Introduction à la Programmation par Contraintes

EJCP 2018

Charlotte Truchet

LS2N, UMR 6004, Université de Nantes, France

27 juin 2017

Crédits

Certains slides sont empruntés à

Marie Pelleau	Ghiles Ziat	Giovanni Lo Bianco
MCF U. Nice	PhD U. Paris 6	PhD IMT Nantes



Outline

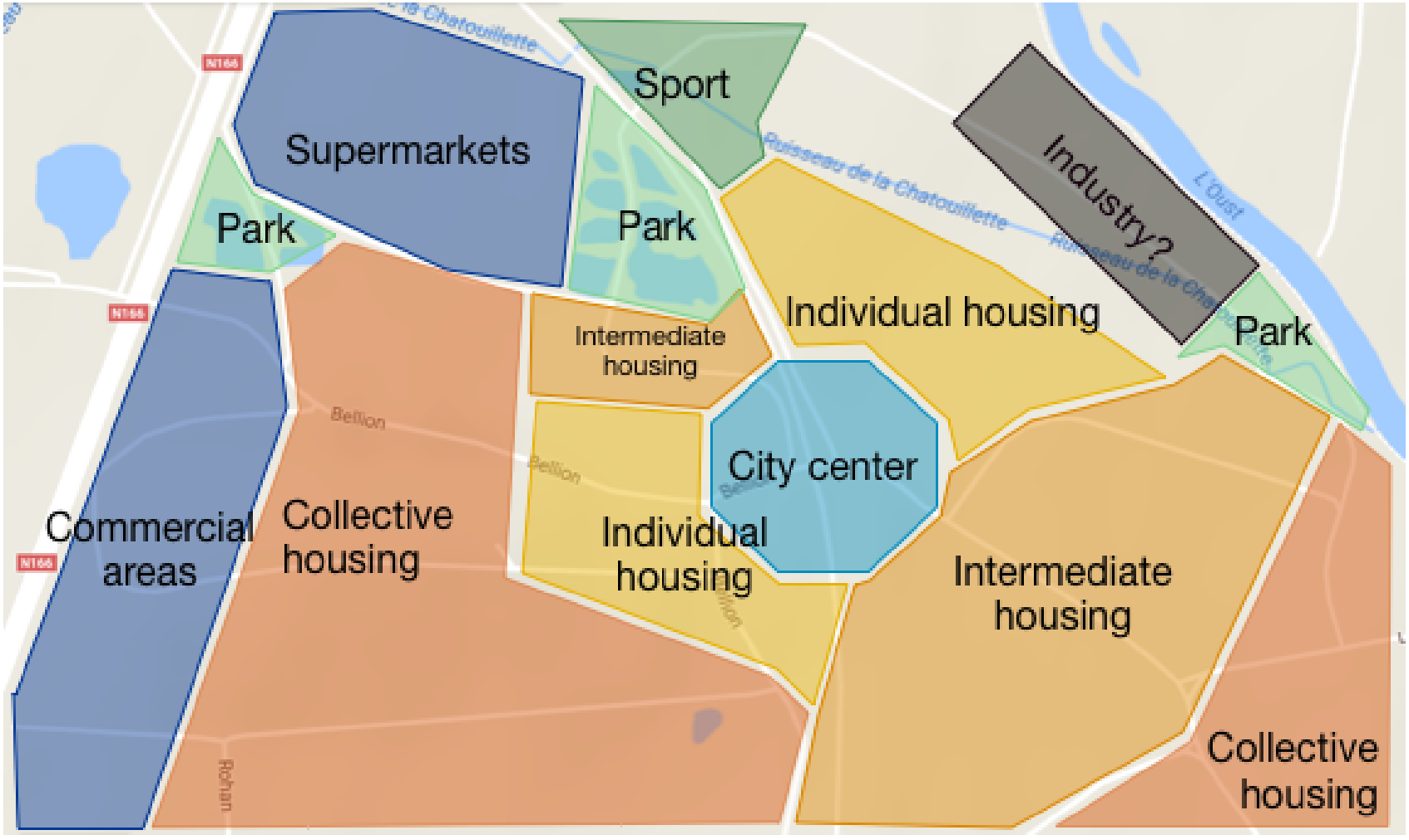
Urban planning

Urban planning

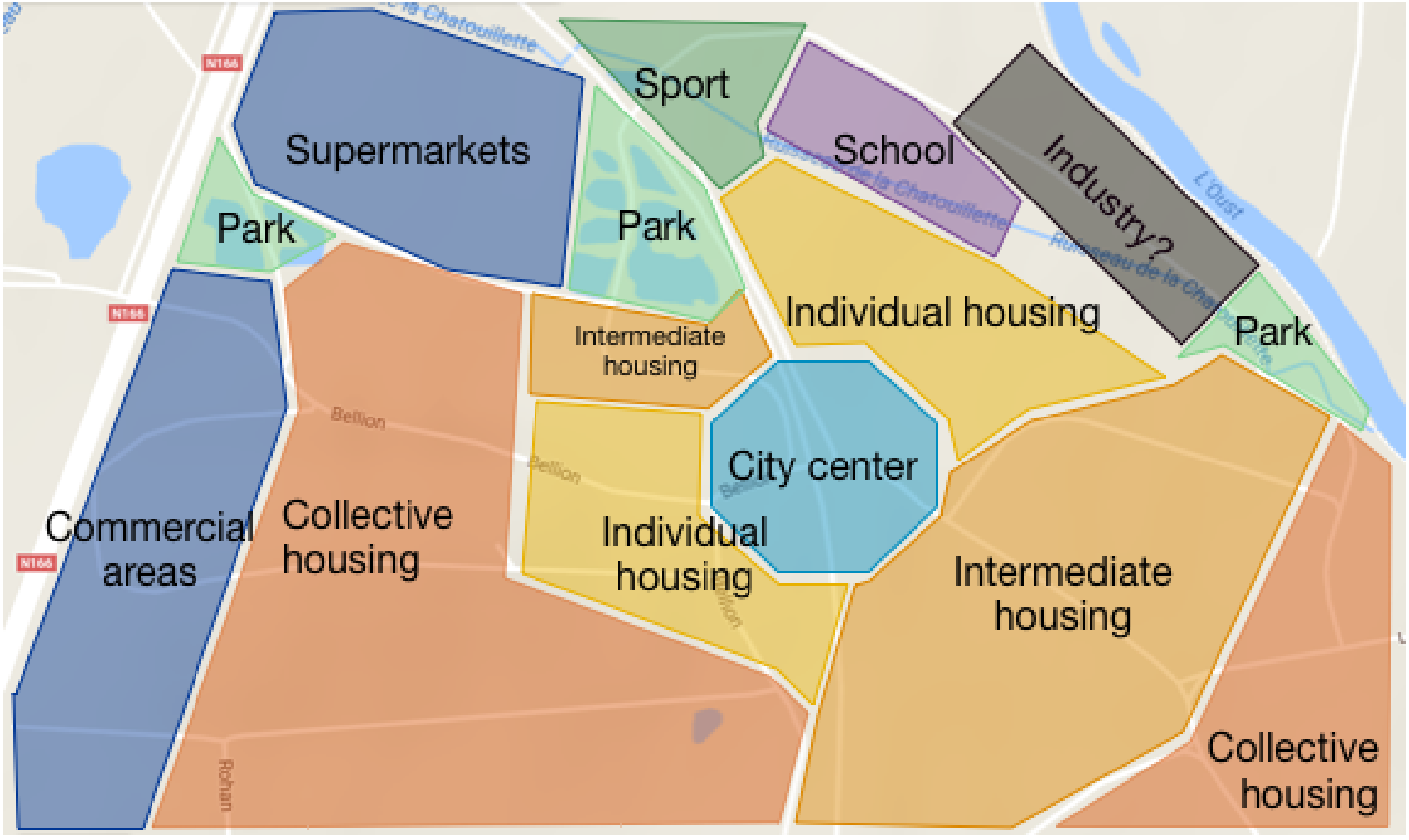
Urban planning

Urban planning

Urban planning



Urban planning



Urban planning

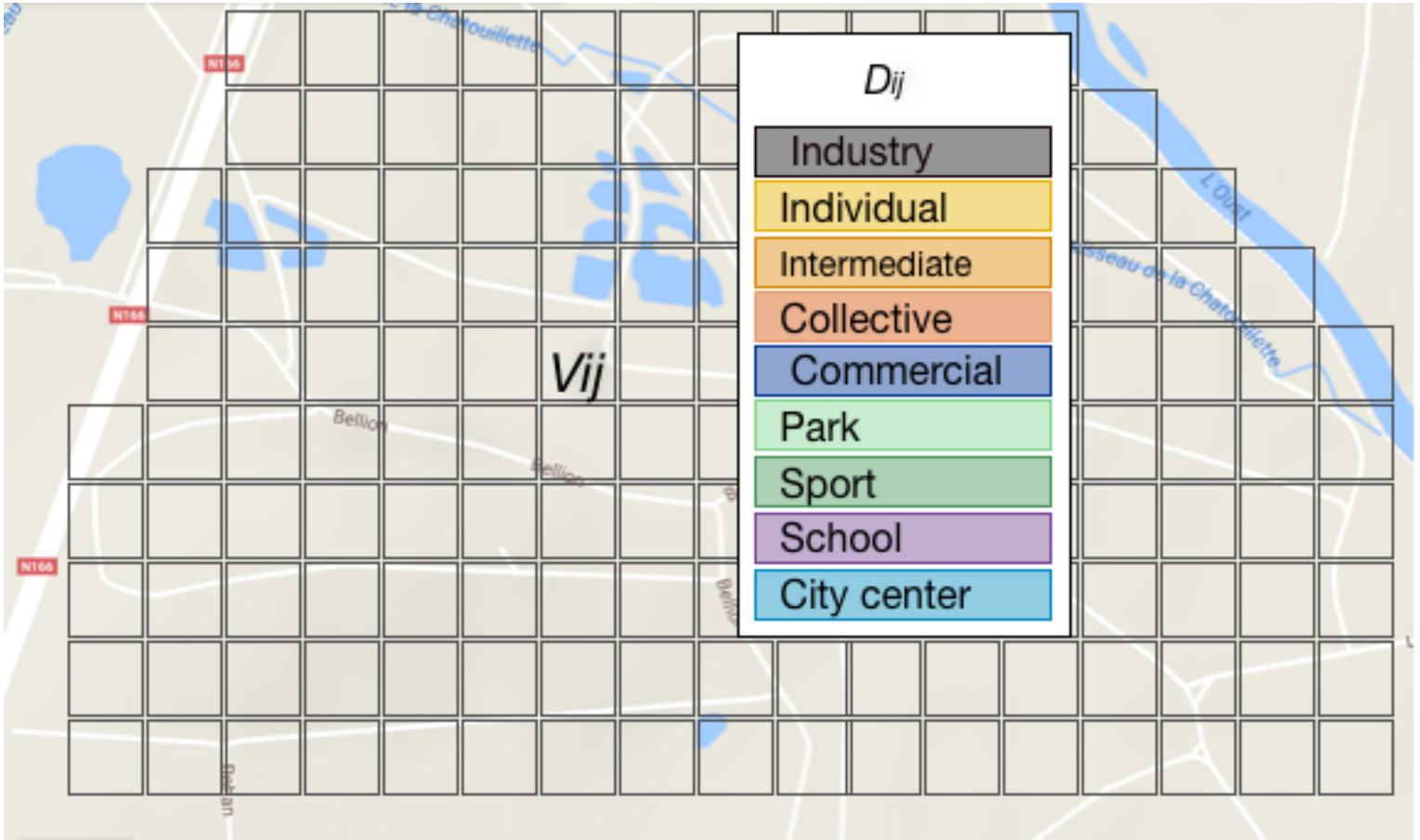
Constraint Programming

In practice:

- combinatorial problem:
  - we need to make choices,
  - choices may have consequences long after they have been made,
  - it must be possible to change the choices (mark them).
- declarative problem:
  - checking is easy, based on rules or user knowledge,
  - efficiently building is difficult.

CP on an example

A variable is an unknown of the problem. It has a given domain, set of values the variable can take.



CP on an example

A variable is an unknown of the problem. It has a given domain, set of values the variable can take.

CP on an example

A variable is an unknown of the problem. It has a given domain, set of values the variable can take.

A constraint is a logical relation on variables.

CP on an example

A variable is an unknown of the problem. It has a given domain, set of values the variable can take.

A constraint is a logical relation on variables.

A consistent domain for a given constraint is a domain which does not contain infeasible values.

CP on an example

A variable is an unknown of the problem. It has a given domain, set of values the variable can take.

A constraint is a logical relation on variables.

A consistent domain for a given constraint is a domain which does not contain infeasible values.

A constraint solver finds values for the variables such that the constraints are all satisfied.

- A complete solver alternates propagations for all the constraints, and choices.
- An incomplete solver iteratively modifies a configuration.

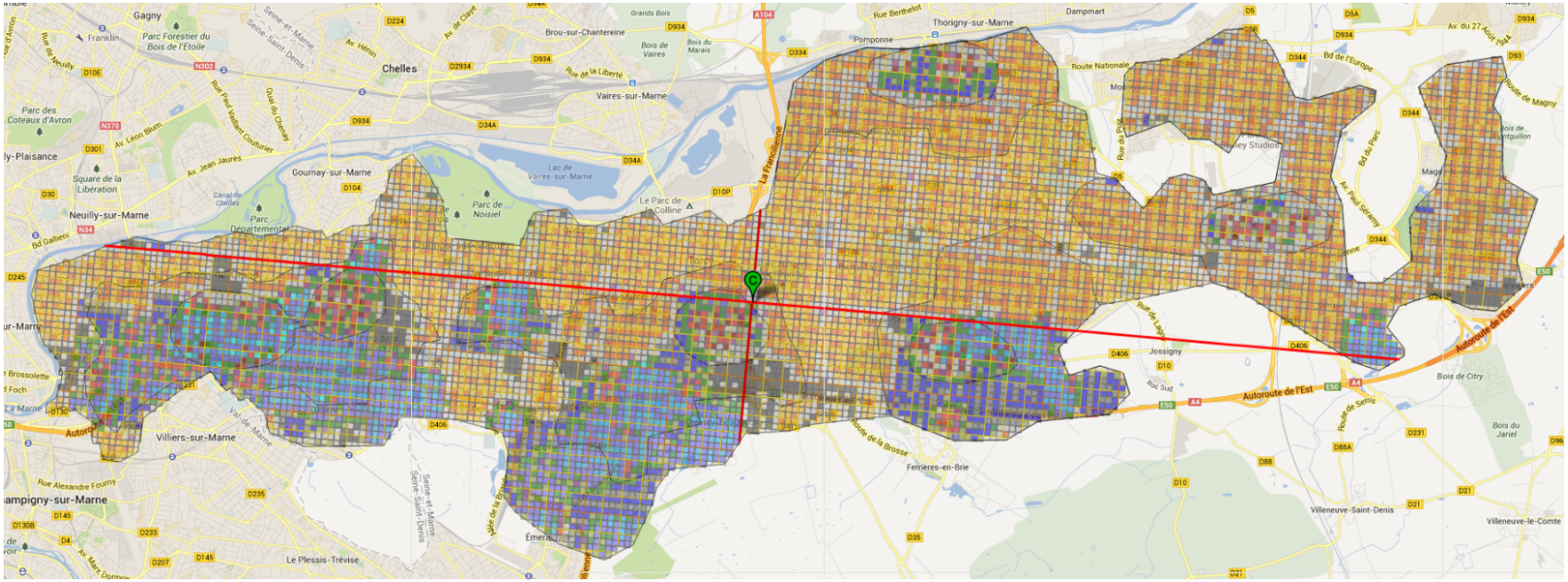
Sustain project

In collaboration with urban planners from EPAMarne

- model of the problem based on urban planners' expertise,
- solver based on a parallelized local search algorithm,
- interactive mode to re-compute partially modified solutions.

PhD of Bruno Belin
collaboration with Marc Christie, Frédéric Benhamou

Sustain project



Simulation on Marne-la-Vallée, a French city of 8728 hectares, with $\sim$ 230 000 inhabitants, $\sim$ 10 000 cells.

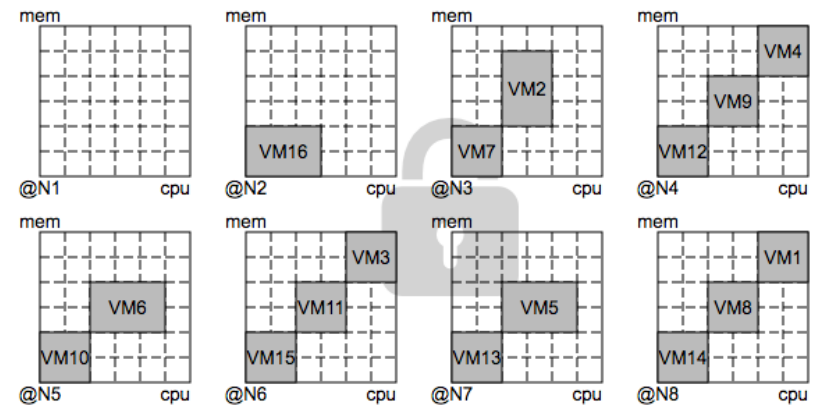
Autres exemples, toujours réels

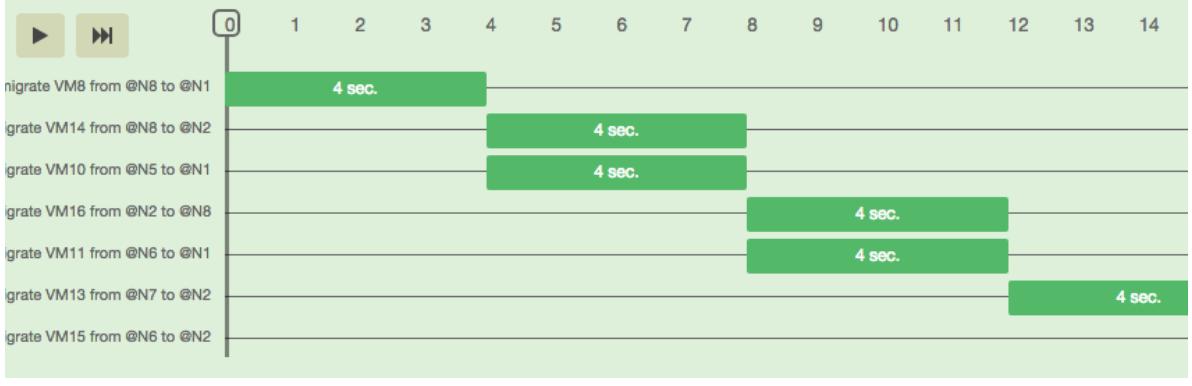
Placement de VMs sur des machines réelles (BtrPlace, projet Entropy), solver Choco

```
namespace sandbox;

VM[1..16]: myVMs;

>>runningCapacity(@N[5..8], 5);
```



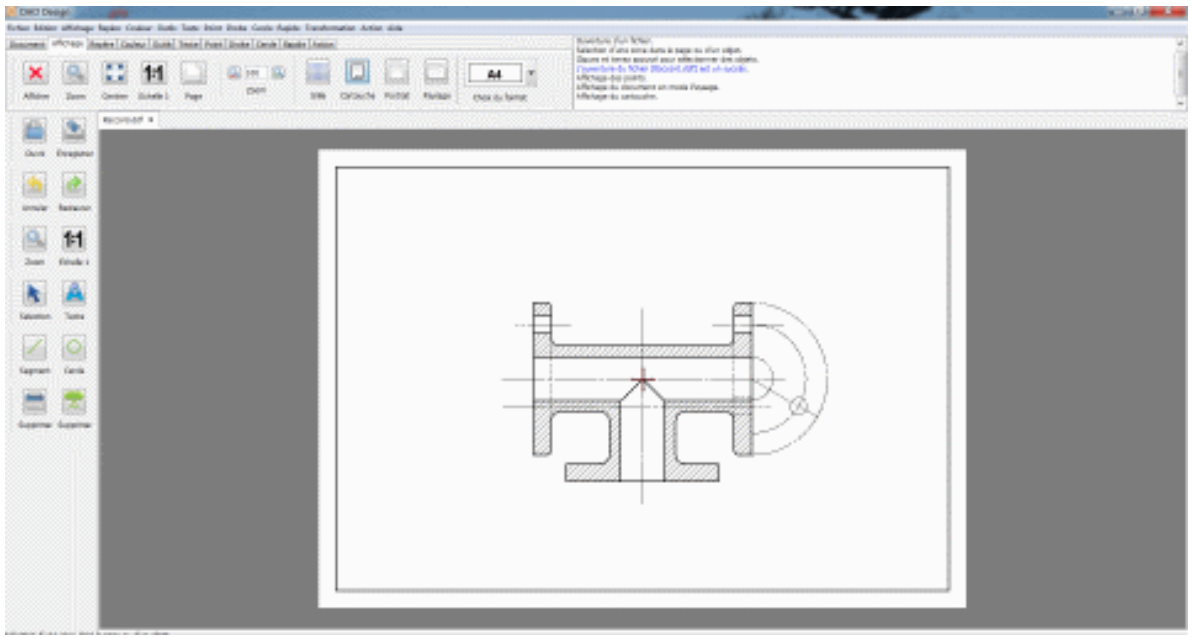


Autres exemples, toujours réels

Planning d'examens dans le domaine de la santé (radio par exemple), Medicalis, solver Choco

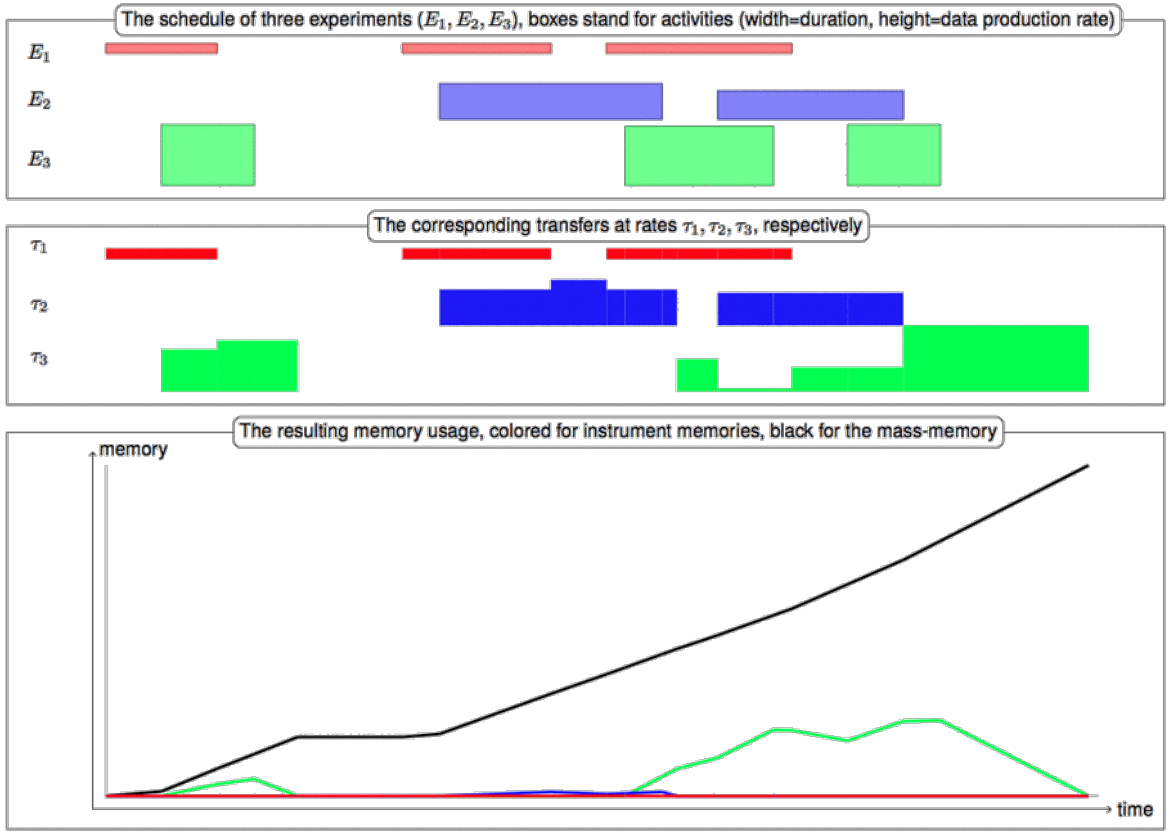
Autres exemples, toujours réels

Calcul de grandeurs géométriques en CAO, DaoDesign (free), solver Choco



Autres exemples, toujours réels

Planification de tâches, notamment transferts de données, pour le robot Philae sous contraintes de ressources (mémoire, énergie).



