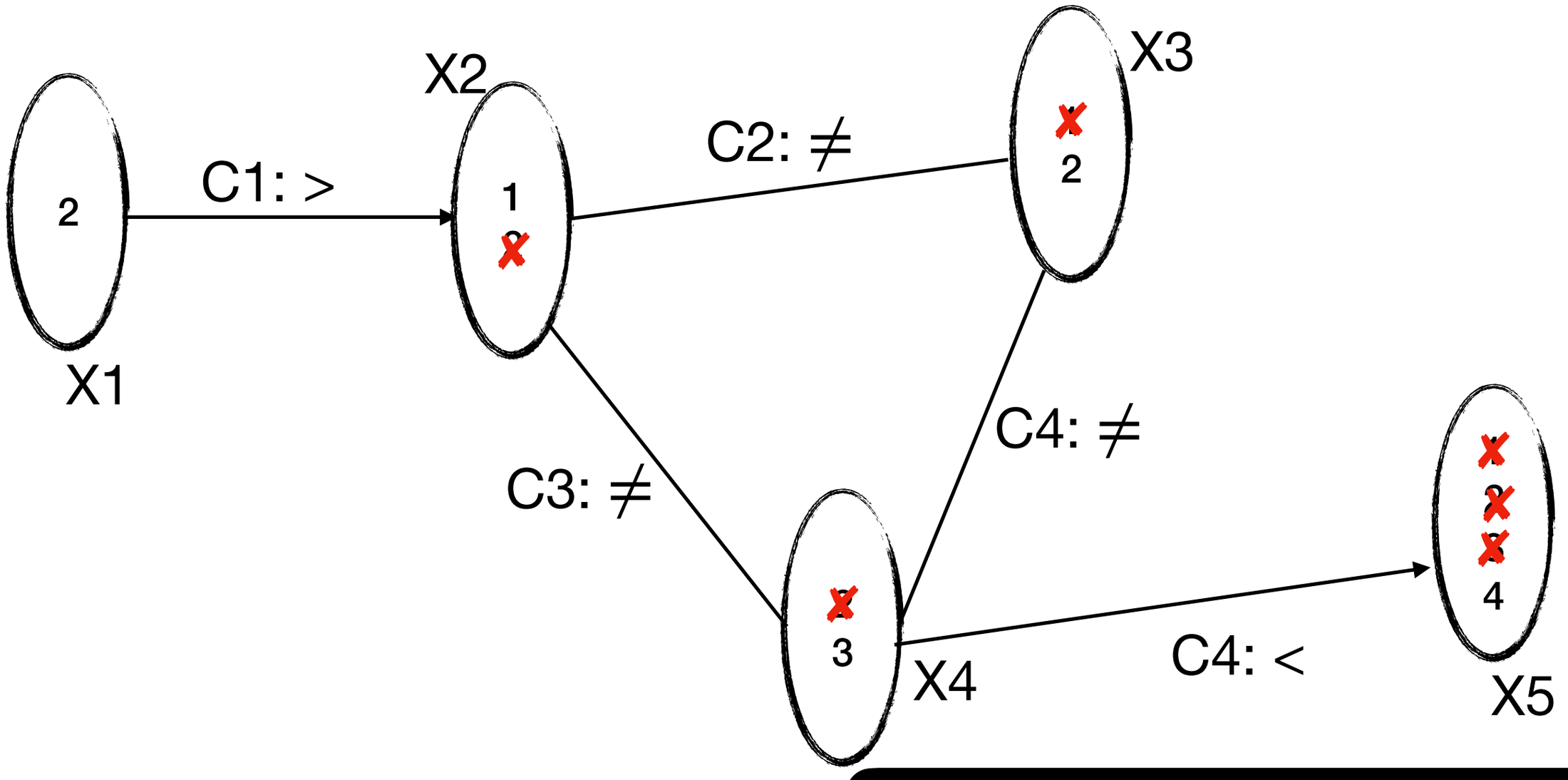
Q1: Taille de l'espace de recherche ?

Q2: Quelles sont les valeurs incohérentes ici ?

Situation 1

Résolution en PPC

Propagation (Consistance Locale)



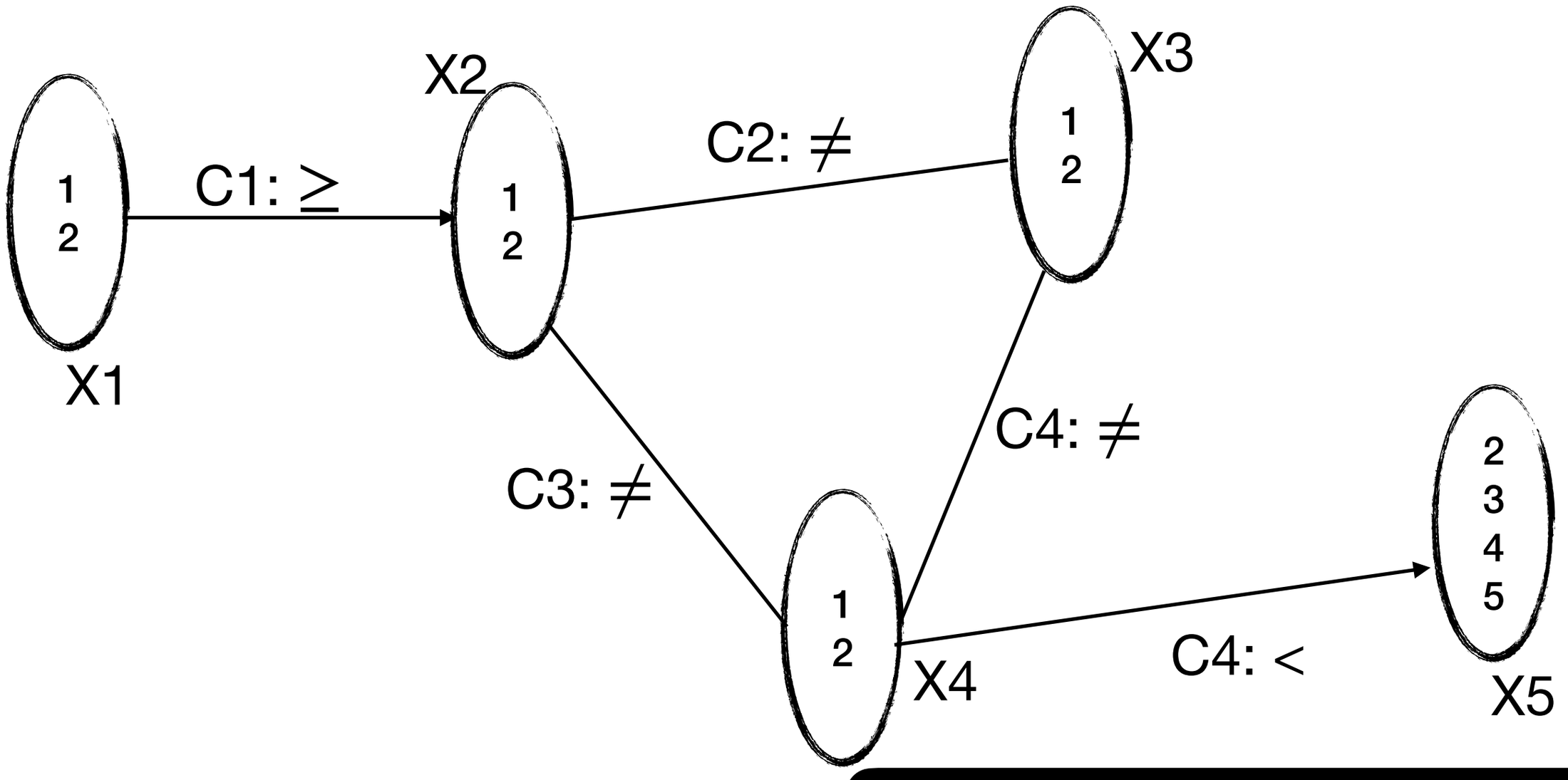
Q1: Taille de l'espace de recherche ?

Q2: Quelles sont les valeurs incohérentes ici ?

Situation 1 (Solution)

Résolution en PPC

Propagation (Consistance Locale)



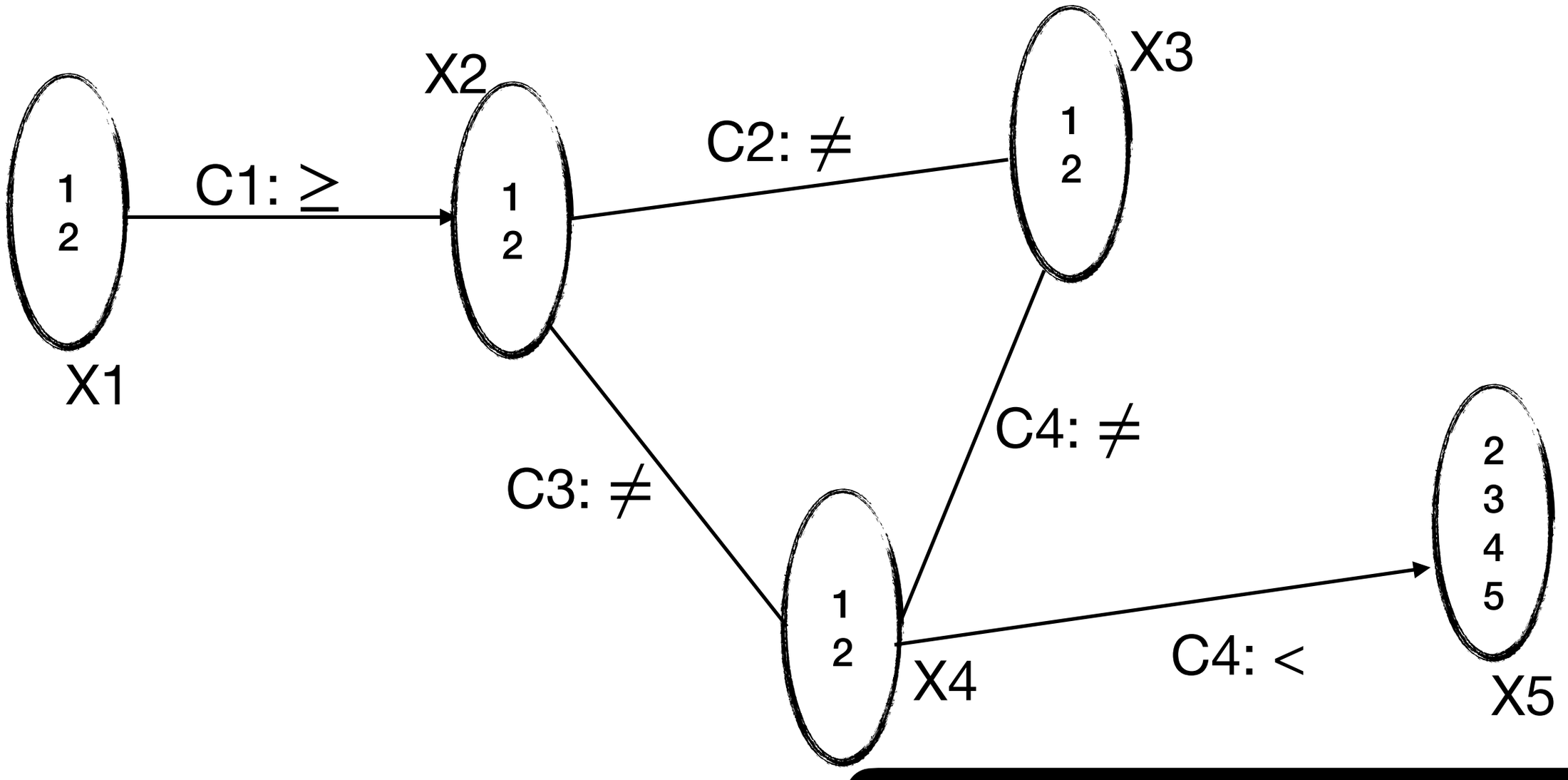
Q1: Taille de l'espace de recherche ?

Q2: Quelles sont les valeurs incohérentes ici ?

