Constraint Programming

The user states the problem,
the computer solve it!

Nadjib Lazaar
Ing - Phd - HDR - Assistant Professor - University of Montpellier - COCONUT Team
http://www.lirmm.fr/~lazaar/

11/09/2023

Constraint Programming

Definition

Constraint Programming

Definition

A formalism to model (**constraint network**) and to solve (**constraint solver**) combinatorial problems (scheduling, planning,…).

Constraint Programming

Definition

A formalism to model (**constraint network**) and to solve (**constraint solver**) combinatorial problems (scheduling, planning,…).

Examples of constraints :

- $X_1+X_2+X_3=5$
- allDifferent(X_1,..X_n)
- c(X_i,X_j)= {(1,1), (2,1),(2,3), (4,4)}

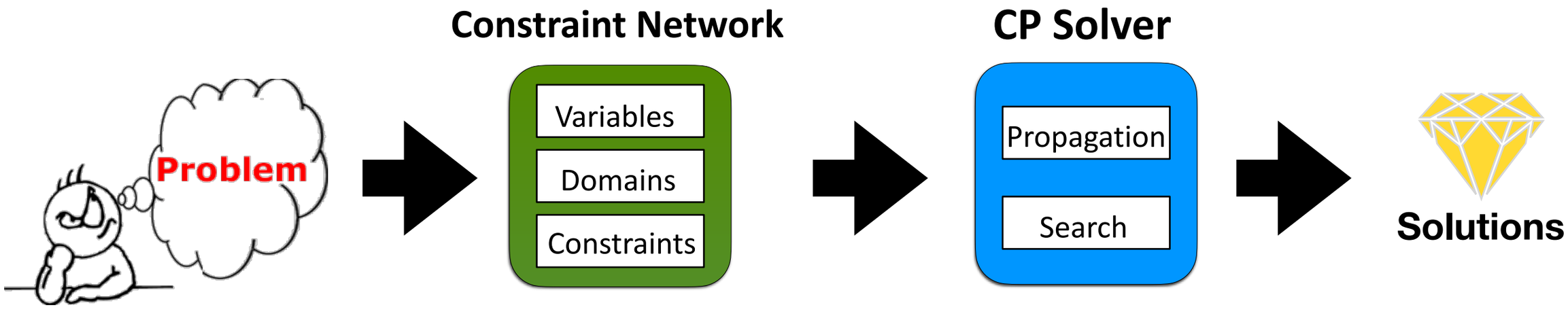
Constraint Programming

Definition

A formalism to model (**constraint network**) and to solve (**constraint solver**) combinatorial problems (scheduling, planning,…).

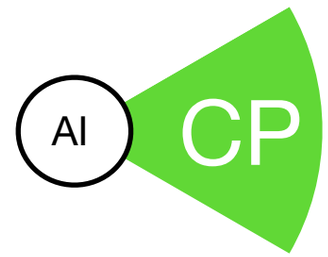
Examples of constraints :

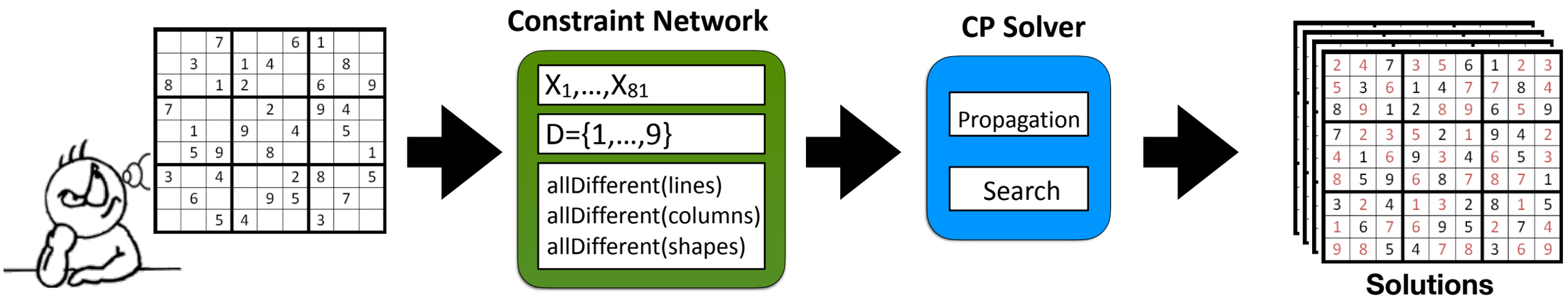
- $X_1+X_2+X_3=5$
- allDifferent(X_1,..X_n)
- c(X_i,X_j)= {(1,1), (2,1),(2,3), (4,4)}



Constraint Programming

Example



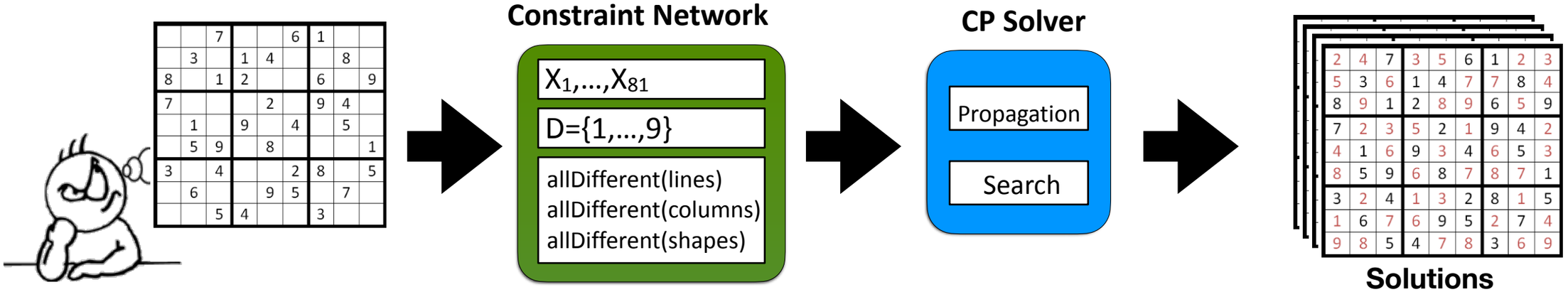


Switch

Constraint Programming

Example

AI CP



Propagation:

X1={~~1~~,2,~~3~~,4,5,~~6~~,~~7~~,~~8~~,9}

…

Search:

X2 <— 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		

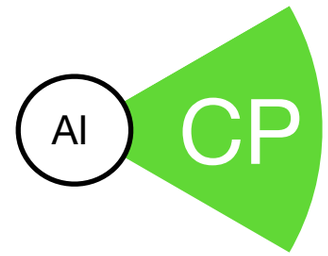
Solution:

X1 = 2
x2 = 4
X3 = 7
…
X81 = 9

Switch

Constraint Programming

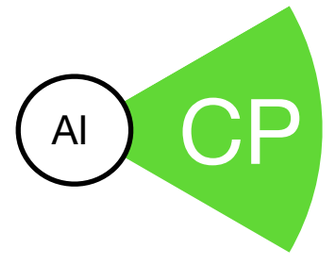
Why CP?



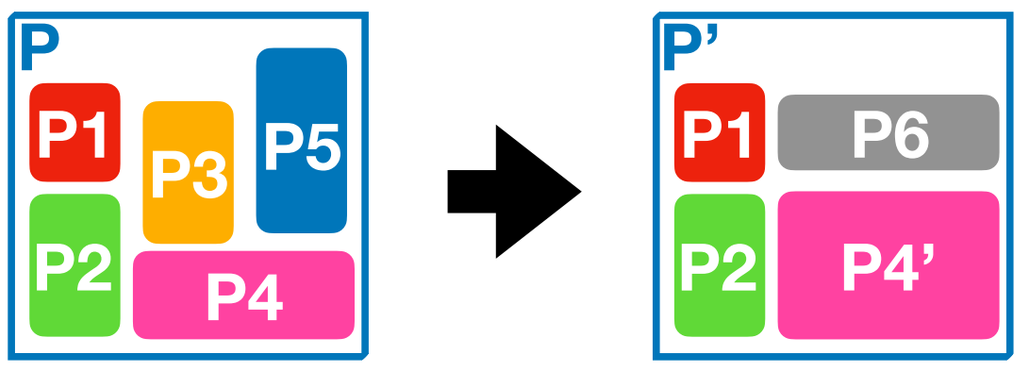
- Combinatorial problems can be solved by Integer Linear Programming (ILP) or by propositional satisfiability (SAT)
- Advantages of CP :
  - Compactness
  - Expressiveness
  - And (often) efficiency

Constraint Programming

Why CP?

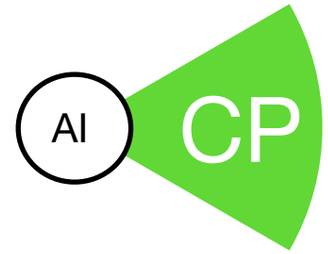


- Combinatorial problems can be solved by Integer Linear Programming (ILP) or by propositional satisfiability (SAT)
- Advantages of CP :
  - Compactness
  - Expressiveness
  - And (often) efficiency



Constraint Programming

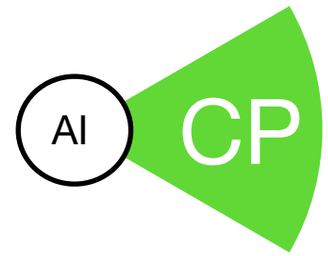
Why CP?



- Sudoku 9x9:
  - SAT = 6 500 clauses
  - CP: `sudoku.mod` with n=9

Constraint Programming

Why CP?



- Sudoku 9x9:
  - SAT = 6 500 clauses
  - CP: `sudoku.mod` with n=9

Variables $X = \{X_{1,1}, \ldots X_{n,n}\}$

Domaine des variables : {1,…,n}

Contraintes :

allDifferent($X_{i,1}$,..,$X_{i,n}$), ∀i (rows)

allDifferent($X_{i,1}$,..,$X_{i,n}$), ∀i (columns)

allDifferent($X_{i,j}$), ∀i,j (squares)

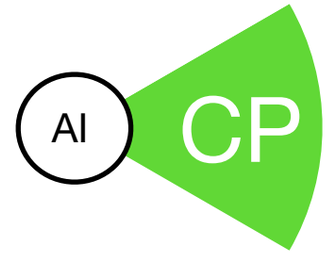
sudoku.mod

Sudoku 36x36

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

Constraint Programming

Why CP?

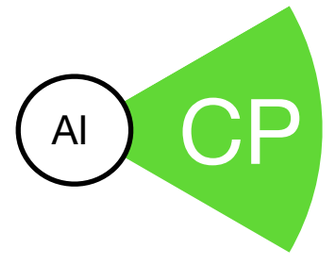


- Sudoku 36x36:
  - SAT ~ 2 millions of clauses : cnf file of 160Gb
  - CP: `sudoku.mod` with n=36

Switch

Constraint Programming

Why CP?



- Sudoku 36x36:
  - SAT ~ 2 millions of clauses : cnf file of 160Gb
  - CP: `sudoku.mod` with n=36

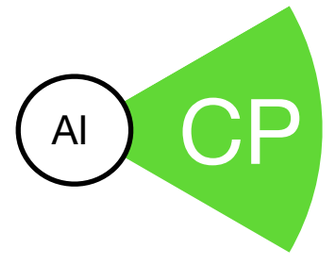
```
Variables X= {X1,1, …Xn,n}

Domaine des variables : {1,…,n}

Contraintes :
allDifferent(Xi,1,..,Xi,n), ∀i (rows)
allDifferent(Xi,1,..,Xi,n), ∀i (columns)
allDifferent(Xi,j), ∀i,j (squares)
```

sudoku.mod

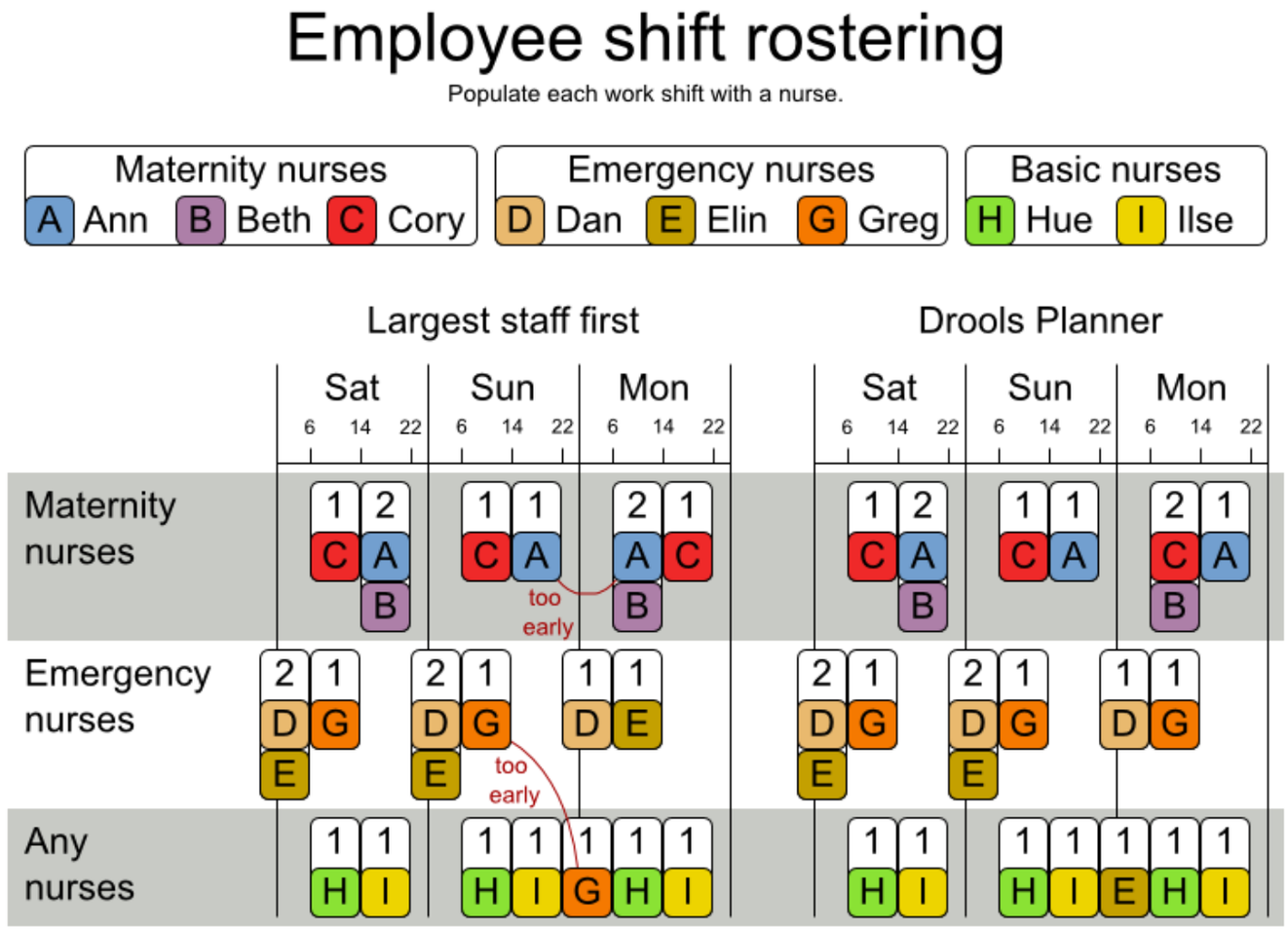
Switch

Constraint Programming

Why CP?



