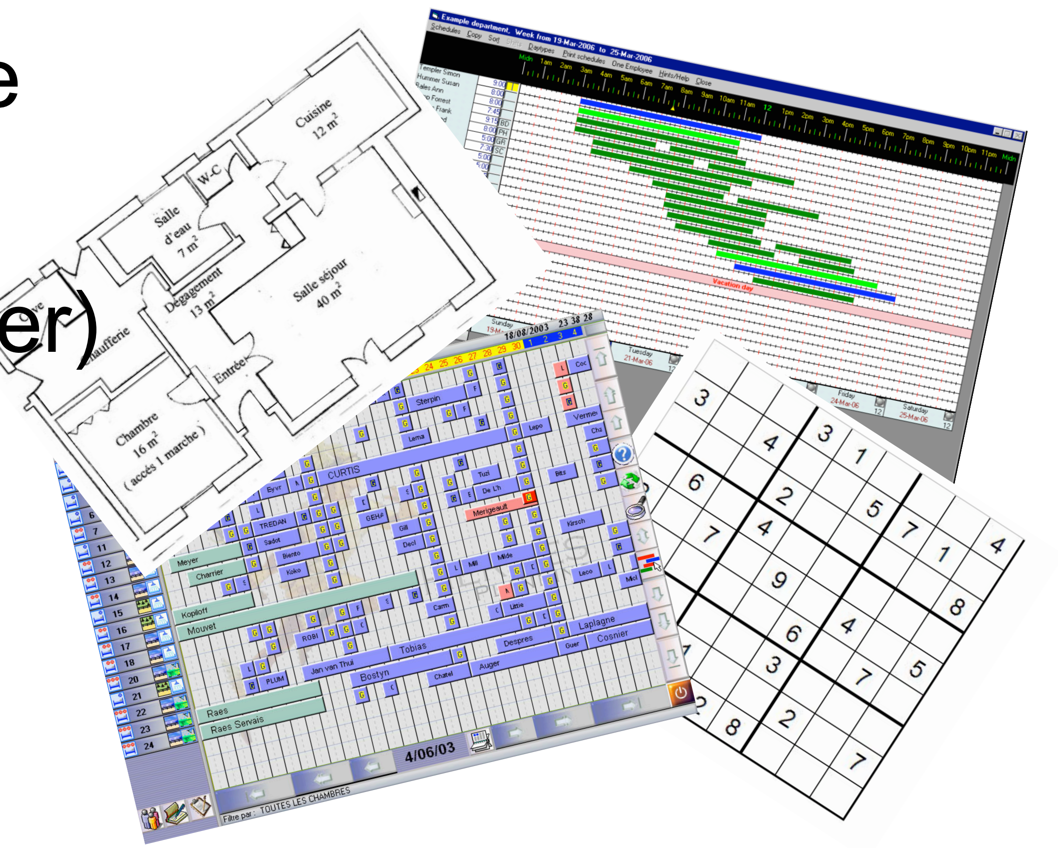
Using CP When You Don't Know CP

Christian Bessiere

LIRMM

(CNRS/U. Montpellier)



An illustrative example

5-rooms flat (bedroom, bath, kitchen, sitting, dining) on a 6-room pattern

Constraints of the builder:

- Kitchen and dining must be linked
- Bath and kitchen must have a common wall
- Bath must be far from sitting
- Sitting and dining form a single room

The pattern:

north -west	north	north-east
south-west	south	south-east

Problem

- How to propose all possible plans?

➔ a constraint network that encodes the constraints of the builder

Library of constraints

nw	n	ne
sw	s	se

- Constraints :
  - X ≠ Y, X = Y
  - **`Next`**(X,Y) = {
    (nw,n),(nw,sw),(n,nw),(n,ne),(n,s),
    (ne,n),(ne,se),(sw,nw),(sw,s),(s,sw),
    (s,n),(s,se),(se,s),(se,ne) }
  - **`Far`**(X,Y) = {
    (nw,ne),(nw,s),(nw,se),(n,sw),(n,se),
    (ne,nw),(ne,sw),(ne,s),(sw,n),(sw,ne),
    (so,se),(s,nw),(s,ne),(se,nw),(se,n),(se,sw) }

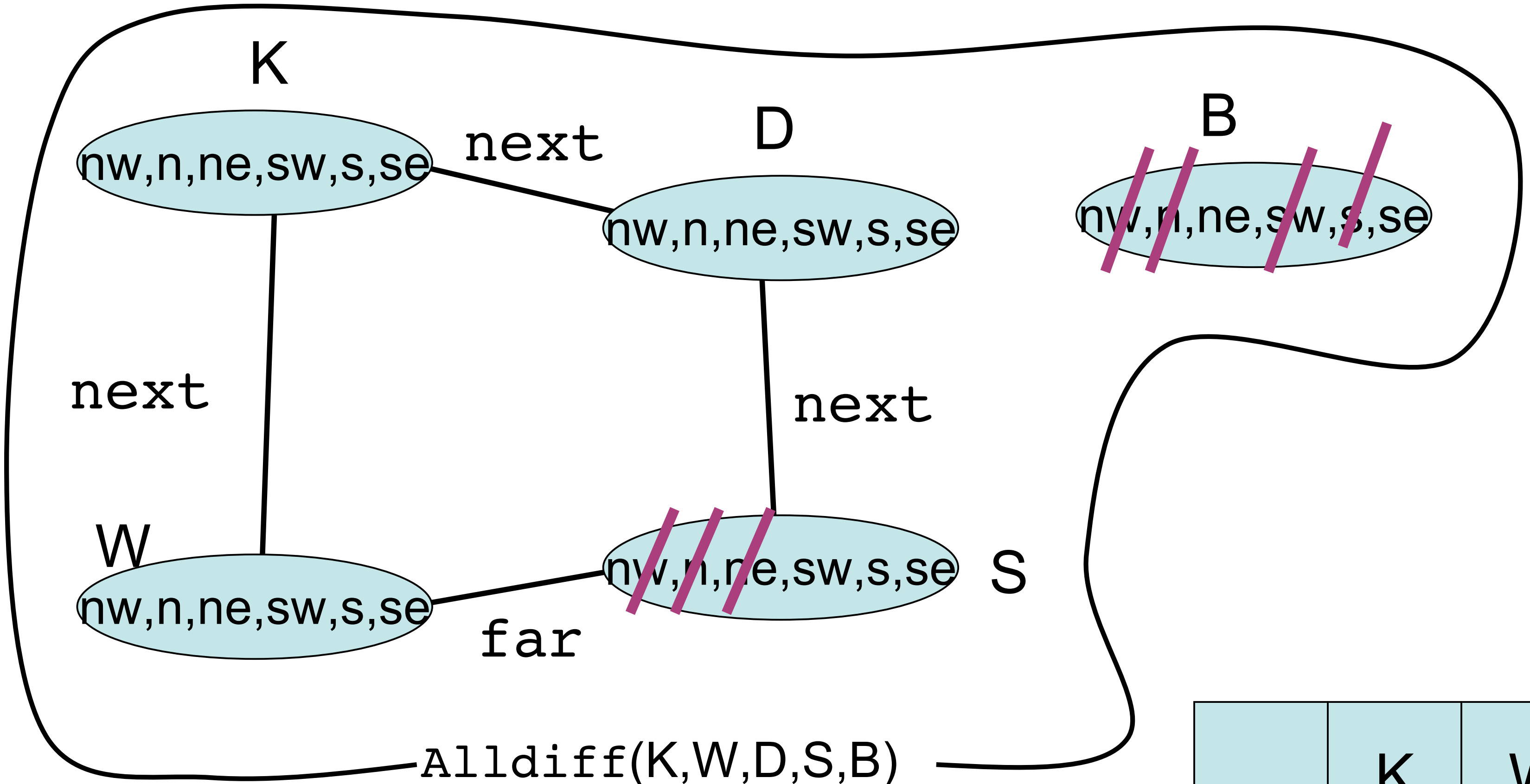
A possible viewpoint (variables, domains)