Situation 2

Résolution en PPC

Propagation (Consistance Locale)



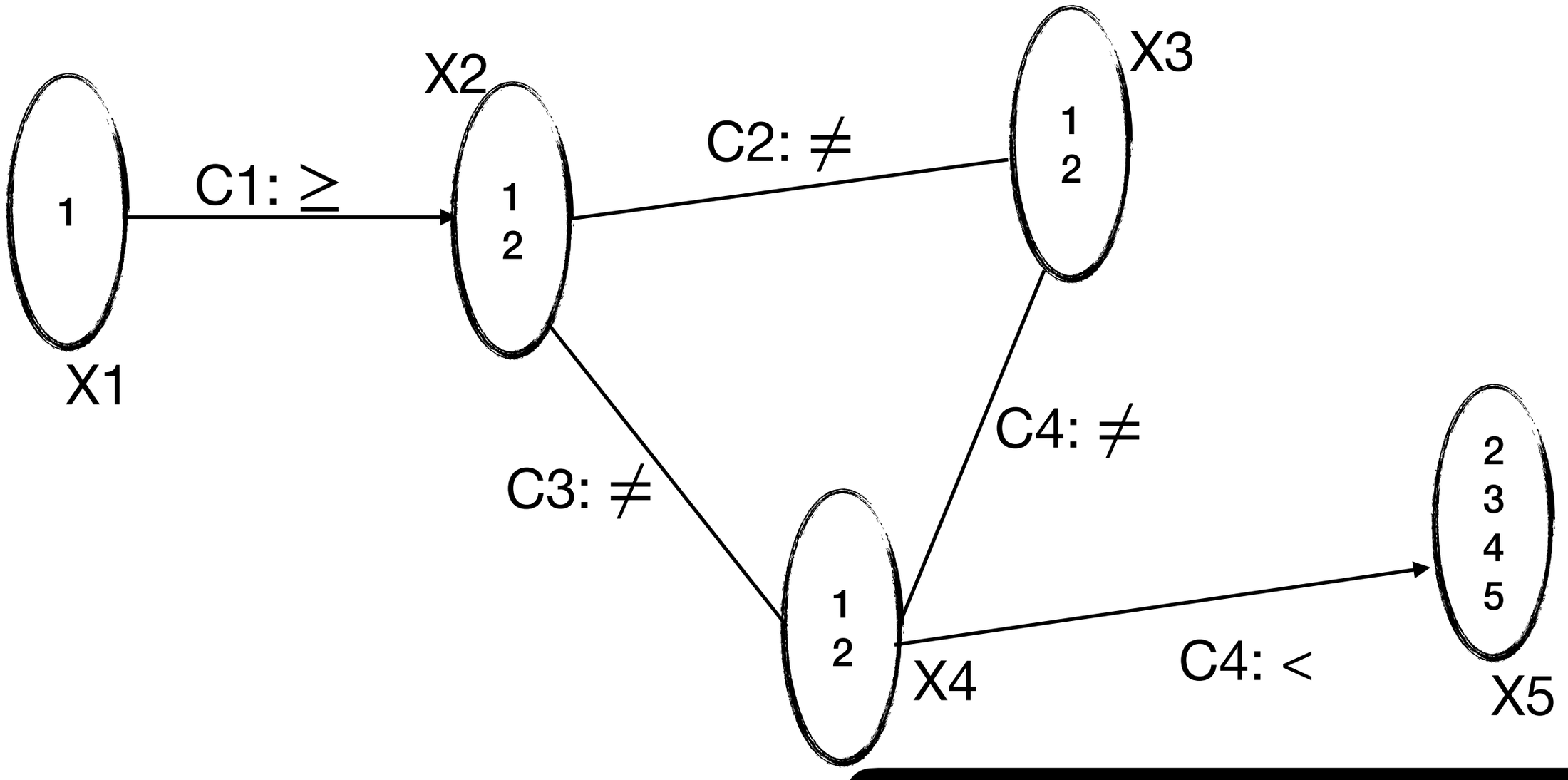
Q1: Taille de l'espace de recherche ?

Q2: Quelles sont les valeurs incohérentes ici ?

Situation 2 (No change)

Résolution en PPC

Propagation (Consistance Locale)



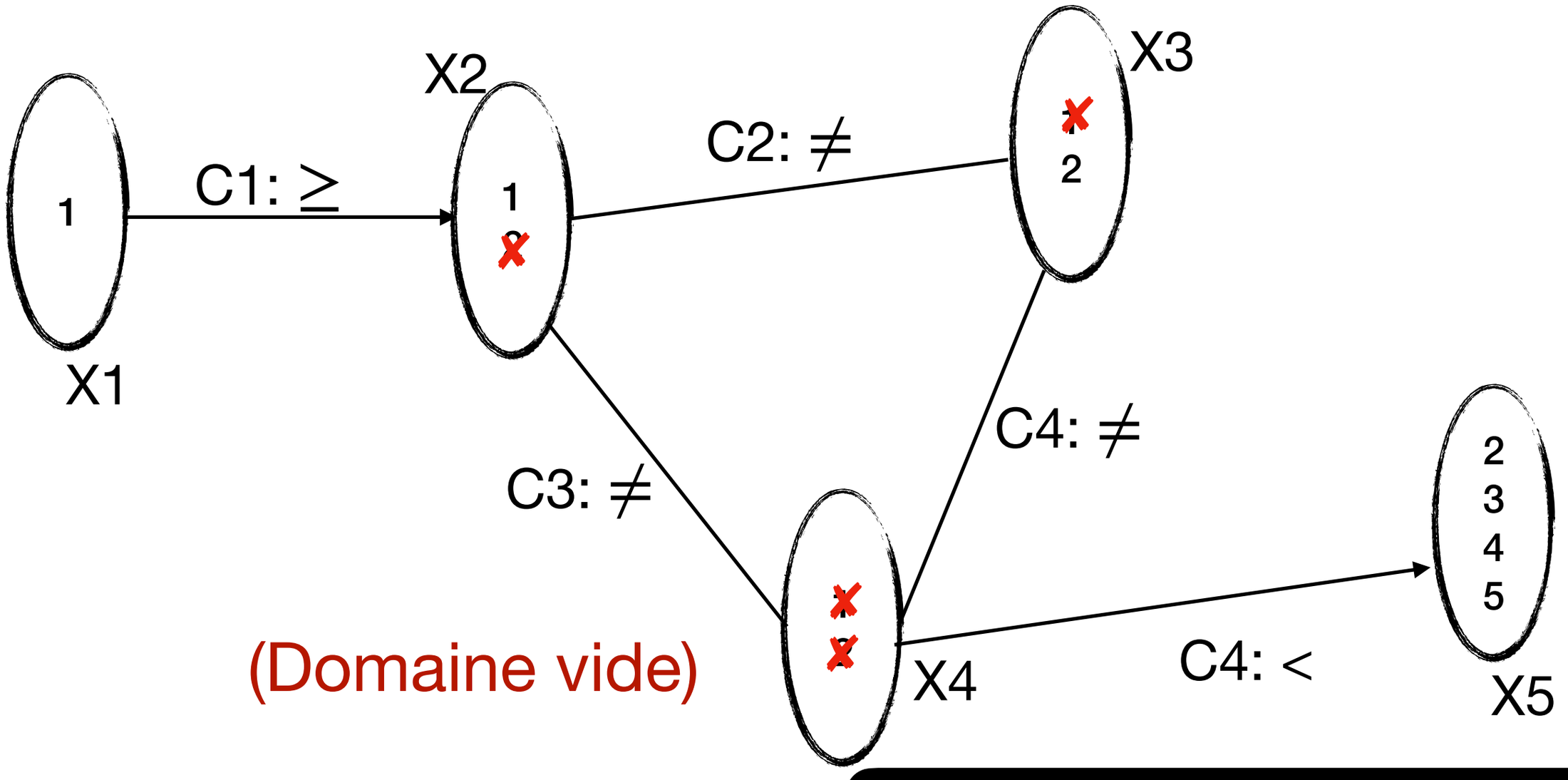
Q1: Taille de l'espace de recherche ?

Q2: Quelles sont les valeurs incohérentes ici ?

Situation 3

Résolution en PPC

Propagation (Consistance Locale)



(Domaine vide)

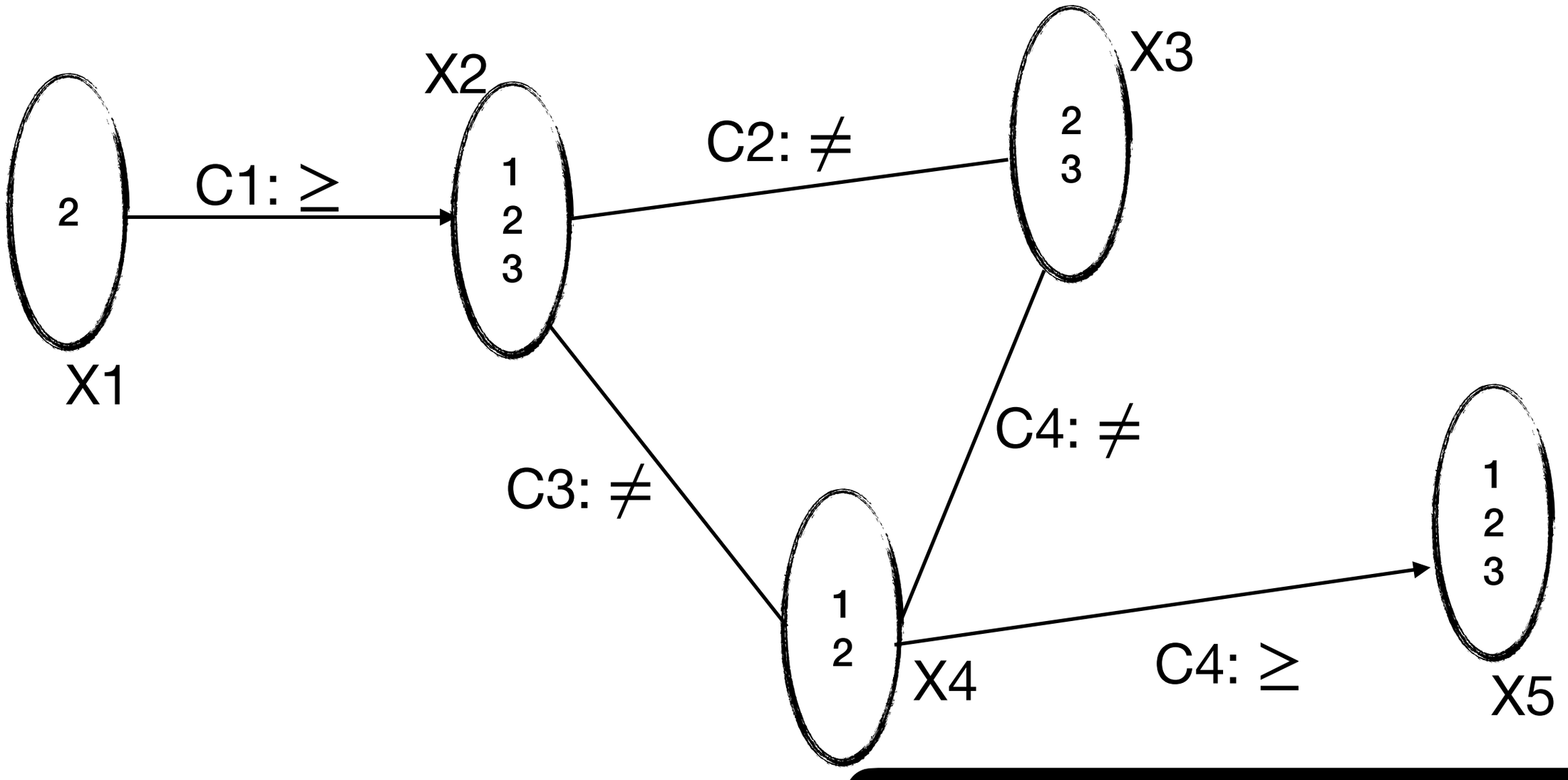
Q1: Taille de l'espace de recherche ?

Q2: Quelles sont les valeurs incohérentes ici ?

Situation 3

Résolution en PPC

Propagation (Consistance Locale)



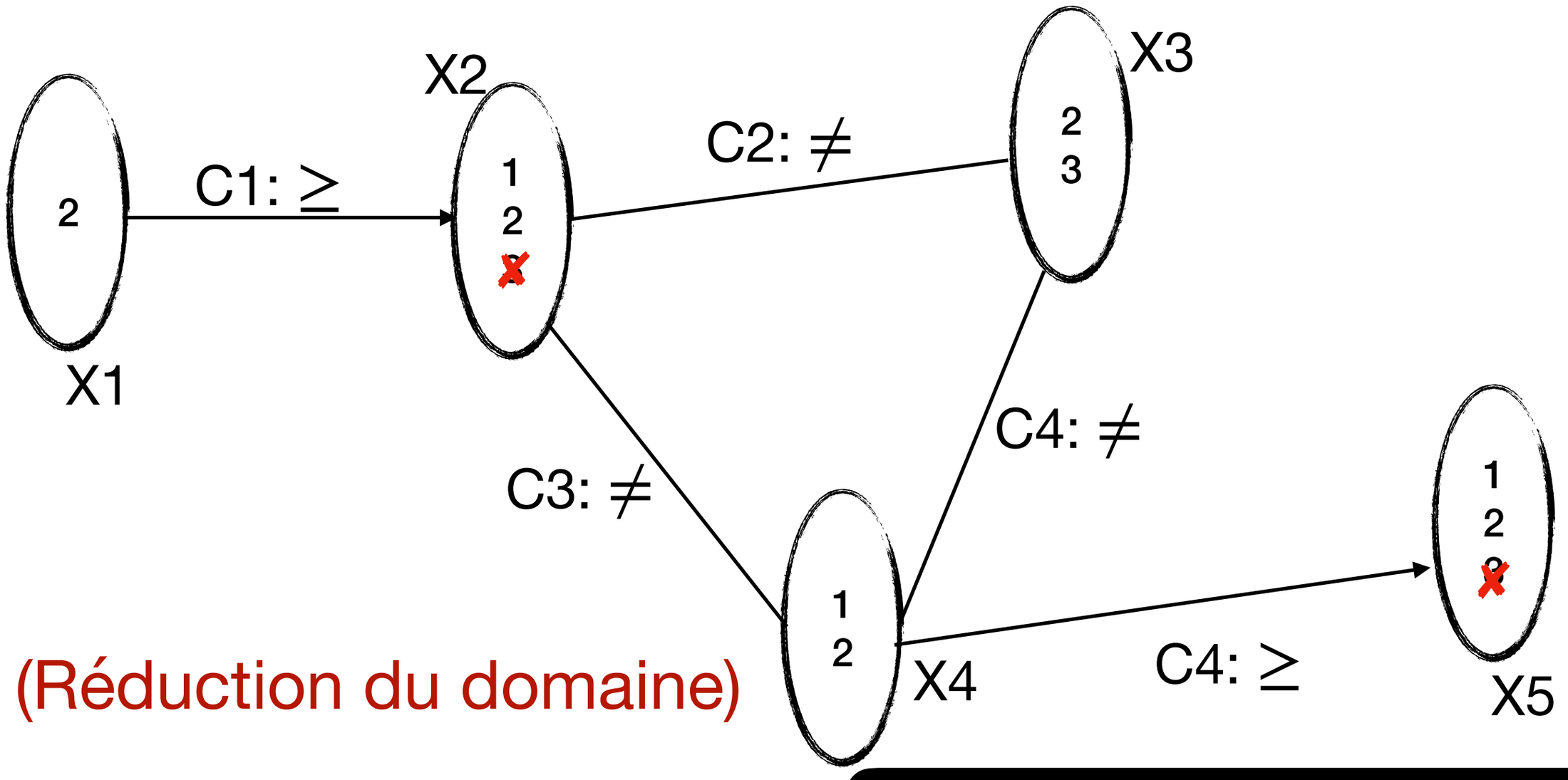
Q1: Taille de l'espace de recherche ?