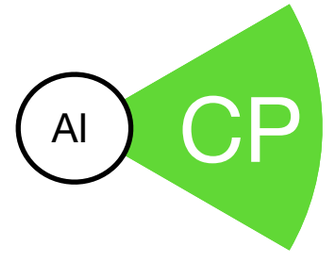
- Nurse Rostering Problem
  - Best Linear Program: more than 10 000 lines



Switch

Constraint Programming

Solvers and CP platforms



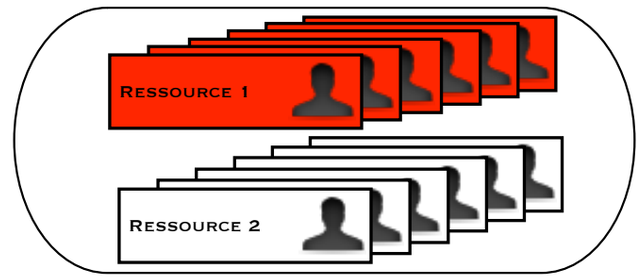
IBM ILOG CP Optimizer (Java, C++, Python, .NET)	IBM
Google OR-Tools (C++, Java, C#, Python)	Google OR-Tools
Artelys Kalis (Java, C++, Python)	ARTELYS KALIS™ Constraint Programming Library
SICStus Prolog (CLPFD bib in prolog)	SICStus 4
Gecode (C++)	G
Choco (Java)	CHOCO
Minizinc (high-level, solver-independent)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

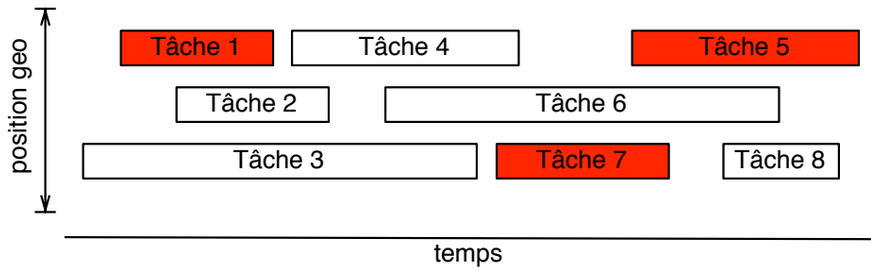
Constraint Programming

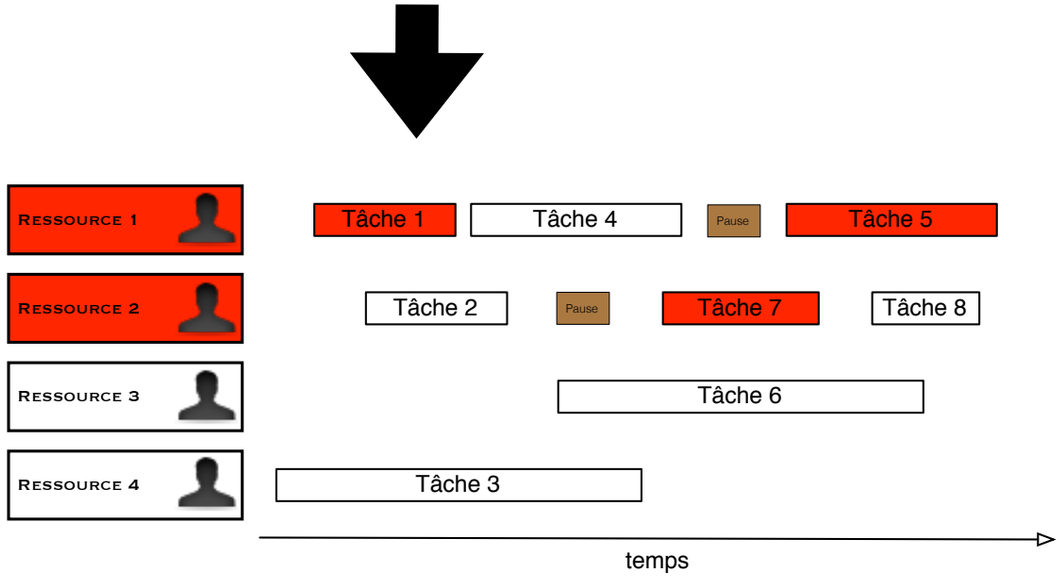
SNCF train driver planning using CP

AI 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
  - …



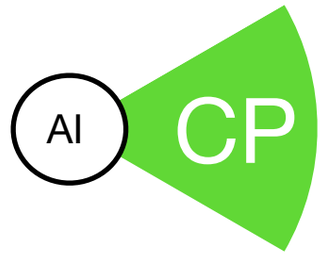




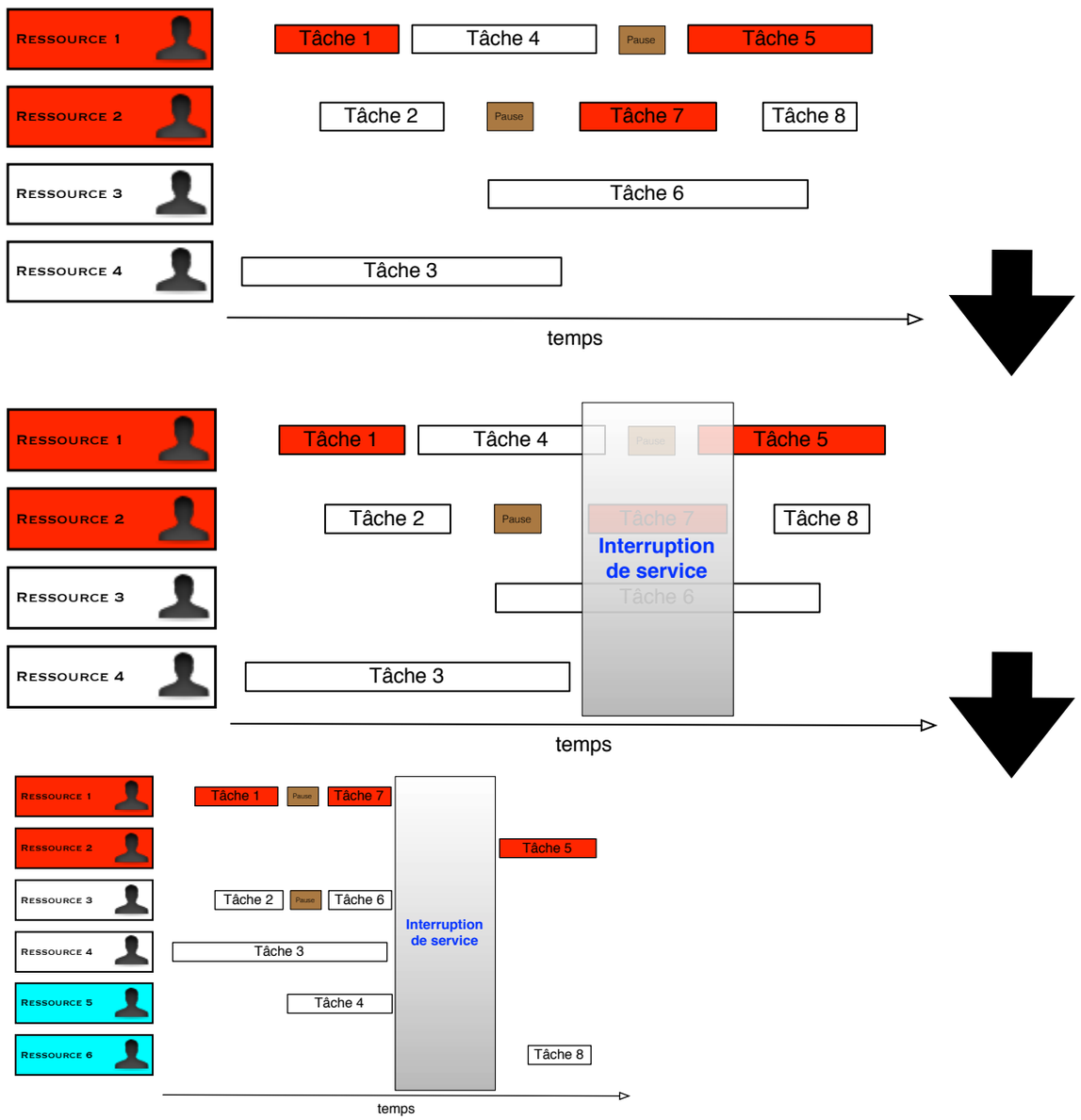
SNCF

Constraint Programming

SNCF train driver planning using CP



- Maintaining an existing planning



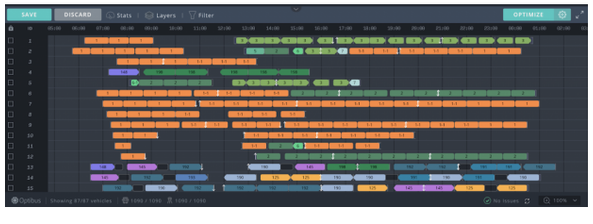
Constraint Programming

ABB Robotics partner projects

AI CP

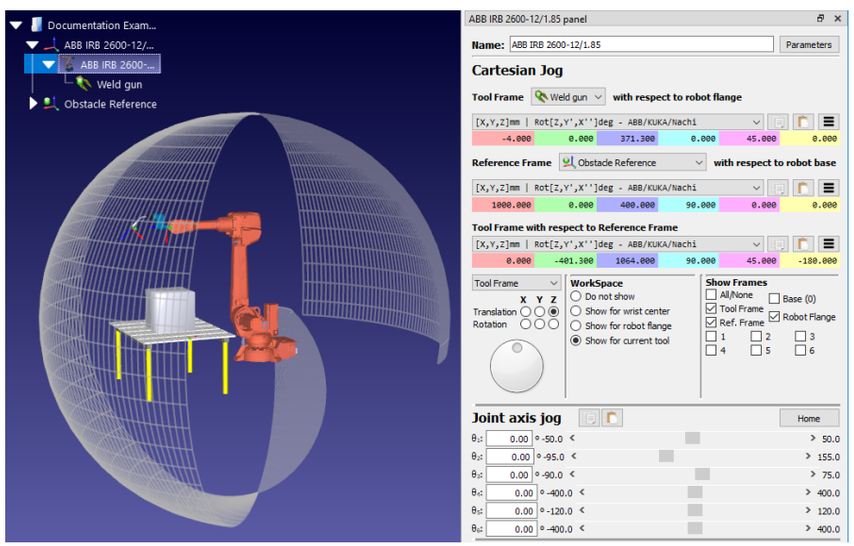
- SWMOD: Test Case Execution Scheduling with CP

"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

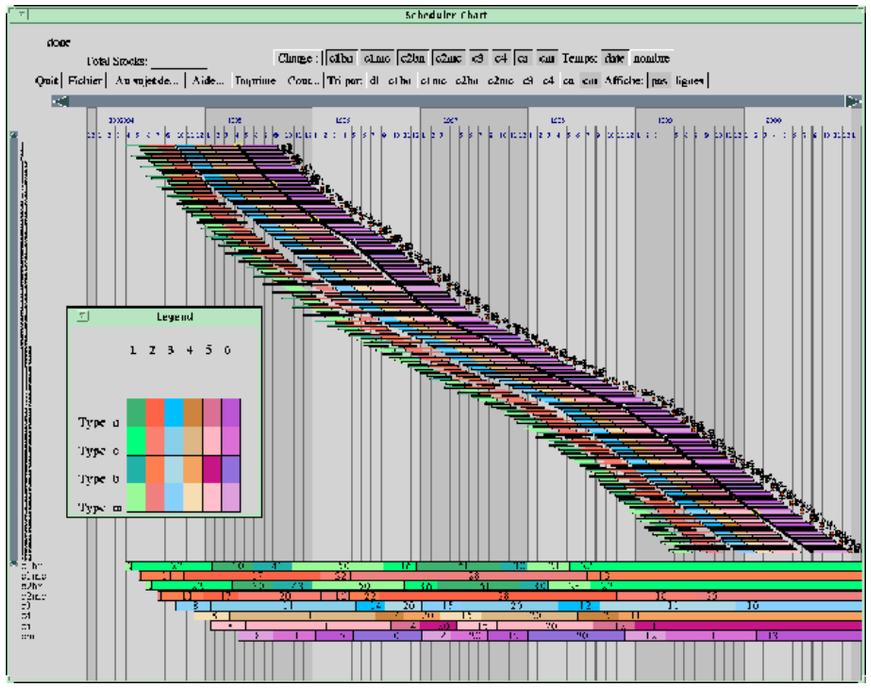
- Robtest: Optimal Stress Test Trajectories for Robots with CP



Constraint Programming

Aircraft Industry - Dassault Aviation

Assembly of Mirage aircraft

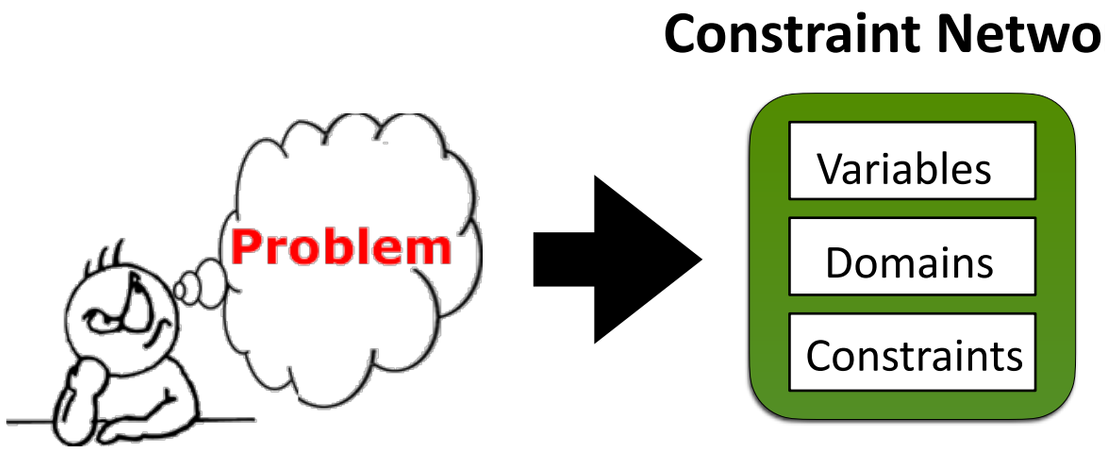


Constraint Programming

Modeling

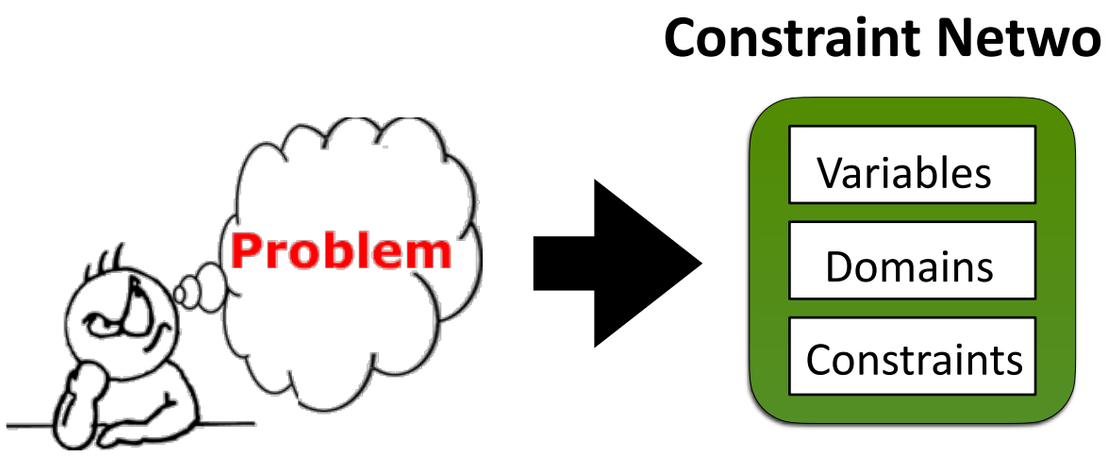
Constraint Programming

Modeling



Constraint Programming

Modeling

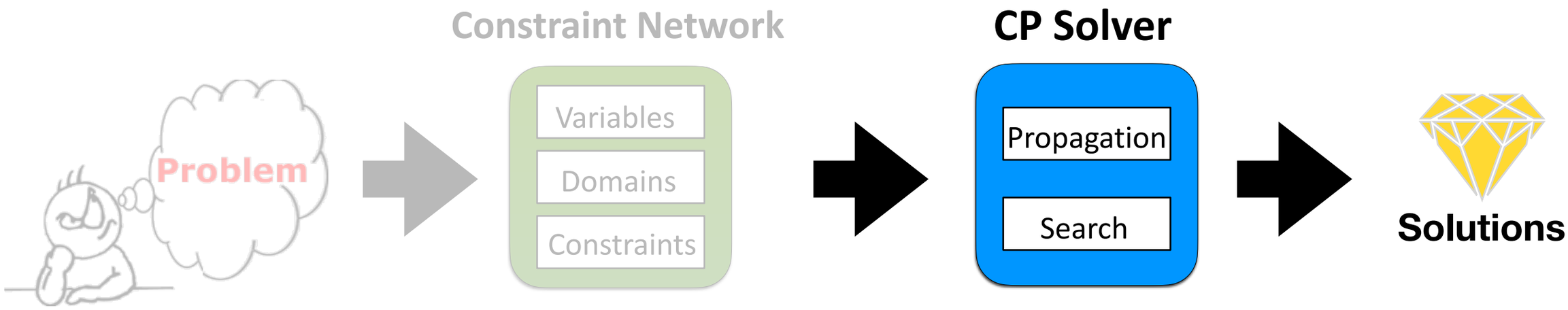


Constraint Network N=(X,D,C)

X: finite set of variables
D: domain on X, where D(Xi) is a finite set of values for Xi
C: a set of constraints