Les points communs ?

A chaque fois on a :

- des problèmes combinatoires (choix)
- avec des contraintes diverses
- pour lesquels il ne semble pas malin de partir de zéro ni de faire un algo *ad hoc* (structures similaires).

CP

Constraint Programming (CP) is both:

- an Artificial Intelligence technique
  for declarative programming,
- a series of efficient algorithms for combinatorial problems.

Applications

Many successful applications on a wide range of problems:

- logistics/planning: vehicle routing, nurse roastering, matching...
- sustainable development: energy optimization, lifetime...
- arts, music, computer graphics: automatic harmonization, assisted composition...
- verification/software engineering: test generation, floating point abstractions...
- medicine, football games, cryptography, ...

CP& friends

In the same family:

- Linear Programming: constraints are all linear (or linearized)
  Branch and Bound: same solving scheme, different constraints computations.
- Numerical Analysis: real functions
  Newton: numerical algorithms
- SAT/SMT: constraints are logical clauses
  DPLL with clause learning: same solving scheme, different constraints computations.
- Planning: optimization with time.
  A^* : a common idea of heuristic.

CP& friends

In the same family:

- Linear Programming: constraints are all linear (or linearized)
  Branch and Bound: same solving scheme, different constraints computations.
- Numerical Analysis: real functions
  Newton: numerical algorithms
- SAT/SMT: constraints are logical clauses
  DPLL with clause learning: same solving scheme, different constraints computations.
- Planning: optimization with time.
  A^* : a common idea of heuristic.

There ~~is no~~ should not be a competition.
In practice, all of these techniques are very often combined.

Outline

Constraint Satisfaction Problem

Un *Constraint Satisfaction Problem* (CSP) est donné par

- un ensemble de variables $V_1 \ldots V_n$ (n fixé),
- un ensemble de domaines $D_1 \ldots D_n$,
  où D_i représente les valeurs que la variable V_i peut prendre,

  NB : les domaines sont le plus souvent finis - mais pas toujours.
- un ensemble de contraintes $C_1 \ldots C_p$, qui sont des relations logiques entre les variables.

Constraint Satisfaction Problem

Une solution du problème est une instanciation de valeurs des domaines aux variables, qui satisfait les contraintes.

Dans le cas continu, si les solutions ne sont pas représentables en machine, une solution peut être donnée:

- par une approximation extérieure de l'ensemble des solutions (solveur complet),
- par une approximation intérieure (solveur correct).

Constraint Satisfaction Problem

Remarques :

- Une contrainte restreint les valeurs que les variables peuvent prendre. C'est dans le cadre classique la seule chose qui relie les variables.
- Par nature, un CSP définit un espace de possibilités $D_1 \times ... \times D_n$, de taille d^n (si $\forall i, |D_i| = d$).
- En général, on utilise la programmation par contraintes sur des problèmes NP-durs. Cela dit, ce n'est pas obligatoire.

Contraintes

Les langages de contraintes incluent en général

- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
  $V_1 + 7 = V_3$,
  $V_1 * V_3 < 10$
  $\sum_i V_i < M$

Contraintes

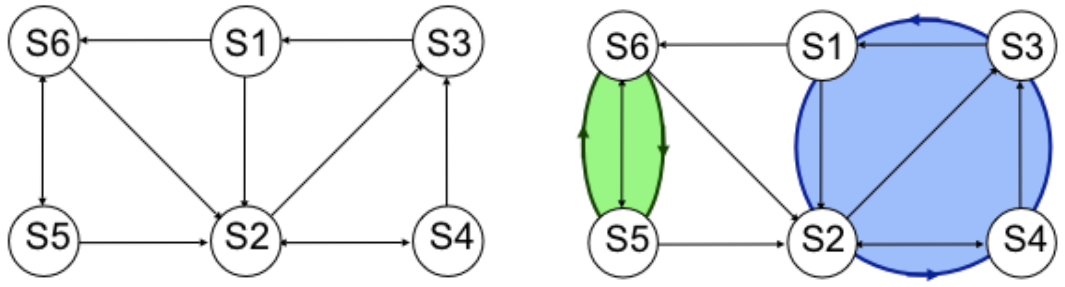
Les langages de contraintes incluent en général

- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :

Contraintes

Les langages de contraintes incluent en général

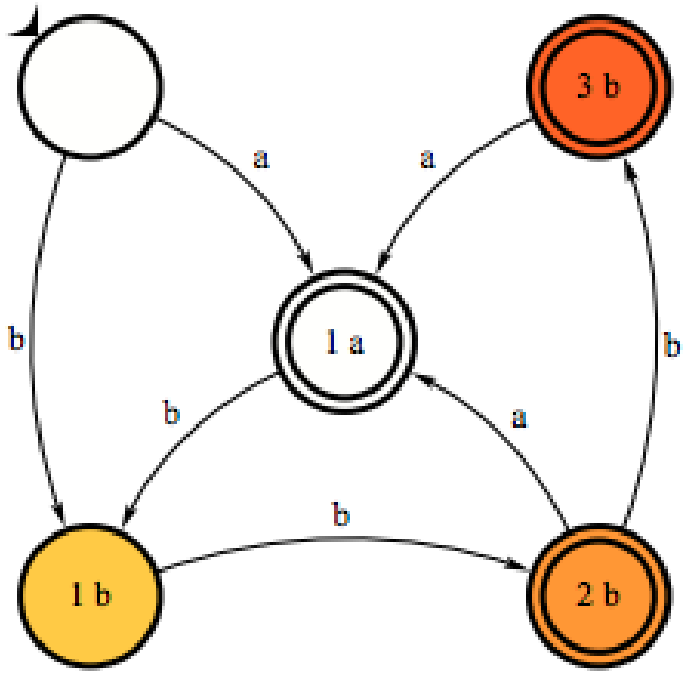
- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...



Contraintes

Les langages de contraintes incluent en général

- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...



Contraintes

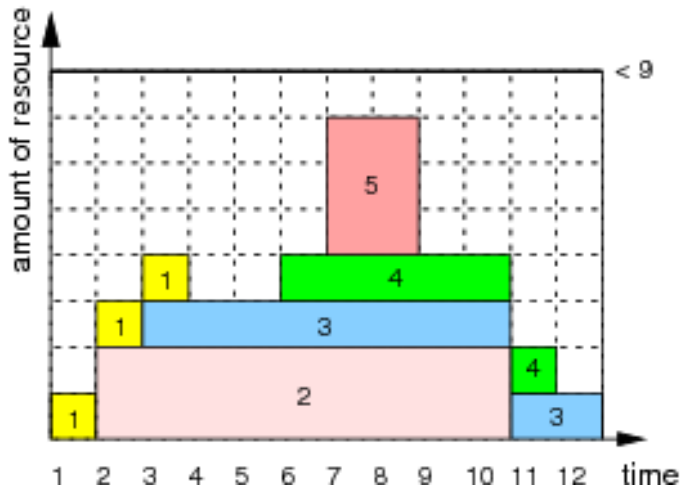
Les langages de contraintes incluent en général

- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...
  - pratiques : `element`, `table`...

Contraintes

Les langages de contraintes incluent en général

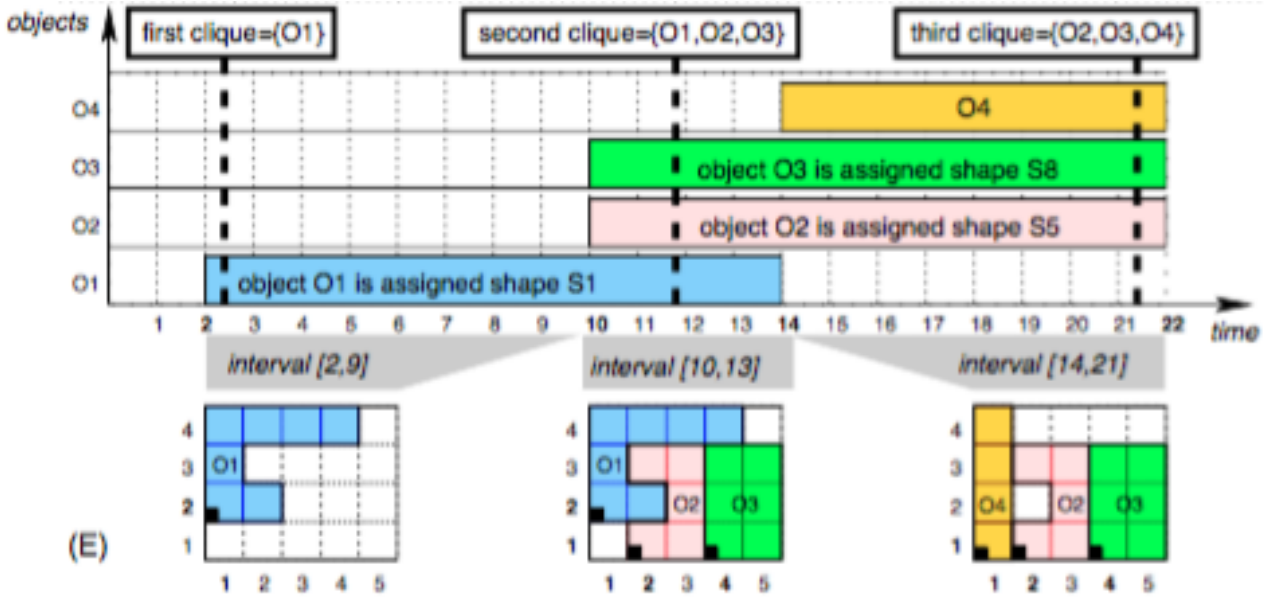
- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...
  - pratiques : `element`, `table`...
  - spécifiques : `cumulative`, `geost`,



Contraintes

Les langages de contraintes incluent en général

- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...
  - pratiques : `element`, `table`...
  - spécifiques : `cumulative`, `geost`,



Contraintes

Les langages de contraintes incluent en général

- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...
  - pratiques : `element`, `table`...
  - spécifiques : `cumulative`, `geost`,
  - de cardinalité : `alldifferent`, `nvalue`, `atleast`, `gcc`...

Contraintes

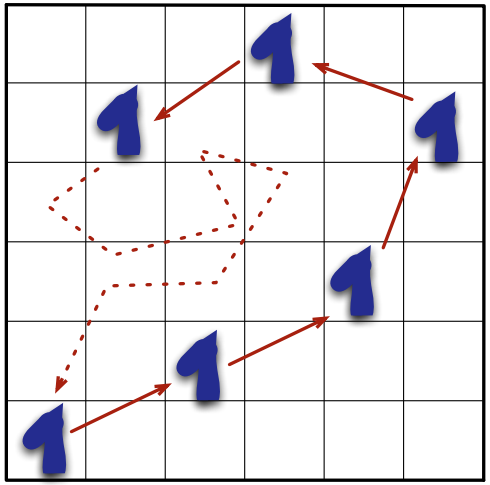
Les langages de contraintes incluent en général

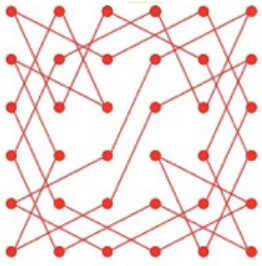
- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...
  - pratiques : `element`, `table`...
  - spécifiques : `cumulative`, `geost`,
  - de cardinalité : `alldifferent`, `nvalue`, `atleast`, `gcc`...

Presque toutes les contraintes globales sont référencées dans le *Global Constraint Catalog*, avec un format commun, et les références bibliographiques.
`http://sofdem.github.io/gccat/`

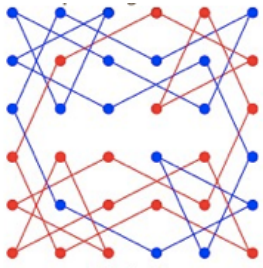
An example

Euler's knight:
A knight (chess game) must travel across a chessboard, it must pass once and only once on each square, and its first and last squares must be equal.

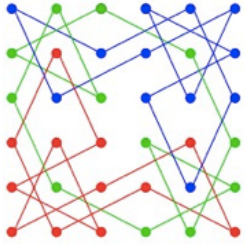




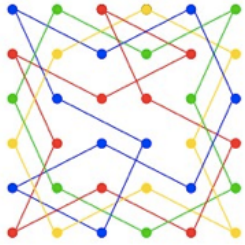
1 knight
(36 moves)



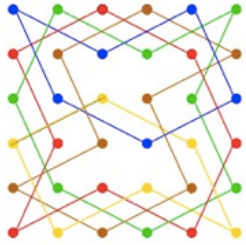
2 knights
(18 and 18 moves)



3 knights
(12, 12 and 12 moves)



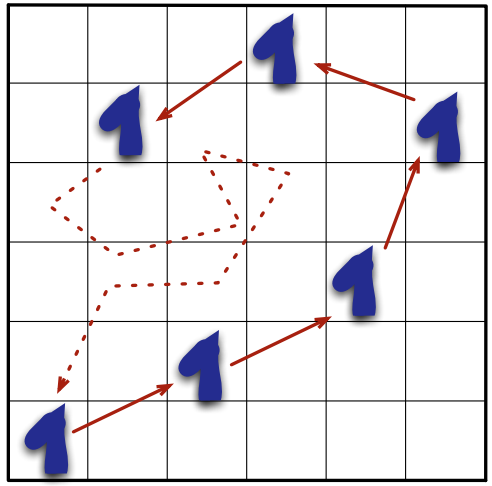
4 knights
(8, 8, 10 and 10 moves)



5 knights
(6, 6, 8, 8 and 8 moves)

An example

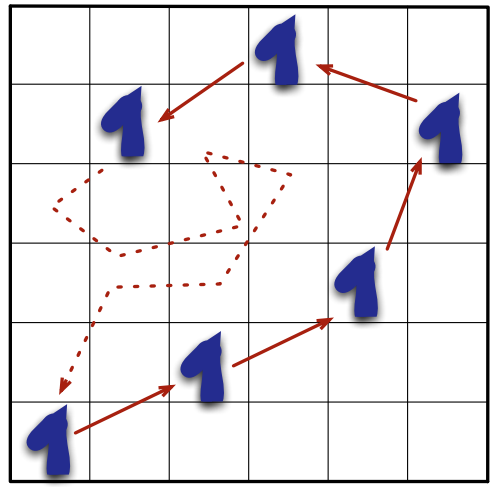
Euler's knight:
A knight (chess game) must travel across a chessboard, it must pass once and only once on each square, and its first and last squares must be equal.



How to compute one (all the) solution(s)?
Find an algorithm?

An example

Euler's knight:
A knight (chess game) must travel across a chessboard, it must pass once and only once on each square, and its first and last squares must be equal.

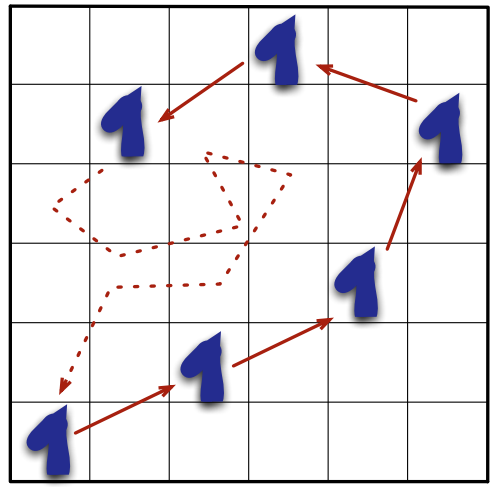


How to compute one (all the) solution(s)?
Find an algorithm?
Remark: describing the problem in a simple language is easy.

An example

Let X_i, Y_i the coordinate of the knight at step i, and f a function enumerating the squares depending on their coordinates. We must have:

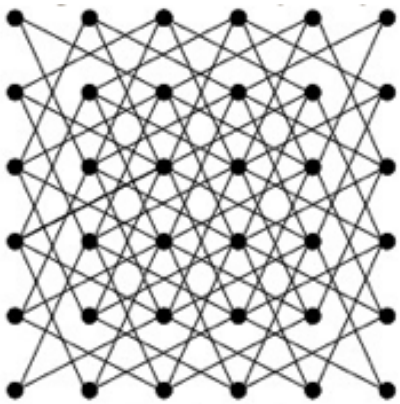
- for $i, j < n$, $X_i \neq X_{i+1} \wedge Y_i \neq Y_{i+1}$ and $|X_{i+1} - X_i| + |Y_{i+1} - Y_i| = 3$ (respect the knight moves)
- for $i \neq j$, $f(X_i, Y_i) \neq f(X_j, Y_j)$ (not the same square twice)
- $1 \leq X_i, Y_i \leq n$

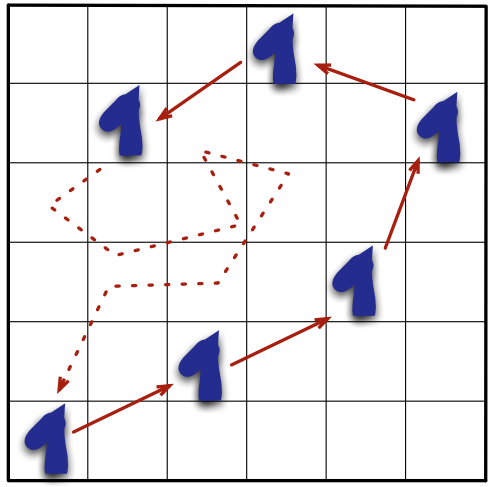


If we know how to solve each constraint, **and** aggregate the results, then we can solve the problem... and many others.

An example

Another model: consider the graph of feasible moves on the chessboard.





Then one constraint is enough:

```
circuit(Sommets, 1).
```

Modeling

Natural question: given a problem, what is the best CSP to solve it?

Modeling

Natural question: given a problem, what is the best CSP to solve it?

Tricky answers:

- define best: fastest? closest to some optimum? best approximations (if no solutions)?

Modeling

Natural question: given a problem, what is the best CSP to solve it?

Tricky answers:

- define best: fastest? closest to some optimum? best approximations (if no solutions)?
- define solve: interactions with the solver internal tuning.

Un cas pratique

In theses frames are real slides from a presentation I made at CPAIOR 2014 on the urban planning applications (only a few of them)

In blue, I tell you how things really happened.

Evolution of the world city population [United Nations reports]

- 1995 the urban population in emerging countries is higher than the urban population of industrial countries,
- 2008 more than 50% of the world population lived in cities,
  1/3 in townships,
- 2025 58% of the world population lives in cities (projection)
- 2050 67.2% of the world population lives in cities (projection)

$\Longrightarrow$ Huge needs for the development of new cities, mostly in Africa and Asia.

We had a meeting with urban planners. They had problems which seemed interesting, highly combinatorial, and hard.

The SUSTAINS project

- a national-funded French research project,
- the project gathers urban planners and computer scientists Areva TA, Armines, Artelys, Artefacto, EPAMarne, LINA, LIMSI,
- goal : deploy a software suite for designing new cities, taking into account sustainability issues (energetic, social and economic),
- end-users are urban planners and decision makers.

Early stage of urban planning:

- start with a beetroot field,
- draw a rough map for the decision makers and the population,
- used to have first estimations of the needs in big equipments.

It is often easier to get money when working on such applications, because: industrial, and original.