Q2: Quelles sont les valeurs incohérentes ici ?

Situation 4

Résolution en PPC

Propagation (Consistance Locale)



(Réduction du domaine)

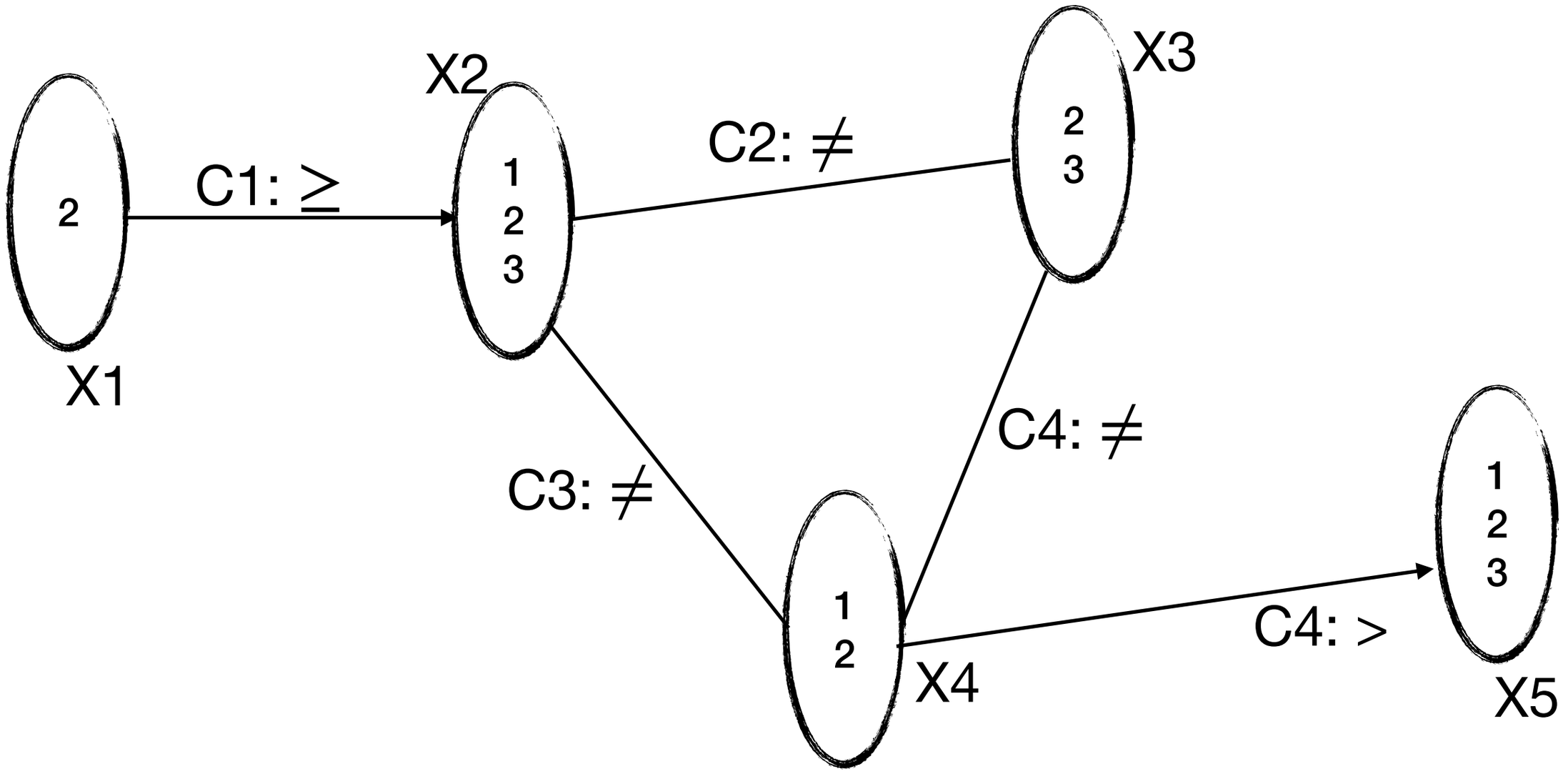
Q1: Taille de l'espace de recherche ?

Q2: Quelles sont les valeurs incohérentes ici ?

Situation 4

Résolution en PPC

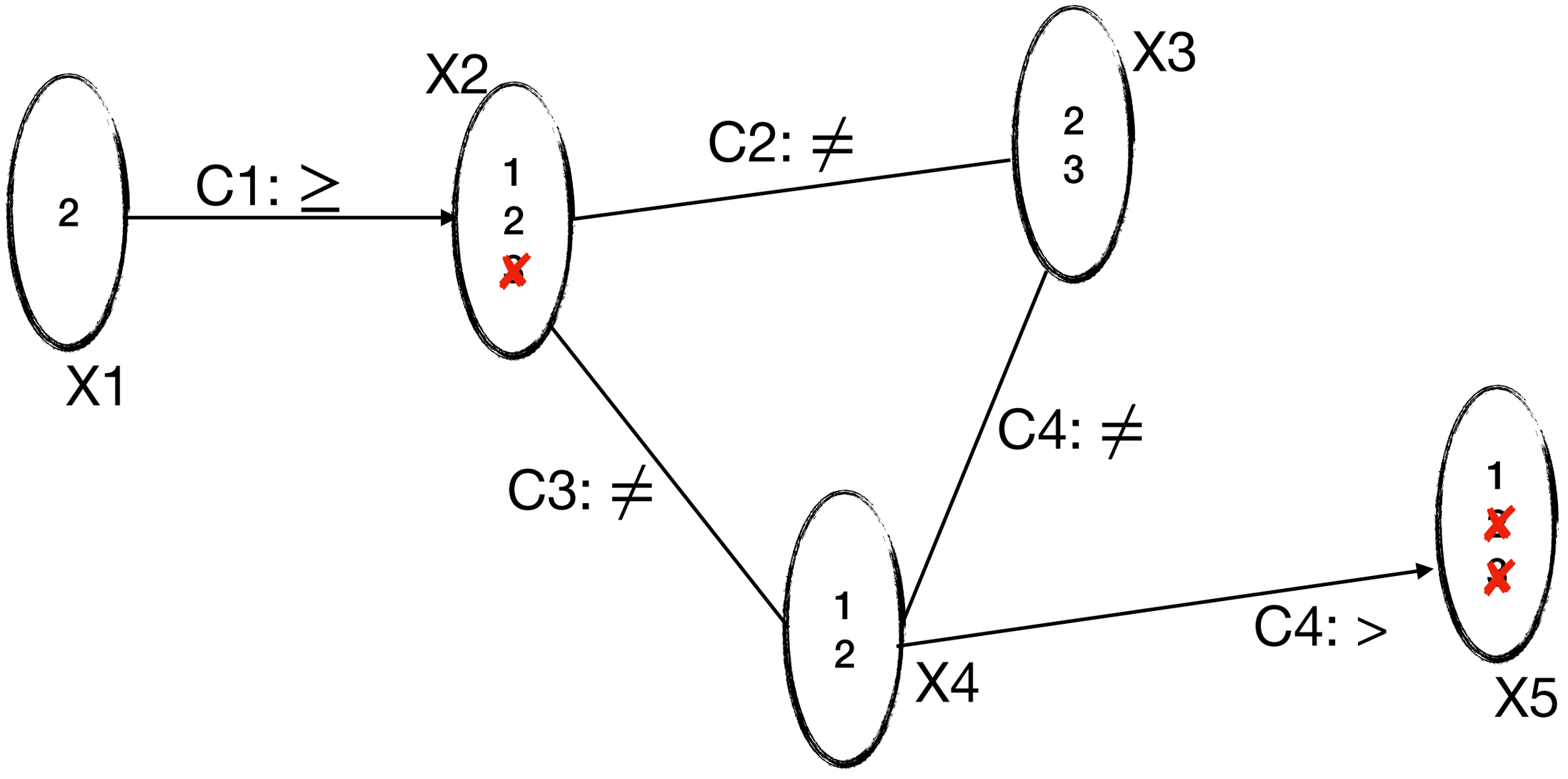
Propagation (Consistance Locale)



Situation 5

Résolution en PPC

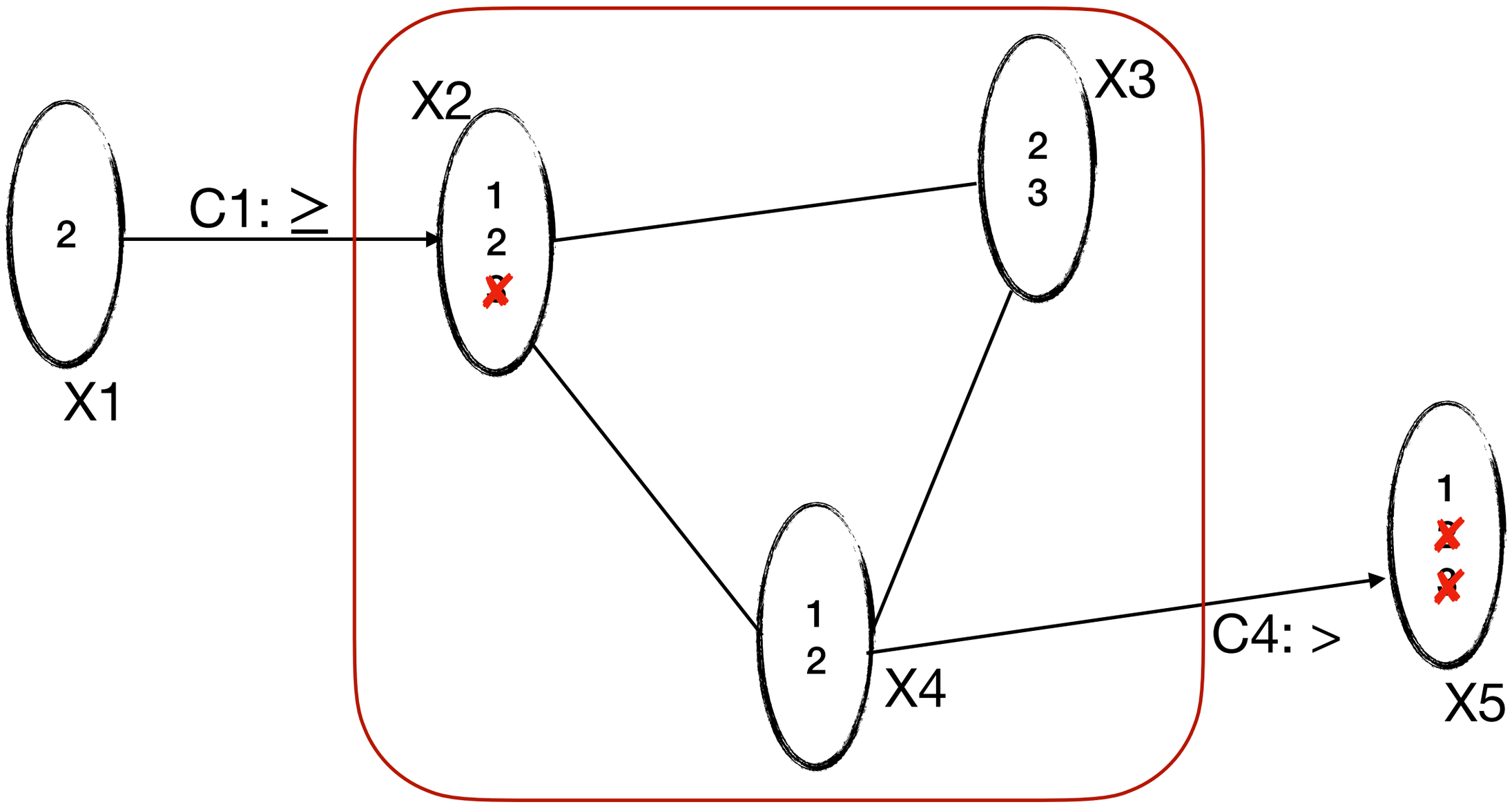
Propagation (Consistance Locale)