- Variables :
  - B (bedroom),
  - W (washroom),
  - K (kitchen),
  - S (sitting),
  - D (dining)

nw	n	ne
sw	s	se

- Domains : {nw,n,ne,sw,s,se}

A constraint network

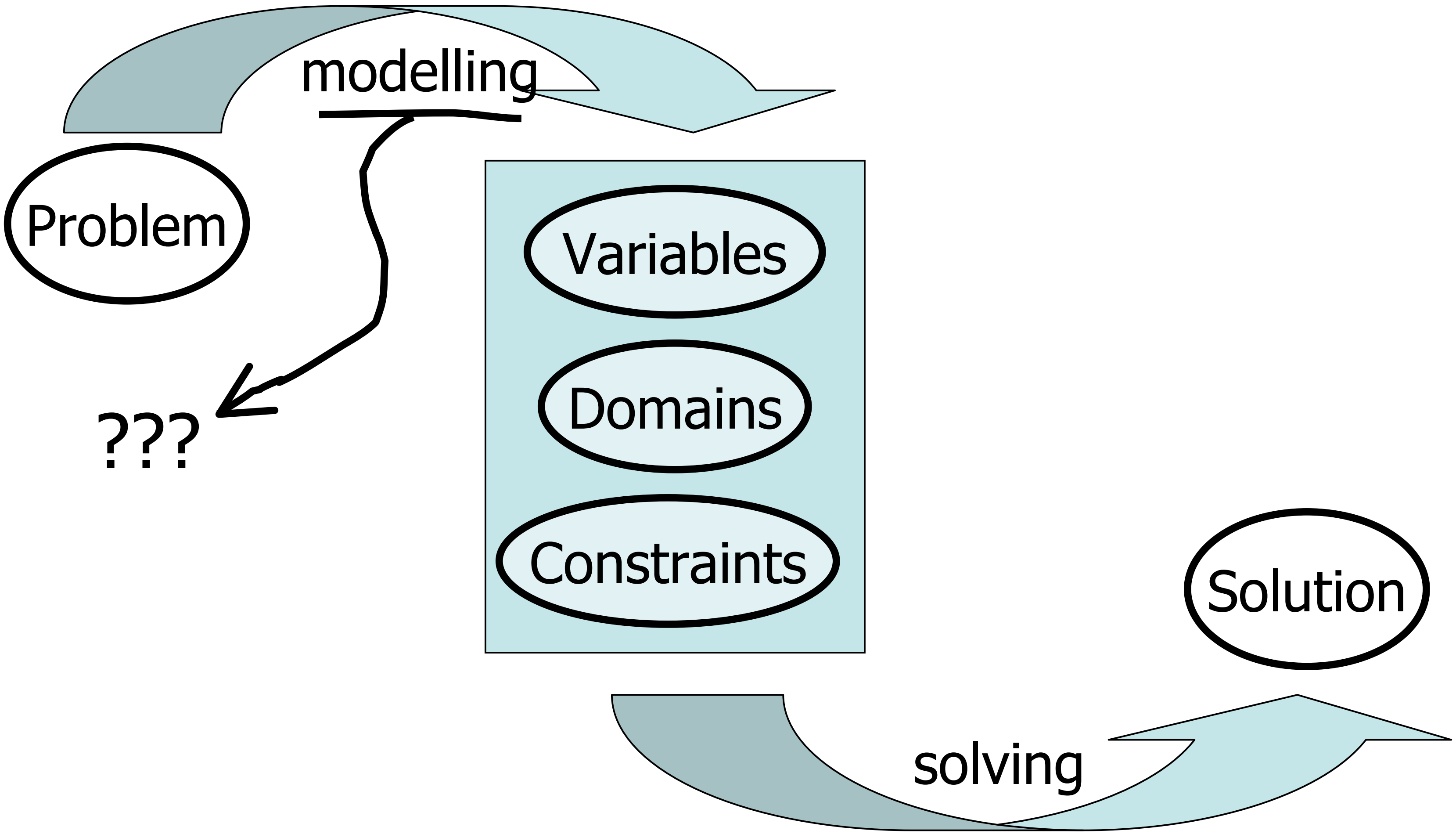


Client wishes: Bedroom: east
Sitting: south

a solution →

	K	W
S	D	B

Constraint Programming



Modelling
(“*it’s an art, not a science*”)

- In the 80s, it was considered as trivial
  - Zebra problem (Lewis Carroll) or random problems
- But on “real” problems:
  - Which variables ? Which domains ?
  - Which constraints for encoding the problem?
- And efficiency?
  - Which constraints for speeding up the solver?
    - Global constraints, symmetries…

➔ All is in the expertise of the user

If you’re not an expert?