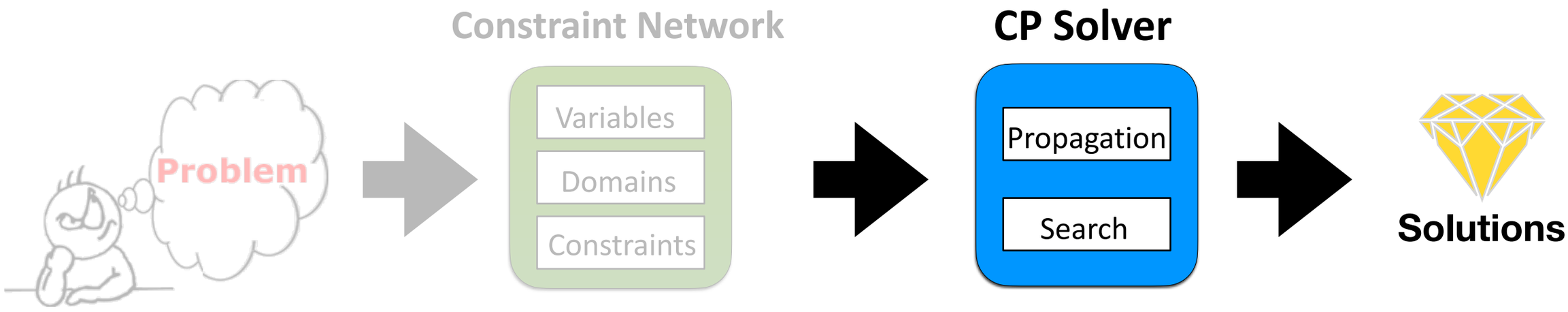
Constraint Programming

Solving



Constraint Programming

Solving



Instantiation I

- Complete: assignment on X
- Partial: assignment on Y subset X
- Valid: values in D
- Violating c: assignment on var(c) is not in c
- Locally consistent: assignment not violating any constraint on Y
- Solution: locally consistent on X

Constraint Programming

Solving - Backtraking (BT)

- A general-purpose search algorithm
- Systematically explores the search space
- Utilizes a depth-first search strategy
- May encounter inefficiency on large search spaces
- Suitable for a wide range of constraint problems
- May require additional pruning techniques

Constraint Programming

Solving - Backtraking (BT)

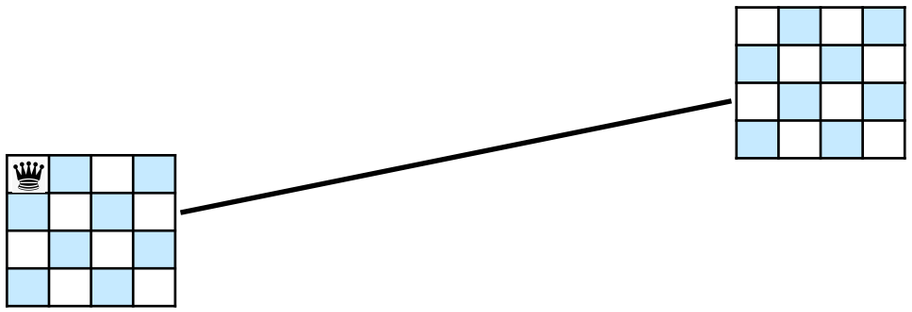
Constraint Programming

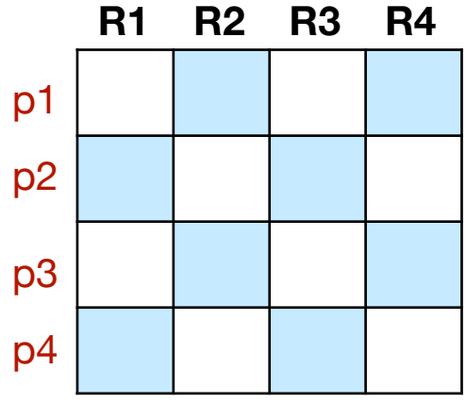
Solving - Backtraking (BT)

```
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
```

Constraint Programming

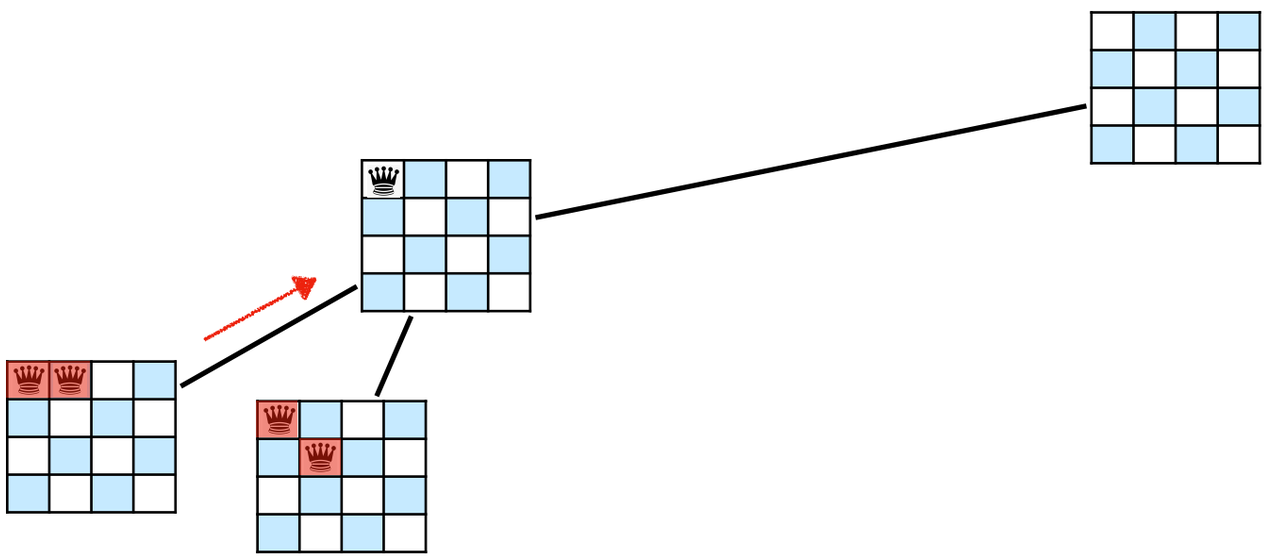
Solving - Backtraking (BT)

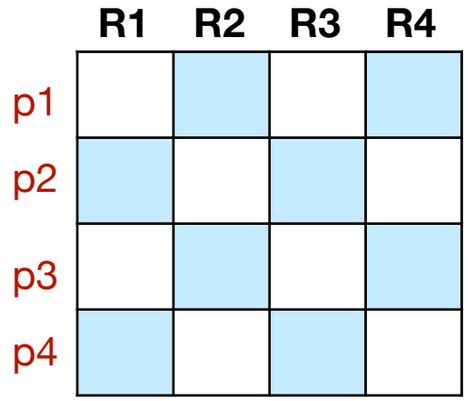




Constraint Programming

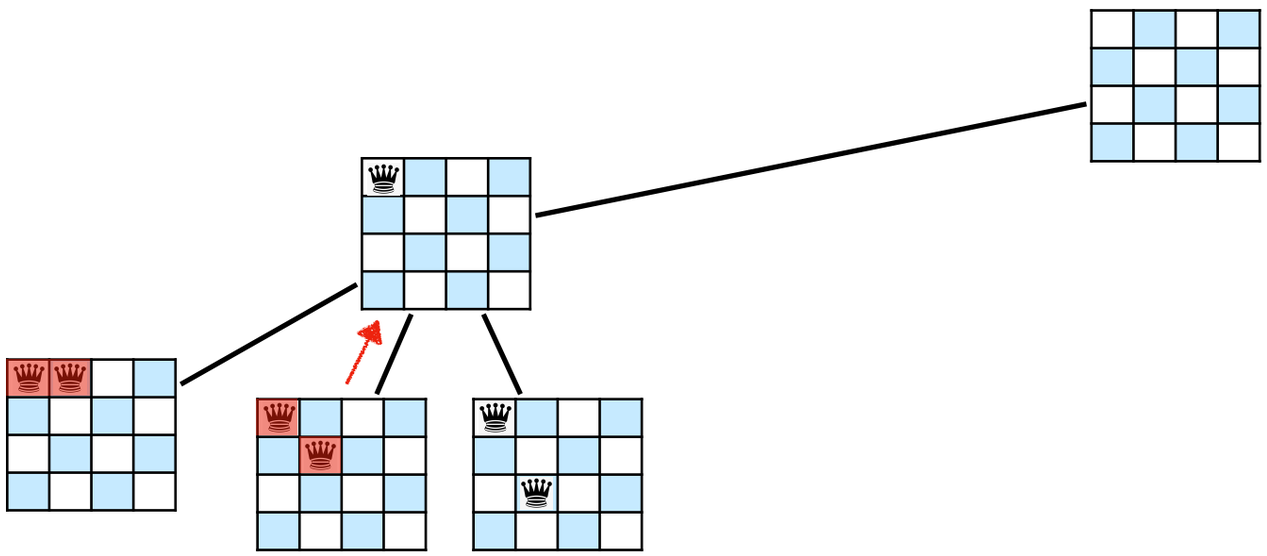
Solving - Backtraking (BT)

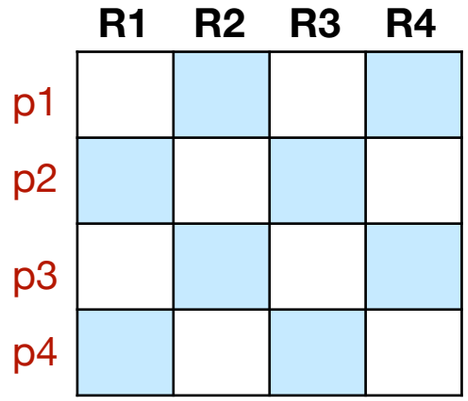




Constraint Programming

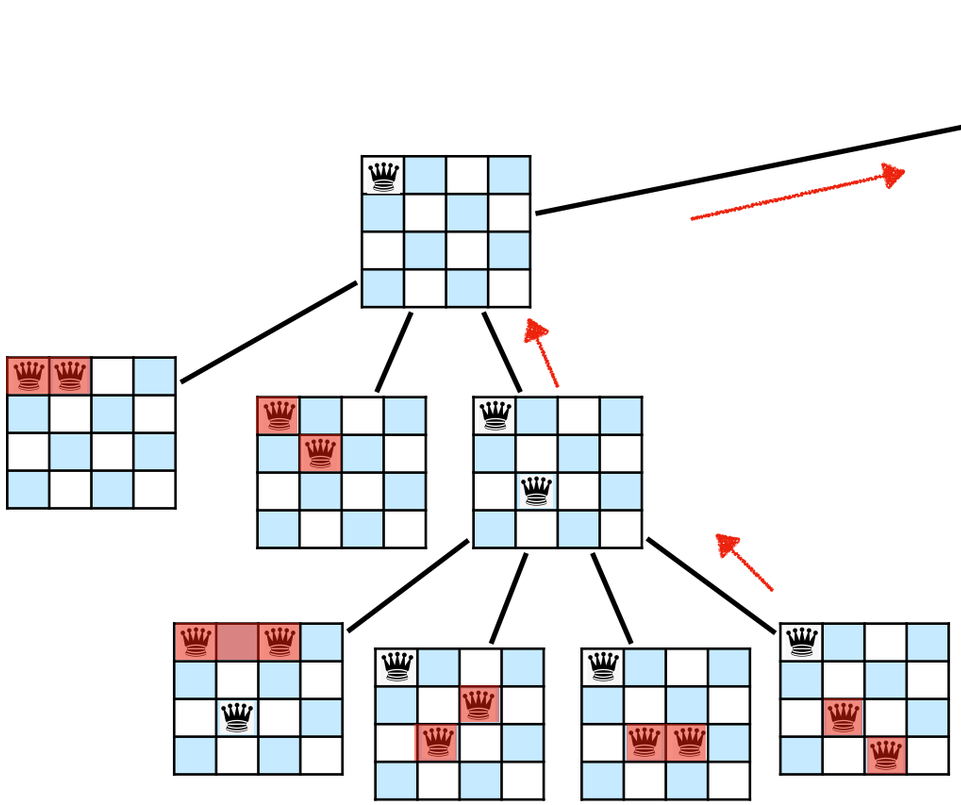
Solving - Backtraking (BT)

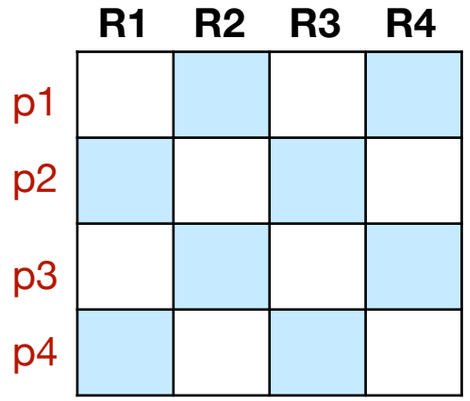




Constraint Programming

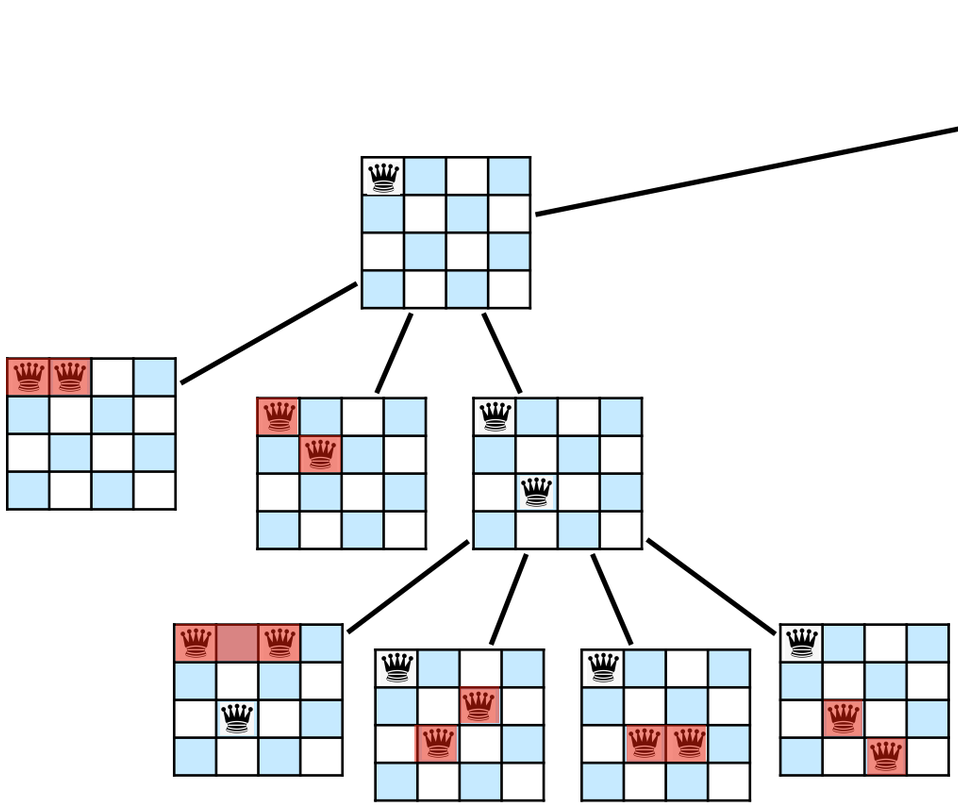
Solving - Backtraking (BT)