Situation 5

Résolution en PPC

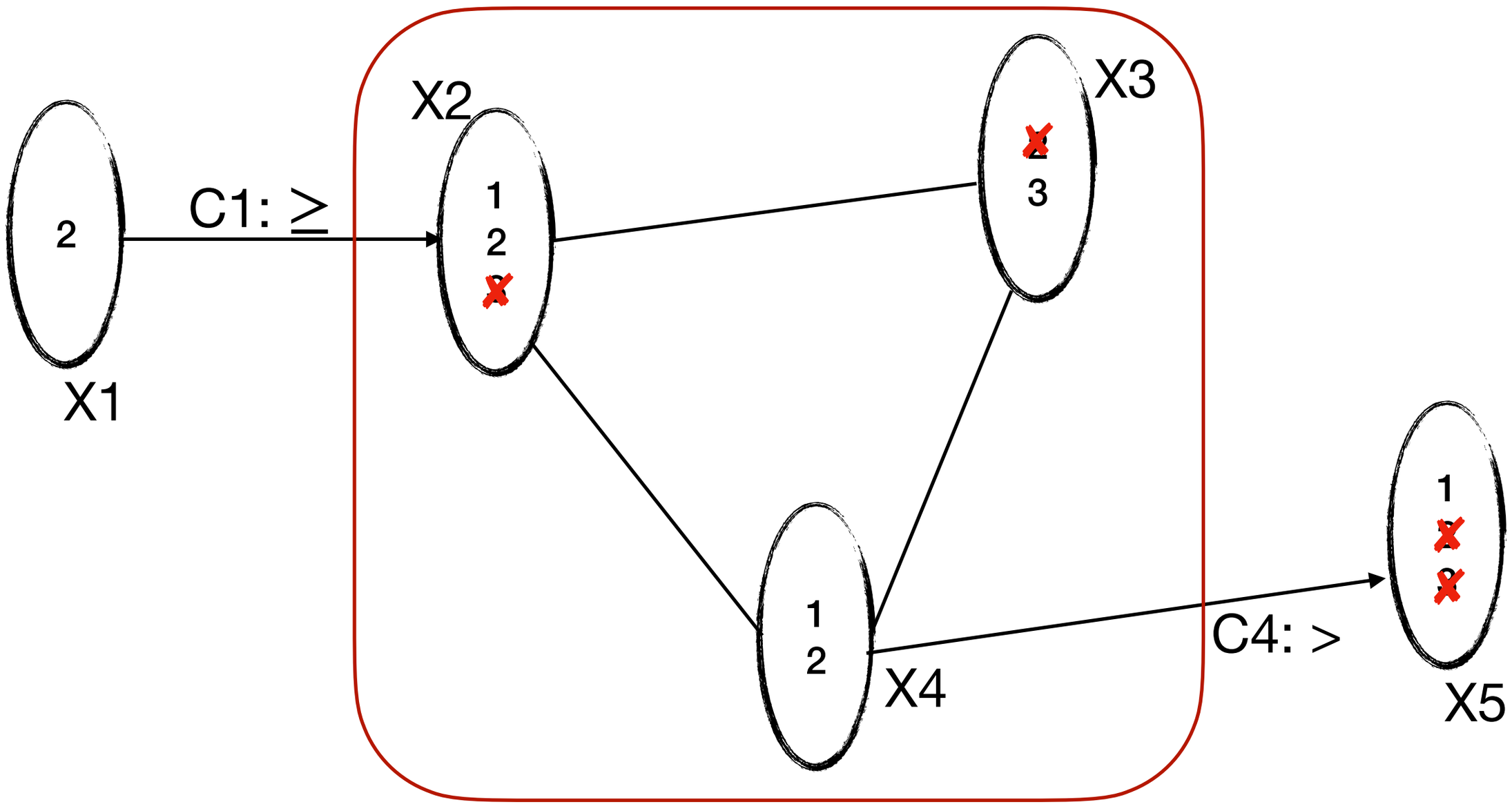
Propagation (Consistance Locale)



Situation 5 (Contraintes Globales)

Résolution en PPC

Propagation (Consistance Locale)



Situation 5 (Contraintes Globales)

Constraint Programming

Résolution

Instantiation I :

- **Complète :** affectation sur X
- **Partielle :** affectation sur $Y \subset X$
- **Valide :** valeurs dans D
- **Violation de c :** l'affectation sur $var(c)$ ne respecte pas c
- **Localement cohérente :** l'affectation ne viole aucune contrainte sur Y
- **Solution :** localement cohérente sur X

Résolution en PPC

Backtraking (BT)

- Un algorithme de recherche généraliste
- Explore systématiquement l’espace de recherche
- Utilise une stratégie de recherche en profondeur (depth-first search)
- Peut rencontrer des inefficacités sur de grands espaces de recherche
- Convient à une large gamme de problèmes de contraintes
- Peut nécessiter des techniques de réduction supplémentaires

Résolution en PPC

Backtraking (BT)

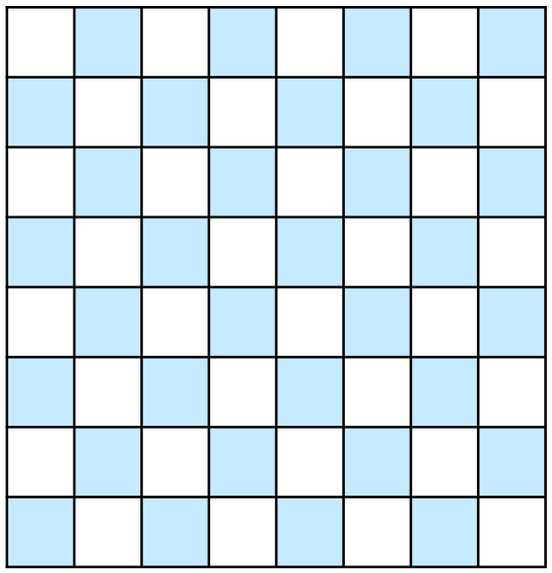
Résolution en PPC

Backtraking (BT)

```
BT(<X,D,C>, I):

If I is complete then return true

Select a variable Xi not in I

ForEach v in D(Xi) do

   If I union <Xi, v> is locally consistent then

      If BT(<X,D,C>, I union <Xi, v>) then
          return true

return false
```