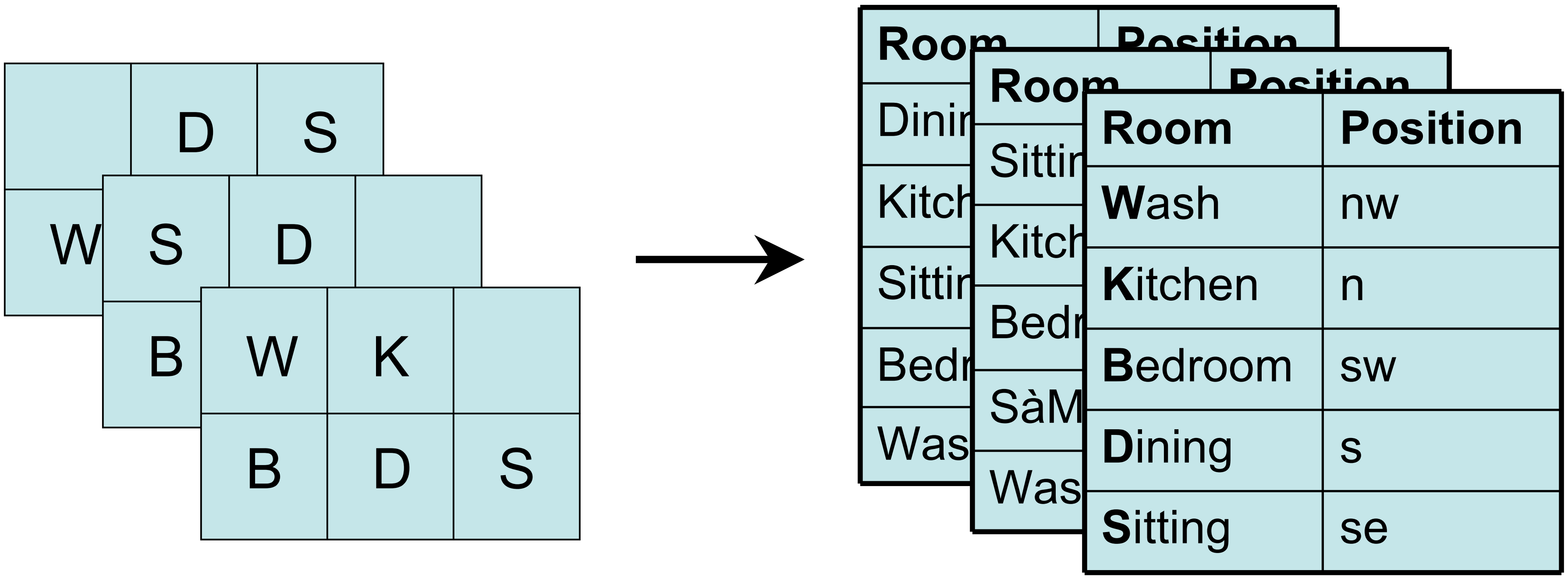
1. Choice of variables/domains

2. Constraint acquisition

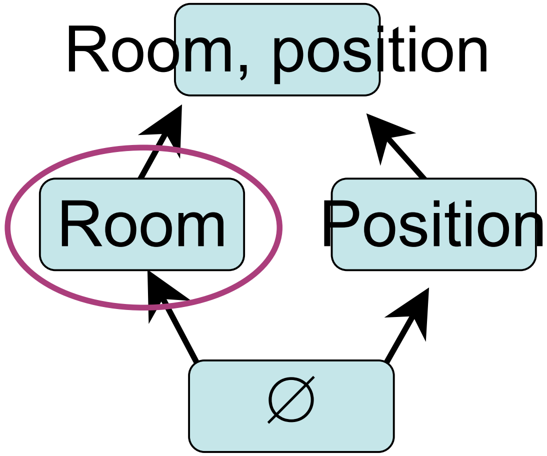
3. Improve a basic model

Choice of variables/domains (viewpoints)

- From ***historical*** data (former solutions)
- Solutions described in tables (flat data)



Extract viewpoints



$X_{Wash} \in \{nw, n, ne, sw, s, se\}$

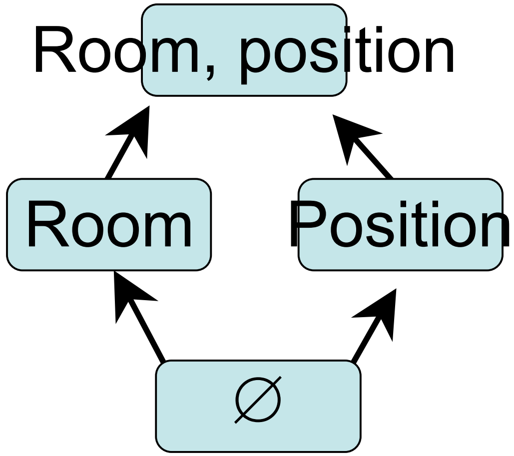
….