Existing urban models:

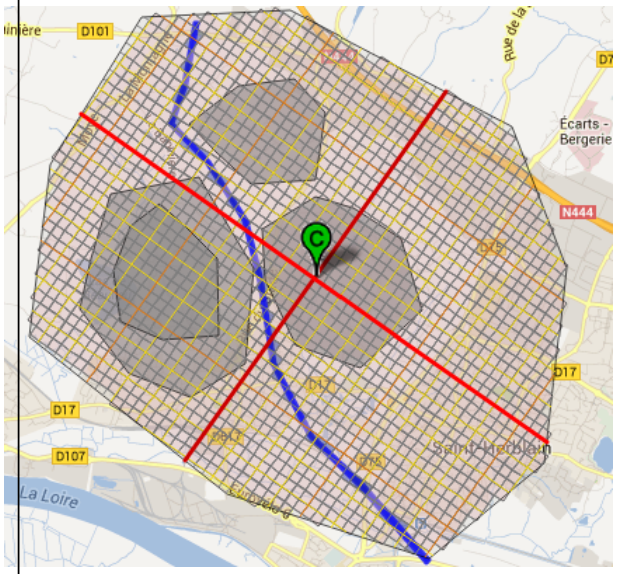
- UrbanSim [Waddell 2002], a land-use model,
- CommunityViz, a fine grain model for urban planners.

We are not interested in fine grain information, pricing issues or time evolution of the future city. We use an ad-hoc urban model proposed by EPAMarne, which need very little information and does not model time evolution.

A two stage approach:

- first step: a urban model calibrates the type of land use of the future city (EPAMarne),
- second step: a geographical constraint model computes the best possible map of the city.

We started in the blind and had to read a totally new bibliography. But think about it: I've read scientific papers on SimCity. For work!



What is given:

- geographic data: area, rivers, landscape, existing elements (highways, historic places...),
- blocks: a 80m $\times$ 80m square, atomic element of the city,
- urban shapes: type of land use and buildings within a single block,
- macroscopic informations: intensities, centralities, economic indicators (number of inhabitants, employment rate, public services...).

We spent hours of discussion, on almost two years, to understand these urban notions. We never fully understood, but we had reasonable assumptions to start with.

The constraint model is a permutation optimization problem:

given data	grid representation of the city, geographical informations,
variables	cells of the grid,
domains	urban shapes provided by the urban model,

We **never had** a CSP.
We made several prototypes in the blind, we improved them based on their remarks, and the CSP/model arrived in the end.

Core constraints:

Interaction

A cell of a given urban shape has location preferences and neighboorhood preferences.

Example: shool units are attracted to residential units, and residential units are repelled by industrial units. Industrial units are attracted to rivers and roads.

Preferences coefficients: one for each urban shape vs each type of landscape, and one for each urban shape vs each other urban shape.

In discussions with the urban planners, the word "attraction" was often used. So we tried this as our first prototype and it happened to fit the urban planners view.

Higher level constraints

Distance

Some urban shapes must be placed with a minimum distance, expressed as a number of cells, between them.

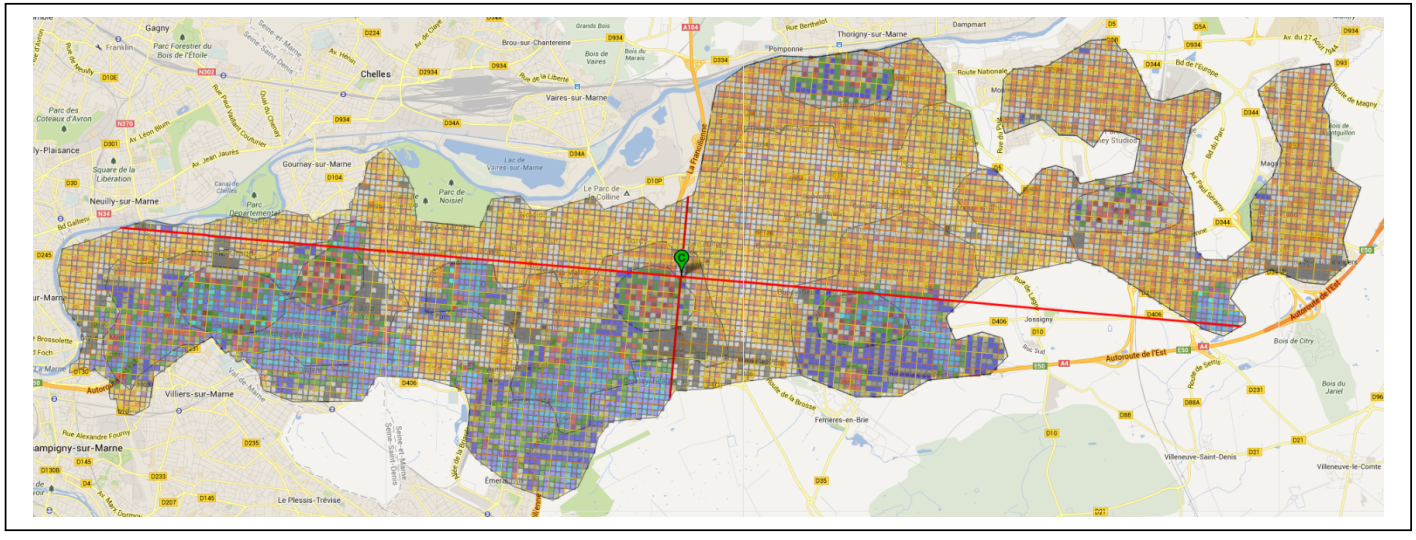
Example: between an individual house and a high building (R+7), there must be a minimum distance of 4 cells

Critical size

Some urban shapes must be grouped to have at minimum size (bigger than one cell).

Example: highschools must be on at least two blocks. Areas with industrial activity must be big enough, otherwise, the area will not be created in practice because it will not be cost-effective.

We showed our results to the urban planners. At once, they saw situations that could not happen, and we refined the model, **adding constraints**, from their observations.



We got scared when we saw the size ot real-life problems (here, the french city of Marne-la-Vallée that we used as a test-case).

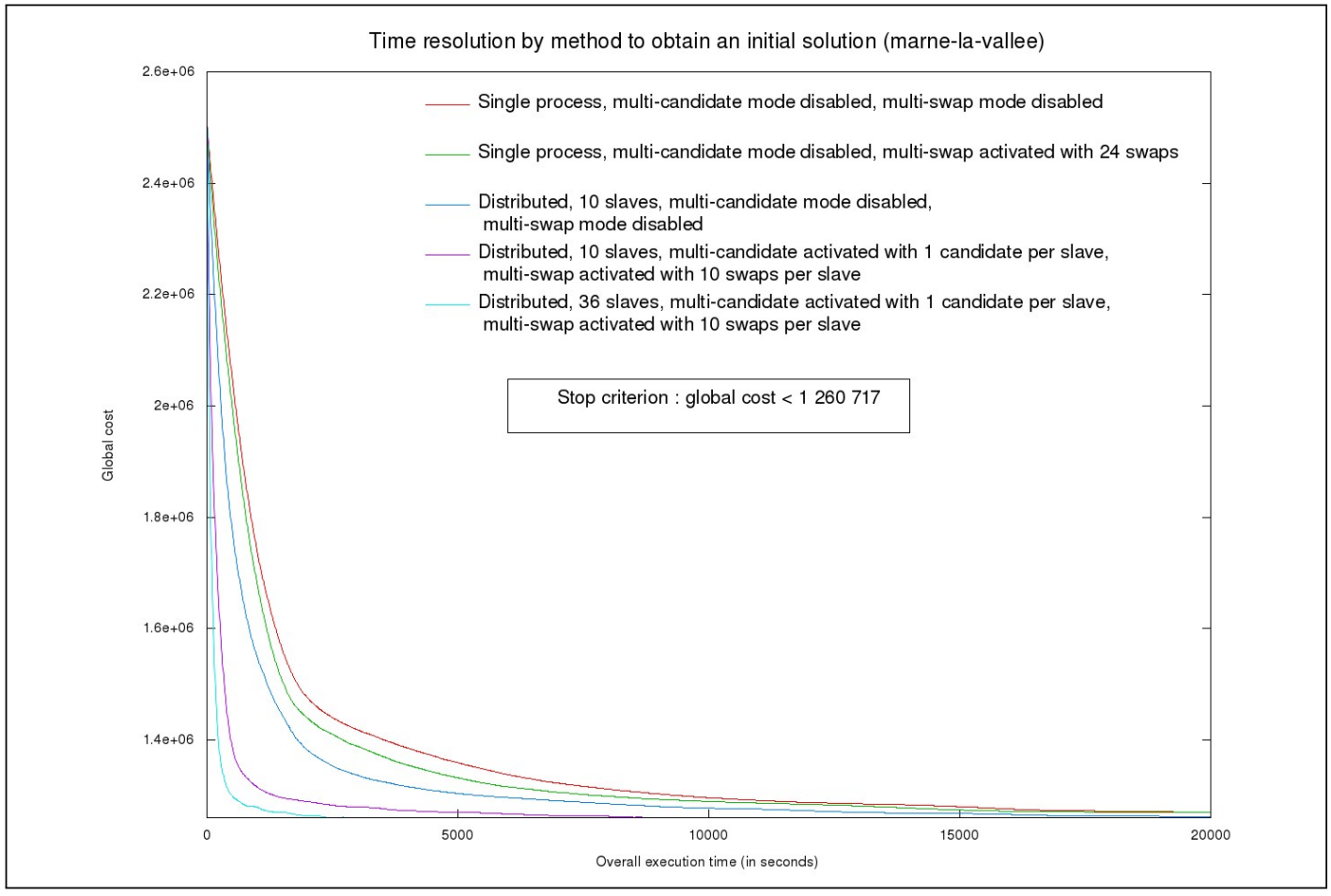
We use a distributed version of Adaptive Search, a local search algorithm in the spirit of Tabu Search [Codognet 2001].

For the distributed version, the map is cut into subareas. We use a master-slave parallel scheme where the slaves can read the whole map, but only work on some subareas.

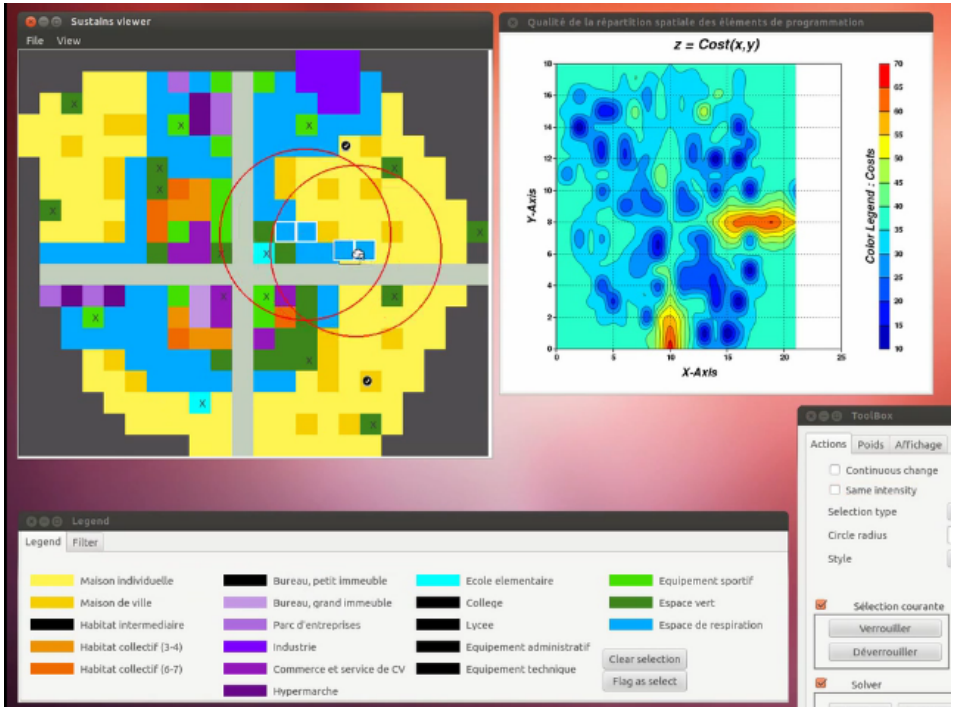
The base iteration becomes:

- compute the cost of the current configuration (master core),
- assign a couple of subareas to each slave (master core),
- select the worst variable within the assigned subareas ,
- select, within the assigned subareas, the best move(s),
- collect all the possible swaps and select the one(s) to apply (master core).

Since the problem is based on a 2D grid, we used its geographic properties to develop a divide-and-conquer distributed approach. Yet, it was quite a lot of (exciting) work to find a good parallel scheme.



We were satisfied. But I am pretty sure that the urban planners never understood why the computation time was so big in the first place.



It was mandatory, for this application, to render the solver interactive. Once a solution is found, we can show its score wrt the attraction model, and the user can modify the variables values.

The project involved people from HMI, who made a very nice 3D interface and implemented it on a huge tablet.

Current researches on modeling

- constraint learning,
  the user gives some solutions, some non-solutions, and the goal is to find a model which separates these
- constraint reformulation,
- model verification.
  check model equivalence

Conclusion on modeling

On big problem instances, you will probably need a CP expert to obtain an efficient model (solving time issue).

On problems of reasonable size, relying on the tools available in the solvers should do the trick.

Outline

Consistances sur les domaines finis

Une contrainte $C(V_1 ... V_n)$ est generalized arc-consistent (GAC) pour des domaines $D_1 ... D_n$ ssi pour toute variable V_i, pour toute valeur $v^i \in D_i$, il existe des valeurs $v^1 \in D_1, ..., v^{i-1} \in D_{i-1}, v^{i+1} \in D_{i+1}, ..., v^n \in D_n$ telles que $C(v^1, ... v^n)$.

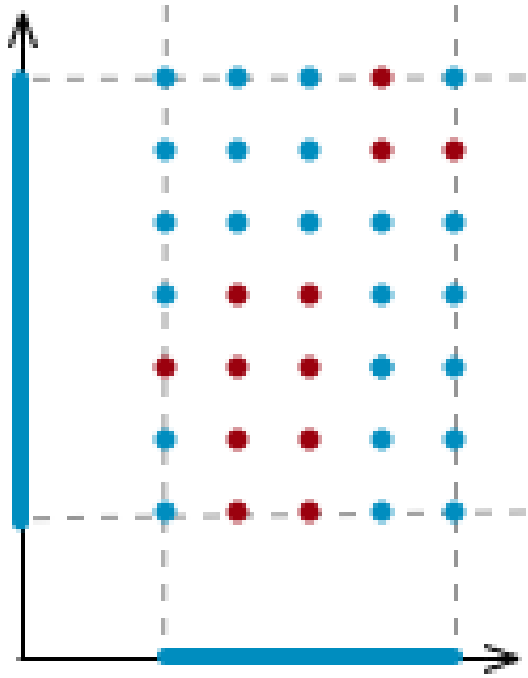
Consistances sur les domaines finis

Une contrainte $C(V_1 ... V_n)$ est generalized arc-consistent (GAC) pour des domaines $D_1 ... D_n$ ssi pour toute variable V_i, pour toute valeur $v^i \in D_i$, il existe des valeurs $v^1 \in D_1, ..., v^{i-1} \in D_{i-1}, v^{i+1} \in D_{i+1}, ..., v^n \in D_n$ telles que $C(v^1, ... v^n)$.

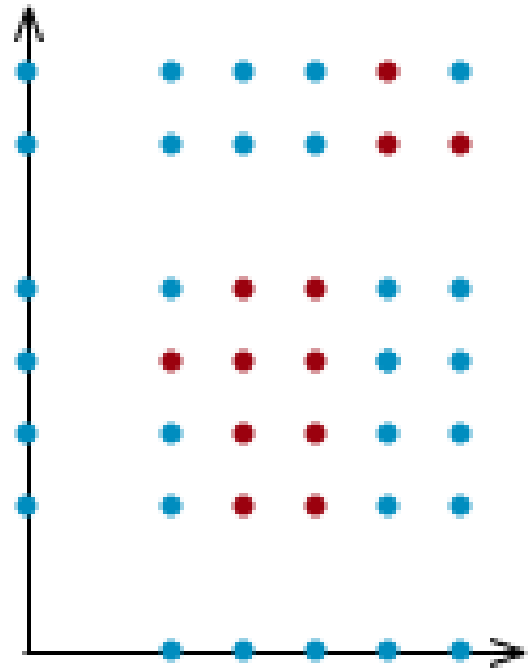
Une contrainte $C(V_1 ... V_n)$ est bound-consistent (BC) pour des domaines $D_1 ... D_n$ ssi les bornes de tous les domaines sont consistantes (au sens ci-dessus).

Consistances sur les domaines finis

En rouge les solutions, en bleu les domaines consistants.



Bound-consistency



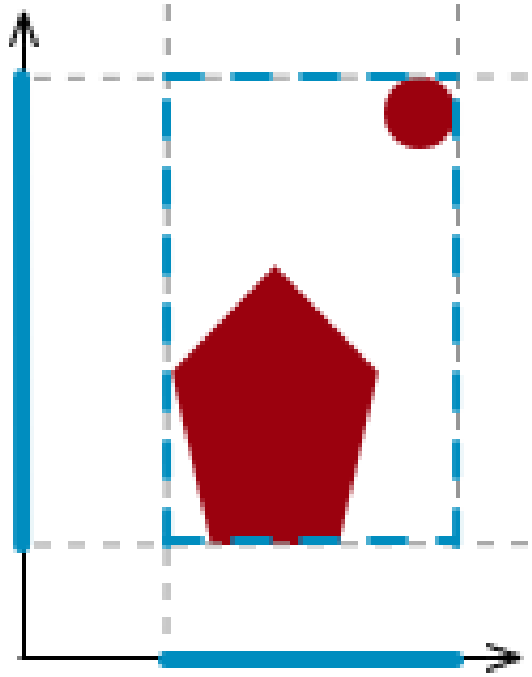
Generalized arc-consistency

Consistances sur les domaines continus

Une contrainte C sur des variables $V_1 \ldots V_n$, de domaines $D_1 \ldots D_n$ est Hull consistent (HC) ssi $D_1 \times \cdots \times D_n$ est la plus petite boîte réelle, à bornes flottantes, contenant les solutions de C dans $D_1 \times \cdots \times D_n$.

Consistances sur les domaines continus

En rouge les solutions, en bleu les domaines consistants.



Hull Consistency

Consistances

Remarques :

- il existe toutes sortes d'autres consistances (sur le discret : path-consistency, singleton arc-consistency, strong consistencies... et sur le continu : Box consistency, MOHCC...)
- ne pas confondre consistance et satsfaisabilité.

Propagation

La propagation d'une contrainte C sur les domaines $D_1 \ldots D_n$ consiste à enlever des domaines toutes les valeurs inconsistantes pour cette contrainte.

Propagation d'une contrainte

- $X = Y + 3 * Z$ si $X = 10$, $Y = 4$ alors on sait que $Z = -2$,

Propagation d'une contrainte

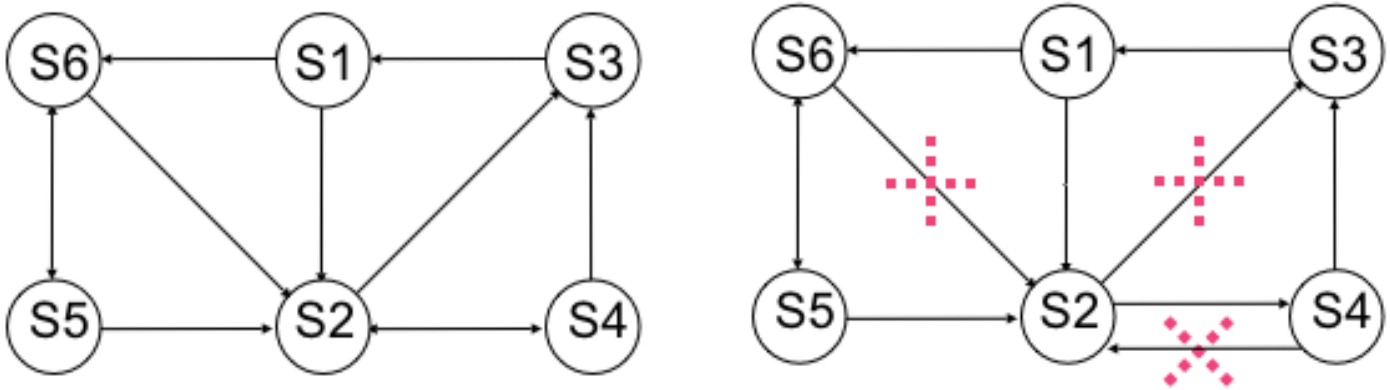
- $X = Y + 3 * Z$ si $X = 10$, $Y = 4$ alors on sait que $Z = -2$,
- $X = Y + 3 * Z$ si $D_Z = \{1..5\}$ et $D_X = \{0..10\}$
  alors D_Y peut être intersecté avec $\{-5, 7\}$,

Propagation d'une contrainte

- $X = Y + 3 * Z$ si $X = 10$, $Y = 4$ alors on sait que $Z = -2$,
- $X = Y + 3 * Z$ si $D_Z = \{1..5\}$ et $D_X = \{0..10\}$
  alors D_Y peut être intersecté avec $\{-5, 7\}$,
- `alldifferent`(X_1, X_2, X_3) : si on sait que D_1 **et** D_2 valent $\{1, 2\}$, on peut éliminer ces deux valeurs pour X_3.