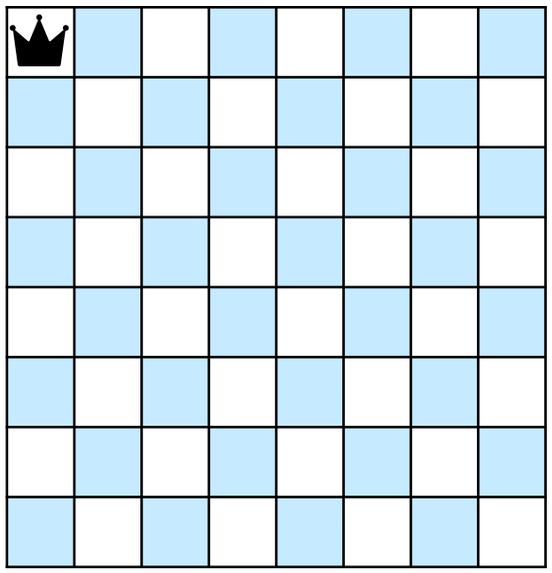
Modélisation <-> Résolution

N-Reines



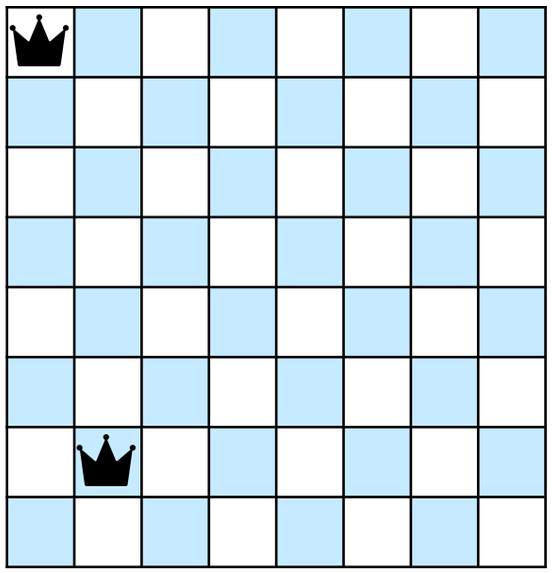
Modélisation <-> Résolution

N-Reines



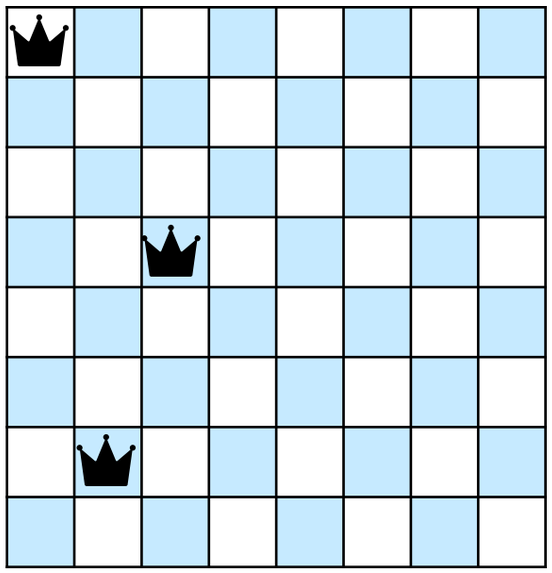
Modélisation <-> Résolution

N-Reines



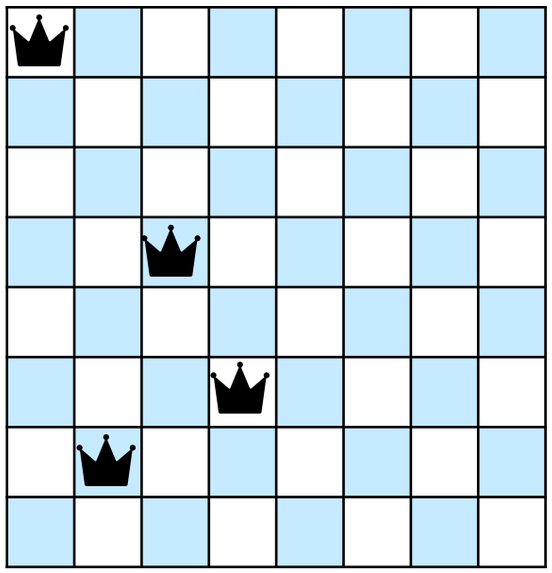
Modélisation <-> Résolution

N-Reines



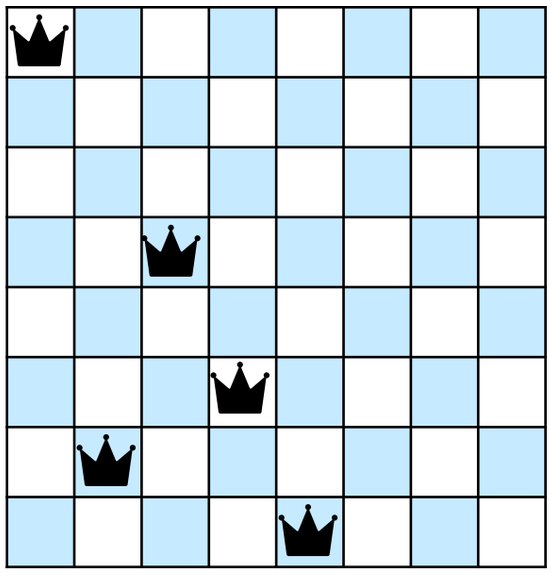
Modélisation <-> Résolution

N-Reines



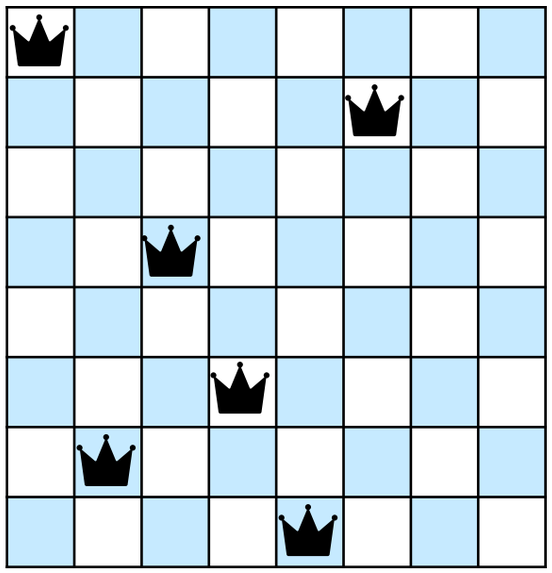
Modélisation <-> Résolution

N-Reines



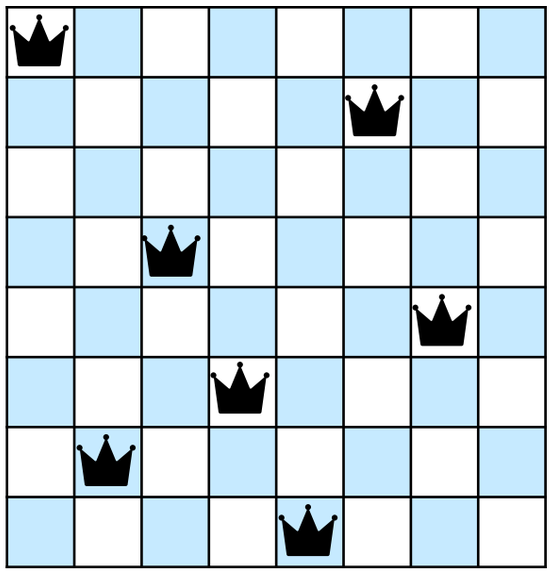
Modélisation <-> Résolution

N-Reines



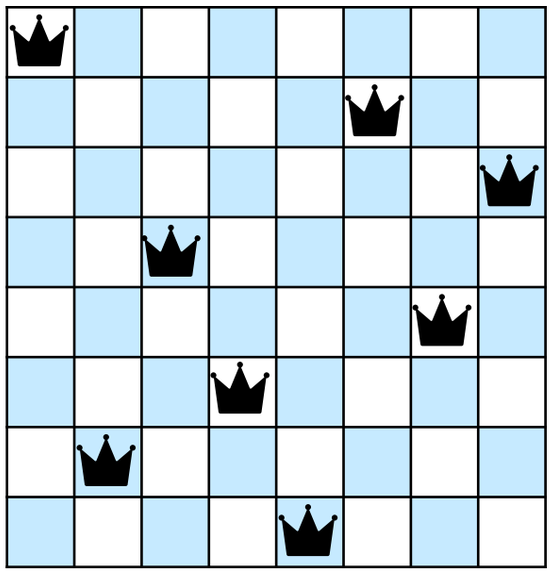
Modélisation <-> Résolution

N-Reines



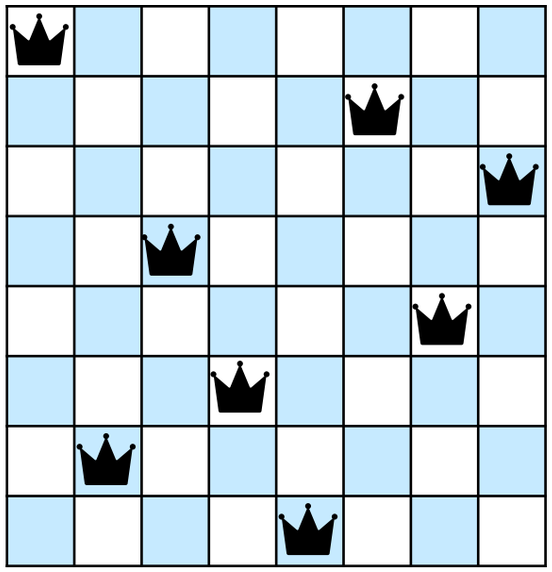
Modélisation <-> Résolution

N-Reines



Modélisation <-> Résolution

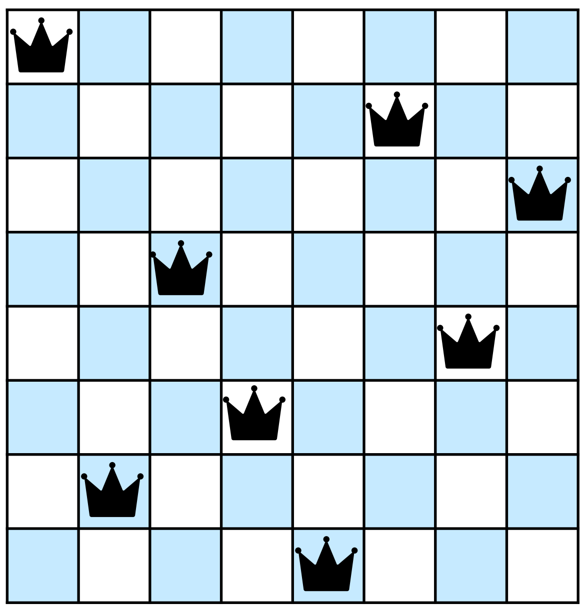
N-Reines



1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0

Modélisation <-> Résolution

N-Reines

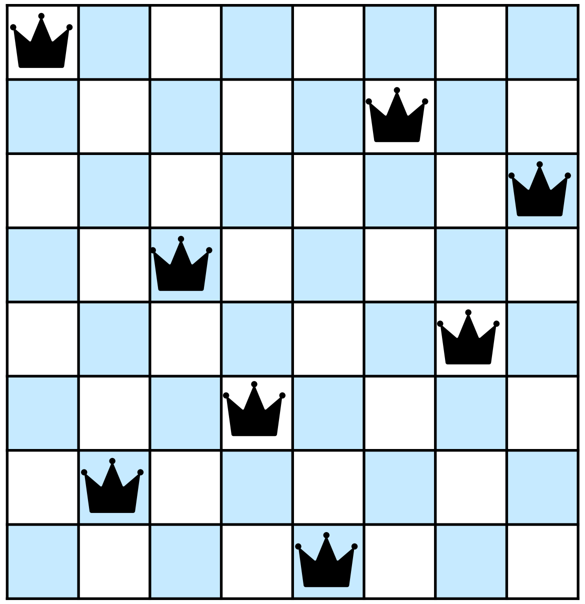


1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0

0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

Modélisation <-> Résolution

N-Reines



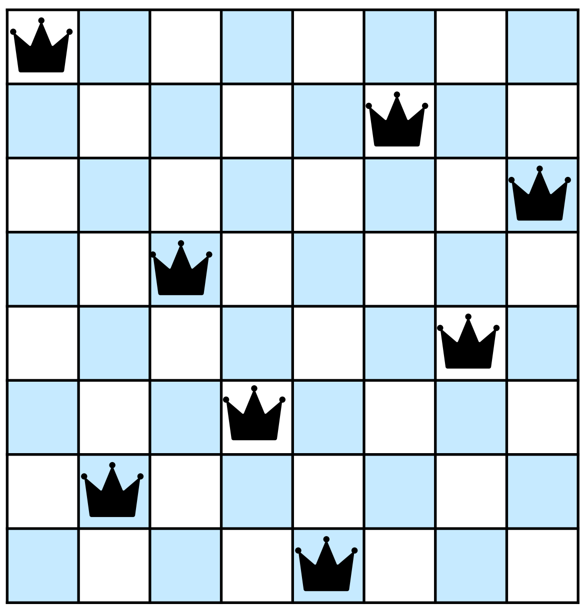
1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0

0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

$$X = \{x_{1,1}, \ldots, x_{n,n}\}$$
$$D(x_{i,j}) \in \{0,1\}$$

Modélisation <-> Résolution

N-Reines



1	7	4	6	8	2	5	3

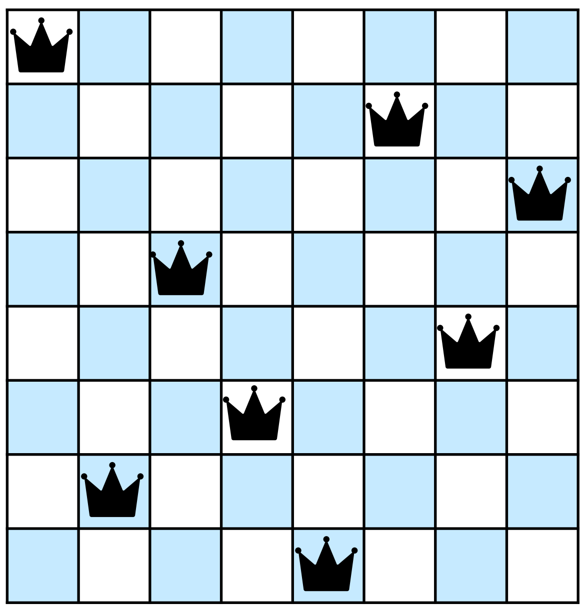
1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0

0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

$$X = \{x_{1,1}, \ldots, x_{n,n}\}$$
$$D(x_{i,j}) \in \{0,1\}$$

Modélisation <-> Résolution

N-Reines



1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0

0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

1	7	4	6	8	2	5	3

$$X = \{x_{1,1}, \ldots, x_{n,n}\}$$
$$D(x_{i,j}) \in \{0,1\}$$

$$X = \{x_1, \ldots, x_n\}$$
$$D(x_i) \in 1..n$$

Modélisation <-> Résolution

N-Reines

$X = \{x_1, \ldots, x_n\}$
$D(x_i) \in 1..n$

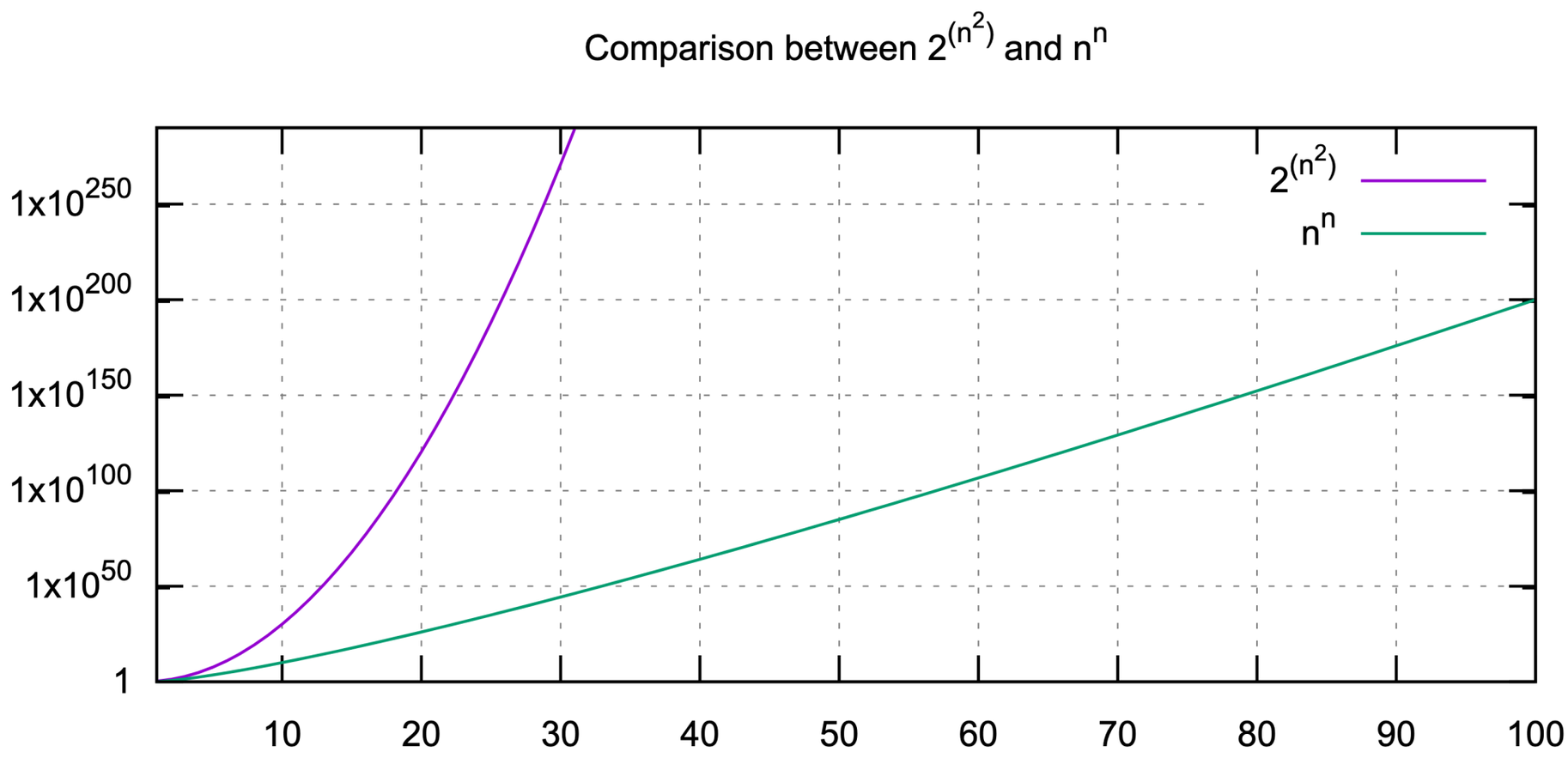
$X = \{x_{1,1}, \ldots, x_{n,n}\}$
$D(x_{i,j}) \in \{0,1\}$

Modélisation <-> Résolution

N-Reines

$X = \{x_1, \ldots, x_n\}$
$D(x_i) \in 1..n$

$X = \{x_{1,1}, \ldots, x_{n,n}\}$
$D(x_{i,j}) \in \{0,1\}$



Résolution en PPC

Backtraking (BT)

- Un algorithme de recherche à usage général
- Explore systématiquement l’espace de recherche
- Utilise une stratégie de recherche en profondeur
- Peut rencontrer des inefficacités sur de grands espaces de recherche
- Adapté à un large éventail de problèmes de contraintes
- Peut nécessiter des techniques de taille supplémentaires

Résolution en PPC

Backtraking (BT)

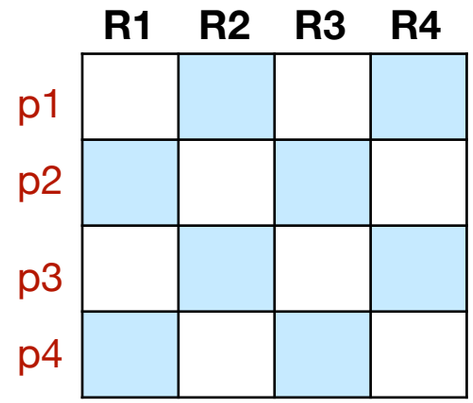
Résolution en PPC

Backtraking (BT)

```
BT(<X,D,C>, I):

If I is complete then return true

Select a variable Xi not in I

ForEach v in D(Xi) do

   If I union <Xi, v> is locally consistent then

      If BT(<X,D,C>, I union <Xi, v>) then
          return true

return false
```

Résolution en PPC

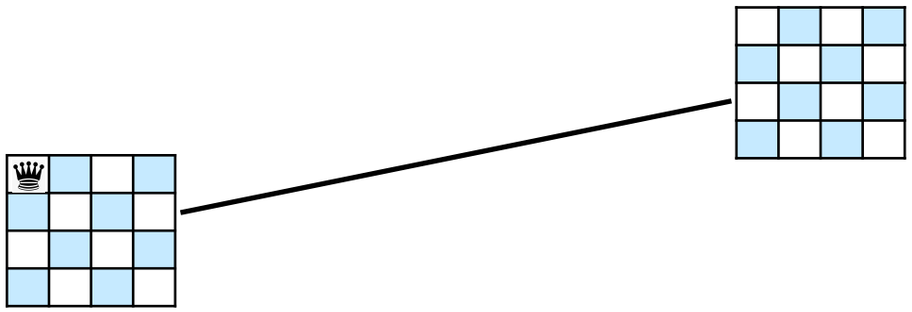
Backtraking (BT)

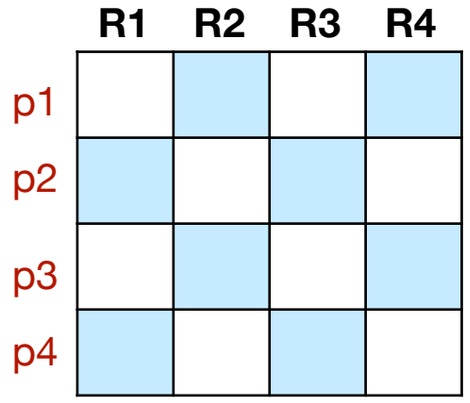


N-queens problem

Résolution en PPC

Backtraking (BT)