$X_{Sitting} \in \{nw, n, ne, sw, s, se\}$

Room	**Position**
Wash	nw
Kitchen	n
Bedroom	sw
Dining	s
Sitting	se

Extract viewpoints

- Two viewpoints:
  - $X_B,\ldots,X_S \in \{nw,n,ne,sw,s,se\}$
  - $X_{nw},\ldots,X_{se} \in \{W,B,K,D,S,\nabla\}$



- Trivial viewpoints:
  - $X_1,\ldots,X_5 \in \{B\text{-}nw,B\text{-}n,B\text{-}sw,\ldots,$ $S\text{-}s,S\text{-}se\}$
  - $X_{B\text{-}nw},\ldots,X_{S\text{-}se} \in \{0,1\}$

Room	Position
wash	nw
kitchen	n
bedroom	sw
dining	s
sitting	se

Connect viewpoints

- VP1: $X_B,\ldots,X_S \in \{nw,n,ne,sw,s,se\}$
- VP2: $X_{nw},\ldots,X_{se} \in \{B,W,K,D,S,\nabla\}$

- Channelling constraints:
  - $X_B = nw \leftrightarrow X_{nw} = B$

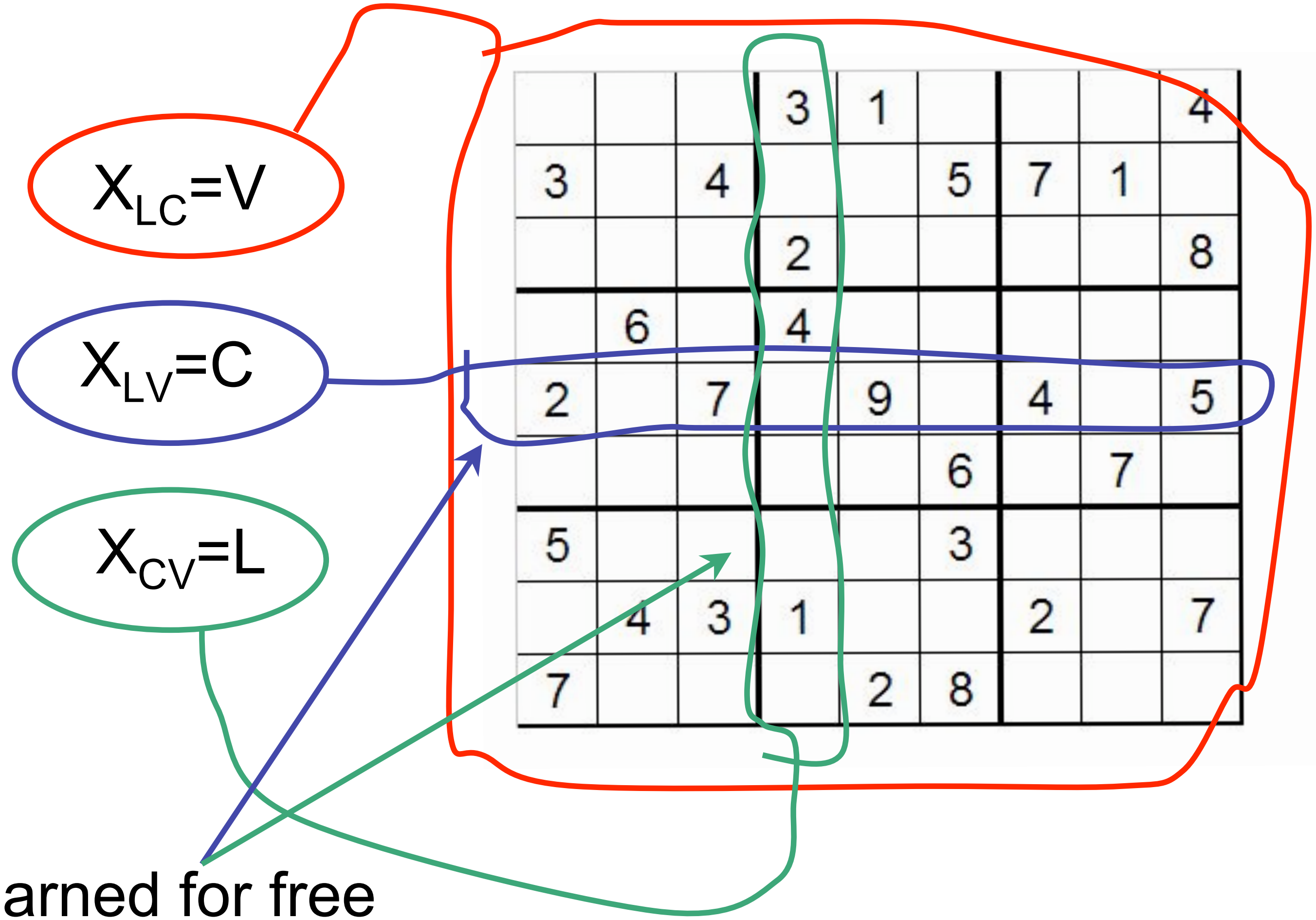
➔“nw” is taken at most once in VP1

➔ `alldiff`($X_B,\ldots,X_S$) is a constraint in VP1

[like in Law,Lee,Smith07]

Application: sudoku

L	C	V
L1	C4	3
L1	C5	1
L2	C1	3
L2	C3	4
L3	C4	2
L3	C9	8
…	…	…



Alldiffs learned for free

Connect viewpoints

- We can derive more than just alldiff
- **Cardinality** constraints can be detected

- Example: a timetabling in which 3 math courses are given

➔ one of the viewpoints will contain 3 variables representing these 3 courses

➔ In all other viewpoints, we can put a cardinality constraint forcing value “math” to be taken 3 times

If you're not an expert?

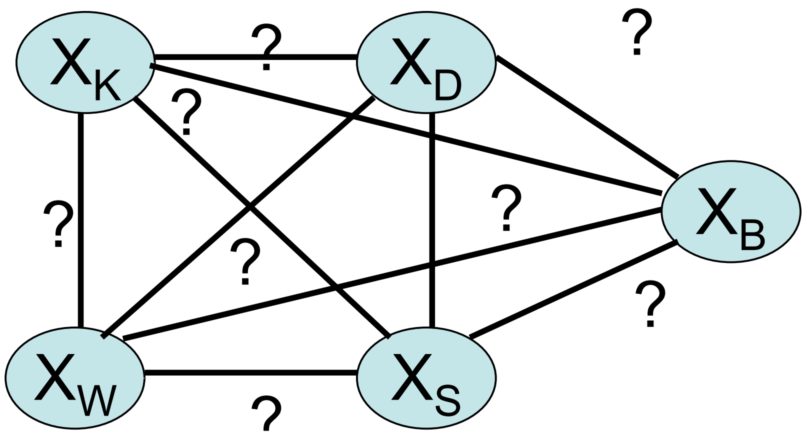
- Choice of variables/domains
- **Constraint Acquisition**
  - Space of networks
  - Redundancy
  - Queries
- Improve a basic model