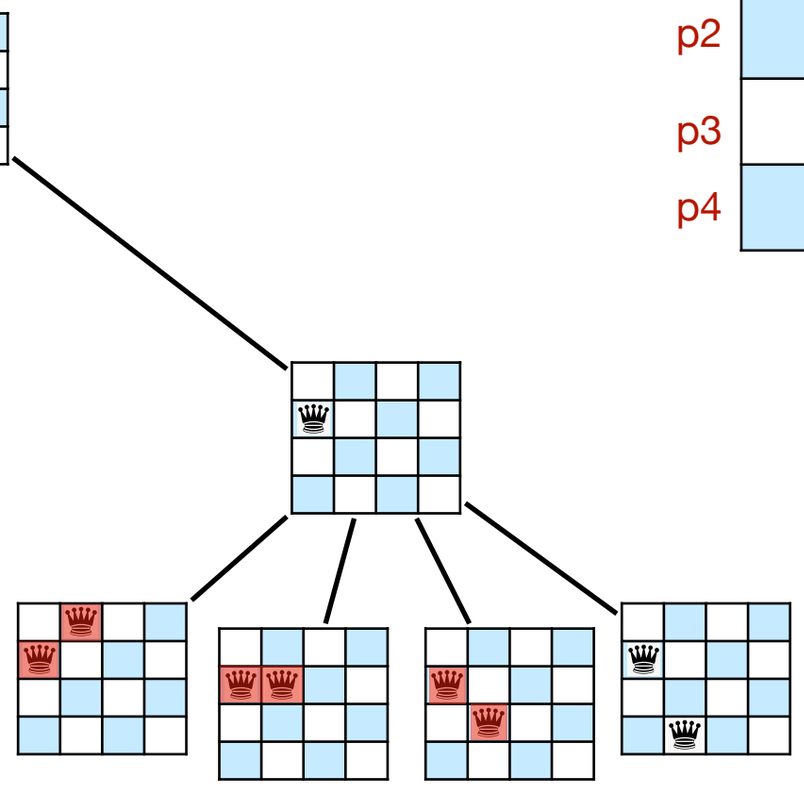


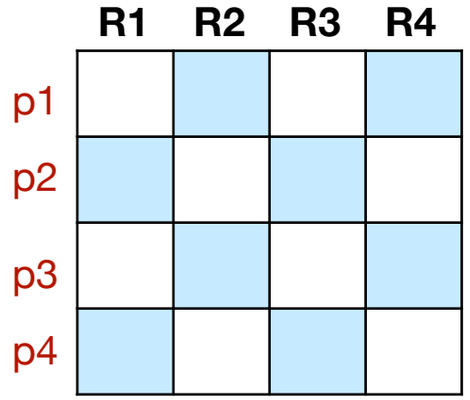


Constraint Programming

Solving - Backtraking (BT)

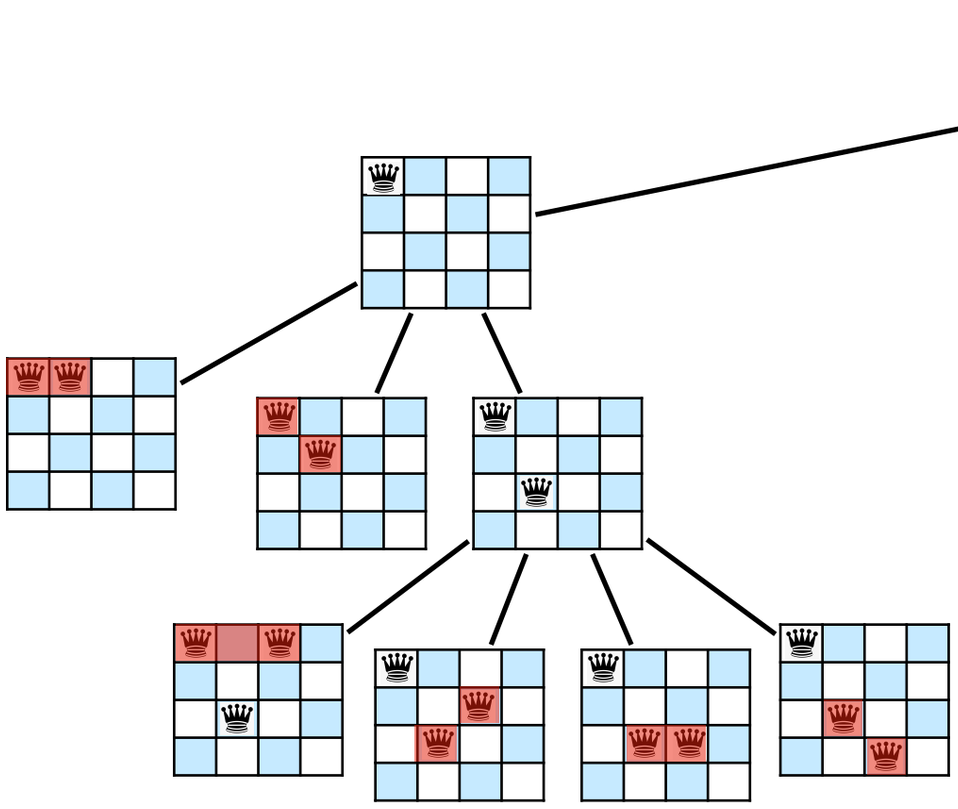


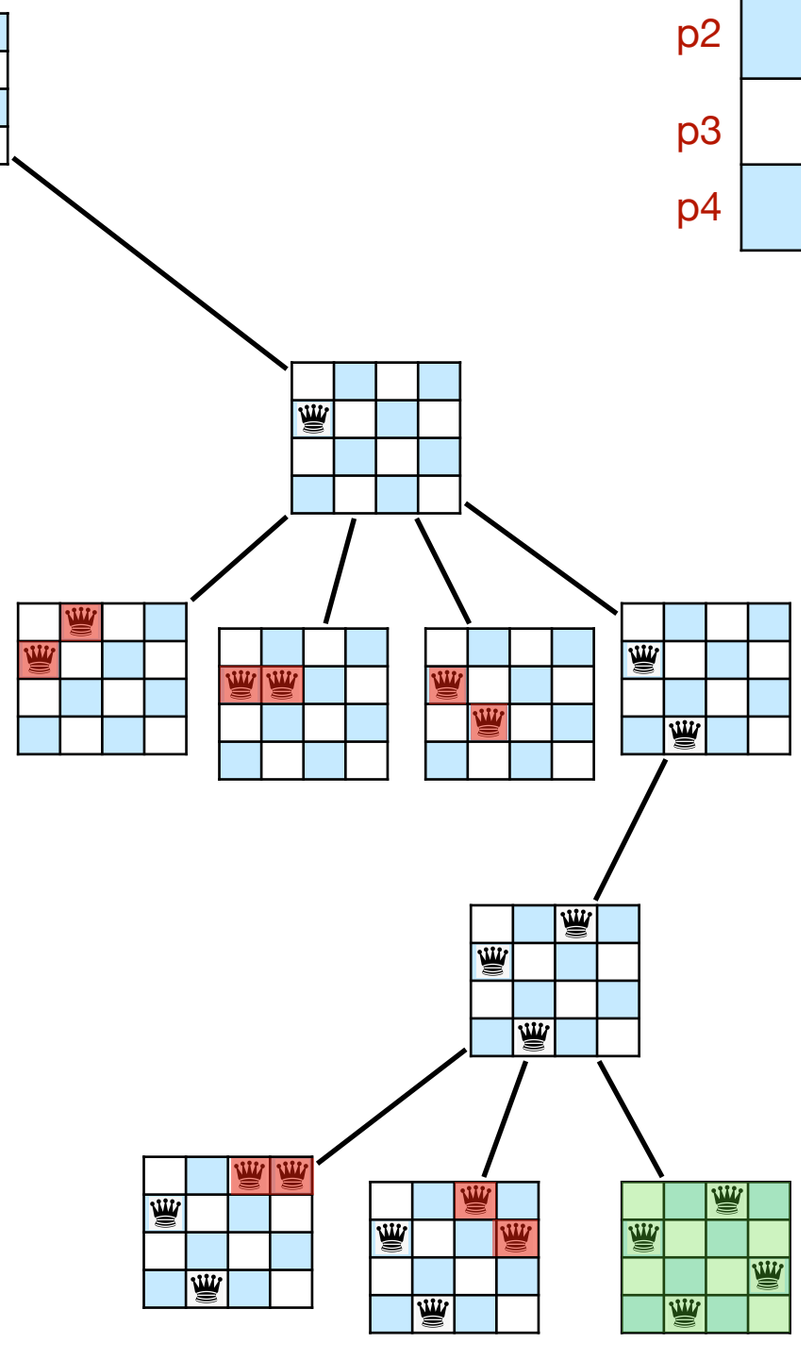


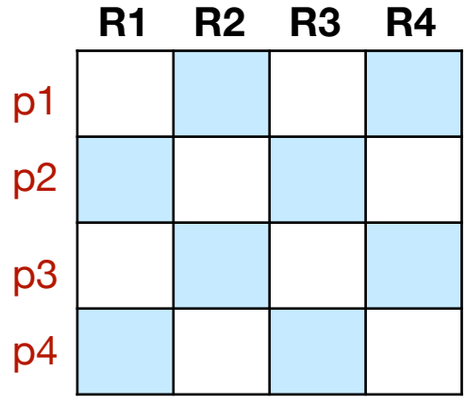


Constraint Programming

Solving - Backtraking (BT)

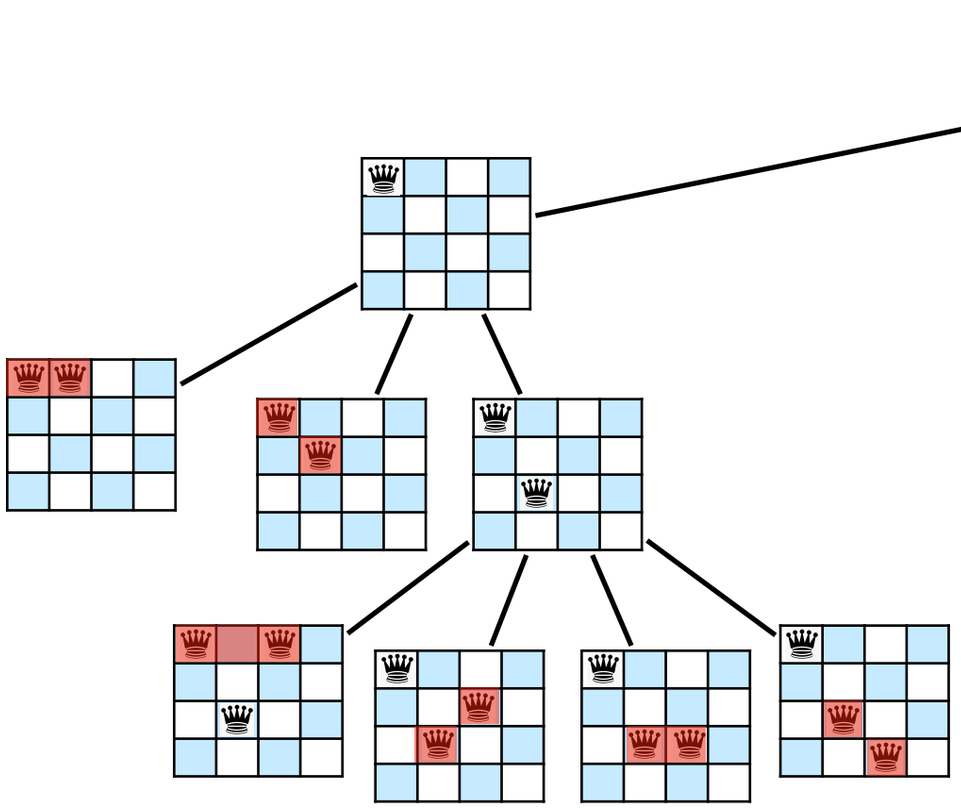


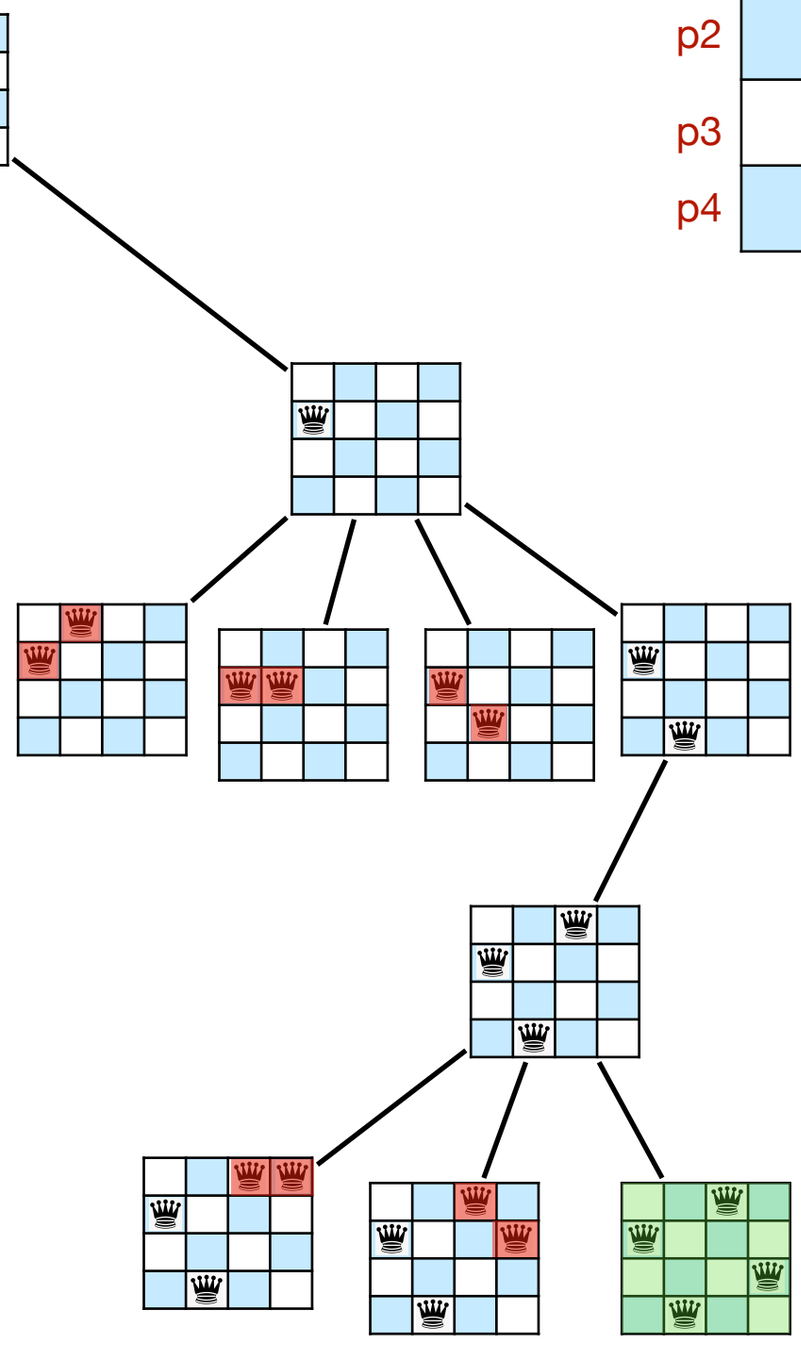


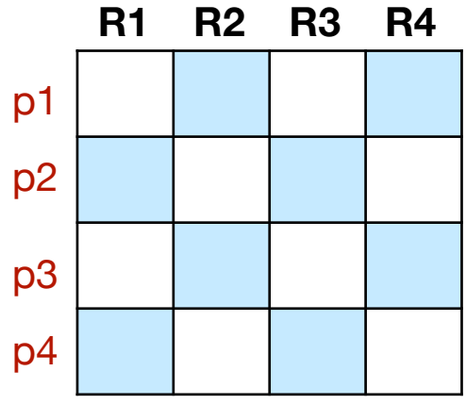


Constraint Programming

Solving - Backtraking (BT)







Constraint Programming

Solving - propagation (local consistency)

- AC closure implemented by several AC algorithms
- AC3 [Mackworth 1977]:

Constraint Programming

Solving - propagation (local consistency)

- AC closure implemented by several AC algorithms
- AC3 [Mackworth 1977]:

```
AC(<X,D,C>):                                                    [Mackworth77]

Q gets {(Xi, c) | c in C, Xi in var(c)}

While Q != emptyset do

  pick (Xi, c) in Q

  If revise(Xi, c) then

    If D(Xi) = emptyset then return false

    Else Q gets Q union {(Xj,c')|c' in C and Xi, Xj in var(c)}

return true
```

Constraint Programming

Solving - propagation (local consistency)