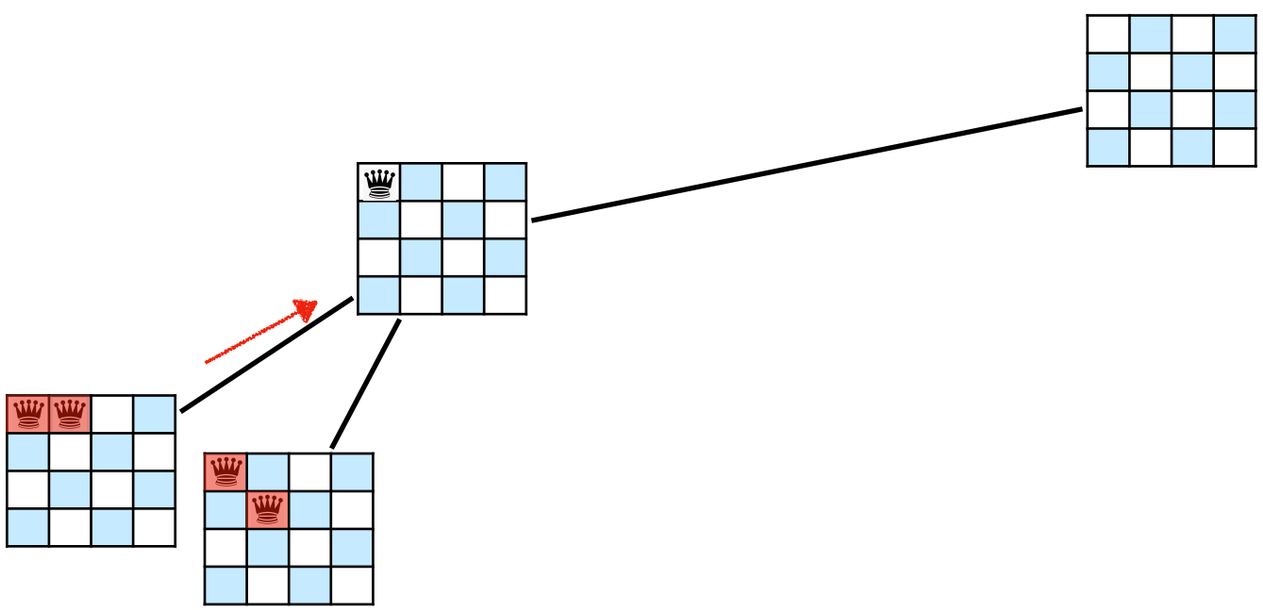


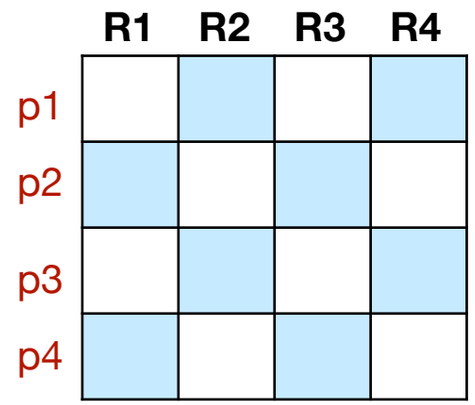


N-queens problem

Résolution en PPC

Backtraking (BT)

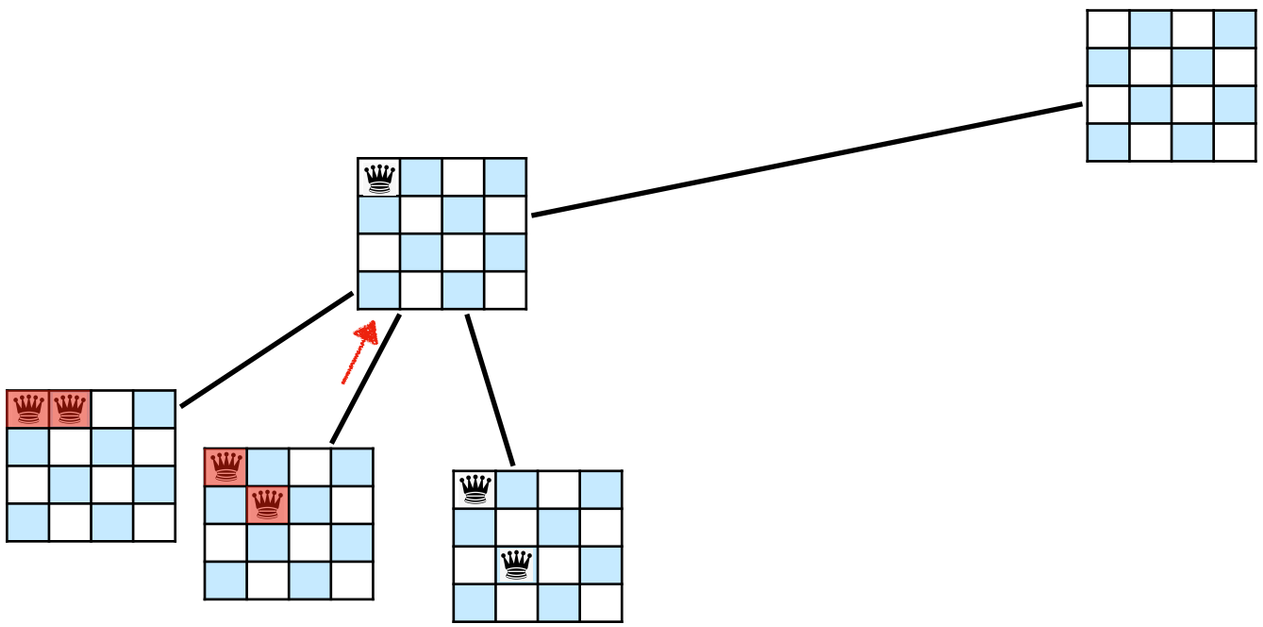


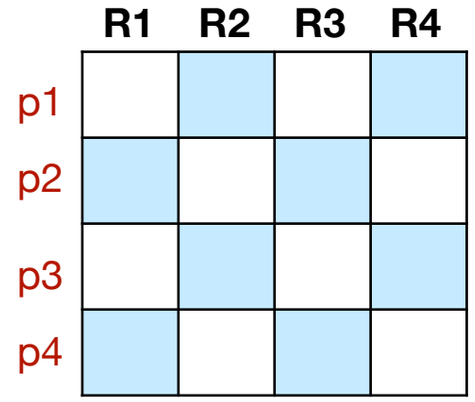


N-queens problem

Résolution en PPC

Backtraking (BT)

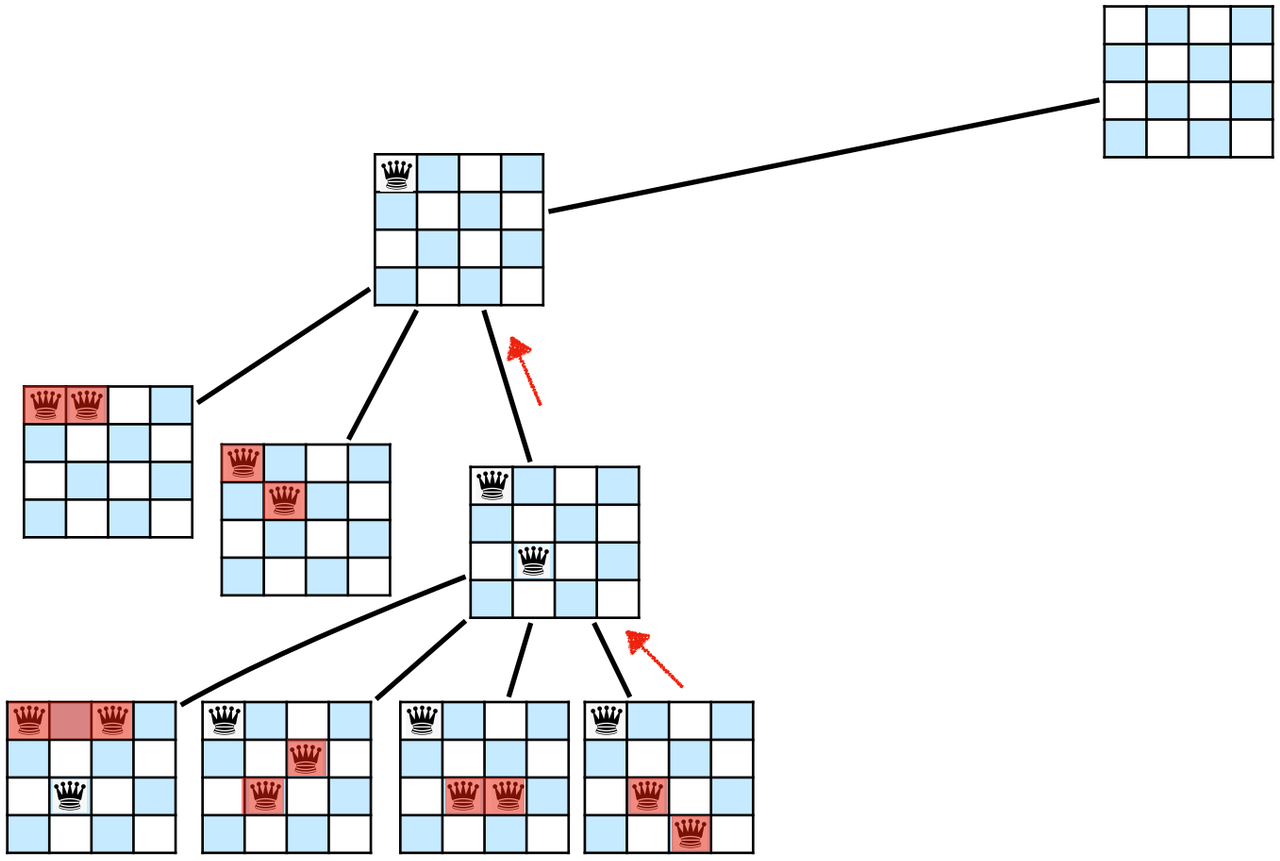


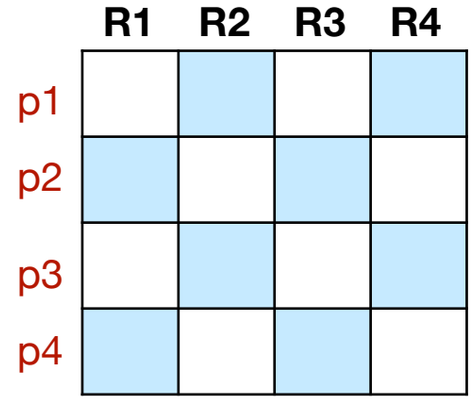


N-queens problem

Résolution en PPC

Backtraking (BT)

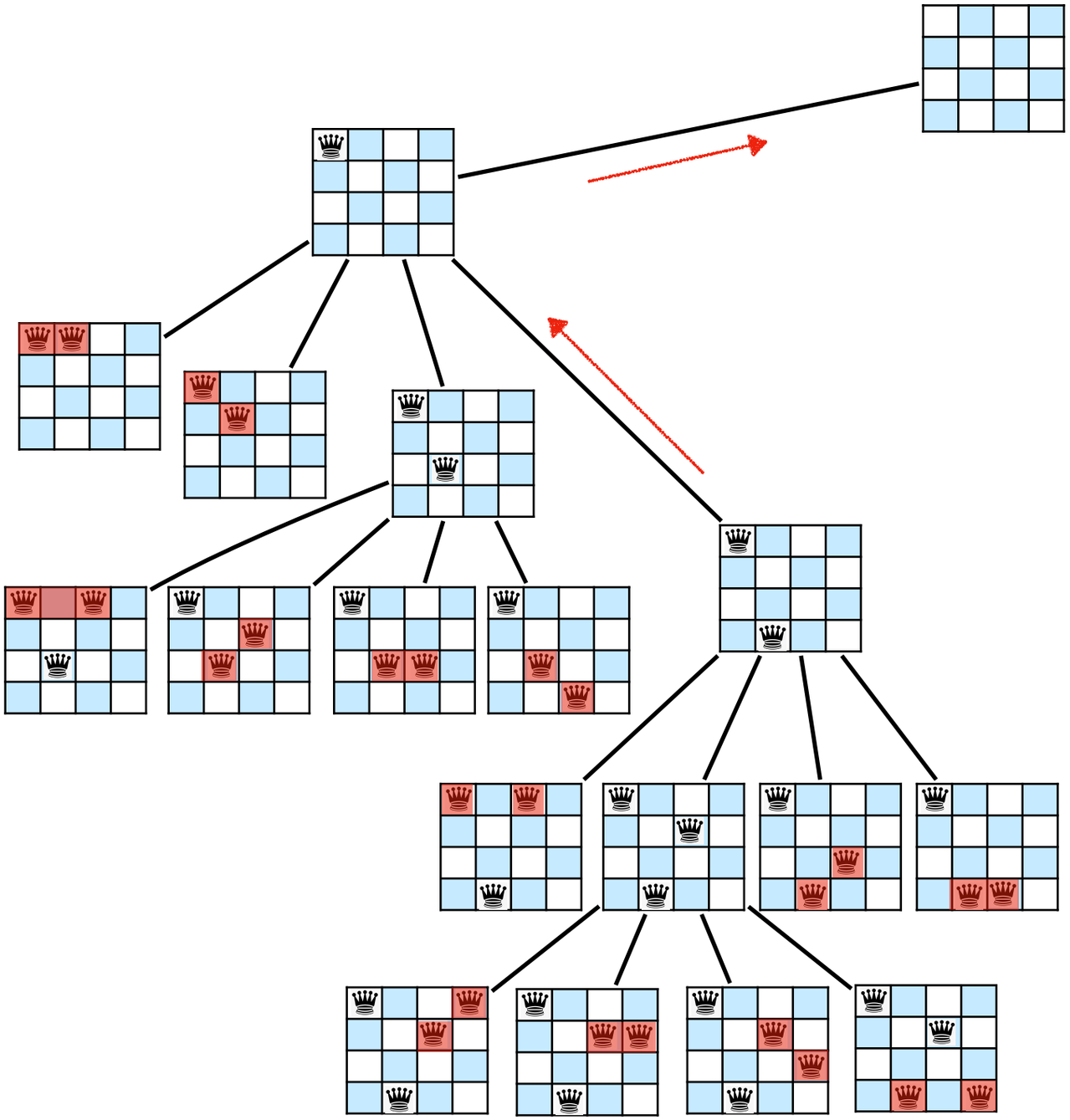


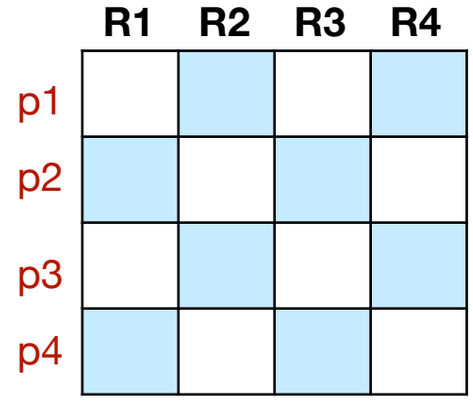


N-queens problem

Résolution en PPC

Backtraking (BT)