Acquire constraints

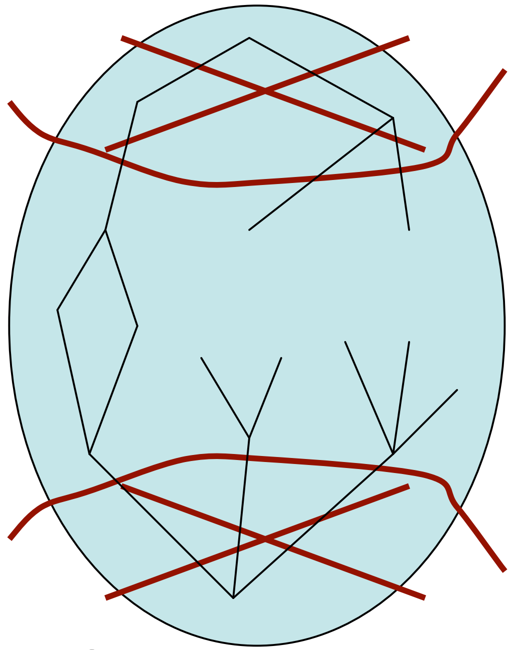
- The user doesn't know how to specify constraints
- She knows how to discriminate solutions from non-solutions
  - Ex: valid flat vs invalid flat

➔Use of machine learning techniques
  - Interaction by examples (positive *e*+ or negative *e*-)
  - Acquisition of a network describing the problem

Space of possible networks



Some negative accepted



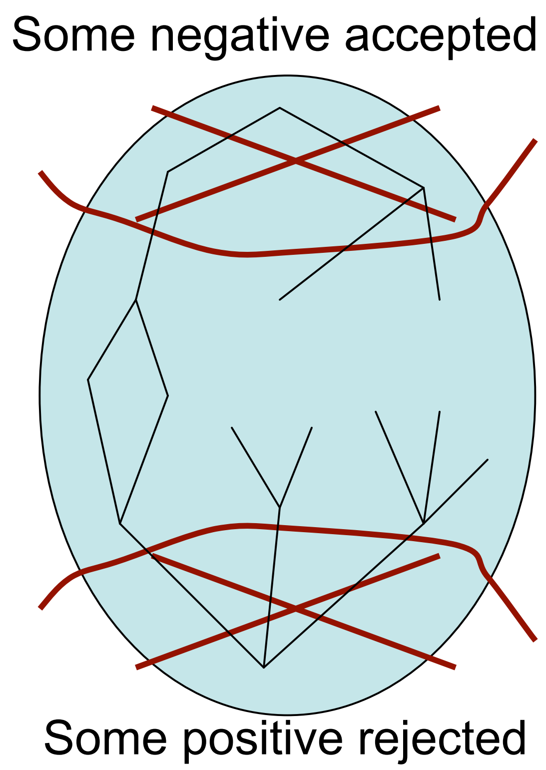
Some positive rejected

- Language :
  ? $\rightarrow$ { =, $\neq$, `next`, `far`}
- Bias :
  X_S=X_W; `next`(X_S,X_B);… …;
$X_K$$\neq$$X_D$; `far`($X_K$,$X_D$)

Compact SAT encoding

- A SAT formula $\mathcal{K}$ representing all possible networks:
  - Each constraint c_i
    $\rightarrow$ a literal b_i
  - Models($\mathcal{K}$) = version space
  - Example *e*- rejected by $\{c_i, c_j, c_k\}$
    $\rightarrow$ a clause $(b_i \vee b_j \vee b_k)$
  - Example *e*+ rejected by c_i
    $\rightarrow$ a clauses $(\neg b_i)$

- $m \in$ models($\mathcal{K}$)
  $\Rightarrow \varphi(m) = \{c_i \mid m(b_i)=1\}$ accepts all positive examples and rejects all negative examples



Reduce the space

$C(X_K,X_D)$: $\neq$, $\cancel{=}$, ~~far~~, next

$C(X_D,X_S)$: $\neq$, $\cancel{=}$, ~~far~~, next

$C(X_K,X_S)$: $\neq$, $\cancel{=}$, far, ~~next~~

$\mathtt{next}(X_D,X_S) \vee \mathtt{far}(X_K,X_S)$

e^{+}_{1}

B	K	
S	D	W

e^{-}_{2}

K		W
B	S	D

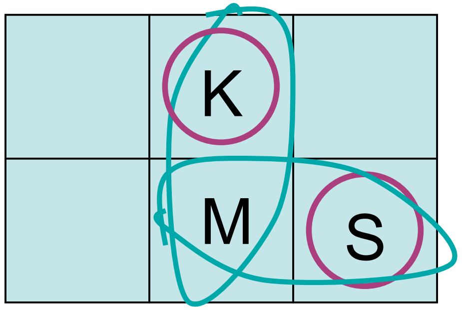
e^{+}_{3}

	K	B
S	D	W

e^{-}_{4}

S	K	B
	D	W

Redundancy



- Constraints are not ***independent***

- "$\mathtt{next}(X_K,X_D) \wedge \mathtt{next}(X_D,X_S) \Rightarrow \mathtt{far}(X_K,X_S)$"

- See local consistencies

- It's different from attribute-value learning

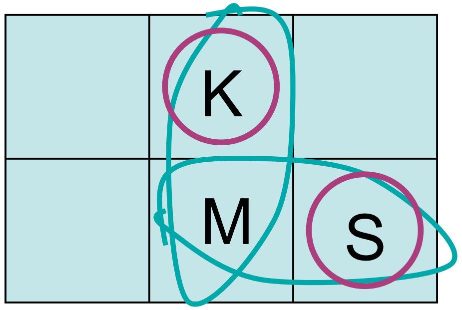
Redundancy

- Redundancy prevents *convergence*

➔ a set $\mathcal{R}$ of redundancy rules:

$\texttt{alldiff}(X_1,\ldots,X_n) \Rightarrow X_i \neq X_j,\ \forall i,j$

$\texttt{next}(X_K,X_D) \wedge \texttt{next}(X_D,X_S) \Rightarrow \texttt{far}(X_K,X_S)$