Propagation d'une contrainte

- $X = Y + 3 * Z$ si $X = 10$, $Y = 4$ alors on sait que $Z = -2$,
- $X = Y + 3 * Z$ si $D_Z = \{1..5\}$ et $D_X = \{0..10\}$ alors D_Y peut être intersecté avec $\{-5, 7\}$,
- `alldifferent`(X_1, X_2, X_3) : si on sait que D_1 **et** D_2 valent $\{1, 2\}$, on peut éliminer ces deux valeurs pour X_3.
- contrainte cycle dans un graphe :



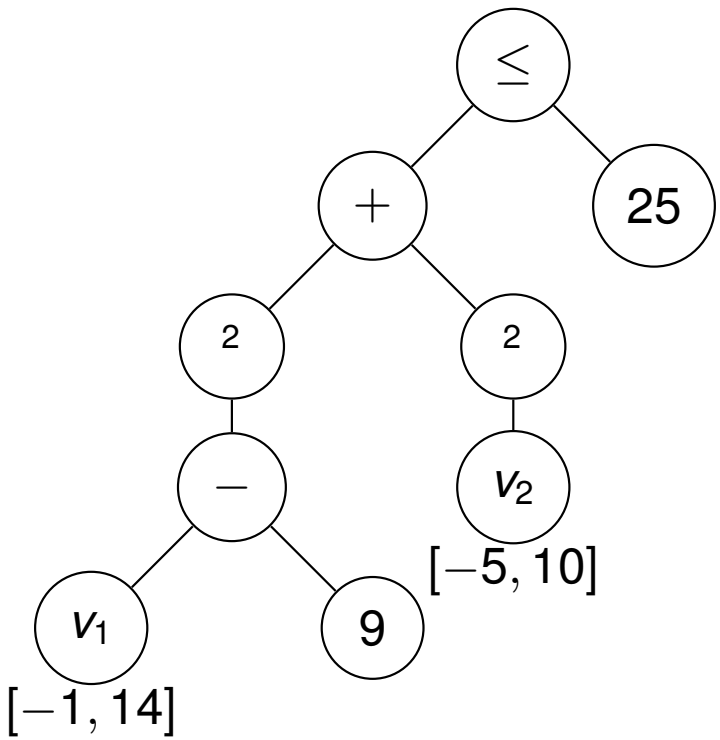
Two examples of propagation algorithms

- HC4-Revise [Benhamou, 1996] : a classical algorithm for hull consistency (real constraints)
- GAC for the `alldifferent` constraint [Régin, 1994, van Hoeve, 2001].

NB : there are plenty of other propagations!
See
`http://ktiml.mff.cuni.cz/~bartak/constraints/`
for a tutorial on basic propagations and
`http://sofdem.github.io/gccat/`
for global constraints.

HC4-Revise

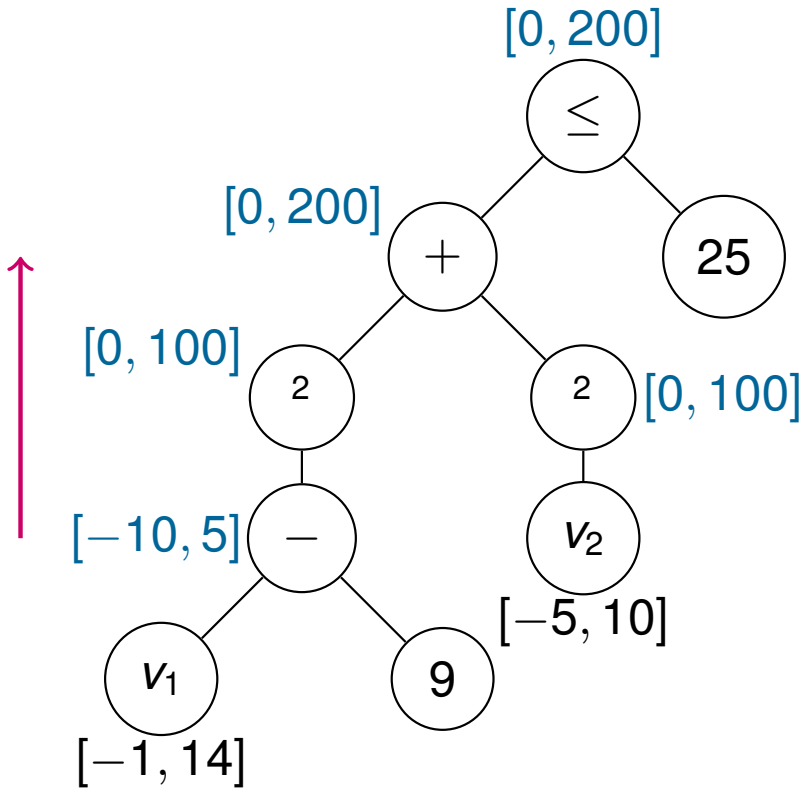
Let $(v_1 - 9)^2 + v_2^2 \leq 25$ be a constraint,
$D_1 = [-1, 14]$, $D_2 = [-5, 10]$ the domains



- Constraint syntactic tree

HC4-Revise

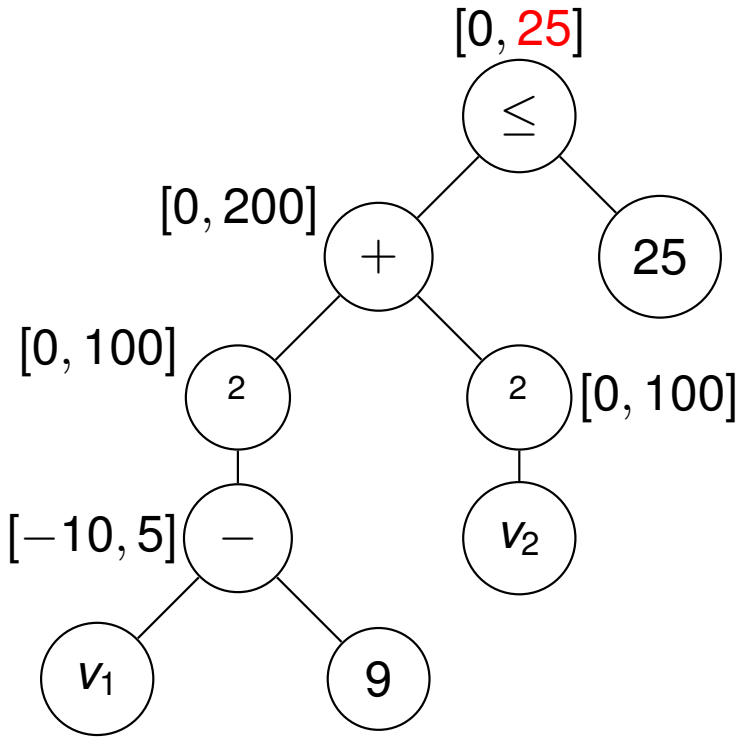
Let $(v_1 - 9)^2 + v_2^2 \leq 25$ be a constraint,
$D_1 = [-1, 14]$, $D_2 = [-5, 10]$ the domains



- Constraint syntactic tree
- Interval arithmetics

HC4-Revise

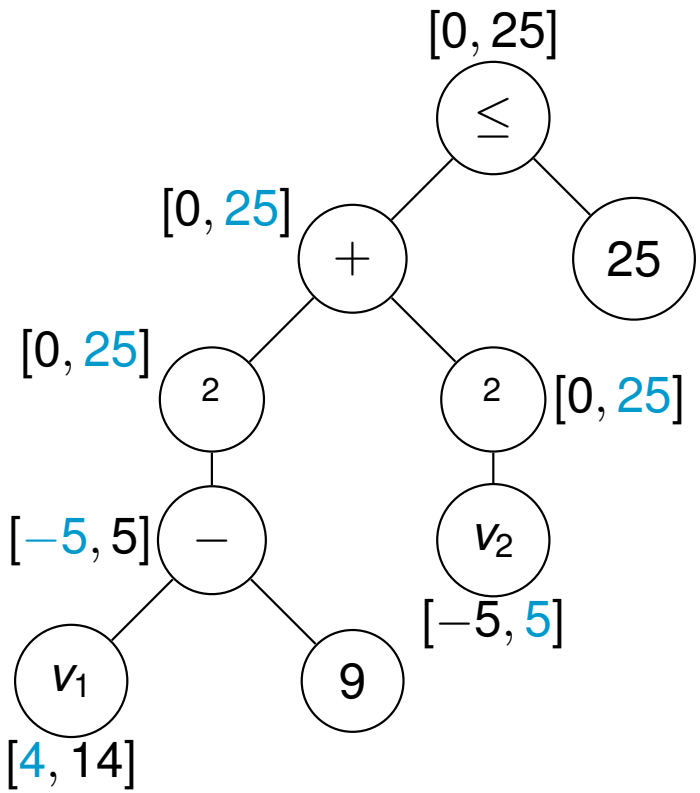
Let $(v_1 - 9)^2 + v_2^2 \leq 25$ be a constraint,
$D_1 = [-1, 14]$, $D_2 = [-5, 10]$ the domains



- Constraint syntactic tree
- Interval arithmetics

HC4-Revise

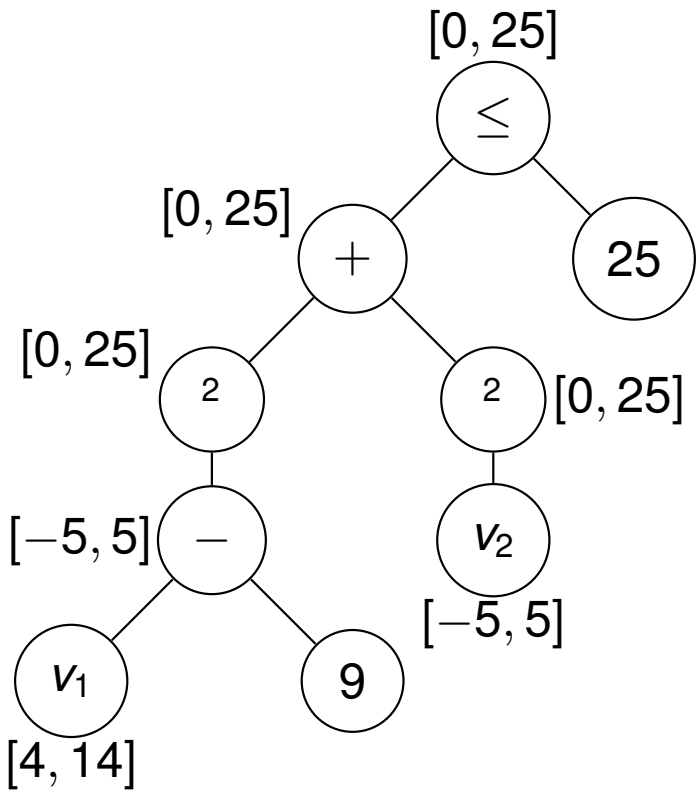
Let $(v_1 - 9)^2 + v_2^2 \leq 25$ be a constraint,
$D_1 = [-1, 14]$, $D_2 = [-5, 10]$ the domains



- Constraint syntactic tree
- Interval arithmetics

HC4-Revise

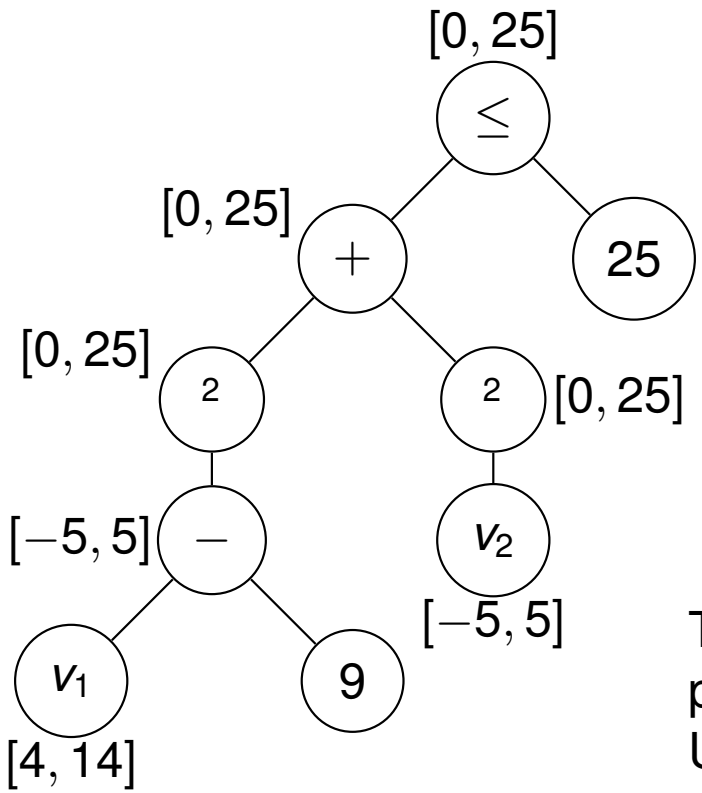
Let $(v_1 - 9)^2 + v_2^2 \leq 25$ be a constraint,
$D_1 = [-1, 14]$, $D_2 = [-5, 10]$ the domains



- Constraint syntactic tree
- Interval arithmetics
- This algorithm achieves Hull-Consistency only under certain conditions

HC4-Revise

Let $(v_1 - 9)^2 + v_2^2 \leq 25$ be a constraint,
$D_1 = [-1, 14]$, $D_2 = [-5, 10]$ the domains



- Constraint syntactic tree
- Interval arithmetics
- This algorithm achieves Hull-Consistency only under certain conditions

This algorithm has also been defined independently in AI under the name of Bottom-Up Top-Down [Cousot and Cousot, 1977]

Cas des contraintes globales

Les algorithmes de propagation des contraintes globales sont donnés au cas par cas.

Grande variété des méthodes utilisées :

- souvent issus des graphes ou des flots, par exemple pour les contraintes de cardinalité,
- ou bien de méthodes *ad hoc* : `sweep` adapté pour la contrainte `geost`
- ou inspirées de domaines proches : relaxations et programmation linéaire, newton et consistances continues,
- etc.

Contraintes de cardinalité

`alldifferent`: les valeurs des variables sont toutes différentes.

utile pour : des salles (edt), certaines ressources (planification)...

`nvalue`,`atleast`, `atmost` : les variables prennent exactement (au plus, au moins) *N* valeurs différentes

utile pour : problèmes d'emploi du temps, frequency allocation...

NB : satisfiaisabilité et consistance sont NP-dures.

`gcc` : le nombre de chaque valeur pouvant être prise par les variables est dans un intervalle donné.

utile pour : problèmes d'emploi du temps, séries magiques...

Contrainte `alldifferent`

Notation

Soit $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ un ensemble de variables, $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ leur domaines respectifs (finis et sur les entiers). On note m la taille de $\bigcup_{1 \leq i \leq n} D_i$.

Quelle différence entre `alldifferent`$(\mathcal{X})$ et $\bigwedge_{X_i, X_j \in \mathcal{X}, i \neq j} X_i \neq X_j$?

Contrainte `alldifferent`

Notation

Soit $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ un ensemble de variables, $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ leur domaines respectifs (finis et sur les entiers). On note m la taille de $\bigcup_{1 \leq i \leq n} D_i$.

Quelle différence entre `alldifferent`$(\mathcal{X})$ et $\bigwedge_{X_i, X_j \in \mathcal{X}, i \neq j} X_i \neq X_j$?

- Sémantiquement : aucune !

Contrainte `alldifferent`

Notation

Soit $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ un ensemble de variables, $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ leur domaines respectifs (finis et sur les entiers). On note m la taille de $\bigcup_{1 \leq i \leq n} D_i$.

Quelle différence entre `alldifferent`$(\mathcal{X})$ et $\bigwedge_{X_i, X_j \in \mathcal{X}, i \neq j} X_i \neq X_j$?

- Sémantiquement : aucune !
- Opérationnellement : tout !

Efficacité de la propagation

Critère d'efficacité : nombre de valeurs inconsistantes détectées.

Exemple :
$\mathcal{X} = \{X_1, X_2, X_3, X_4\}$ avec
$\mathcal{D} = \{\{1,2\},\{1,2\},\{2,3,4\},\{1,3,4,5\}\}$.

Efficacité de la propagation

Critère d'efficacité : nombre de valeurs inconsistantes détectées.

Exemple :
$\mathcal{X} = \{X_1, X_2, X_3, X_4\}$ avec
$\mathcal{D} = \{\{1,2\},\{1,2\},\{2,3,4\},\{1,3,4,5\}\}$.

- $\bigwedge_{X_i, X_j \in \mathcal{X}, i \neq j} X_i \neq X_j$ ne détecte aucune valeur inconsistante

Efficacité de la propagation

Critère d'efficacité : nombre de valeurs inconsistantes détectées.

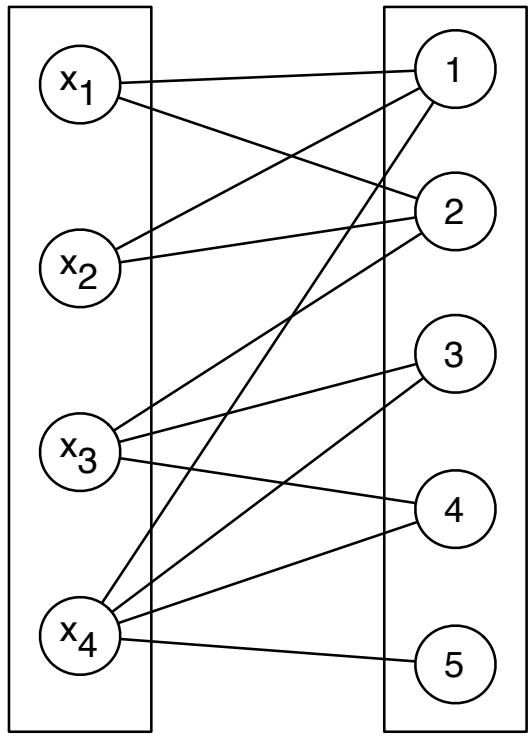
Exemple :
$\mathcal{X} = \{X_1, X_2, X_3, X_4\}$ avec
$\mathcal{D} = \{\{1,2\},\{1,2\},\{2,3,4\},\{1,3,4,5\}\}$.

- $\bigwedge_{X_i, X_j \in \mathcal{X}, i \neq j} X_i \neq X_j$ ne détecte aucune valeur inconsistante
- $\texttt{alldifferent}(X)$: les valeurs 2 et 1 sont respectivement inconsistantes pour X_3 et X_4.

Propagation de `alldifferent`:structure de données

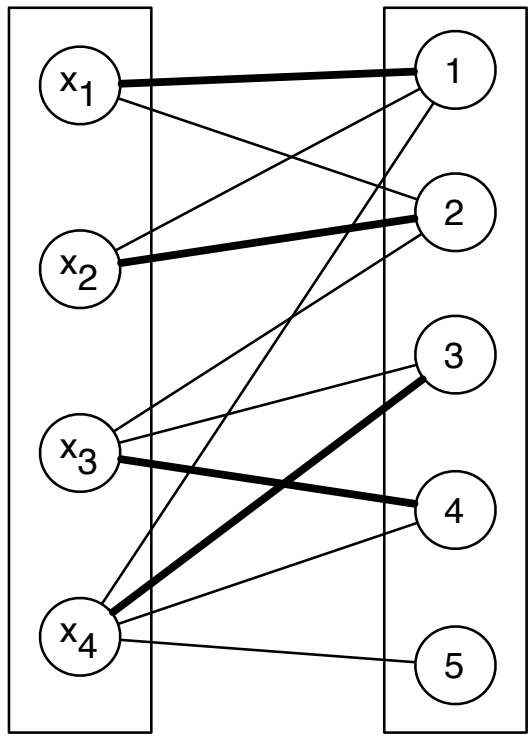
Basée sur un graphe bipartite $G = (\mathcal{X}, V, E)$:

- chaque sommet de $\mathcal{X}$ est une variable du `alldifferent`,
- chaque sommet de V est une valeur de $\bigcup_{D_i \in \mathcal{D}} D_i$,
- une arête $e = (X_i, k)$ appartient à E ssi $k \in D_i$.



Propagation de `alldifferent`:structure de données

Une contrainte `alldifferent`($\mathcal{X}$) est satisfaite ssi il existe un couplage maximal dans G, saturant les sommets de $\mathcal{X}$.



Propagation de `alldifferent`:GAC

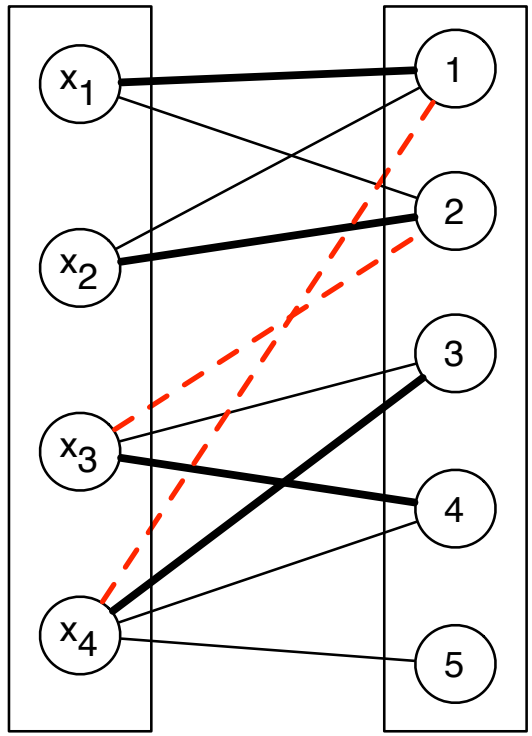
Dans notre contexte, un algorithme de filtrage se décrit informellement de deux manières équivalentes :

- détecte et supprime les valeurs inconsistantes dans $\bigcup_{D_i \in \mathcal{D}} D_i$,
- détecte et supprime des **arêtes incompatibles** dans G.