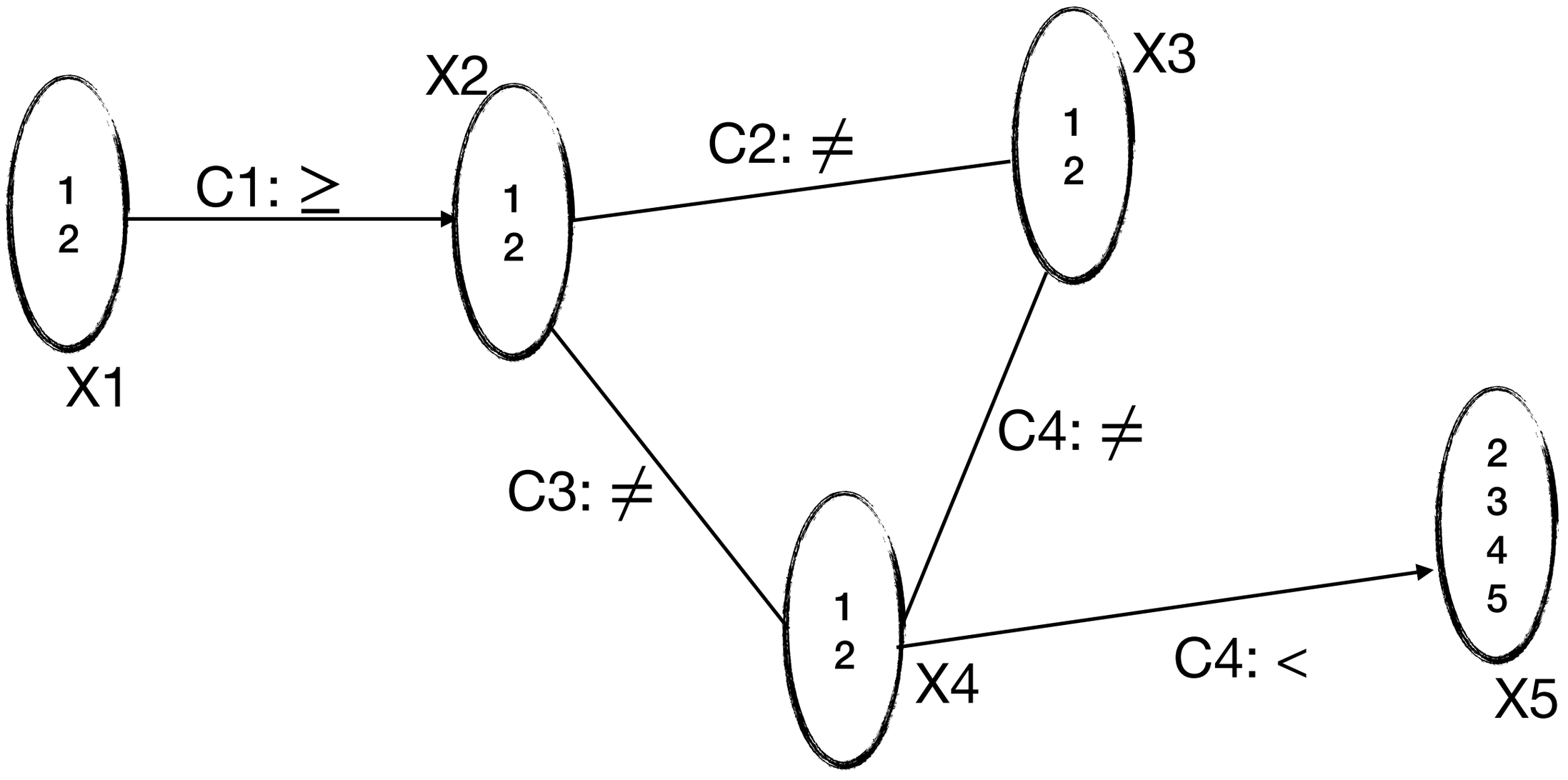
Constraint Programming

Solving - propagation (local consistency)

```
revise(Xi, c):                                              [Mackworth77]

CHANGE gets false

ForEach v in D(Xi) do

  If no allowed pairs (v, vi) for c then

     Remove v from D(Xi)

     CHANGE gets true

return CHANGE
```

Constraint Programming

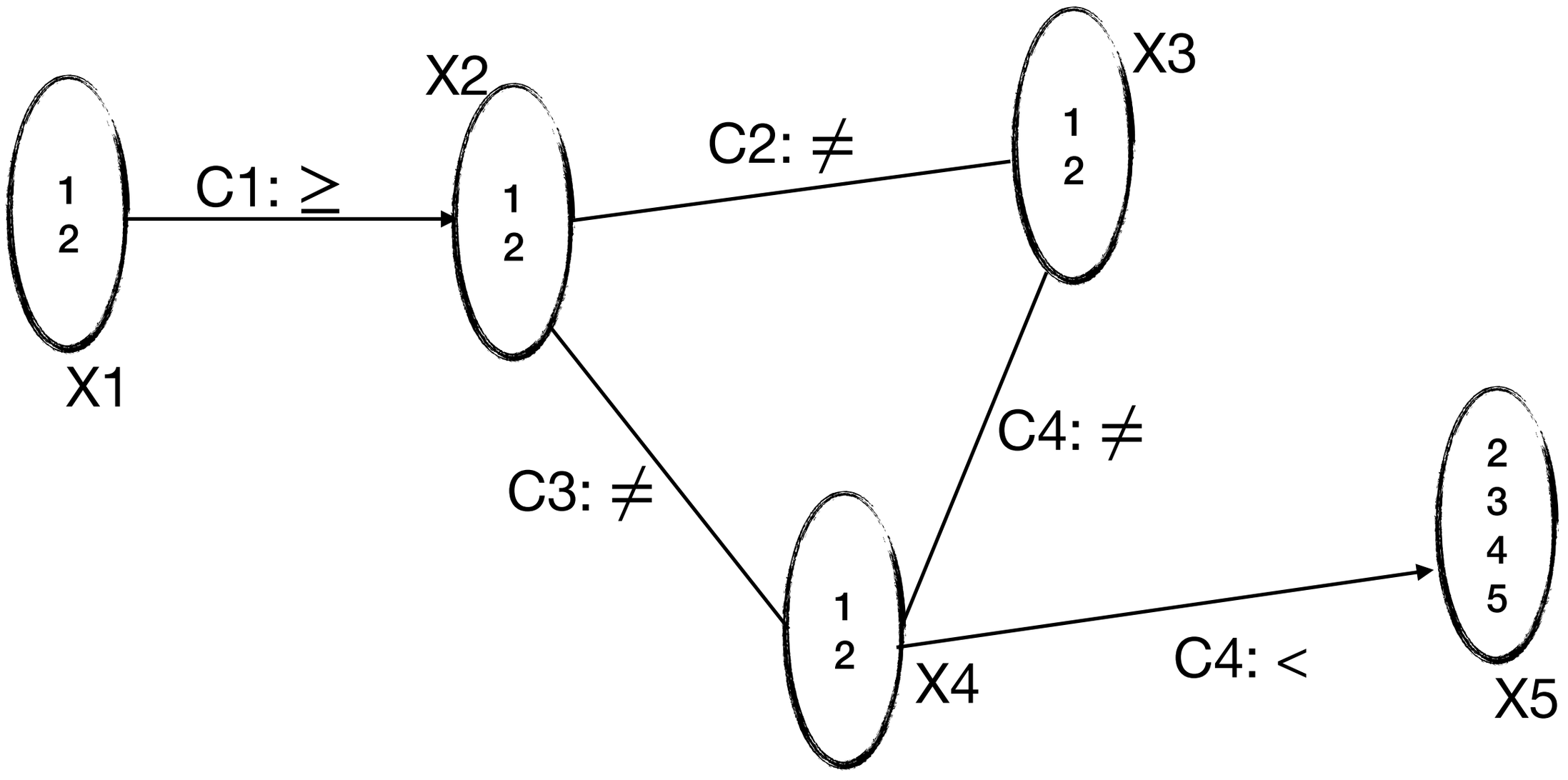
Solving - propagation (local consistency)



Situation 1

Constraint Programming

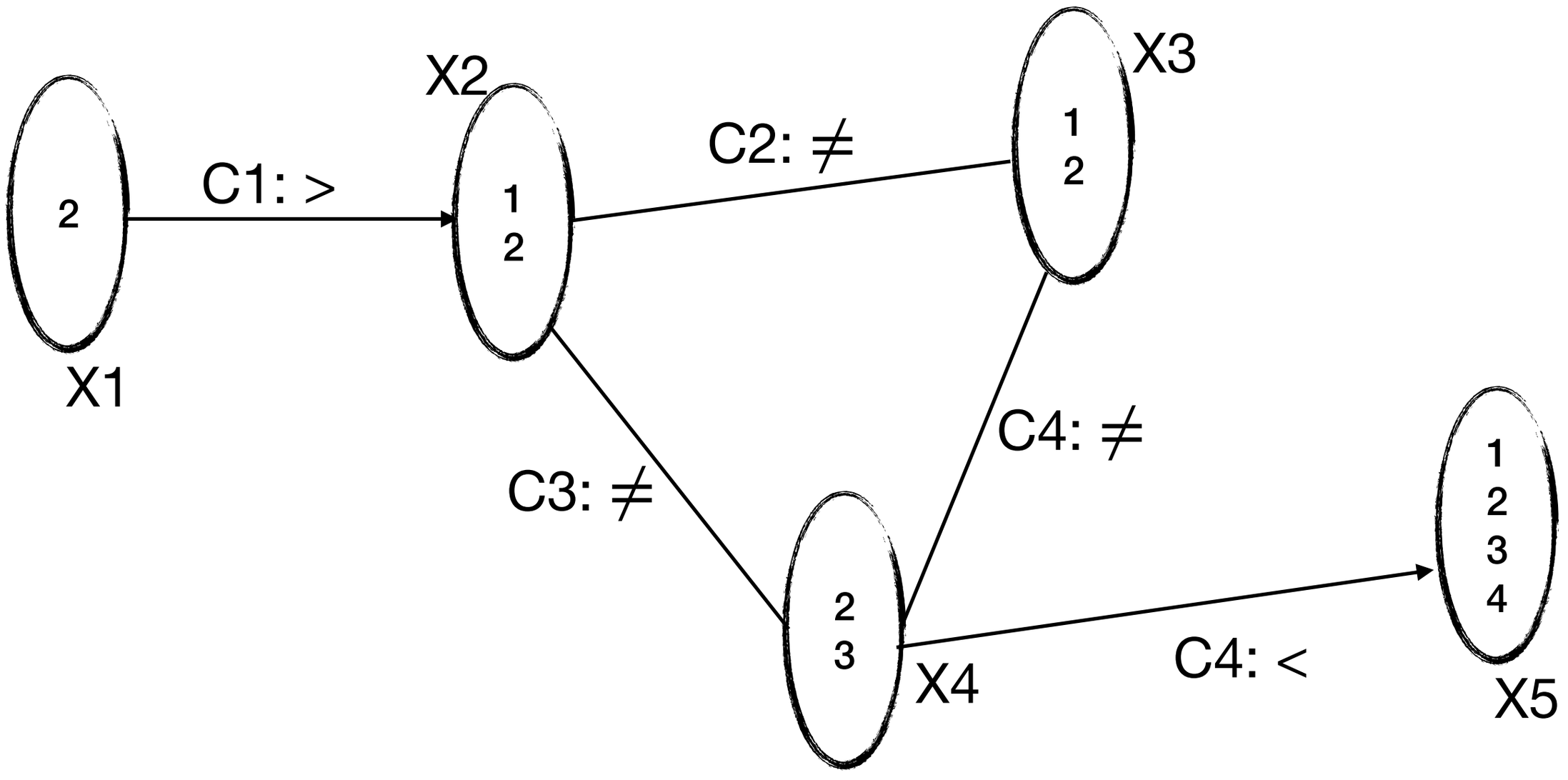
Solving - propagation (local consistency)



Situation 1 (No change)

Constraint Programming

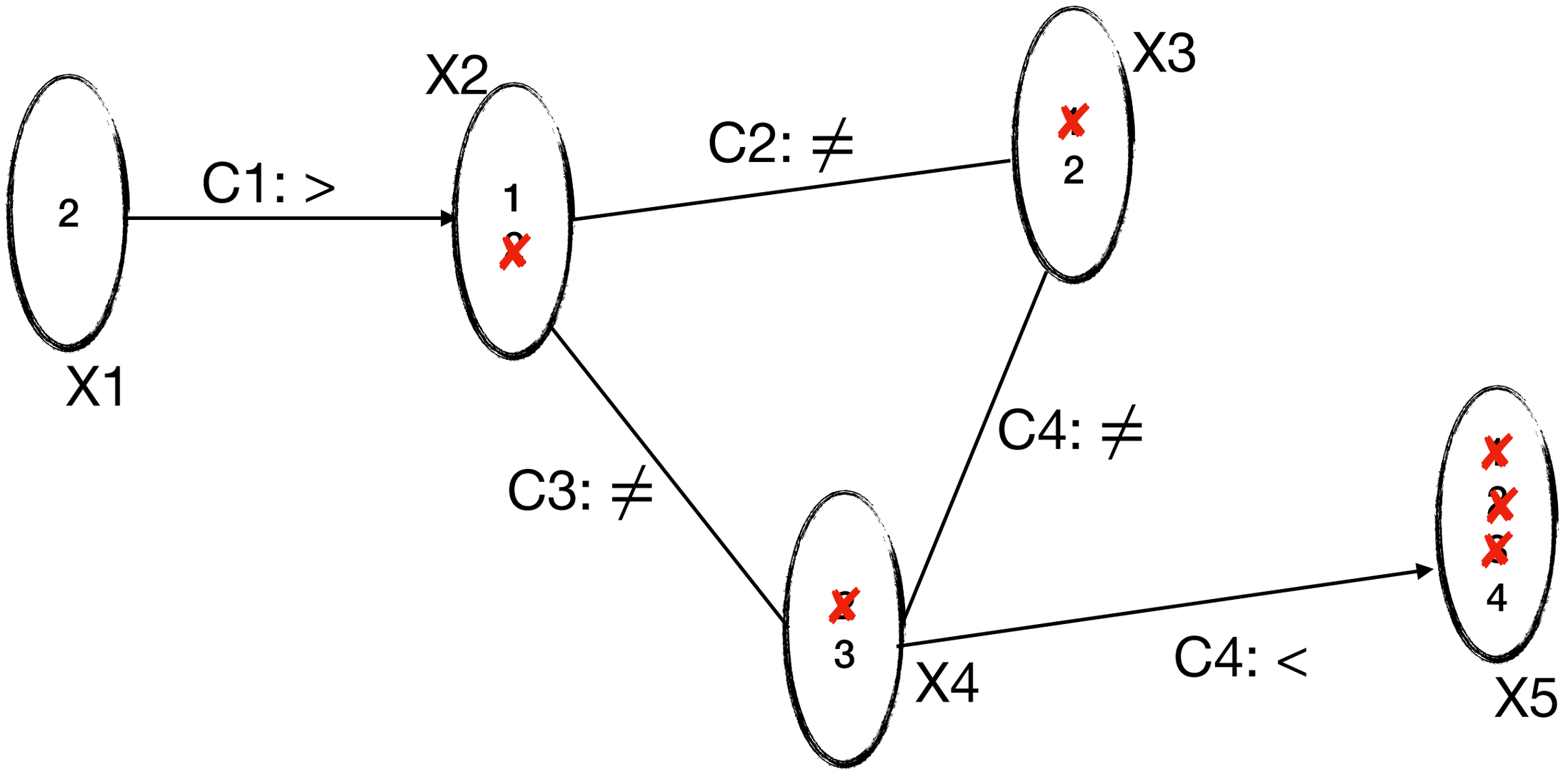
Solving - propagation (local consistency)



Situation 2

Constraint Programming

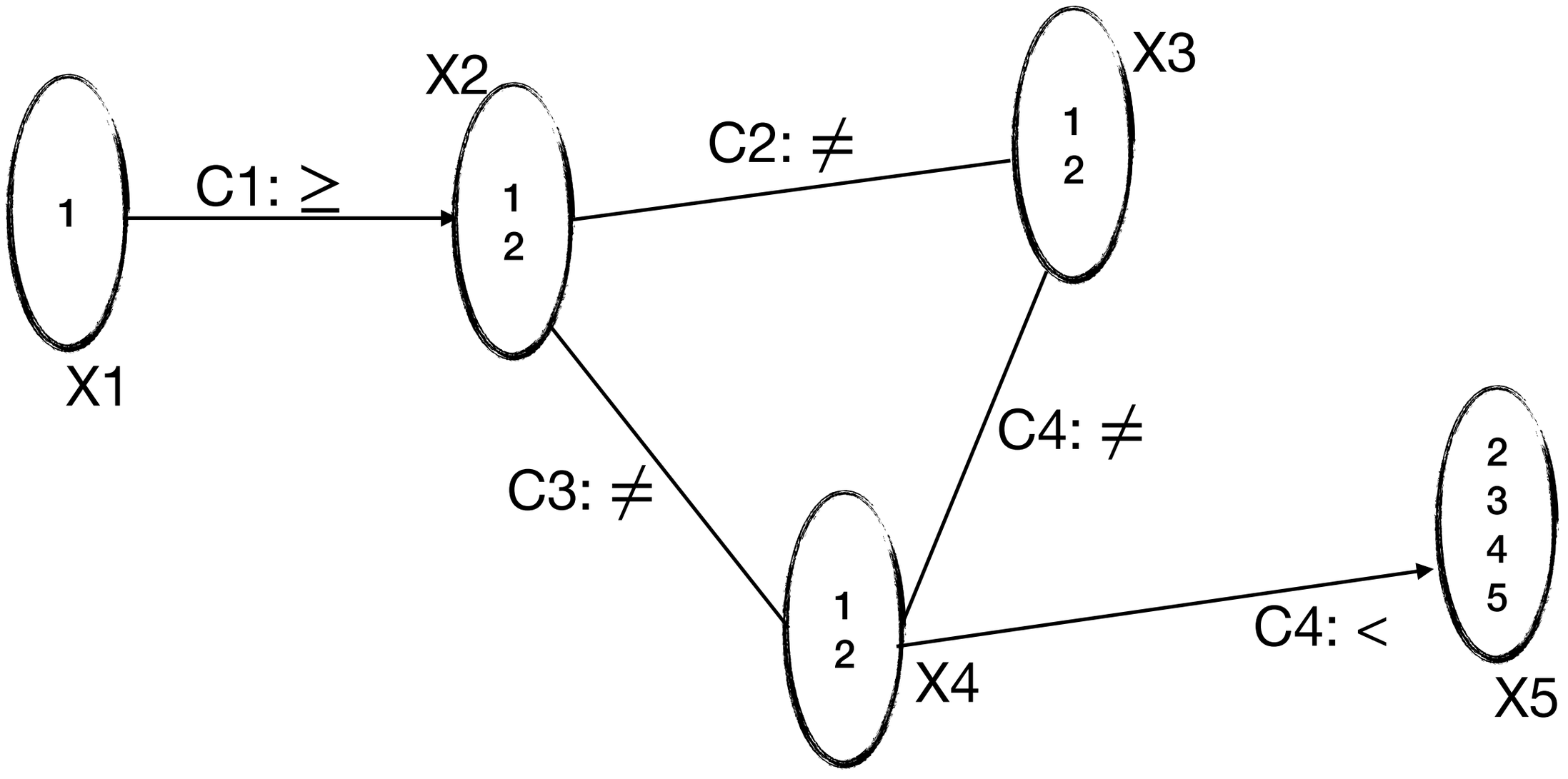
Solving - propagation (local consistency)