- In $\mathcal{K}$ we already have:
  - $\texttt{next}(X_D,X_S) \vee \text{far}(X_K,X_S)$
  - $\texttt{next}(X_K,X_D)$
- So, from $\mathcal{K}+\mathcal{R}$ we deduce $\texttt{far}(X_K,X_S)$

- Version space = Models($\mathcal{K}+\mathcal{R}$)
  - Good properties when $\mathcal{R}$ is complete

Queries
(active learning)

- Examples often lead to little new information (eg, negative plan with kitchen far from dining)

- The system will propose examples (queries) to speed up convergence

- Example *e* rejected by *k* constraints from the space
  - *e* positive ⇒ *k* constraints discarded from the space
  - *e* negative ⇒ a clause of size *k*

- Good query = example which reduces the space as much as possible whatever the answer

Queries

- Negative example *e1*:
  ➔ $cl_{e1} = b_1 \vee \ldots \vee b_k \in \mathcal{K}+\mathcal{R}$
  – find $m \in$ models($\mathcal{K}+\mathcal{R}$) such that a single literal b_i in cl_{e1} is false
  – find $e2 \in sol(\varphi(m))$:
    → e2 violates only constraint c_i
    ➔ b_i or $\neg b_i$ will go in $\mathcal{K}$

- If sol($\varphi(m)$)=∅: any *conflict-set* is a new redundancy rule ➔ quick convergence

$\mathcal{K}+\mathcal{R}$

$b_1 \vee \ldots \vee b_k$

↓

m contains $\neg b_i$

↓

$\varphi(m)$

↓

$e2 \in$ sol($\varphi(m)$)

↓

Query: "*e2*" ?

An example of constraint acquisition in robotics

(by Mathias Paulin)

- The goal is to automate the burden of implementing elementary actions of a robot
- Elementary actions are usually implemented by hand by engineers (complex physic laws, kinetic momentum, derivative equations, etc.)

No need for a user

- Instead of interacting with a user, classification of examples will be done by a run of the robot with given values of its sensorimotor actuators
- If the action has correctly performed, this is positive
- With expensive humanoid robots, a simulator allows easy classification without actually running the robot

Elementary actions

- Each action has variables representing
  - the observed world before the action,
  - the power applied to each actuator
  - the world after the action
- Constraint acquisition will learn a constraint network on these variables such that its solutions are valid actions

Planning a task

- The overall goal is to build a plan composed of elementary actions
- The planning problem is solved by a CP solver
- It is convenient to encode actions as sub-CSPs

Tribot Mindstorms NXT

- 3 motors
- 4 sensors
- 5 elementary actions to combine
- Discretization of variables

Experiment

- Modelling by CONACQ
- Conacq generates a CHOCO model used by CSP-Plan [Lopez2003]

$\Rightarrow$ Objective : catch the mug!

are needed to see this picture.

If you're not an expert?

- Choice of variables/domains
- Constraint acquisition
- Improve the basic model

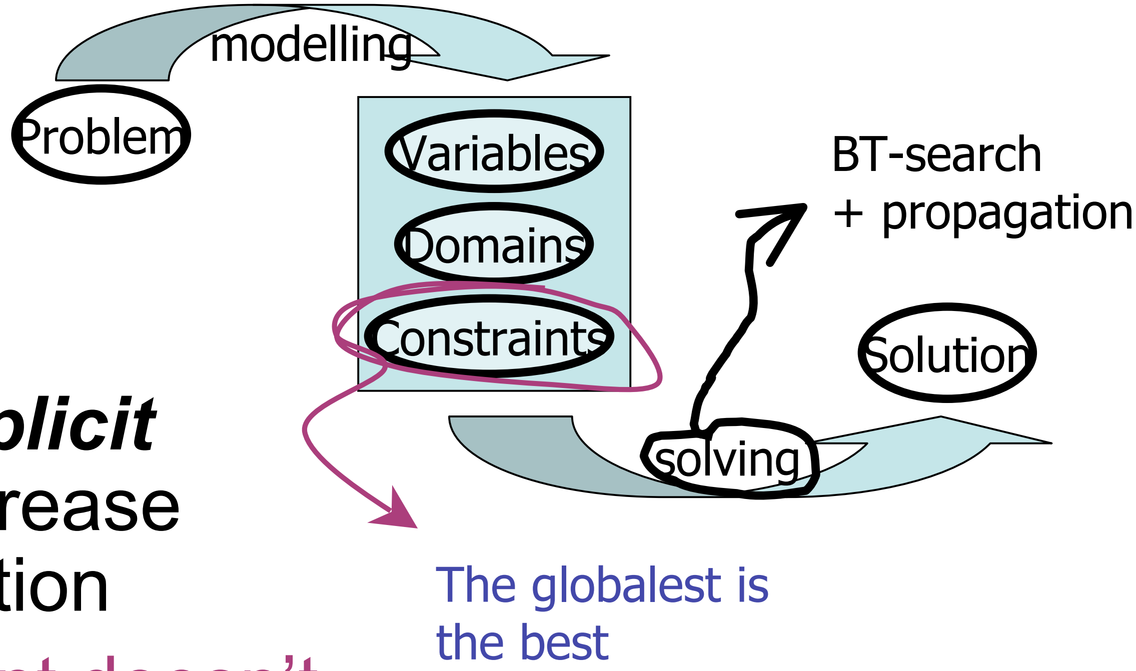
Improve the model

- Basic model M1 :
  solve(M1) ≈ ∞

  ➔ Experts add ***implicit*** constraints that increase constraint propagation

- An implicit constraint doesn't change the set of solutions

  ➔ We will learn implicit **global** constraints



Implicit global constraints

- Model M1:

  at most two 1 per solution

- M1+{`card`[#1≤2](X_1..X_n)}:
  same solutions as M1

- But solve(M1+ `card`) is faster than solve(M1)