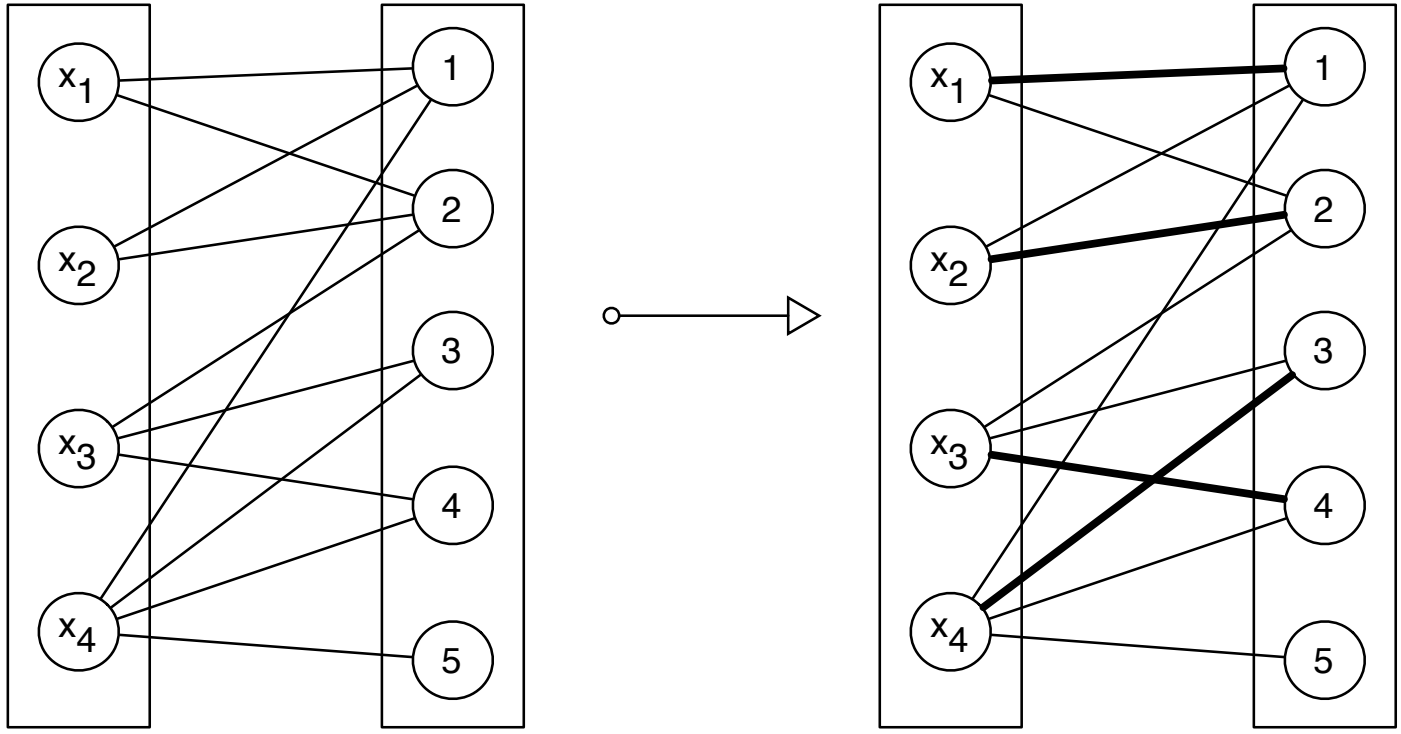
Propagation de `alldifferent`:GAC

Arêtes incompatibles

Pour une variable $X_i \in \mathcal{X}$ et une valeur $v \in D_i$, v est inconsistente pour la contrainte $\texttt{alldifferent}(\mathcal{X})$ ssi l'arête dans G correspondant à la paire (X_i, v) n'appartient à aucun couplage maximal de G.



Propagation de `alldifferent`:GAC



Calcul d'un couplage maximal en $O(m\sqrt{n})$

Propagation de `alldifferent`:GAC

[Berge, 70]

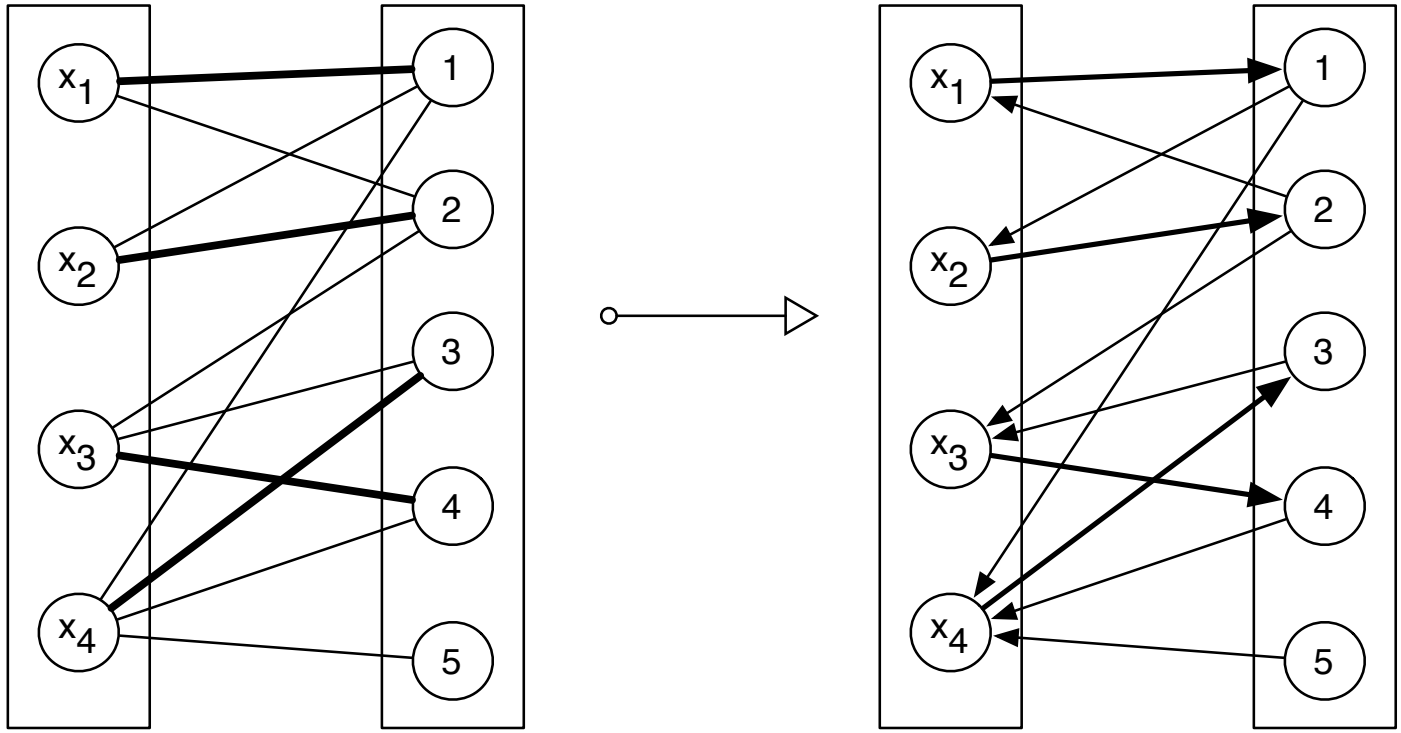
Une arête *e* appartient à au moins un couplage maximal ssi $\forall \mathcal{M}$ un couplage maximal, *e* appartient

- à un chemin alterné de longueur paire commençant sur un sommet libre (pas dans $\mathcal{M}$),
- ou à un cycle alterné de longueur paire.

Si on a un couplage maximal $\mathcal{M}$ donné, les arêtes n'appartenant à aucun couplage maximal sont celles qui:

- ne sont pas dans $\mathcal{M}$,
- n'appartiennent pas à un chemin alterné de longueur paire commençant sur un sommet libre,
- n'appartiennent pas à un cycle alterné de longueur paire.

Propagation de `alldifferent`:GAC



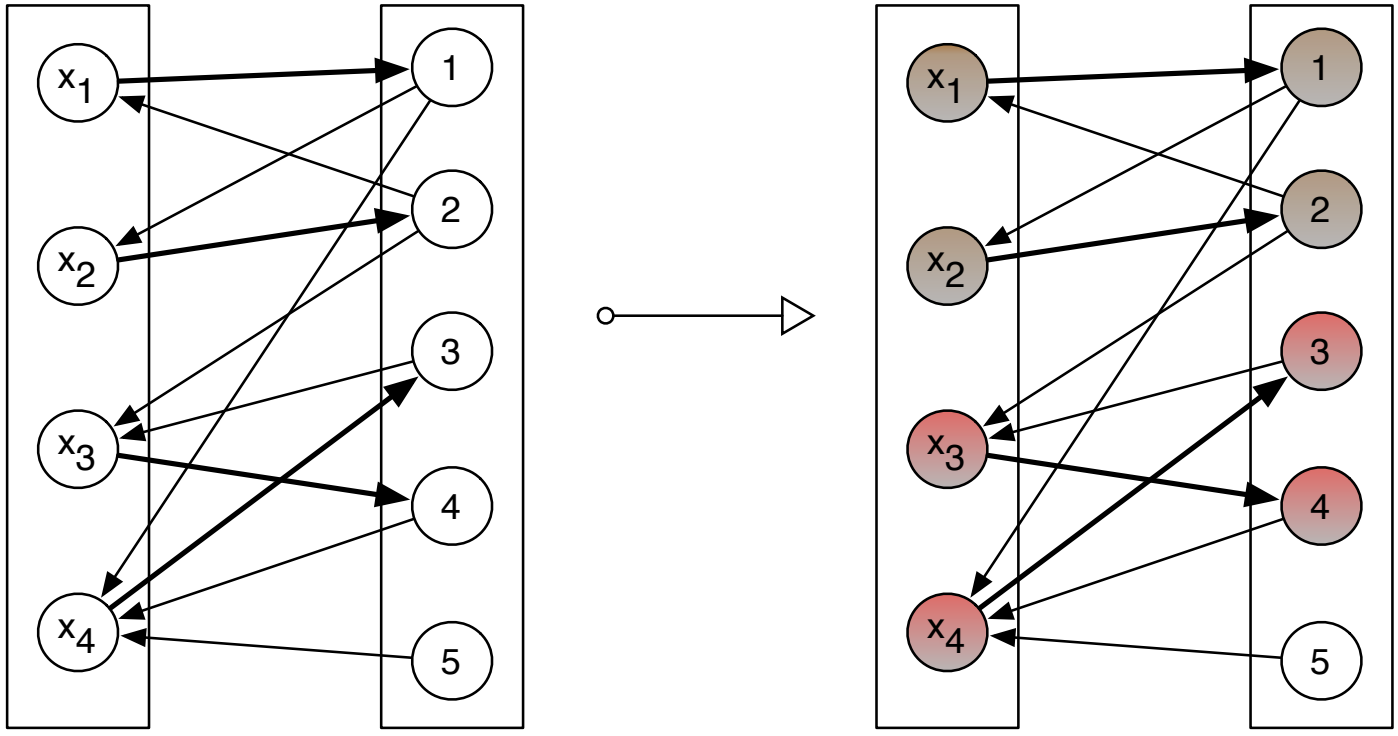
Transformation en un graphe orienté G', graphe résiduel, en $O(m+n)$

Propagation de `alldifferent`:GAC

On peut donc détecter les arêtes appartenant à une chaine alternée de longueur paire :

- Parcours en profondeur d'abord depuis un sommet libre,
- Marquer les arêtes utilisées, elles sont par construction dans une chaine alternée de longueur paire.

Propagation de `alldifferent`:GAC



Identification des cfc's dans G' en $O(m+n)$

Propagation de `alldifferent`:GAC

Complétons l'algorithme précédent :

- Les arêtes internes à chaque cfc sont à marquer car elles sont, par construction, sur un cycle alterné de longueur paire,
- Supprimer toutes les arêtes non marquées.

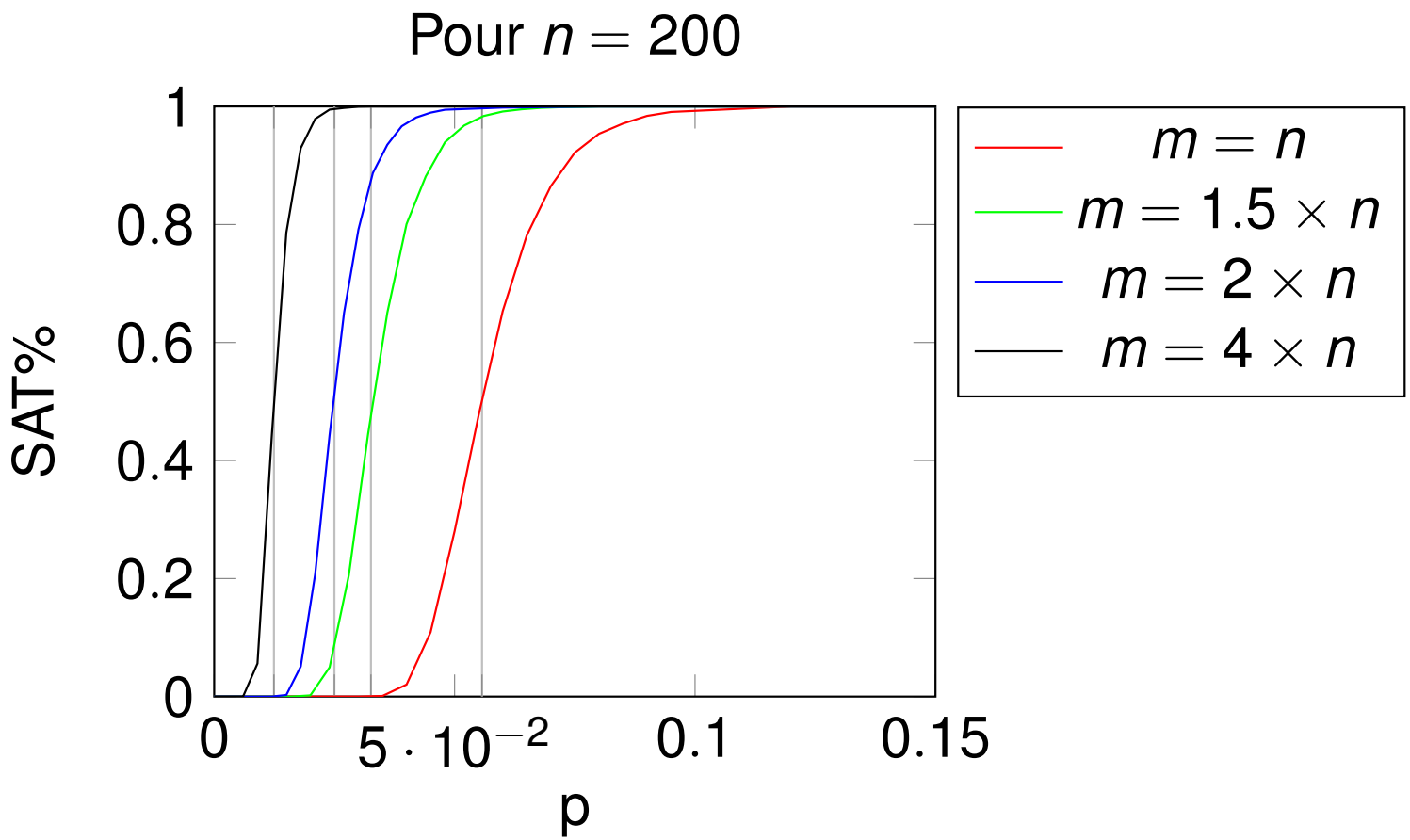
Les limites

En théorie, il s'agit d'un bon algorithme. Mais en pratique :

- re-calcul à chaque réveil (peut être coûteux),
- implémenté de façon incrémentale :
  - on ne recalcule le couplage que si le précédent n'est plus valide,
  - on ne recalcule pas toujours les cfc's.

Quelques propriétés : nombre de solutions

Etude de la satisfiabilité d'une l'instance `alldifferent` en fonction de sa densité d'arêtes.



Quelques propriétés : propagation

Propagation of the `alldifferent`

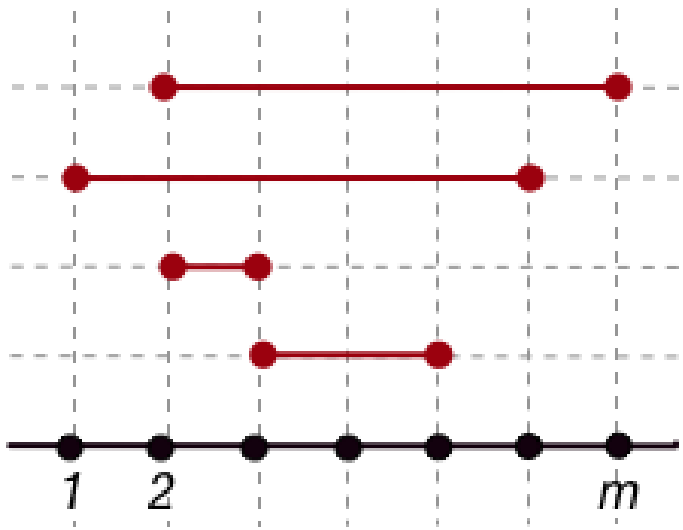
- propagation of the $n * (n-1)/2$ `neq` constraints,
- propagation of the global constraint.

Question: does an algorithm based on BC remove values?
Search for a fair balance efficiency / cost of the propagation ([Katriel, 2006]).

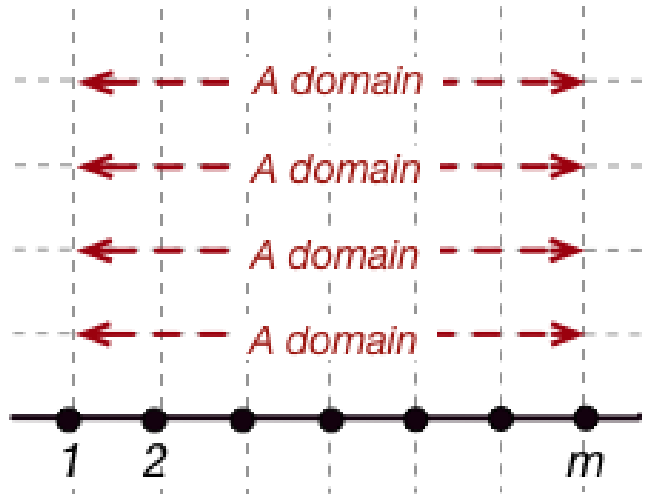
A probabilistic model

Consider a given `alldifferent` on given domains $D_1 \ldots D_n$.

Real situation



Probabilistic model



Propagation, asymptotically

Key result: identification of two different asymptotical regimes for the probability of remaining consistent after an instanciation.

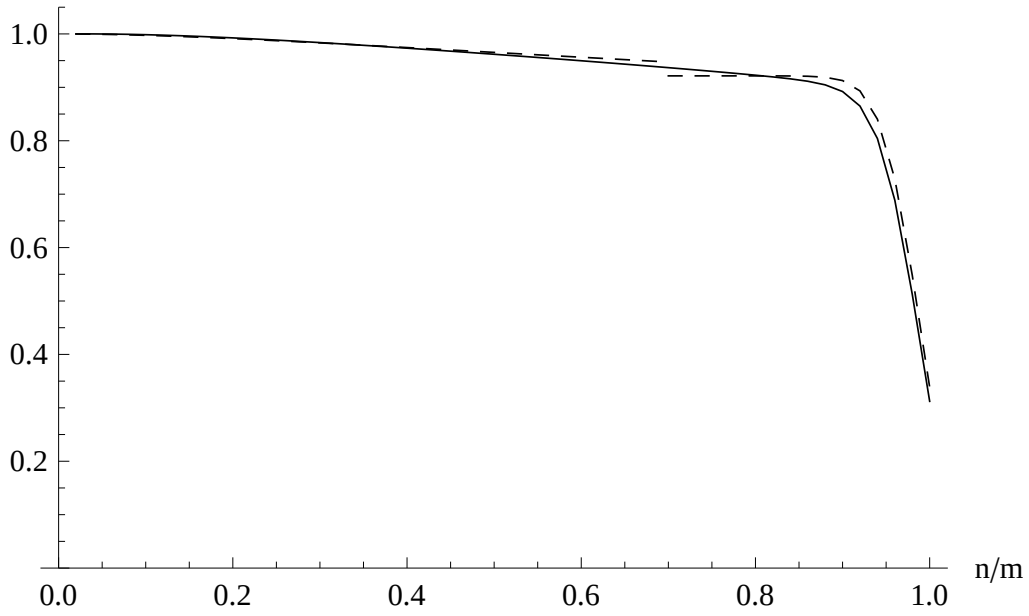
- if $n/m = \rho, \rho < 1$,
  then the limit is 1,

  the BC propagation is useless
- if $n = m - o(m)$,
  then the limit is strictly smaller than 1.

  it depends

In practice

Numerical evaluation of the theoretical (plain) and approached (dashed) probability of remaining BC for $m = 50$ and n varying from 1 to m.



A criterion

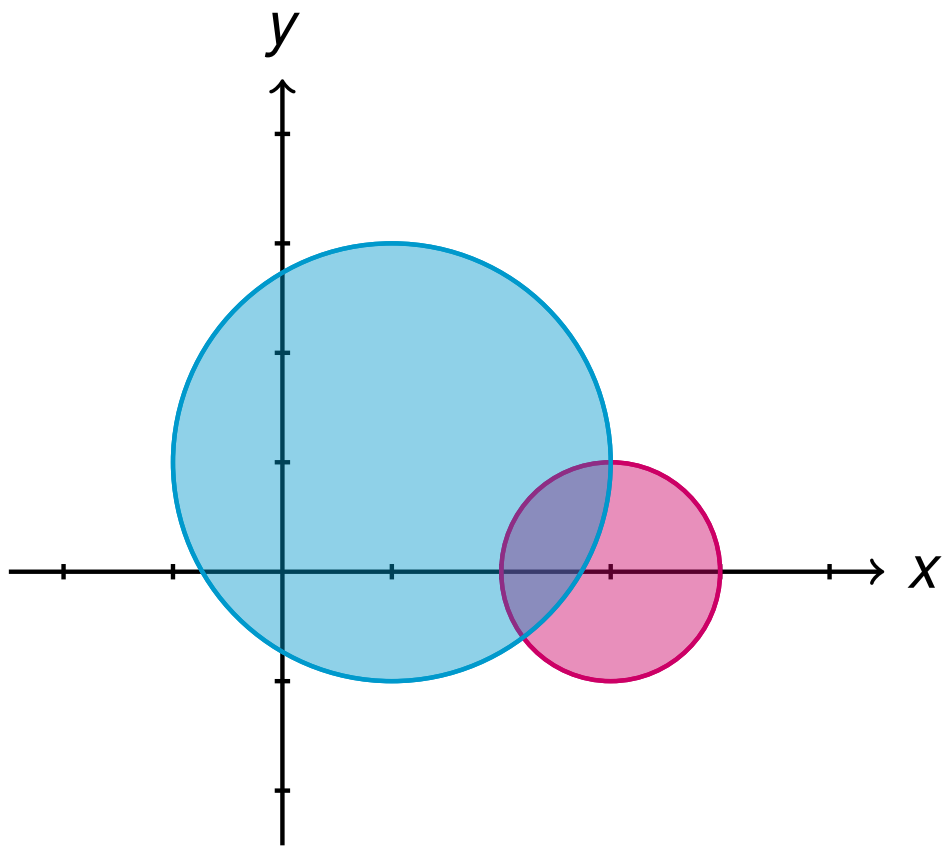
An `alldifferent` constraint on n variables and m values is sharp iff $m - n < 2\sqrt{m}$.