Situation 2 (Solution)

Constraint Programming

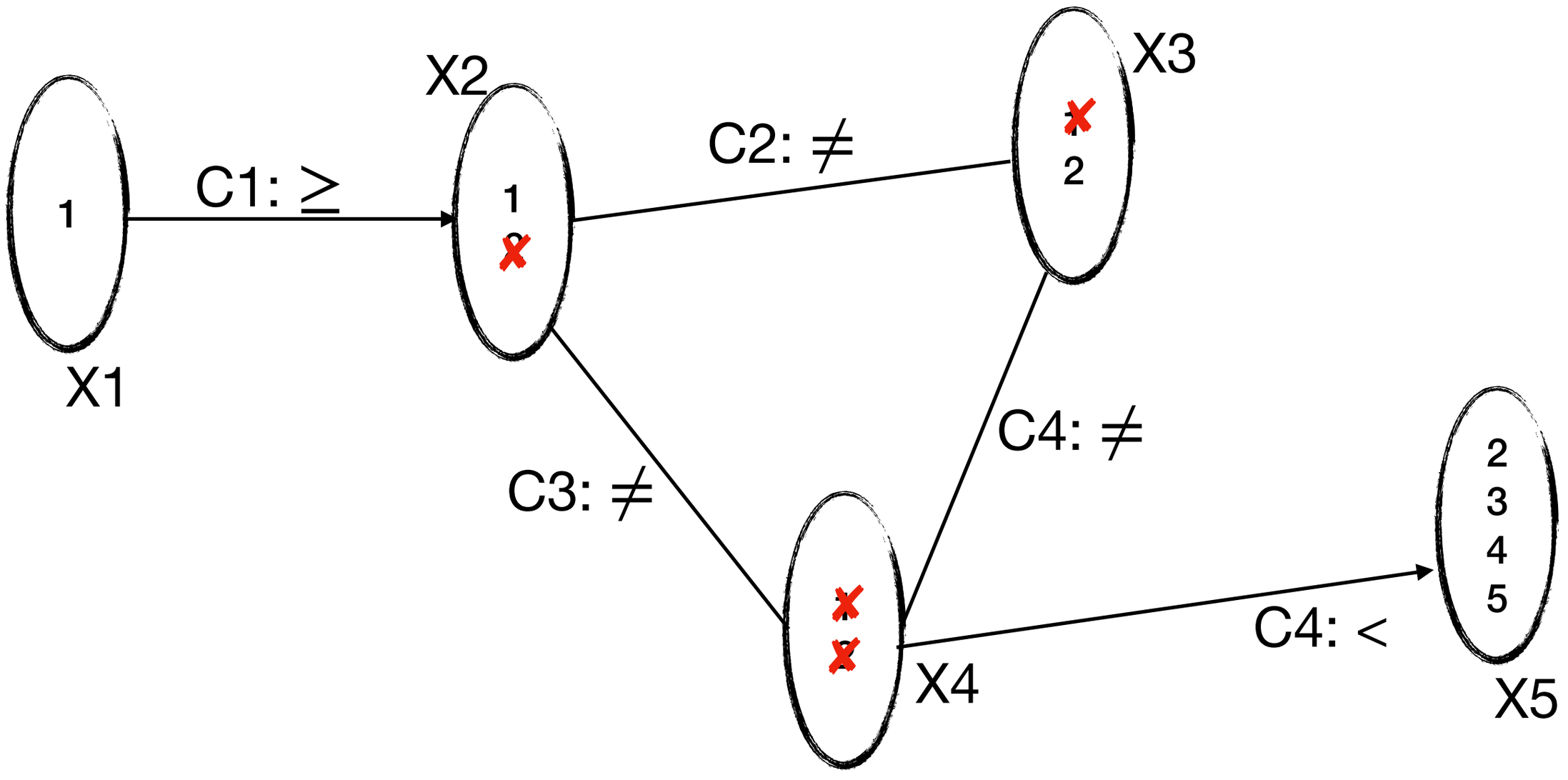
Solving - propagation (local consistency)



Situation 3

Constraint Programming

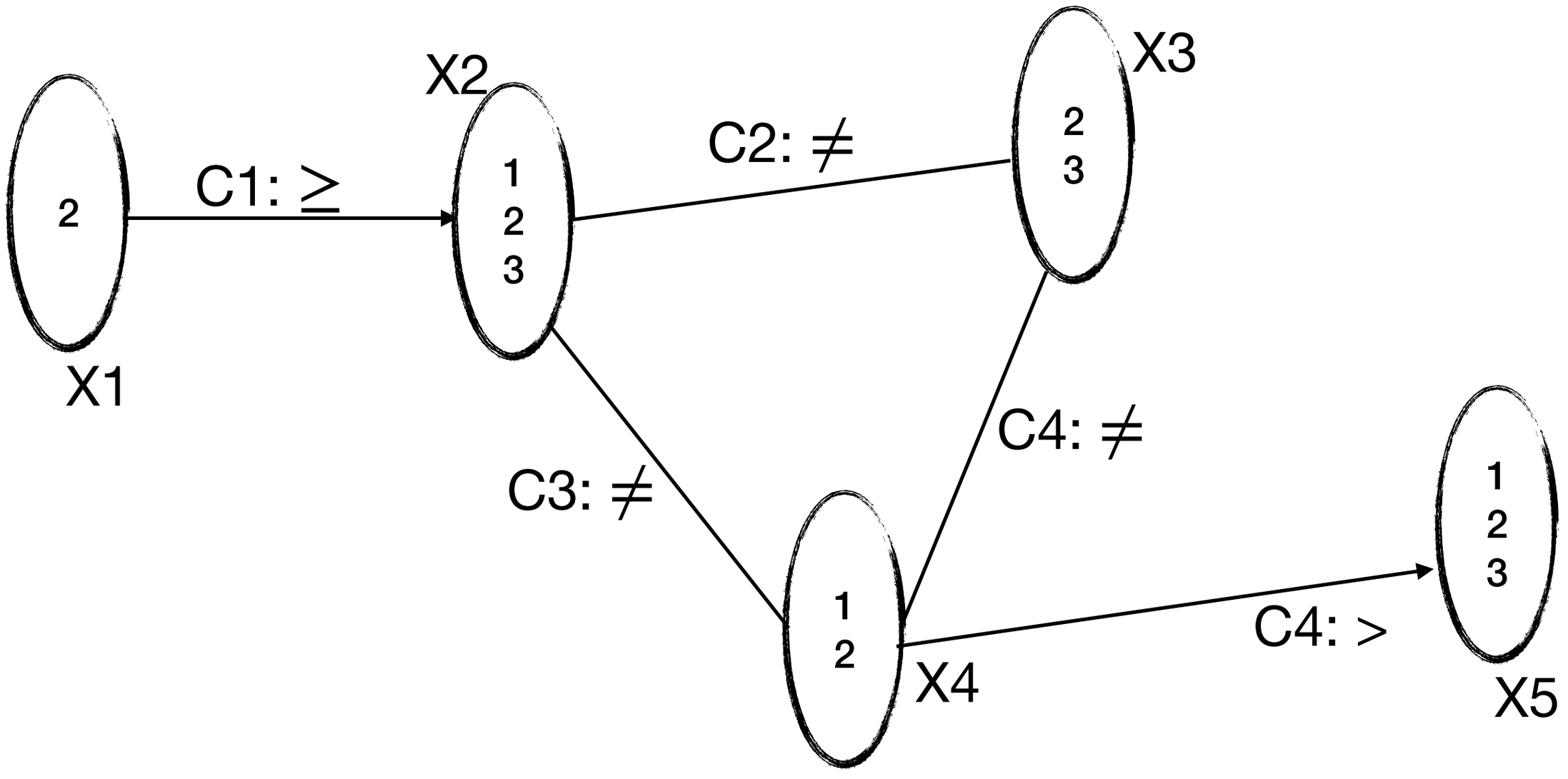
Solving - propagation (local consistency)



Situation 3 (domain wape-out)

Constraint Programming

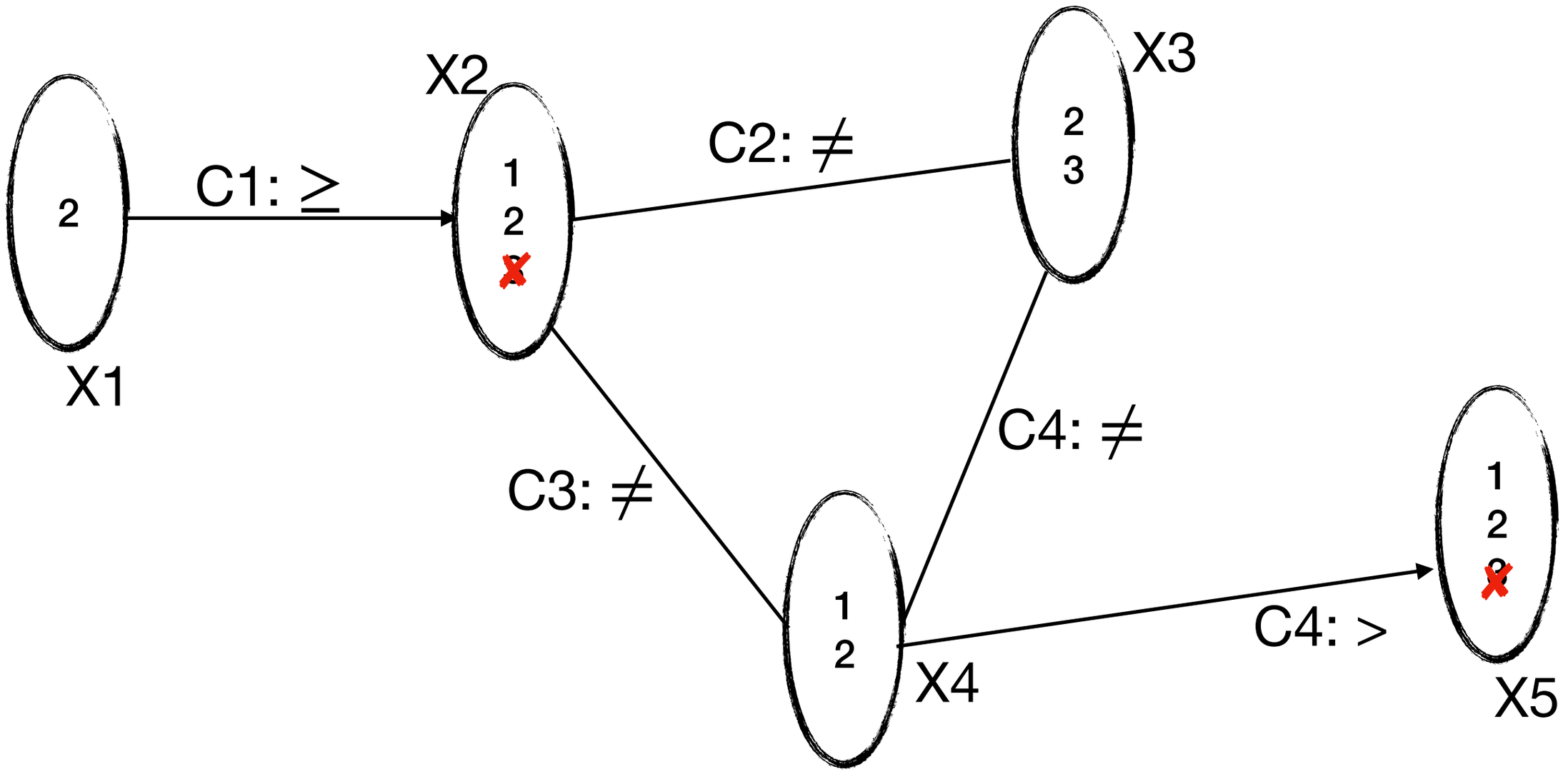
Solving - propagation (local consistency)



Situation 4

Constraint Programming

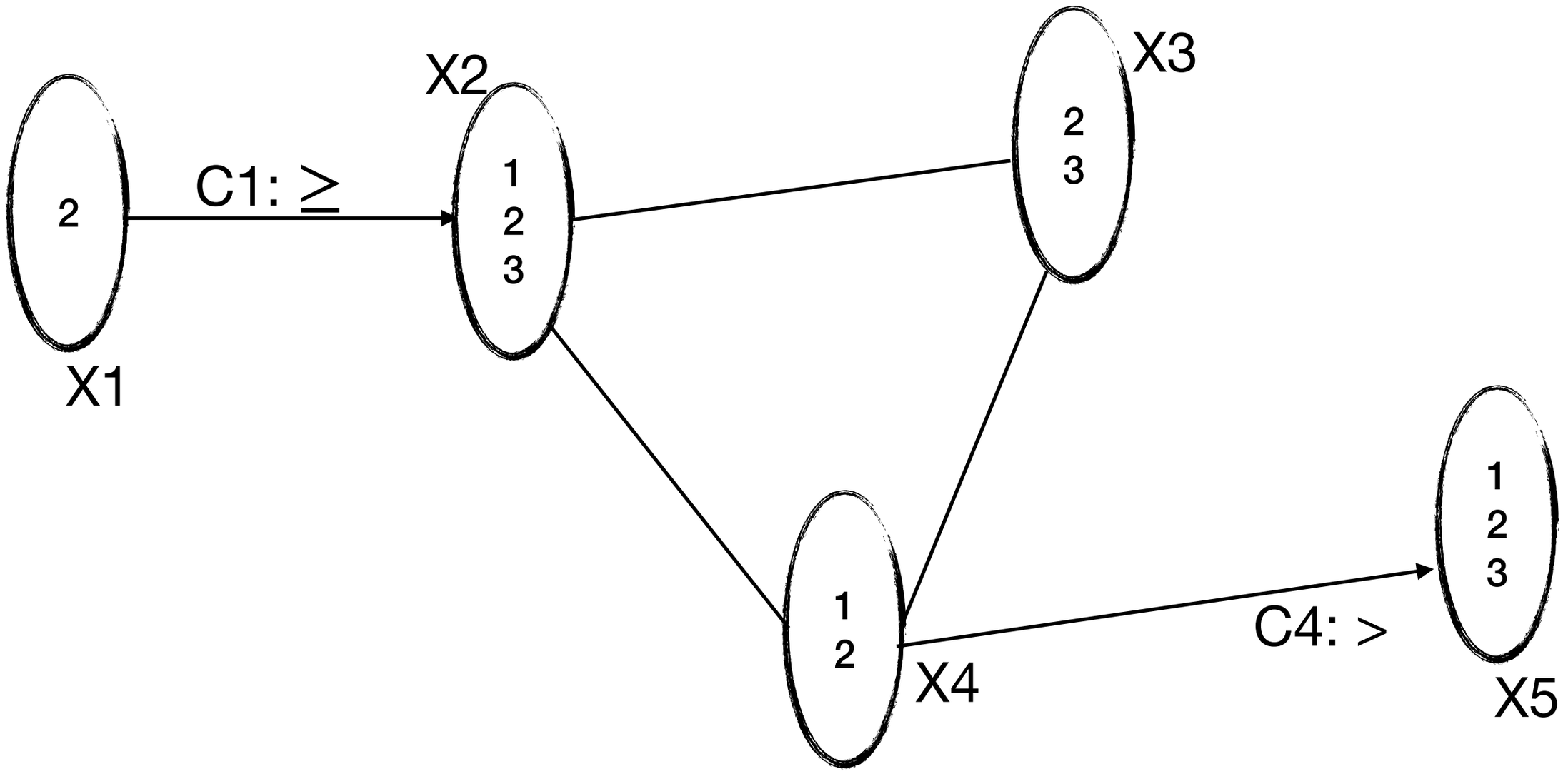
Solving - propagation (local consistency)