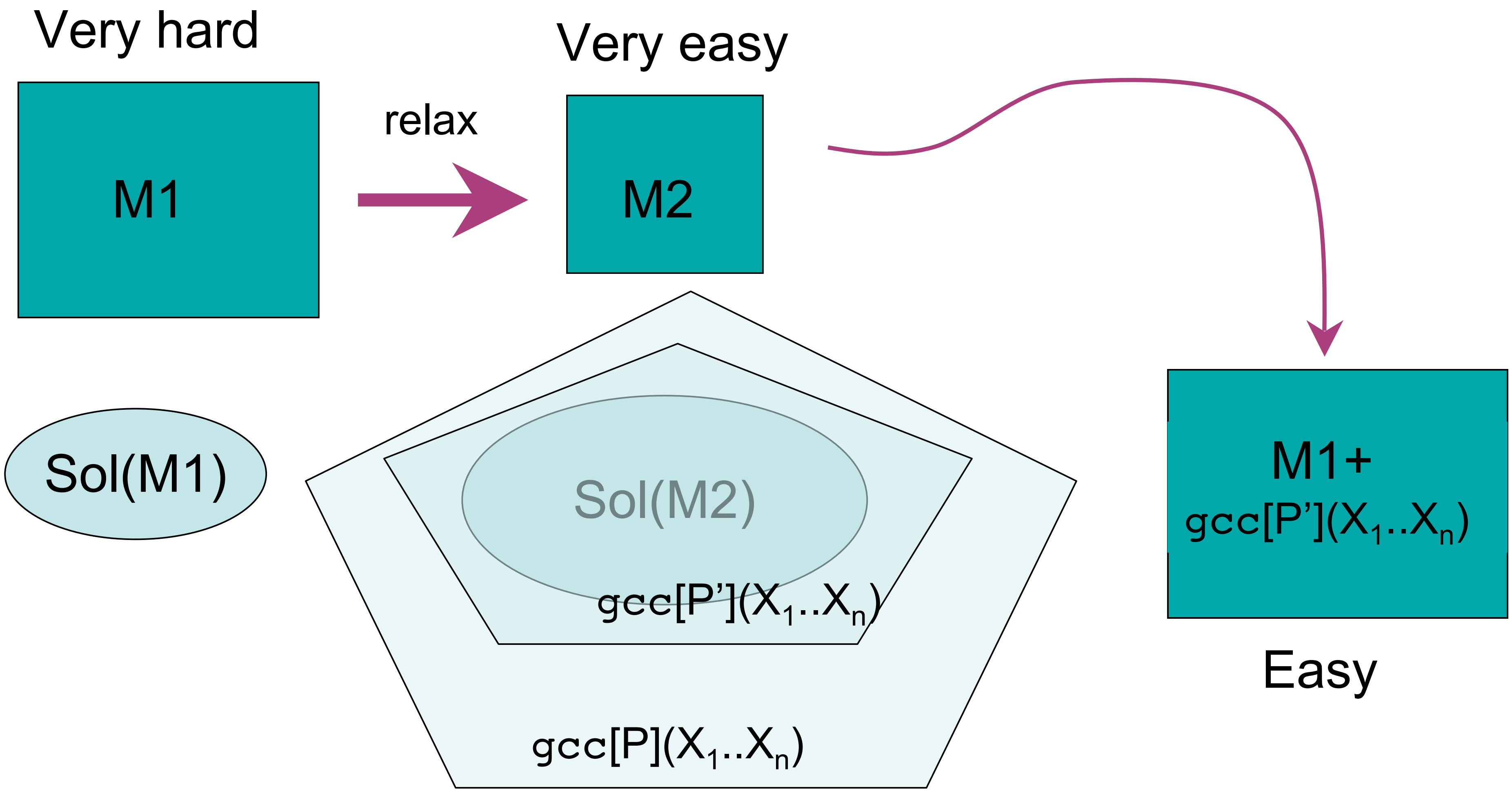
sol(M1):

X_1 … X_n
112345
332223
551554
124135
…..

`Card[..]+card[..]+card[..]`
 `=` **`gcc`**`[P]`

`gcc` = propagation with a flow

Learn parameters P of $gcc[P](X_1..X_n)$



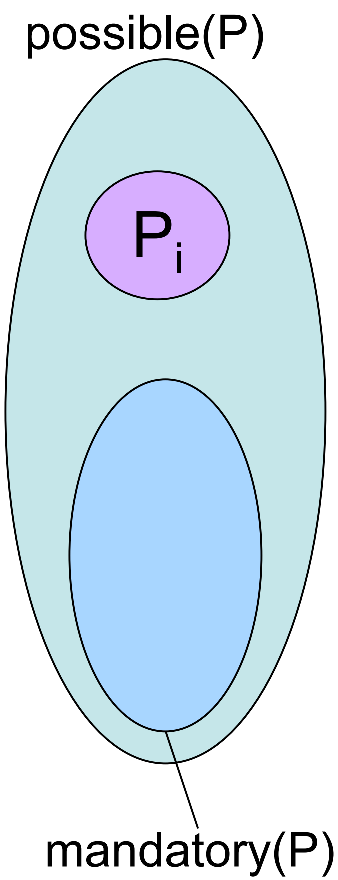
Example: Task allocation

- Projects to be assigned to students while *minimising* disappointment
- Model **M1** designed by some of the students (2h of courses on CP) :
- *optimize*(M1) > 12h

Task allocation

- Launch *optimize*(M1) during 1 sec.
  - Solution s_0 of cost F_0
- M2 = M1+($cost<F_0$)
- *mandatory*(P) ← cardinalities(s_0); *possible*(P) ← Z
- choose $P_i \subseteq$ *possibles*(P) \ *mandatory*(P)
  - s ← *solve*(M2 + gcc[P_i]($X_1..X_n$))
  - **If** s= ∅ **then***possibles*(P) ← *possibles*(P) \ P_i
  - **Else** *mandatory*(P) ← *mandatory*(P) + cardinalities(*s*)

- *optimize*(M1 + gcc[*possibles*(P)]($X_1..X_n$))
- ➔ optimal solution in 43mn instead of >12h



Summary

- There are possible ways to assist a non expert user in:
  - Finding viewpoints
  - Specifying constraints
  - Improving models
- Once CP modelling is automated, this opens new fields where to use CP

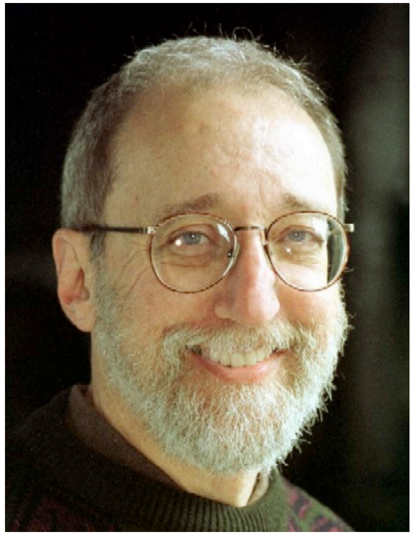
Perspectives

- Take into account background knowledge (eg, ontologies in a company)
  ➔ reduce the size of the learning space

- Robustness to errors from the user

- Vizualization tools for novices

Thanks to...