Propagation Loop

- For a conjunction of constraints, for each constraint the consistency is applied until a fixpoint is reached [Benhamou, 1996, Apt, 1999]
- At each iteration, not all the constraints are propagated [Mackworth, 1977]
- The number of iterations depends on:
  - the constraints, and
  - the order in which the constraints consistency is applied
- Designing an efficient propagation loop is still a challenge [Schulte and Tack, 2001]

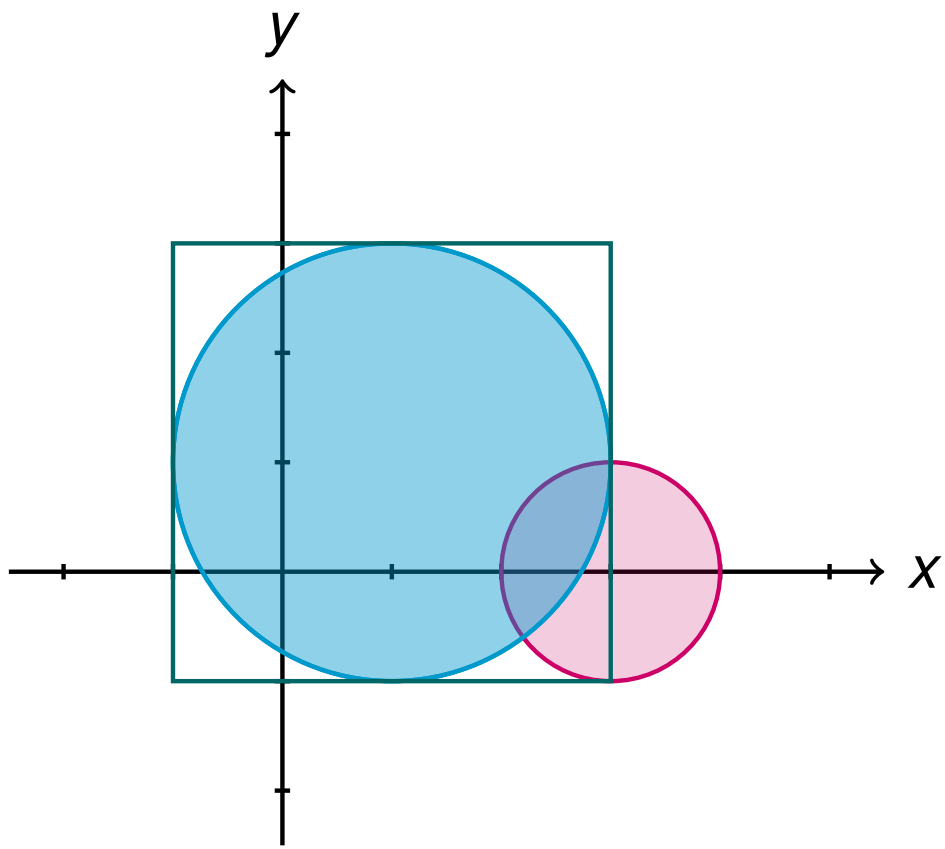
Propagation of several constraints: a continuous example

Computation of a fixpoint of the different propagators.



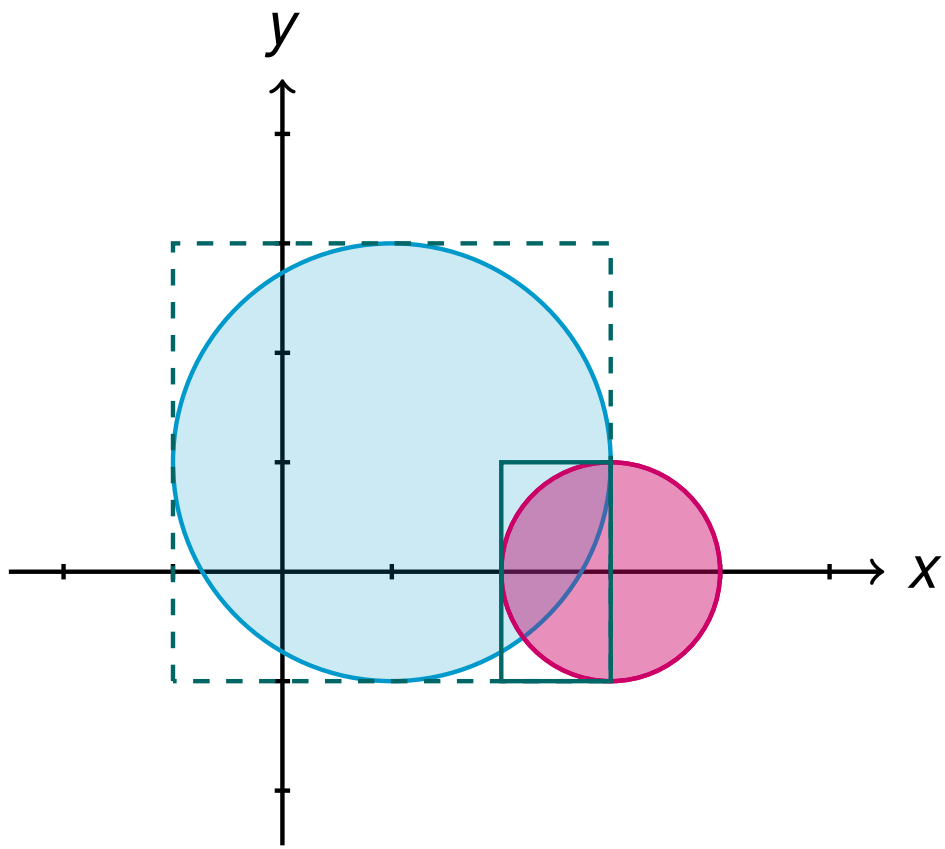
Propagation of several constraints: a continuous example

Computation of a fixpoint of the different propagators.



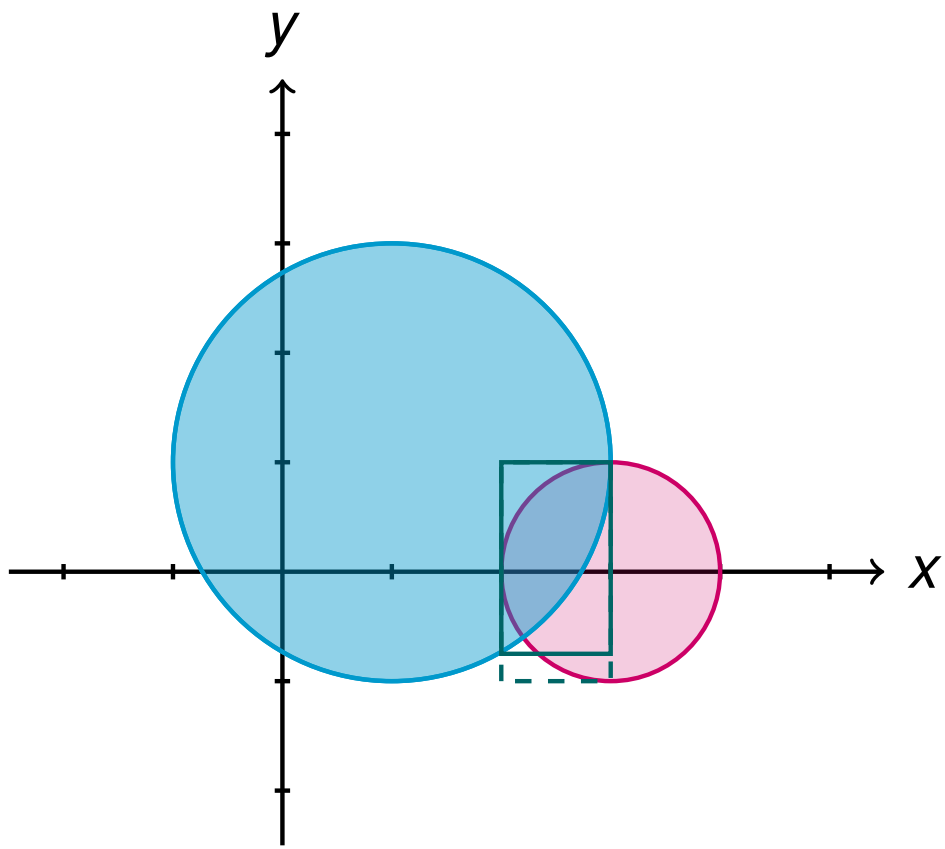
Propagation of several constraints: a continuous example

Computation of a fixpoint of the different propagators.



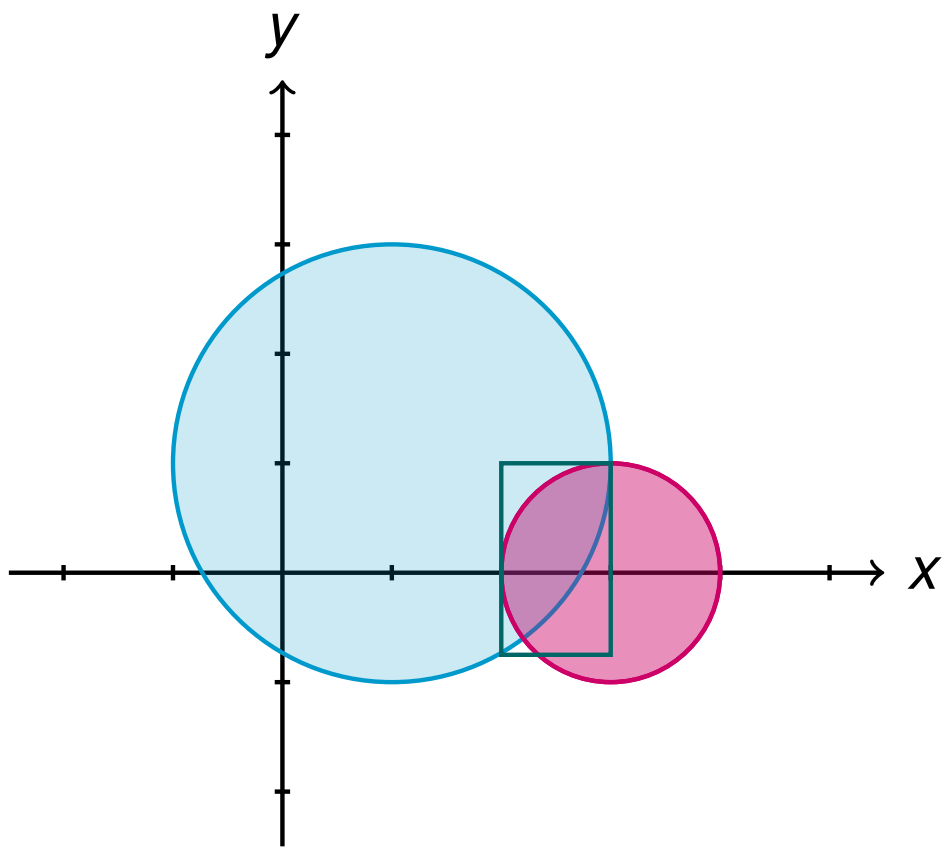
Propagation of several constraints: a continuous example

Computation of a fixpoint of the different propagators.



Propagation of several constraints: a continuous example

Computation of a fixpoint of the different propagators.



Classical propagation loops

Sur des contraintes binaires, algorithme d'arc-consistance : on suppose connue une procédure REVISE(V_i, V_j) qui calcule la consistance des contraintes $C(V_i, V_j)$, et renvoie vrai ssi les domaines ont été modifiés.

Algorithme AC1 (1977)

```
procedure AC1
bool change
Q ← constraints on Vi and Vj for i ≠ j
repeat
    change ← false
    for C(Vi,Vj) in Q do
        change ← REVISE(Vi,Vj) or change
until not change
```

Complexité : $O(\sharp var * \sharp ctes * \sharp tailledom^3)$

Classical propagation loops

Il n'est pas utile de réveiller toutes les contraintes : algorithme AC3.

Algorithme AC3 (1977)

procedure AC3
bool change
Q $\leftarrow$ constraints on V_i and V_j for $i \neq j$
while Q not empty **do**
  C(V_i, V_j) $\leftarrow$ pop Q
  if REVISE(V_i, V_j) **then**
    Q $\leftarrow$ Q $\cup$ { constraints on V_i, V_k pour tout $k \neq i, j$ }

Complexité : $O(\sharp ctes * \sharp tailledom^3)$

Classical propagation loops

Et puis...

- AC4 (1986) : stocke les supports des valeurs ($O(\sharp var * \sharp ctes * \sharp tailledom^2)$)
- AC6 (1994) : maintient un seul support,
- AC7 (1999) : améliore AC4 et AC6 et exploite les symétries
- AC2000, AC2001, etc

Accélérer la convergence au point fixe ?

Aujourd'hui, beaucoup de solveurs utilisent des boucles de propagations à base d'évènements comme:

- une variable a été fixée,
- un domaine a été modifié.

Le tuning précis de la boucle de propagation, pour accélérer la convergence au point fixe, est encore un sujet de recherche [Prudhomme et al. 2016].

Outline

Résolution ?

La consistance ne suffit pas, en général, à trouver une / toutes les solutions.

Principe général d'un solveur

On alterne

- propagation des contraintes (raisonnement),
- splits / instanciations : hypothèses sur les domaines, qui donnent plusieurs branches à explorer.

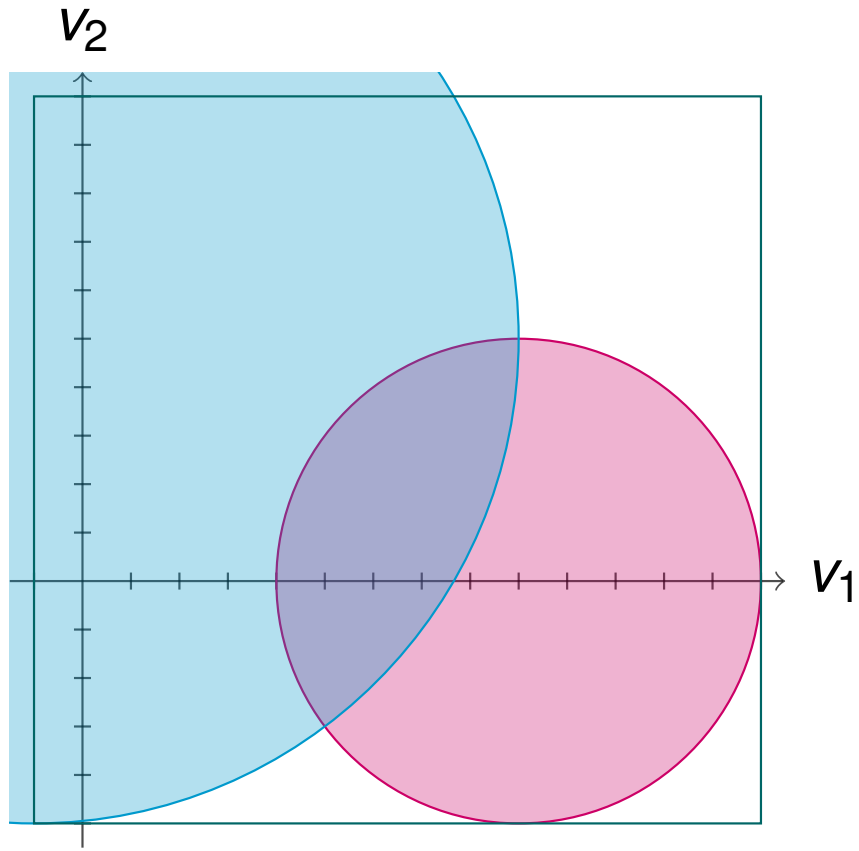
Continuous Solving Method

```
Parameter: float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Hull-Consistency(e)
   if e ≠ ∅ then
      if maxDim(e) ≤ r or isSol(e)
      then
         sols ← sols ∪ e
      else
         split e in two boxes e1 and
         e2
         push e1 and e2 in toExplore
```



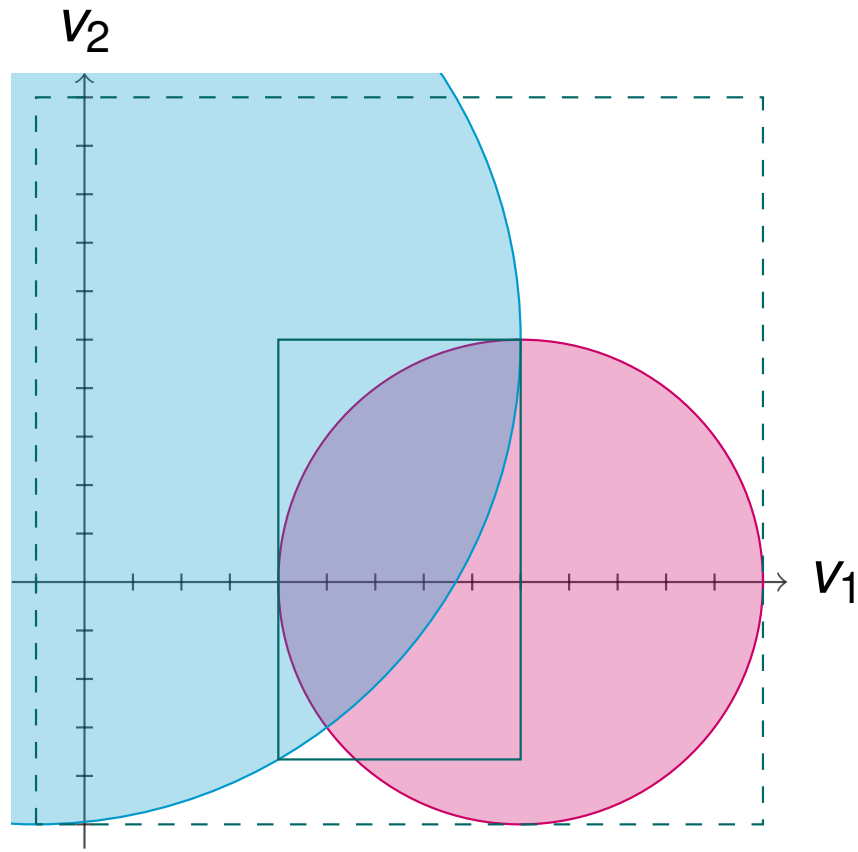
Continuous Solving Method

```
Parameter: float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Hull-Consistency(e)
   if e ≠ ∅ then
      if maxDim(e) ≤ r or isSol(e)
      then
         sols ← sols ∪ e
      else
         split e in two boxes e1 and
         e2
         push e1 and e2 in toExplore
```



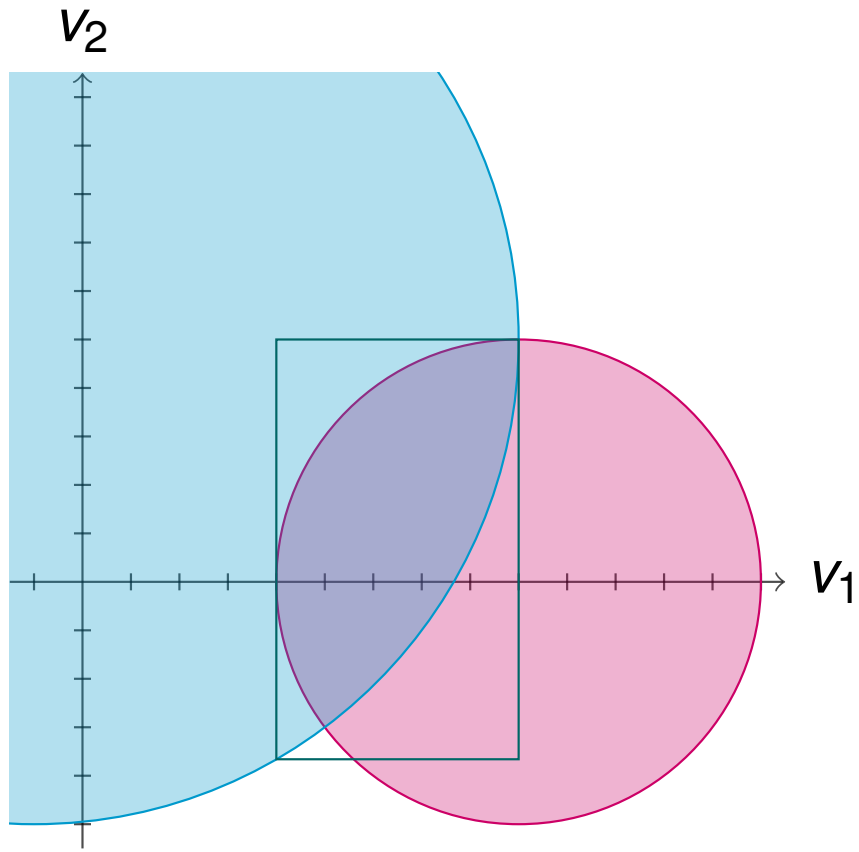
Continuous Solving Method

```
Parameter: float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Hull-Consistency(e)
   if e ≠ ∅ then
      if maxDim(e) ≤ r or isSol(e)
      then
         sols ← sols ∪ e
      else
         split e in two boxes e1 and
         e2
         push e1 and e2 in toExplore
```