Situation 4 (domain reduction)

Constraint Programming

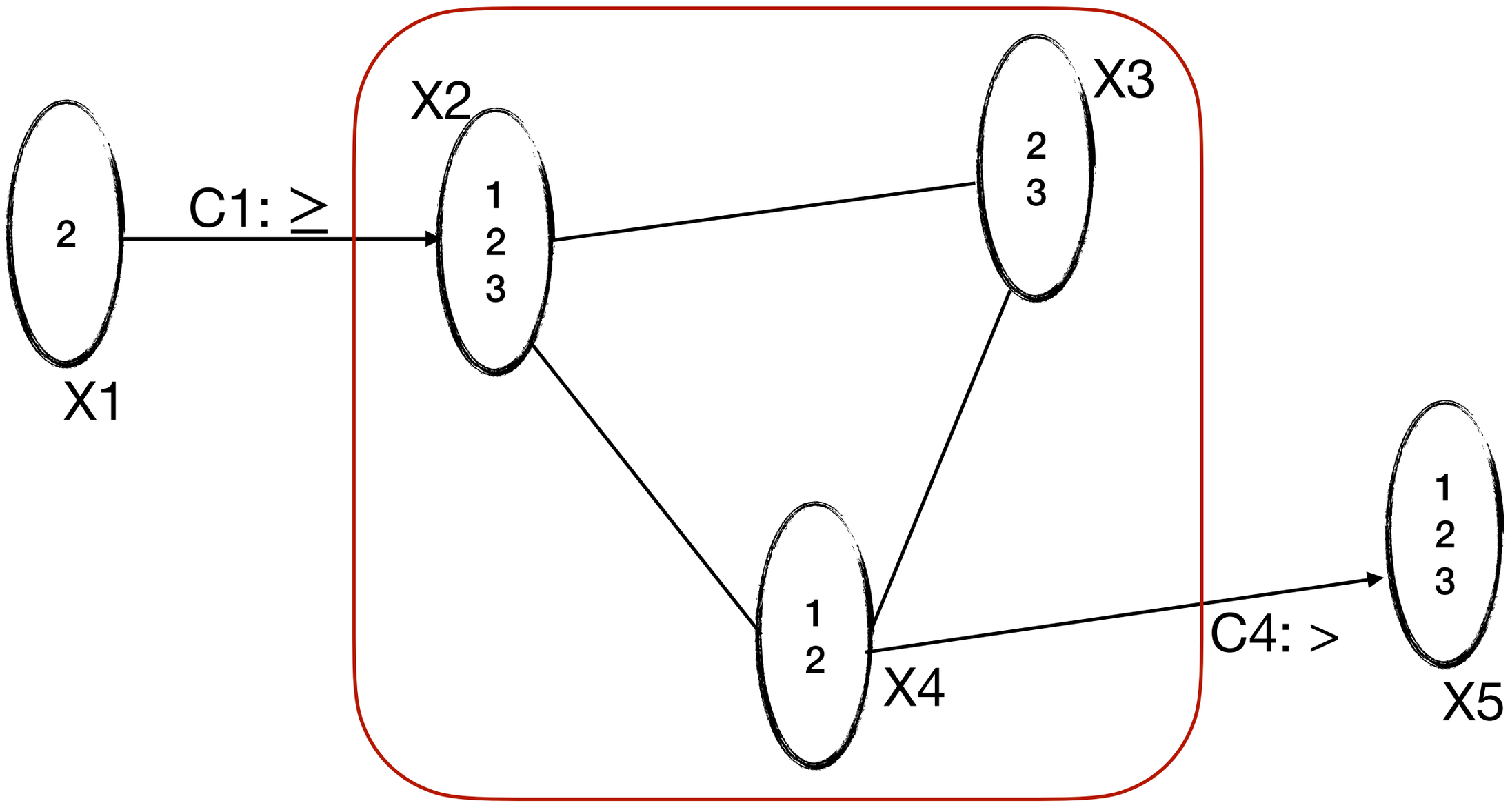
Solving - propagation (local consistency)



Situation 5

Constraint Programming

Solving - propagation (local consistency)



Situation 5 (Global constraints)

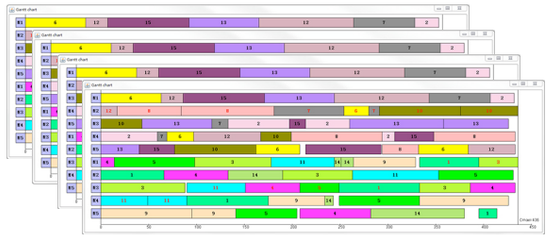
Constraint Programming

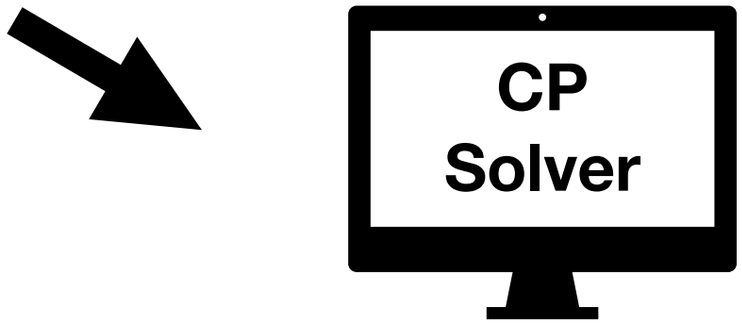
Global constraints

Constraint Programming

Global constraints

Scheduling instances

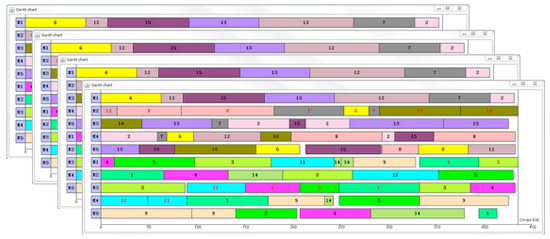


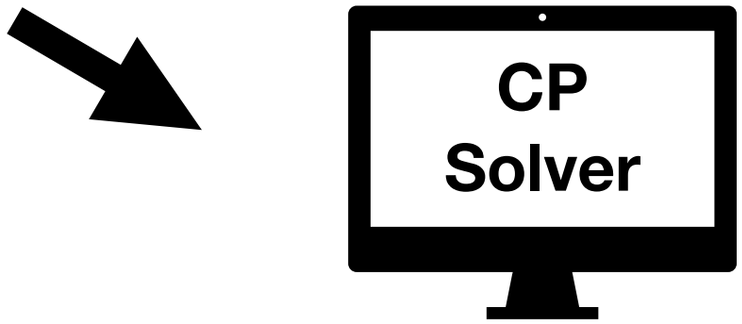


Constraint Programming

Global constraints

Scheduling instances



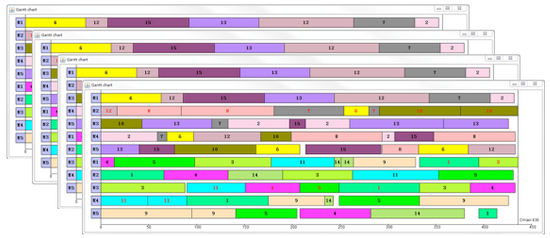


`Cumulative`

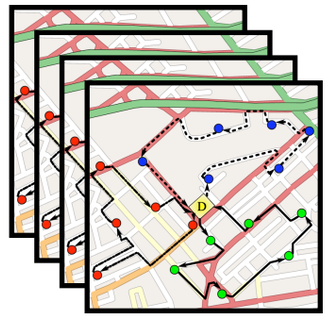
Constraint Programming

Global constraints

Scheduling instances



Vehicule routing instances

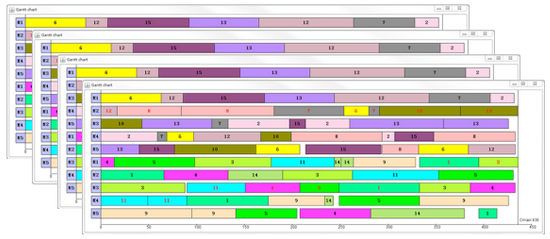


`Cumulative`

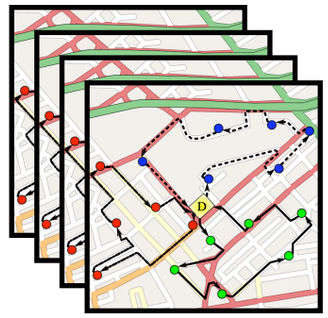
Constraint Programming

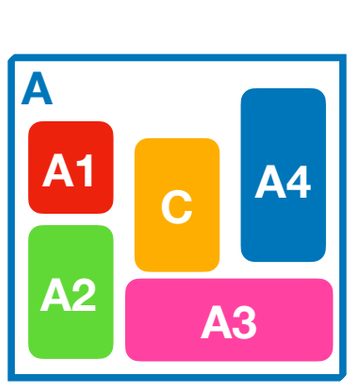
Global constraints

Scheduling instances



Vehicule routing instances



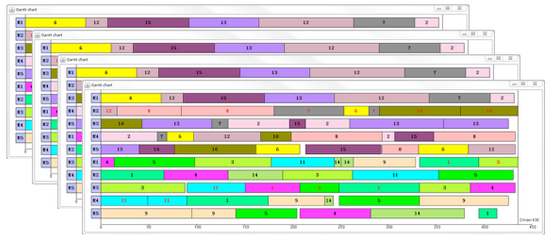


`Cumulative`

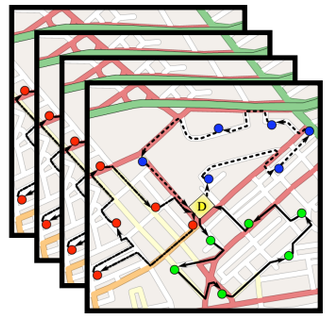
Constraint Programming

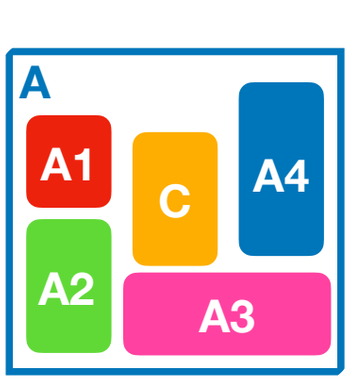
Global constraints

Scheduling instances



Vehicule routing instances



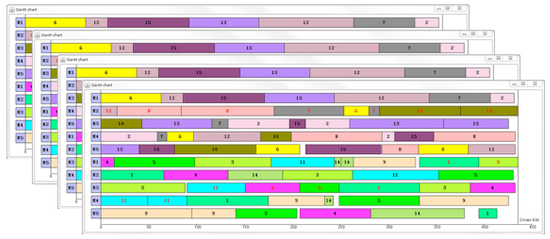


`Cumulative`

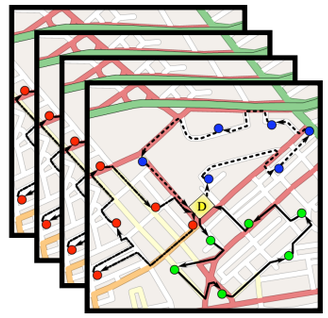
Constraint Programming

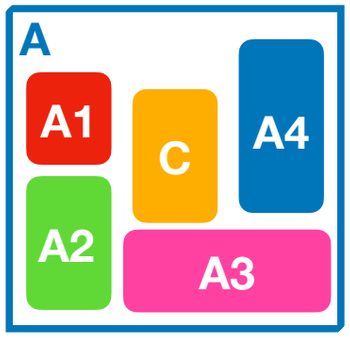
Global constraints

Scheduling instances

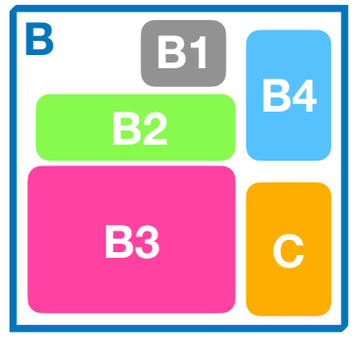


Vehicule routing instances





CP Solver

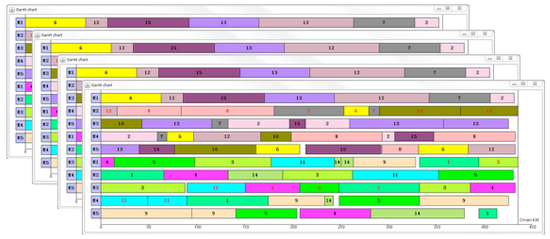



```
Cumulative
Alldifferent
```

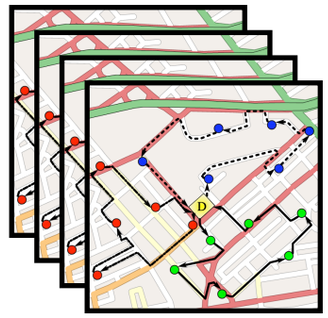
Constraint Programming

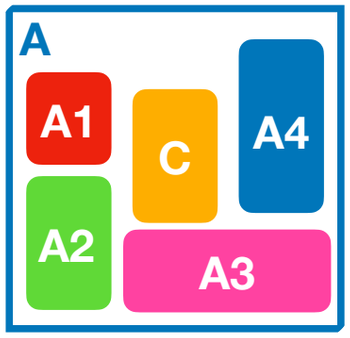
Global constraints

Scheduling instances

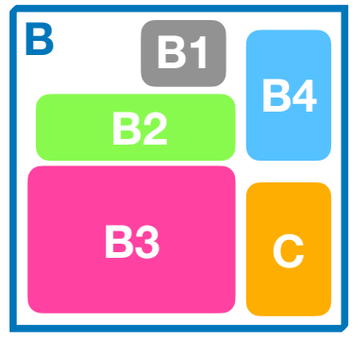


Vehicule routing instances





CP Solver



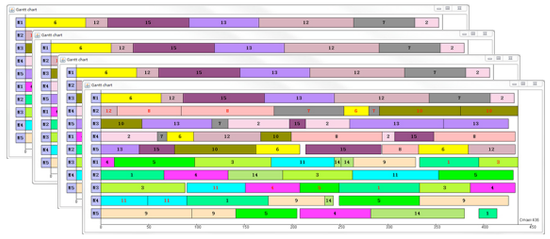

```
Cumulative
Alldifferent
```

•
•
•

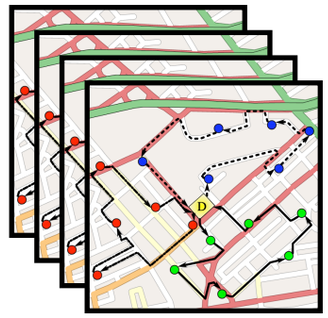
Constraint Programming

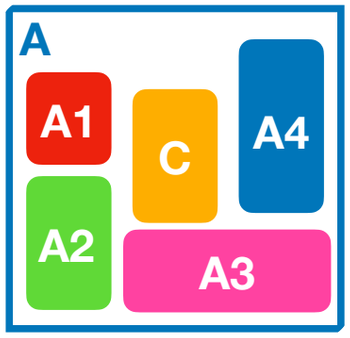
Global constraints

Scheduling instances

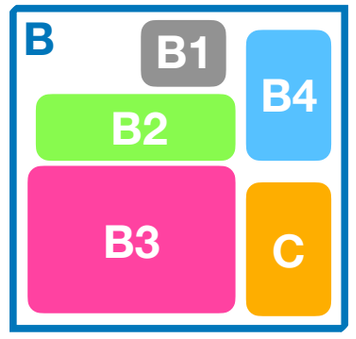


Vehicule routing instances





CP Solver



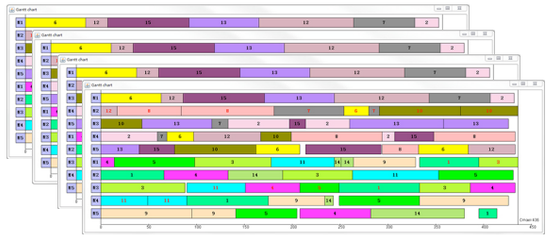

```
Cumulative
Alldifferent
Sum
knapsack
Element
GlobalCardinality
Regular
…
```