LIRMM

Bibliographie

C. Bessiere, R. Coletta, T. Petit.
"Learning Implied Global Constraints"
Proceedings IJCAI'07, Hyderabad, India, pages 50-55.

C. Bessiere, R. Coletta, B O'Sullivan, M. Paulin.
"Query-driven Constraint Acquisition"
Proceedings IJCAI'07, Hyderabad, India, pages 44-49.

C. Bessiere, R. Coletta, F. Koriche, B. O'Sullivan.
"Acquiring Constraint Networks using a SAT-based Version Space Algorithm"
Proceedings AAAI'06, Nectar paper, Boston MA, pages 1565-1568.

C. Bessiere, J. Quinqueton, G. Raymond.
"Mining historical data to build constraint viewpoints"
Proceedings CP'06 Workshop on Modelling and Reformulation, Nantes, France, pages 1-16.

C. Bessiere, R. Coletta, F. Koriche, B. O'Sullivan.
"A SAT-Based Version Space Algorithm for Acquiring Constraint Satisfaction Problems"
Proceedings ECML'05, Porto, Portugal, pages 23-34.

C. Bessiere, R. Coletta, E. Freuder, B. O'Sullivan.
"Leveraging the Learning Power of Examples in Automated Constraint Acquisition"
Proceedings CP'04, Toronto, Canada, pages 123-137.

R. Coletta, C. Bessiere, B. O'Sullivan, E. Freuder, S. O'Connell and J. Quinqueton.
"Constraint Acquisition as Semi-Automatic Modelling"
Proceedings AI-2003, Cambridge, UK, pages 111--124.

Optimistic

- e5=(3,3,3) violates the two constraints X≠Z and Y≠Z
  - e5 positive
    - remove 3/4 of the possible CSPs
  - e5 negative
    - remove 1/4 of the possible CSPs
- Works well when the target CSP is under-constrained

e5=(3,3,3)

X+Y+Z≤10

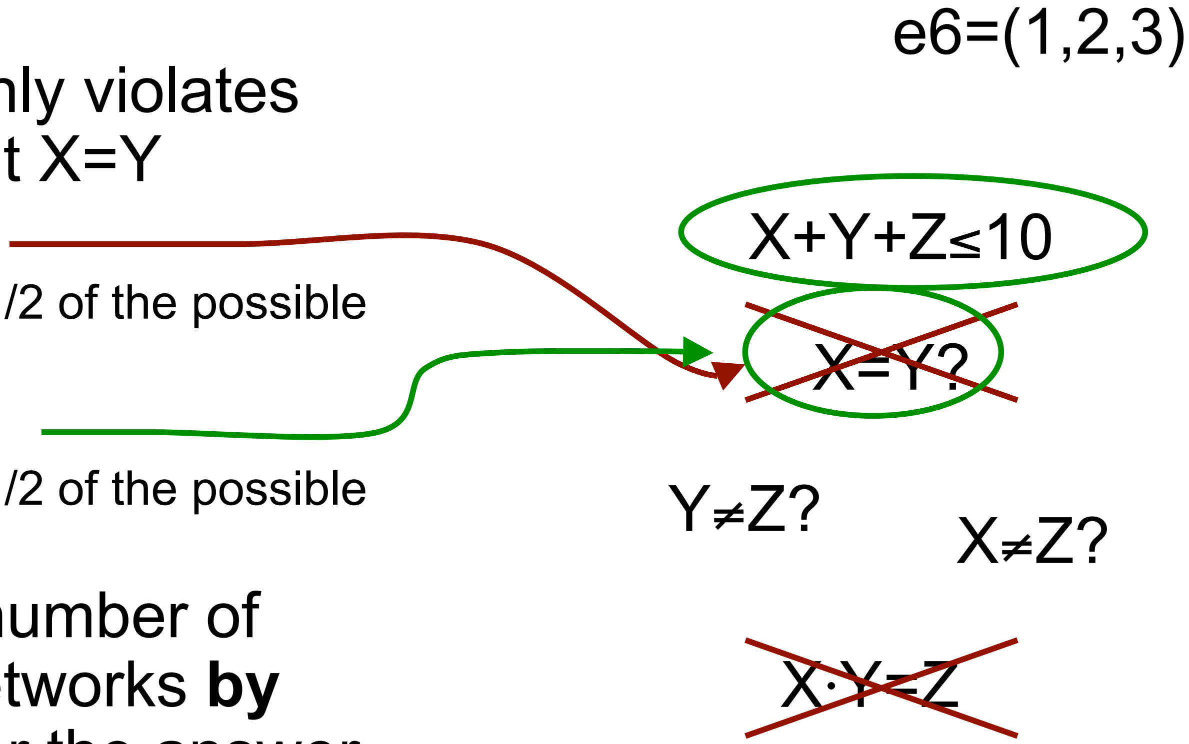
X=Y?

~~Y≠Z?~~ or ~~X≠Z?~~

~~X·Y=Z~~

Optimal

- e6=(1,2,3) only violates the constraint X=Y
  - e6 positive
    - remove 1/2 of the possible CSPs
  - e6 negative
    - remove 1/2 of the possible CSPs
- Divides the number of candidate networks **by half** whatever the answer of the user



Expérimentation : Tribot Mindstorms (2)

- Modélisation **automatique** par CONACQ
- Implémentation en CHOCO du planificateur CSP-Plan [Lopez2003]
- Commande du robot via le langage URBI

⇒ Objectif : Saisie d'un objet par le robot Tribot!

are needed to see this picture.