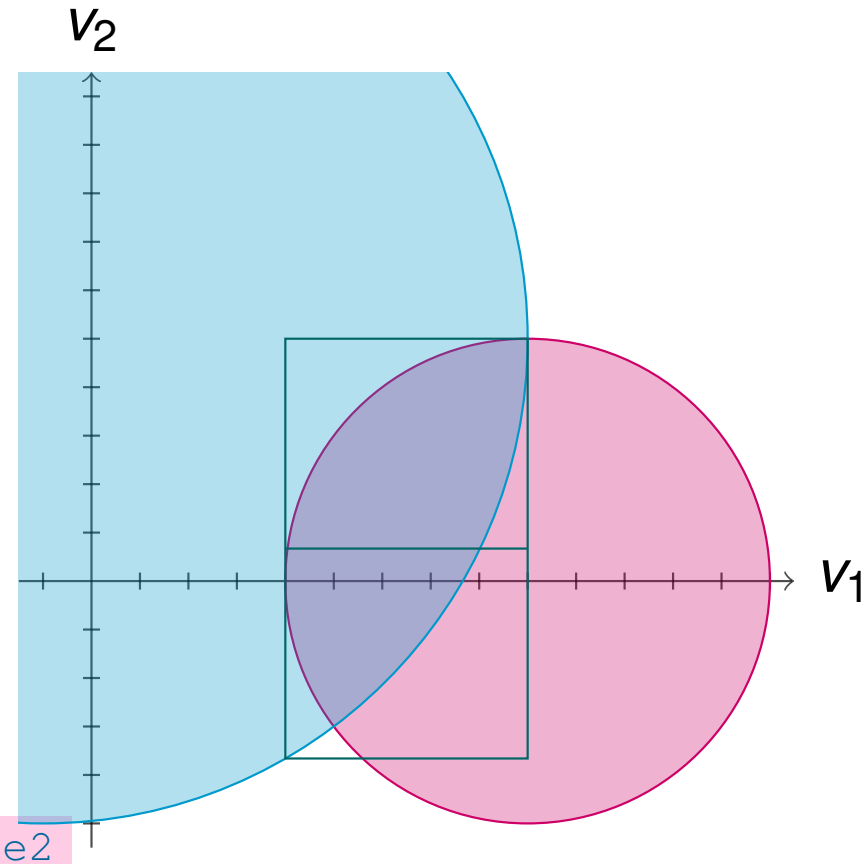
Continuous Solving Method

```
Parameter: float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Hull-Consistency(e)
   if e ≠ ∅ then
      if maxDim(e) ≤ r or isSol(e)
      then
         sols ← sols ∪ e
      else
         split e in two boxes e1 and e2
         push e1 and e2 in toExplore
```



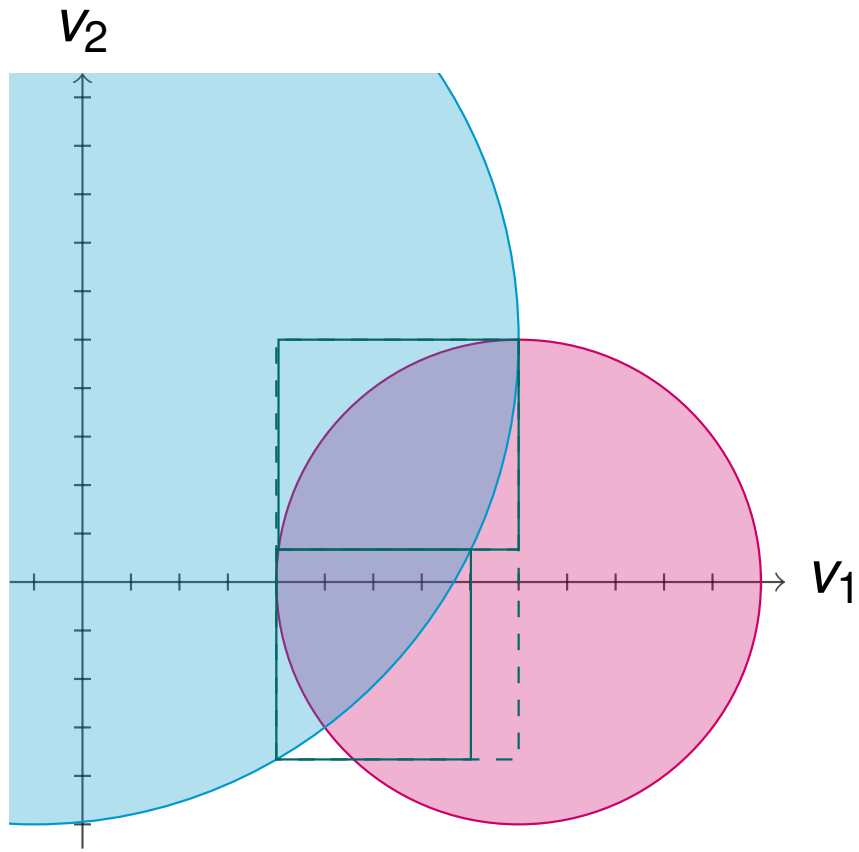
Continuous Solving Method

```
Parameter: float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Hull-Consistency(e)
   if e ≠ ∅ then
      if maxDim(e) ≤ r or isSol(e)
      then
         sols ← sols ∪ e
      else
         split e in two boxes e1 and
         e2
         push e1 and e2 in toExplore
```



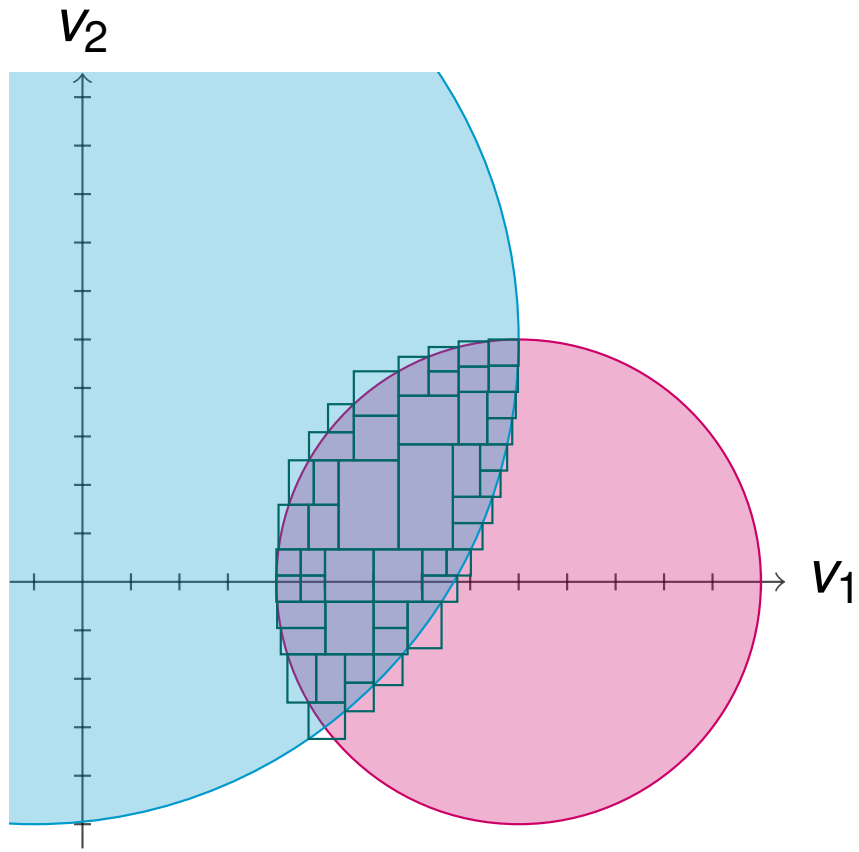
Continuous Solving Method

```
Parameter: float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Hull-Consistency(e)
   if e ≠ ∅ then
      if maxDim(e) ≤ r or isSol(e)
      then
         sols ← sols ∪ e
      else
         split e in two boxes e1 and
         e2
         push e1 and e2 in toExplore
```

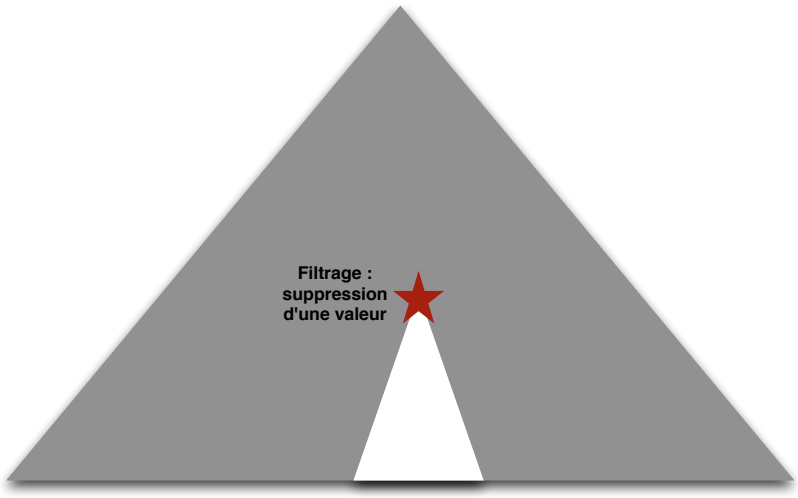


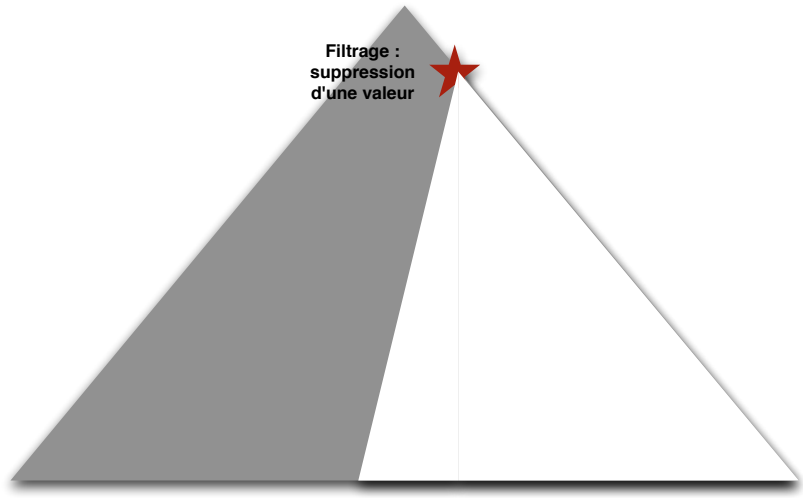
Résolution ?

Dessin au tableau...

Un exemple

Généralement, on essaie de provoquer les échecs au plus tôt.





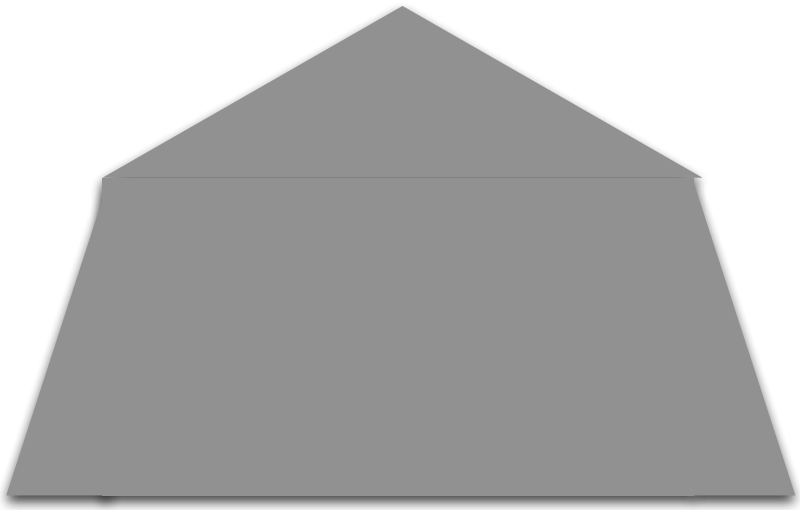
La difficulté

Toutes les structures de données doivent être backtrackables: à chaque backtrack, on doit restaurer un état antérieur du solveur (avec parfois des choses en plus).

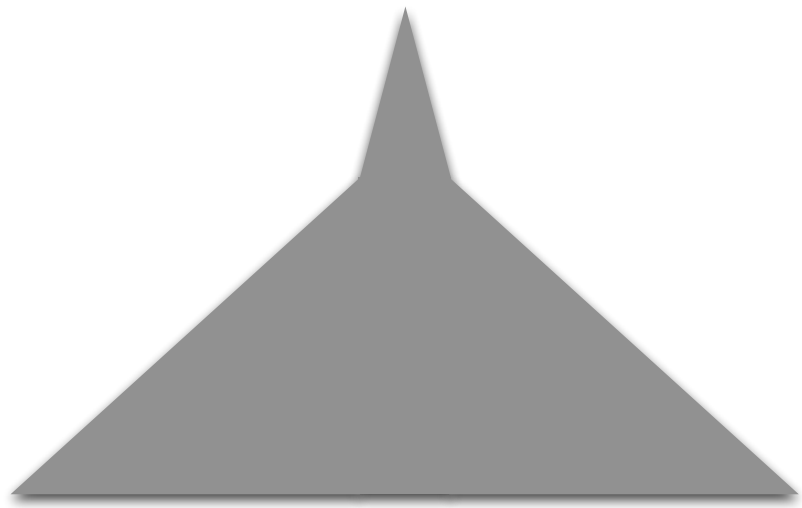
Devinette

Lequel est le meilleur ?
NB : les surfaces sont égales.

Plus grands domaines d'abord



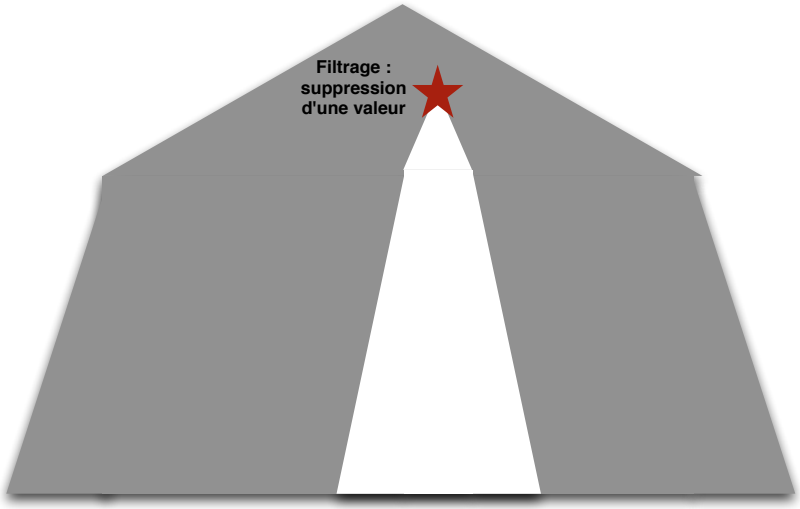
Plus petits domaines d'abord



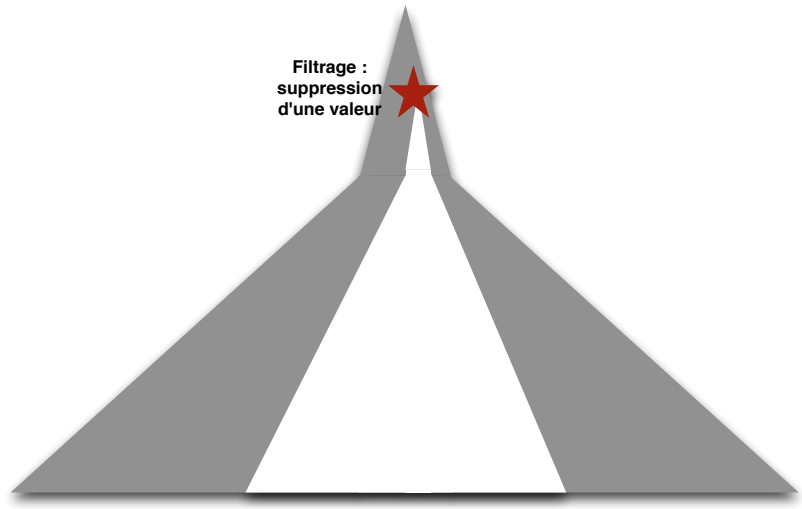
Devinette

Lequel est le meilleur ?
NB : les surfaces sont égales.

Plus grands domaines d'abord



Plus petits domaines d'abord



First fail : plus petit domaine d'abord, variables les plus contraintes d'abord.

Heuristics

- dom: smallest domain first,
- deg, wdeg: most constrained variable first (possibly with weights),
- dom/wdeg: the previous ones combined,
- activity: dynamically adapts to the *efficiency* of the constraints,
- counting-based search: uses estimations (or ub) of the number of solutions for the global constraints (cardinality),
- on continuous domains, largest dimension first,
- *ad hoc* heuristics.

There is no such thing as a Free Lunch.

Some active solvers

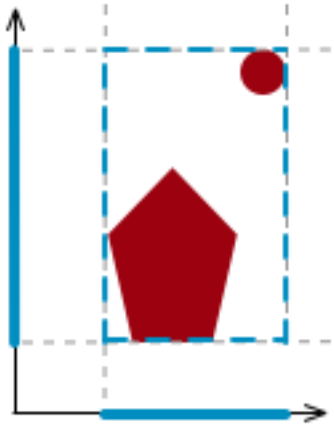
- Choco: java library, free
  `http://www.emn.fr/z-info/choco-solver/`
- gecode: C++ library, free
  `-http://www.gecode.org/`
- ORTools: C++, interface in Python, free,
  `https://code.google.com/p/or-tools/`
- Oscar: Scala, free,
  `https://bitbucket.org/oscarlib/oscar/wiki/Home`
- Prolog family: ECLiPSe, Sicstus
- AbSolute, OCaml, free,
- plenty of others!

Two yearly competitions

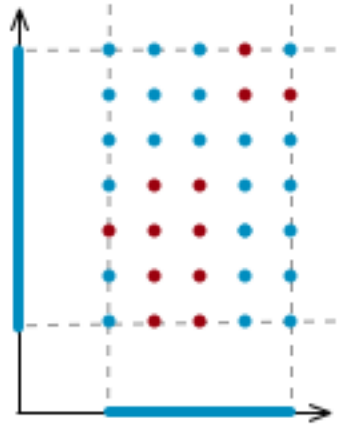
- XCSP3
  `http://www.cril.univ-artois.fr/XCSP18/`
- MiniZinc Challenge
  `http://www.minizinc.org/challenge2018/challenge.html`

Outline

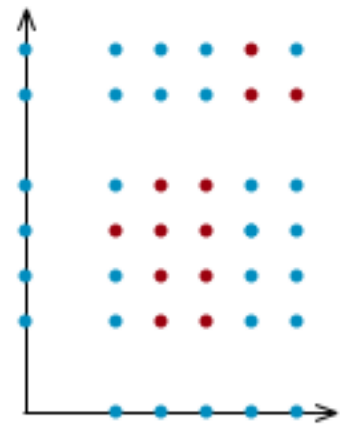
Consistency



Hull-consistency



Bound-consistency



Generalized arc-consistency

Two key remarks

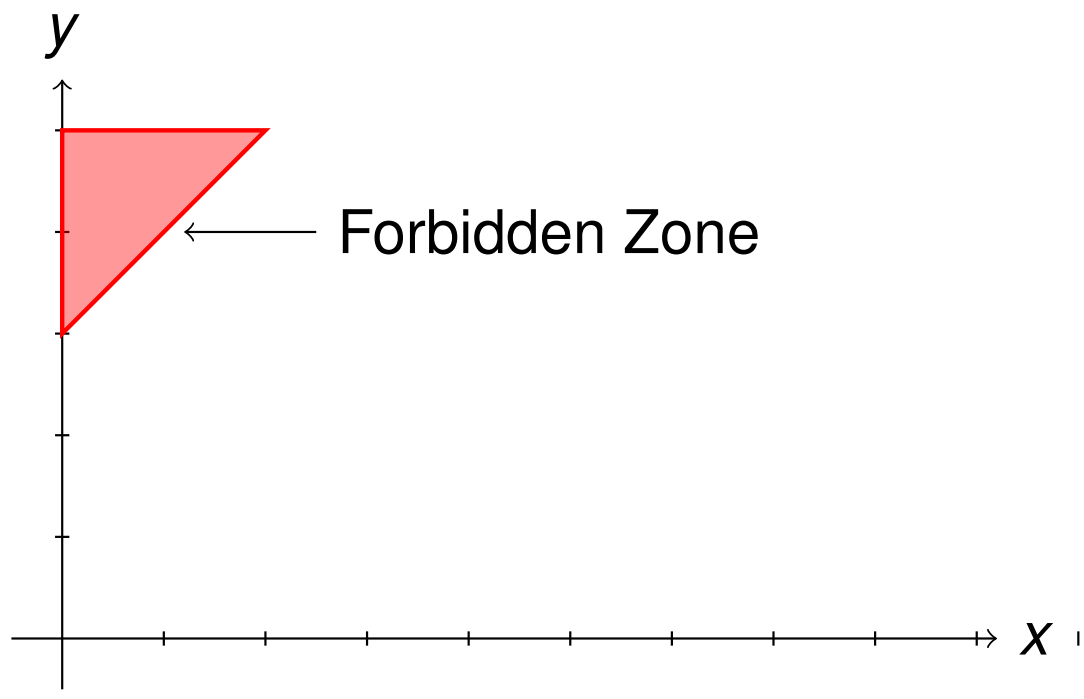
- consistency is not about where the solutions are, it is about where they are *not*,
- why square?