•
•
•

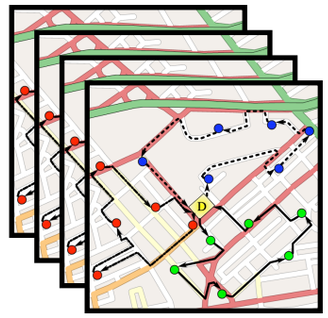
Constraint Programming

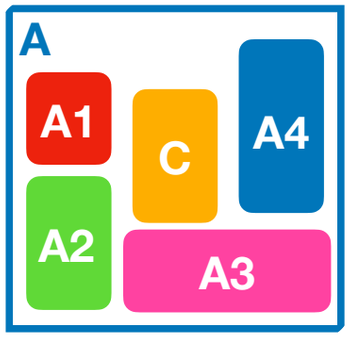
Global constraints

Scheduling instances



Vehicule routing instances

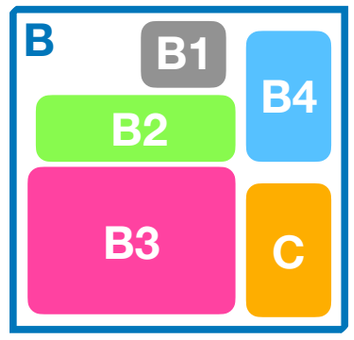




CP Solver

toolbox

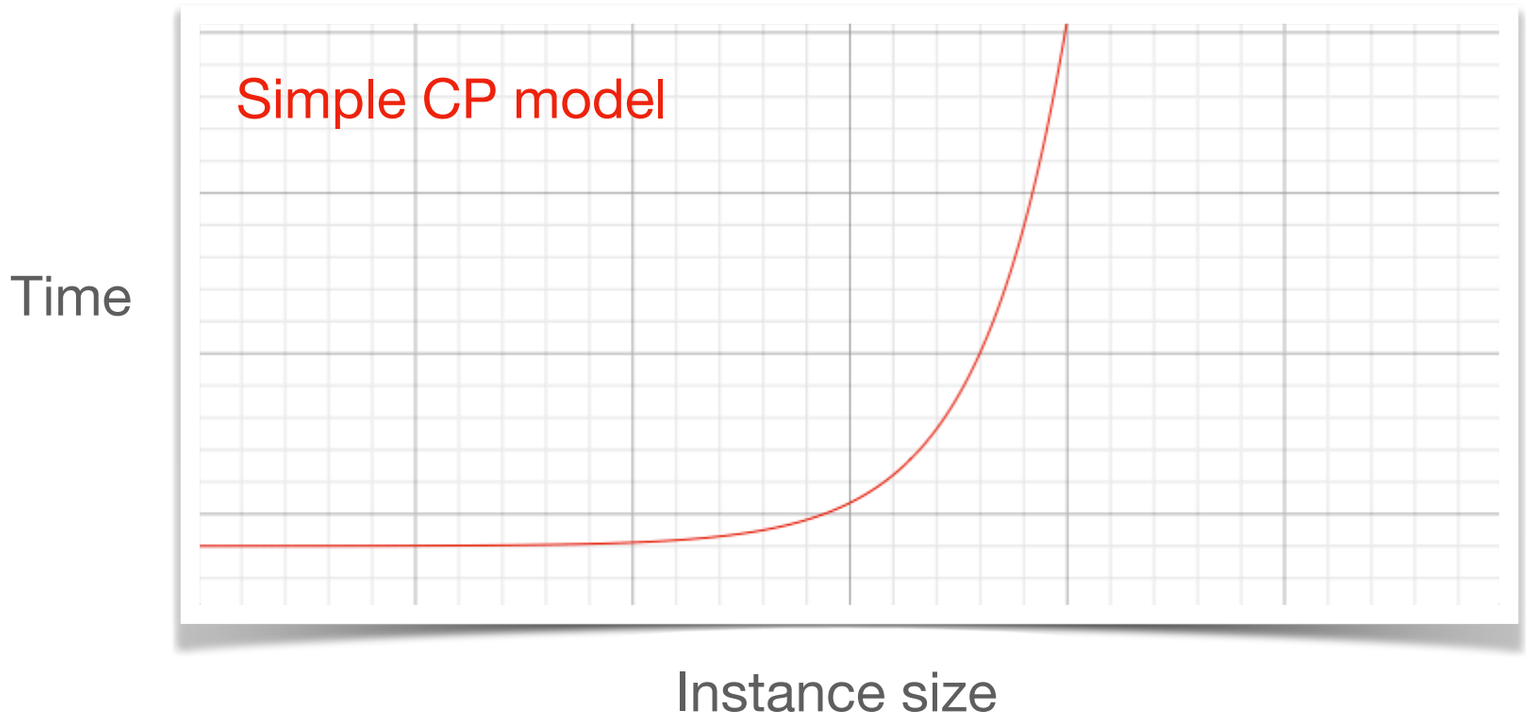
```
Cumulative
Alldifferent
Sum
knapsack
Element
GlobalCardinality
Regular
…
```



•
•
•

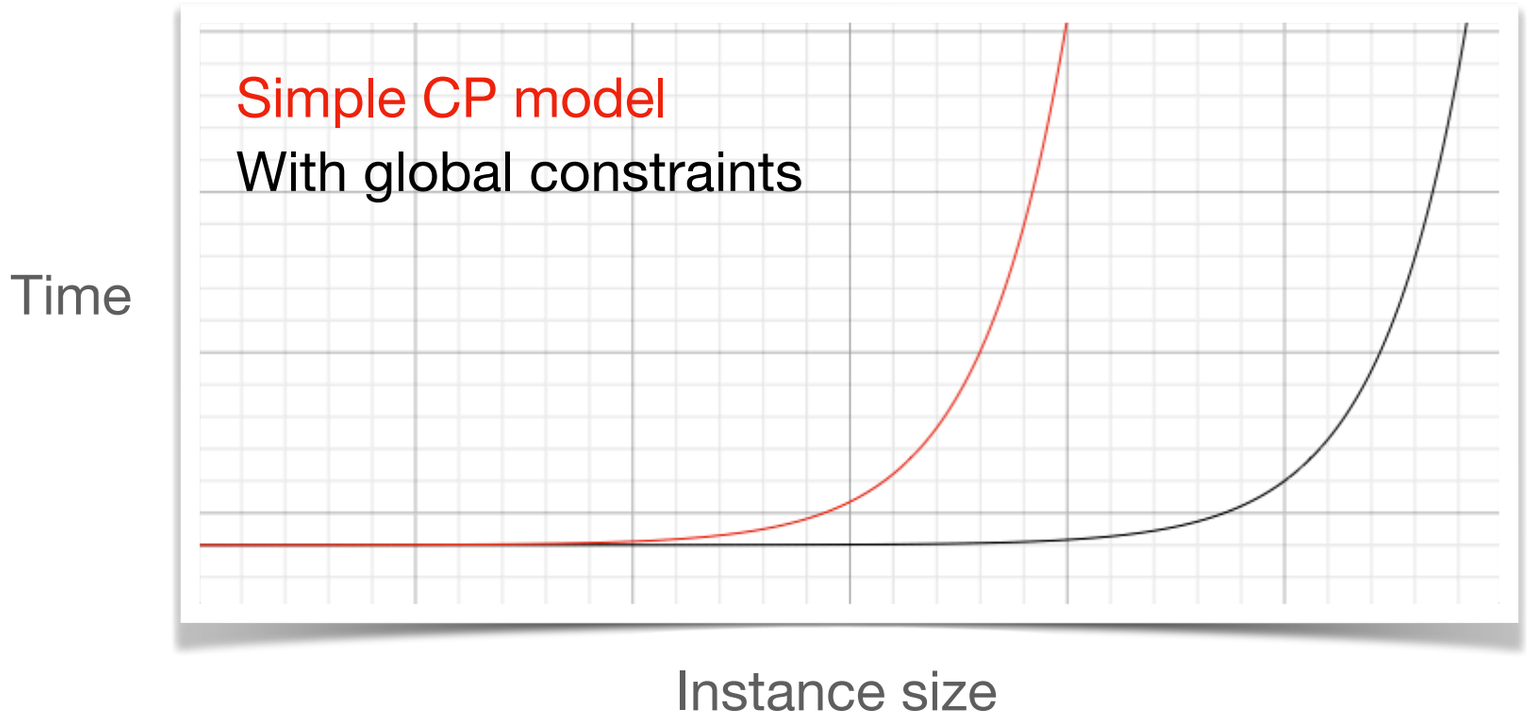
Constraint Programming

Global constraints



Constraint Programming

Global constraints



Constraint Programming

Solving - Forward Checking (FC)

- Checks if a variable's assignment is consistent
- Prunes inconsistent values
- Doesn't necessarily launch assignment propagation
- Simpler and more lightweight compared to MAC
- Suitable for medium-sized constraint problems
- Prunes values after an assignment but not necessarily after each assignment

Constraint Programming

Solving - Forward Checking (FC)

Constraint Programming

Solving - Forward Checking (FC)

```
FC(<X,D,C>, I):      [Haralick&Eliott80, Van Hentenryck89]

If I is complete then return true

Select a variable Xi not in I

ForEach v in D(Xi) do

   Remove all inconsistent (v', Xj) with (v,Xi)

   If no wape-out then

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

   Restore all values pruned because of (v,Xi)

return false
```

Constraint Programming

Solving - Forward Checking (FC)

Constraint Programming

Solving - Forward Checking (FC)

```
FC(<X,D,C>, I):      [Haralick&Eliott80, Van Hentenryck89]

If I is complete then return true

Select a variable Xi not in I

ForEach v in D(Xi) do

   Remove all inconsistent (v', Xj) with (v,Xi)

   If no wape-out then

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

   Restore all values pruned because of (v,Xi)

return false
```

Constraint Programming

Solving - Maintaining Arc Consistency (MAC)

- Enforces arc consistency
- Reduces the search space
- Prunes inconsistent values
- Utilizes an arc consistency restoration algorithm
- Ensures consistency after each assignment
- Suitable for complex constraint problems
- May be more computationally intensive

Constraint Programming

Solving - Maintaining Arc Consistency (MAC)

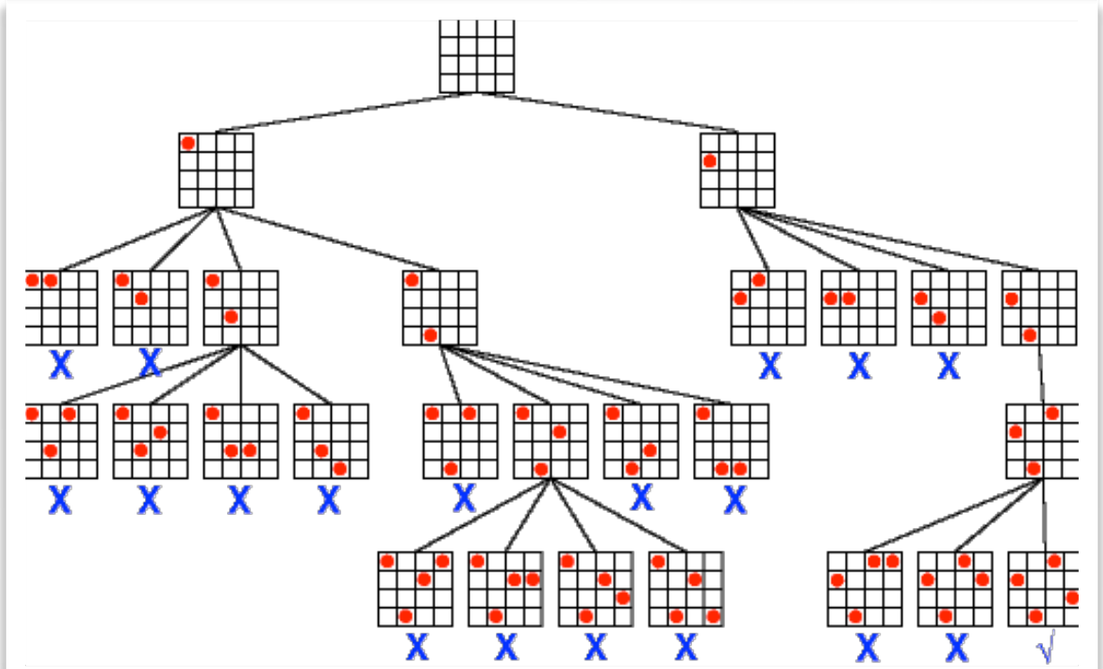
Constraint Programming

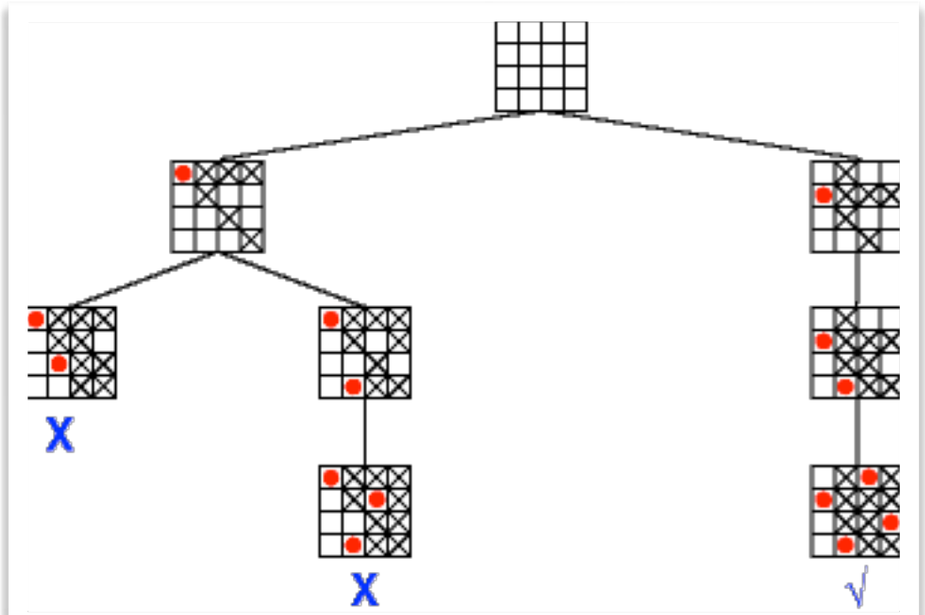
Solving - Maintaining Arc Consistency (MAC)

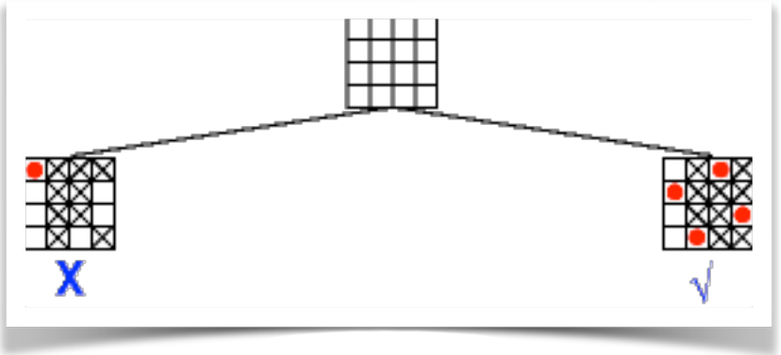
```
MAC(<X,D,C>, I):                                    [Sabin&Freuder94]

If AC(<X,D,C>) then

  If I is complete then return true

  Select a variable Xi not in I

  Select a value v in D(Xi)

  return MAC(<X,D,C>, I union <Xi, v>)
         OR MAC(<X,D,C union Xi!=v >, I)

return false
```

Constraint Programming

Solving - BT, MAC, FC, When to Use Each







Constraint Programming

Solving - BT, MAC, FC, When to Use Each

- **Use BT** when dealing with various types of problems but be cautious of large search spaces

- **Use FC** for medium-sized problems when you need consistency checking and some pruning

- **Use MAC** for complex problems when you require strong consistency enforcement and are willing to invest more computation

Constraint Programming

Solving - BT, MAC, FC, When to Use Each

- **Use BT** when dealing with various types of problems but be cautious of large search spaces

- **Use FC** for medium-sized problems when you need consistency checking and some pruning

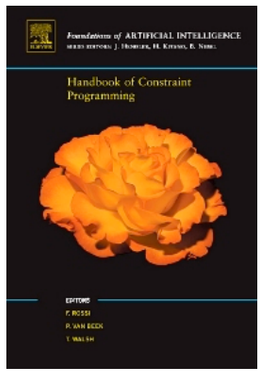
- **Use MAC** for complex problems when you require strong consistency enforcement and are willing to invest more computation

- The choice depends on the problem's size, complexity, and computational resources available.

- It's a trade-off between search efficiency, consistency enforcement, and computational resources.

Constraint Programming

References & links



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

- ACP: Association for Constraint Programming

- Guide to Constraint Programming

- CP conference Series

- Global Constraint Catalog