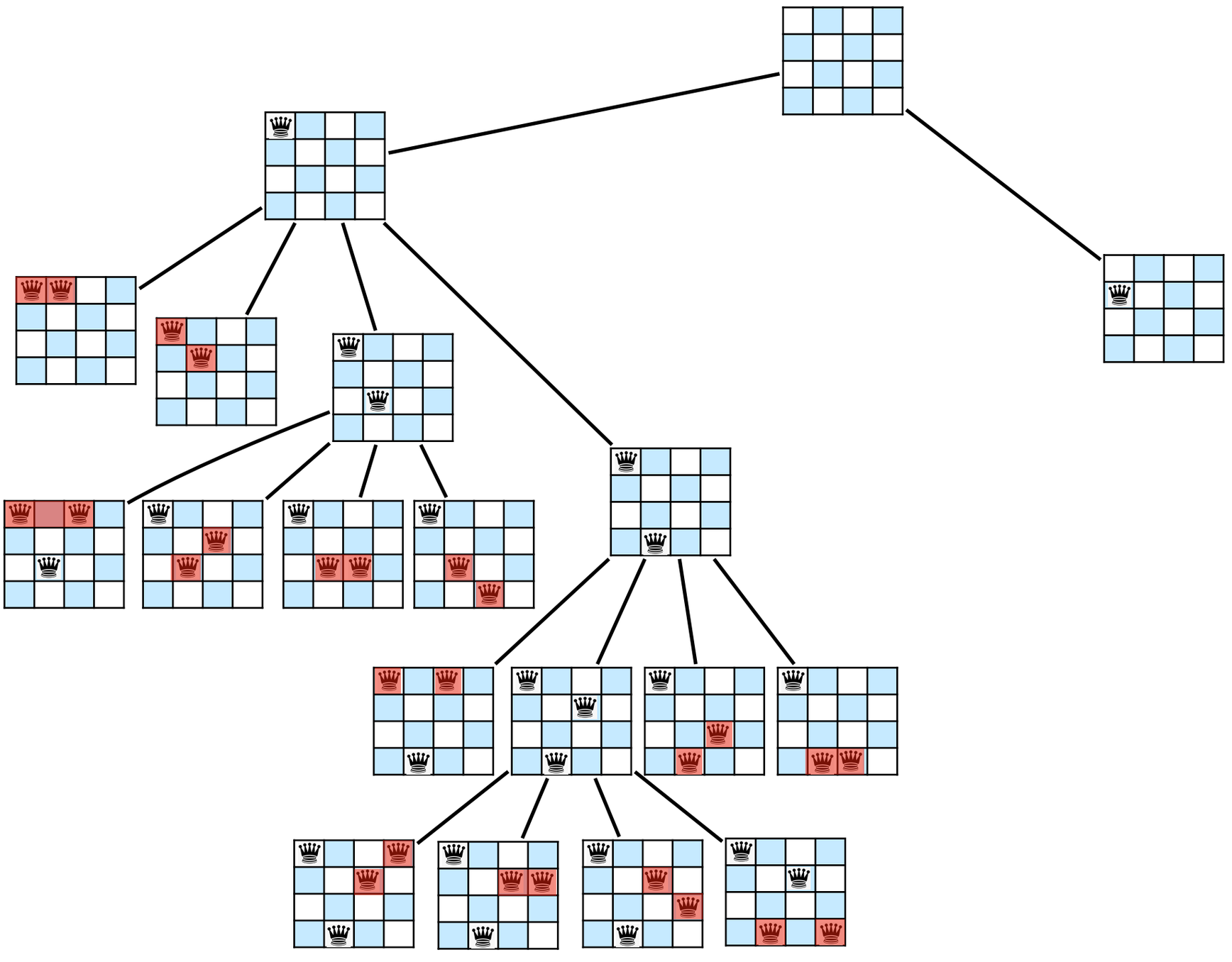


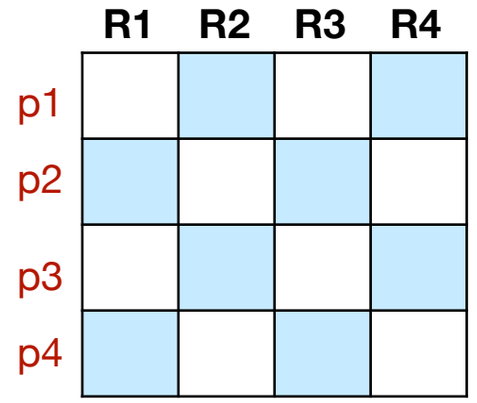


N-queens problem

Résolution en PPC

Backtraking (BT)

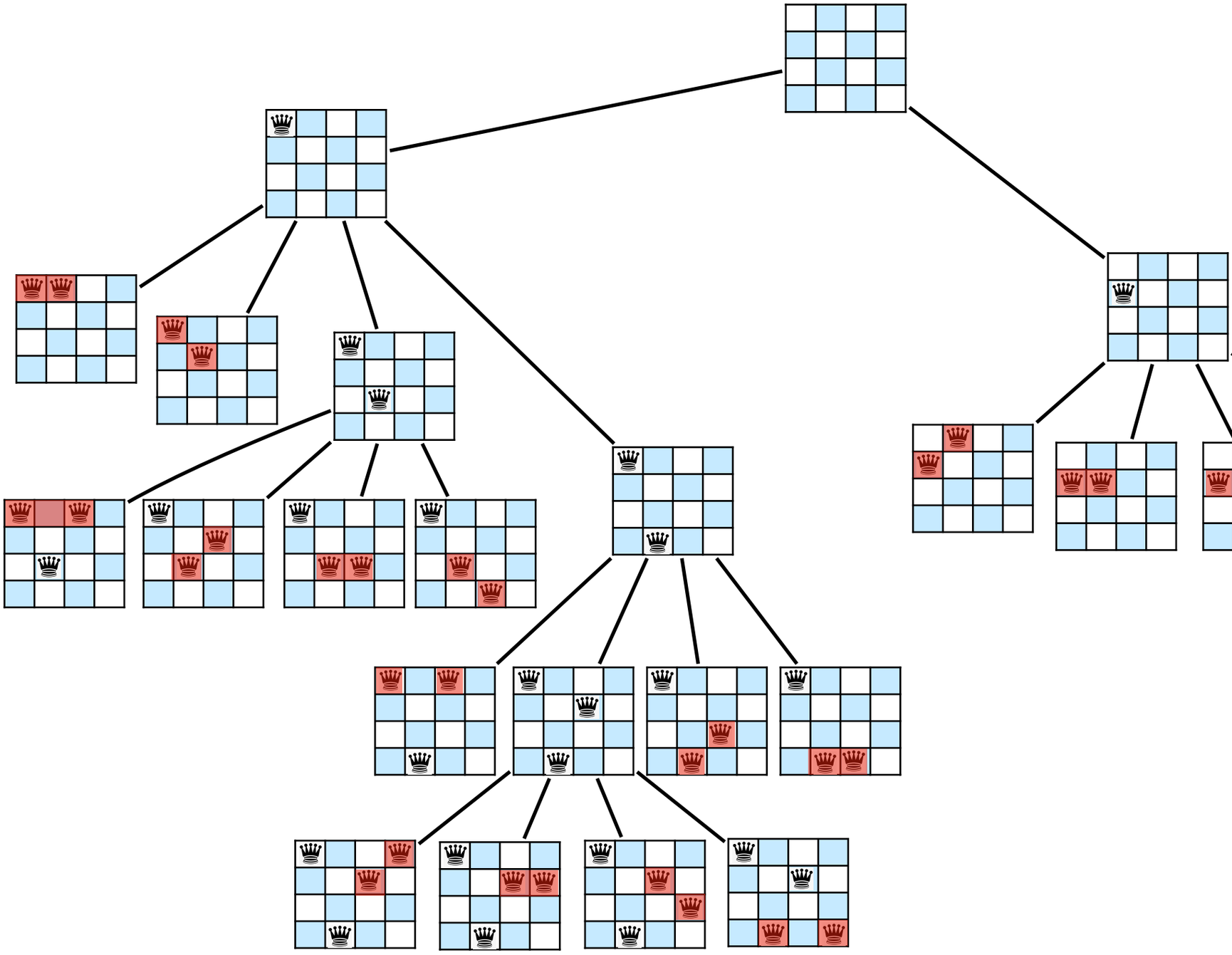


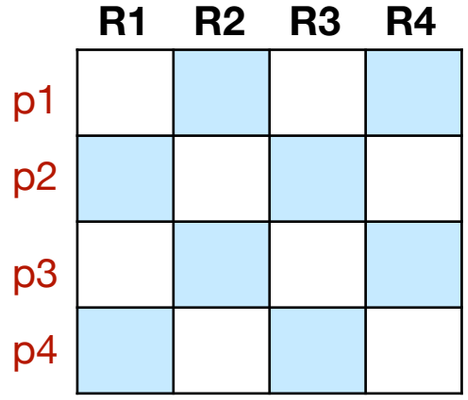


N-queens problem

Résolution en PPC

Backtraking (BT)

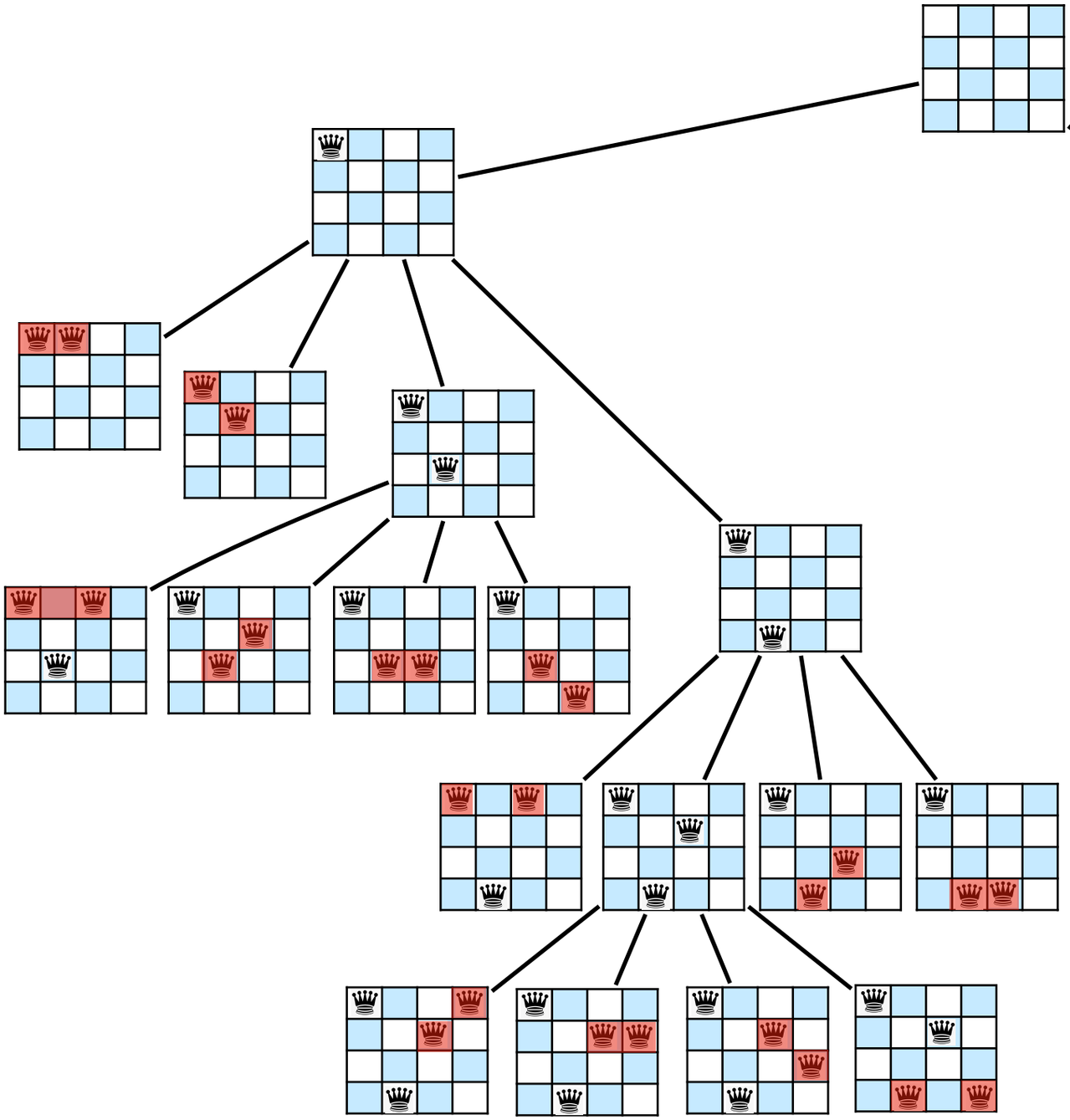


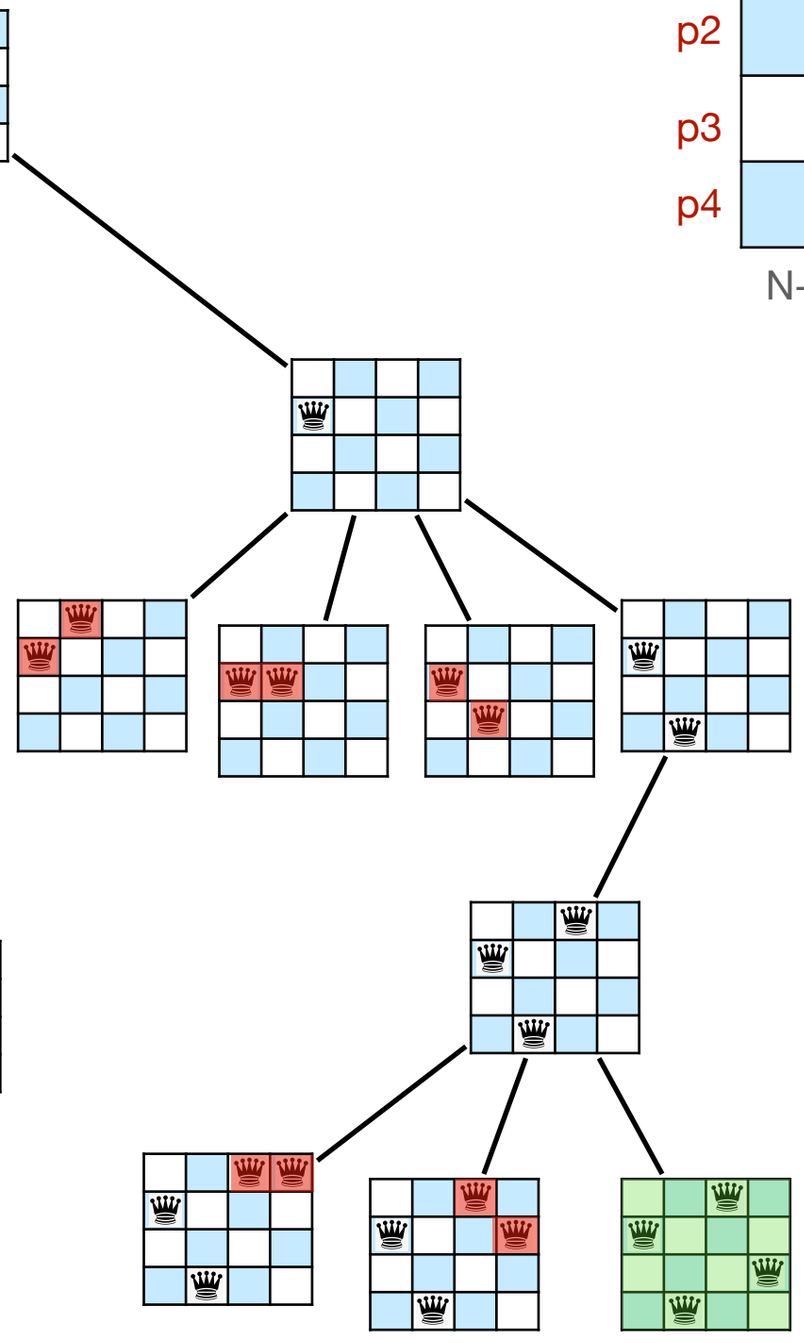


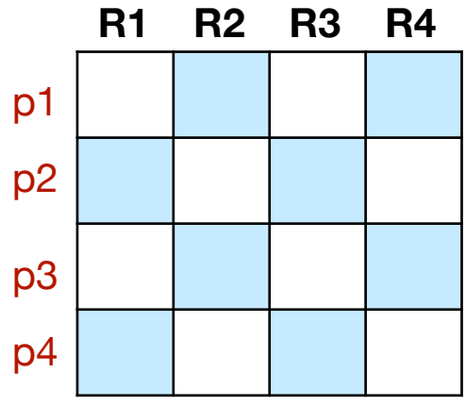
N-queens problem

Résolution en PPC

Backtraking (BT)



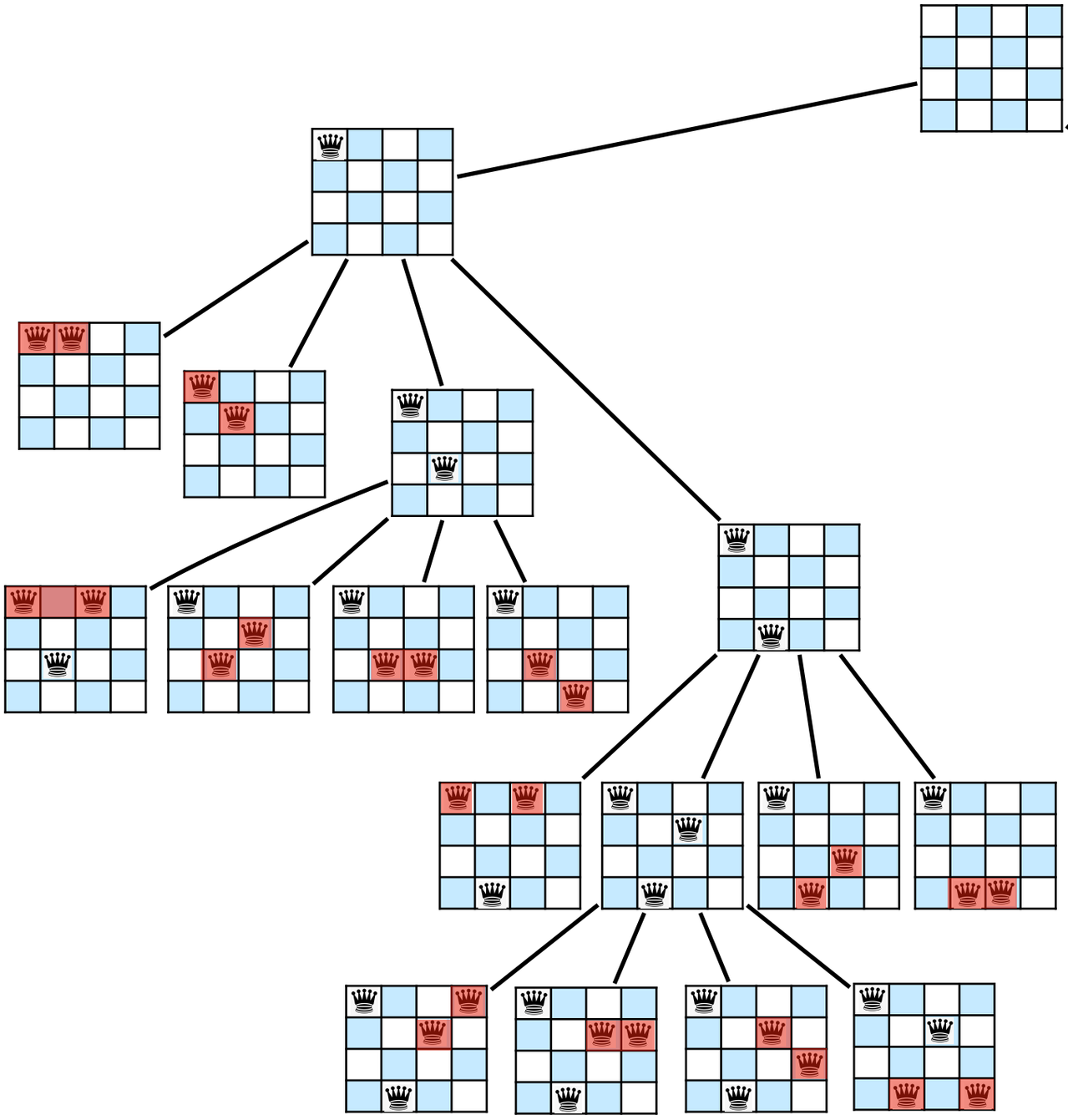


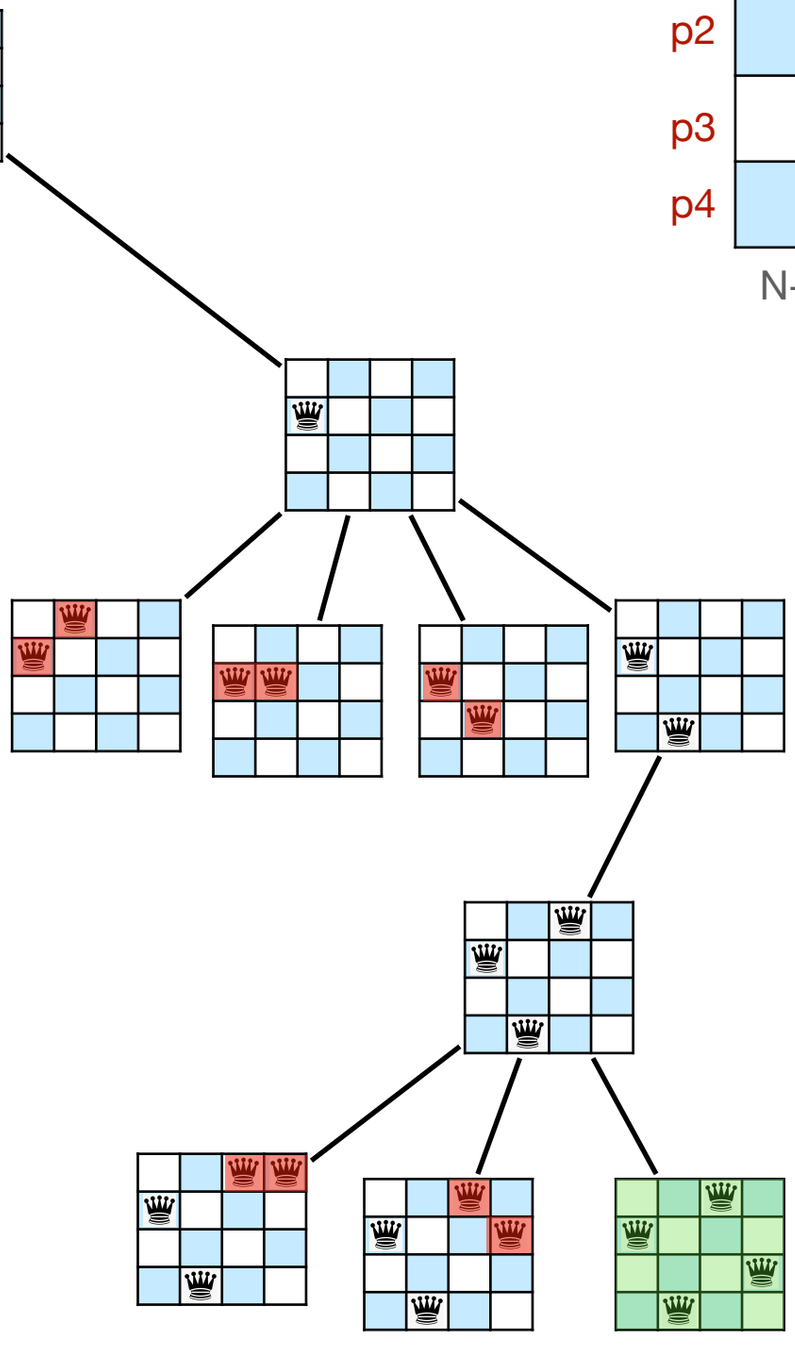


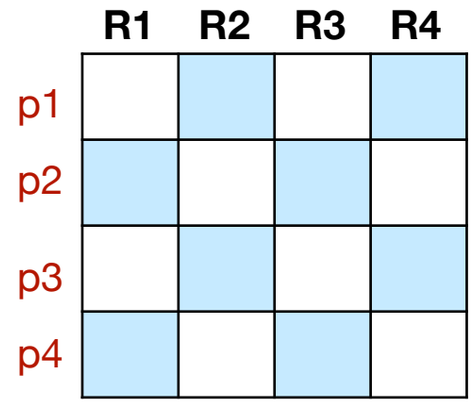
N-queens problem

Résolution en PPC

Backtraking (BT)





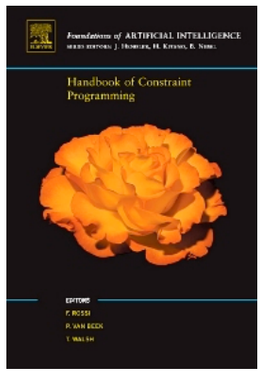


N-queens problem

26 nodes

Constraint Programming

References & links



- **Handbook of Constraint Programming.** Francesca Rossi Peter van Beek Toby Walsh. Elsevier Science 2006

- ACP: Association for Constraint Programming

- Guide to Constraint Programming

- CP conference Series

- Global Constraint Catalog

LISN
LABORATOIRE INTERDISCIPLINAIRE DES SCIENCES DU NUMÉRIQUE

université PARIS-SACLAY

cnrs

Artificial Intelligence

Cours6 - Constraint Programming

L3 - Informatique

Nadjib Lazaar
Ing - Phd - HDR - Professor - Paris-Saclay University - LISN - LaHDAK
lazaar@lisn.fr https://perso.lisn.upsaclay.fr/lazaar/

14/02/2025