Abstract Interpretation

- Abstract Interpretation (AbsInt) is a theory of approximation of the semantics [Cousot and Cousot, 1976]
- Applied to static analysis and verification of software
- Goal: automatically prove that a program does not have execution errors
- Key idea: abstract the valuations of the programs variables

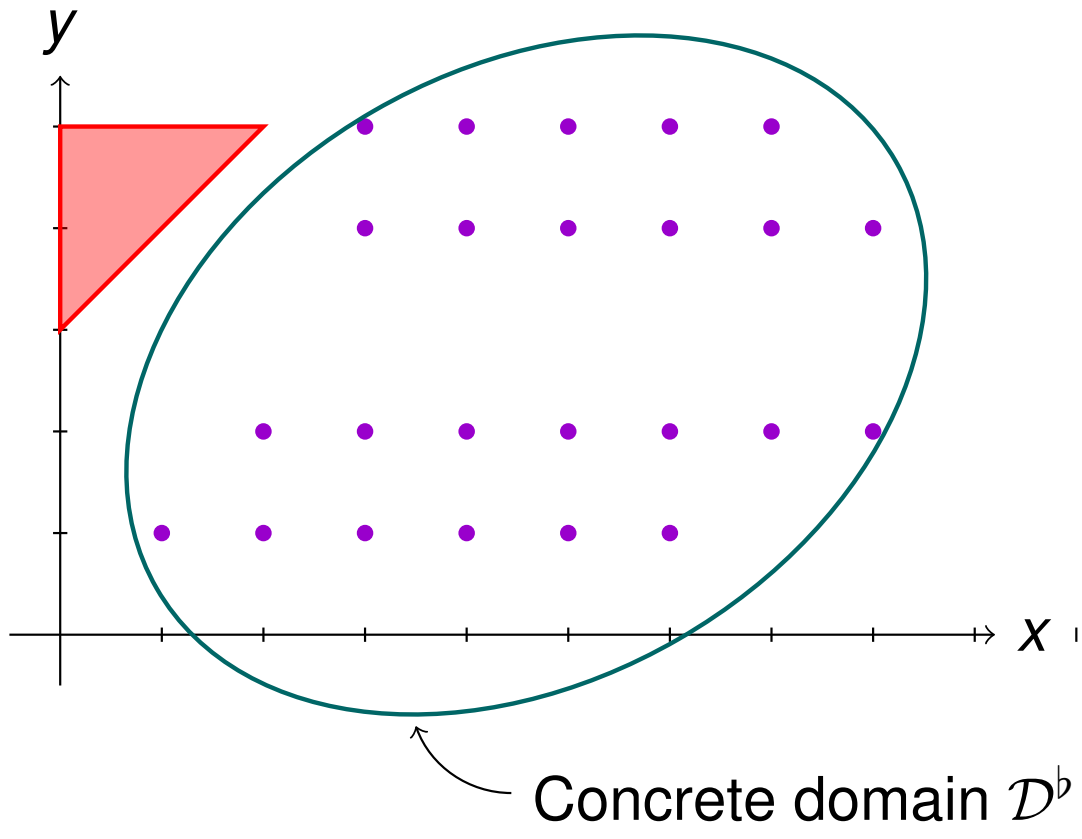
Abstract Domain

```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
   x ← x+y
   y ← 2*y
x ← x-1
y ← y+1
```



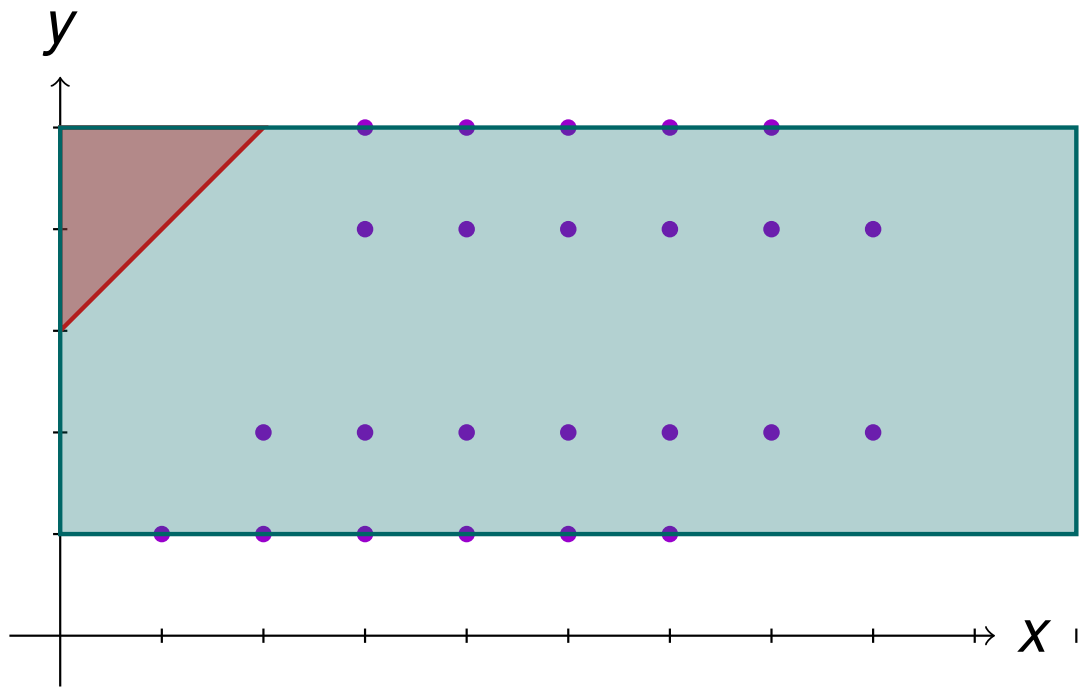
Abstract Domain

```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
   x ← x+y
   y ← 2*y
x ← x-1
y ← y+1
```



Abstract Domain

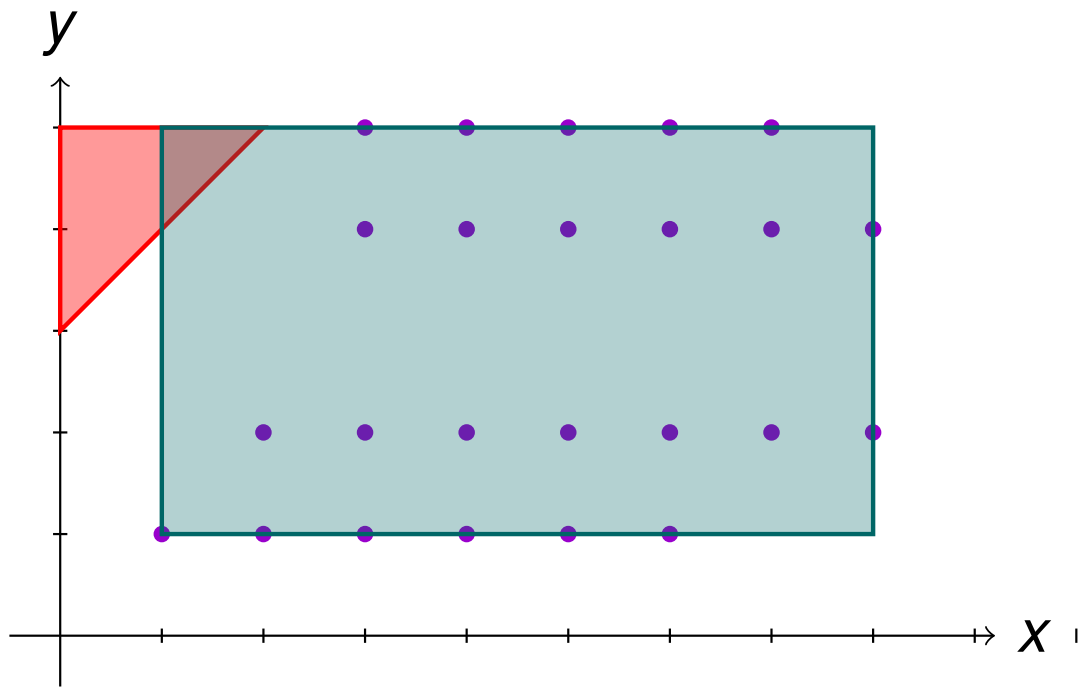
```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
   x ← x+y
   y ← 2*y
x ← x-1
y ← y+1
```



Boxes

Abstract Domain

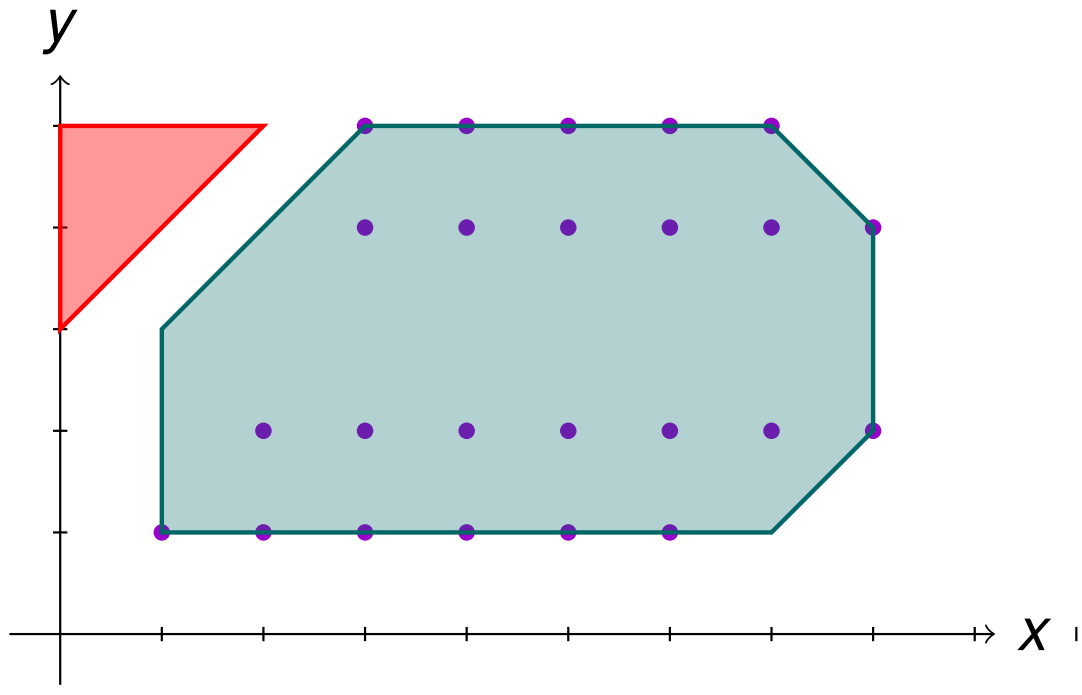
```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
   x ← x+y
   y ← 2*y
x ← x-1
y ← y+1
```



Better boxes

Abstract Domain

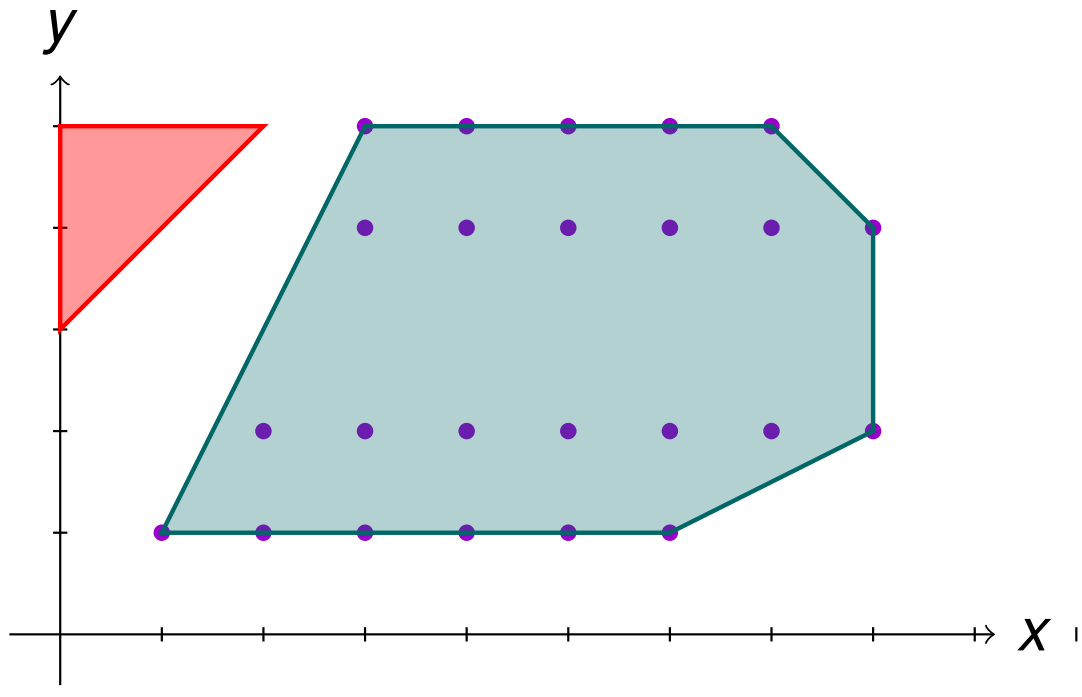
```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
   x ← x+y
   y ← 2*y
x ← x-1
y ← y+1
```



Octagons

Abstract Domain

```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
   x ← x+y
   y ← 2*y
x ← x-1
y ← y+1
```



Convex polyhedra

AI ? CP ?

En AI

peu importe de savoir où on va, du moment qu'on sait où on ne va pas.

AI ? CP ?

En AI

peu importe de savoir où on va, du moment qu'on sait où on ne va pas.

En CP

si on sait où on ne va pas, on le coupe.

Les liens

$PPC \cap IA$

Approximations d'espaces compliqués ou impossibles à calculer exactement.

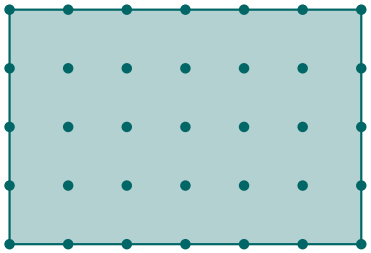
$IA \setminus PPC$

- nombreux domaines abstraits
- cadre théorique commun

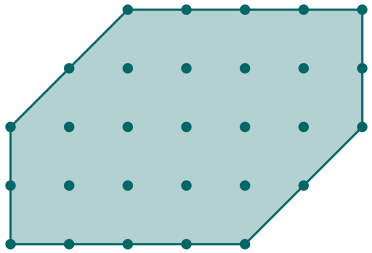
$PPC \setminus IA$

- nombreuses heuristiques
- précision

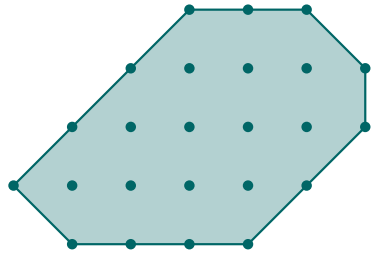
What already exist in AI



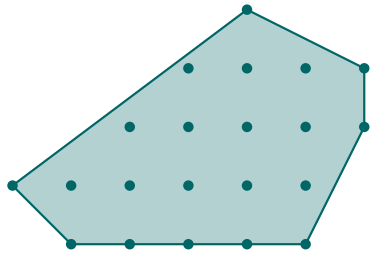
Intervals



Zonotopes



Octagons

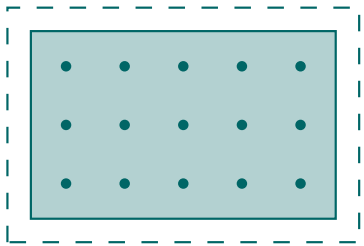


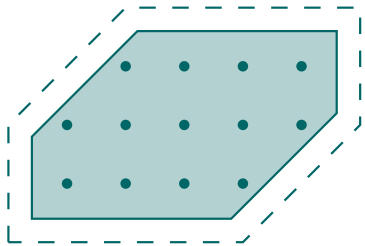
Polyhedron

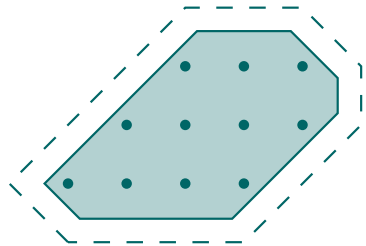
Abstract domains come with:

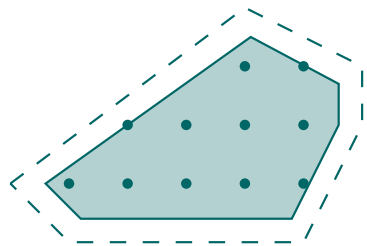
- transfer functions $\rho^\sharp$ (assignment, test, ...)
- meet $\cap^\sharp$ and join $\cup^\sharp$
- widening $\triangledown^\sharp$ and narrowing $\triangle^\sharp$

What already exist in AI


Intervals


Zonotopes


Octagons


Polyhedron

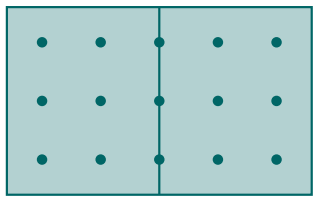
Abstract domains come with:

- transfer functions $\rho^\sharp$ (assignment, test, ...)
- meet $\cap^\sharp$ and join $\cup^\sharp$
- widening $\triangledown^\sharp$ and narrowing $\triangle^\sharp$

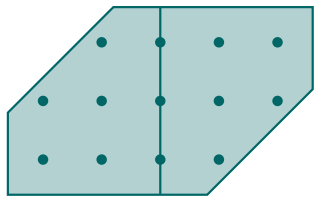
We need:

- a consistency/propagation ρ

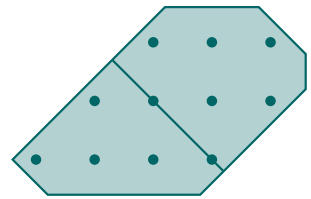
What already exist in AI



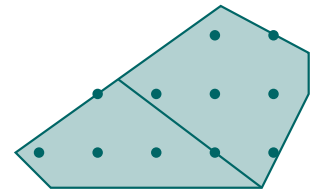
Intervals



Zonotopes



Octagons



Polyhedron

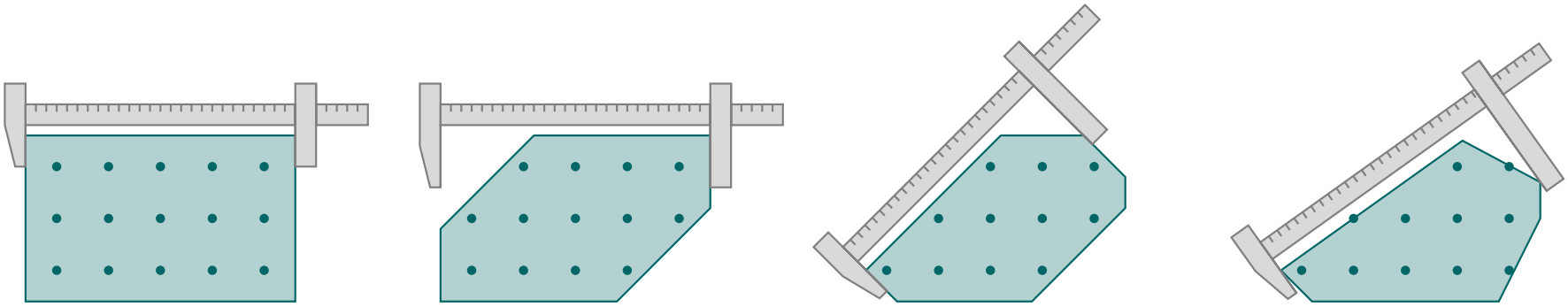
Abstract domains come with:

- transfer functions $\rho^\sharp$ (assignment, test, ...)
- meet $\cap^\sharp$ and join $\cup^\sharp$
- widening $\triangledown^\sharp$ and narrowing $\triangle^\sharp$

We need:

- a consistency/propagation ρ
- a splitting operator $\oplus$

What already exist in AI



Intervals Zonotopes Octagons Polyhedron

Abstract domains come with:

- transfer functions $\rho^\sharp$ (assignment, test, ...)
- meet $\cap^\sharp$ and join $\cup^\sharp$
- widening $\triangledown^\sharp$ and narrowing $\triangle^\sharp$

We need:

- a consistency/propagation ρ
- a splitting operator $\oplus$
- a size function τ

Abstract Solving Method

We define the resolution as a concrete semantics

Propagation

- Consistency, using test transfer functions
- Propagation loop, fixpoint using local iterations [Granger, 1992]

Abstract Solving Method

Exploration

- Splitting operator in disjunctive completion
- Size function

Definition (Disjunctive completion [Cousot and Cousot, 1992])

Given an abstract domain $\mathcal{D}^\sharp$, a disjunctive completion $\mathcal{E}^\sharp = \mathcal{P}_{\mathit{finite}}(\mathcal{D}^\sharp)$ is a subset of incomparable elements of $\mathcal{D}^\sharp$

Continuous Solving Method

```
Parameter: float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e ← D

push e in toExplore

while toExplore ≠ ∅ do
  e ← pop(toExplore)
  e ← propagate(e)
  if e ≠ ∅ then
    if maxDim(e) ≤ r or isSol(e) then
      sols ← sols ∪ e
    else
      split e in two boxes e1 and e2
      push e1 and e2 in toExplore
```

Abstract Solving Method

Parameter: float r

~~list of boxes~~ disjunction sols $\leftarrow \emptyset$
~~queue of boxes~~ disjunction toExplore $\leftarrow \emptyset$
~~box~~ abstract domain e $\leftarrow$ ~~D~~ $\top^\sharp$

push e in toExplore

while toExplore $\neq \emptyset$ **do**
  $e \leftarrow$ **pop**(toExplore)
  $e \leftarrow$ ~~propagate(e)~~ $\rho^\sharp(e)$
  if $e \neq \emptyset$ **then**
    if ~~maxDim(e)~~ $\tau(e) \leq r$ **or** isSol(e) **then**
      sols $\leftarrow$ sols $\cup\, e$
    else
      ~~split e in two boxes e1 and e2~~
      push ~~e1 and e2~~ $\oplus(e)$ in toExplore

Under some conditions on the operators, this abstract solving method terminates, is correct and/or complete.

AbSolute

Un solveur

- écrit en OCaml
- basé sur la librairie Apron (domaines abstraits numériques),
- résolution paramétrable (domaine, précision, profondeur maximum ...),
- architecture modulaire,
- mode visualisation.

`https://github.com/mpelleau/AbSolute`

Bientôt dans opam !

Le langage de contraintes actuel

⟨*bool-expr*⟩ ::= <arith-expr> $\square$ <arith-expr>
$\square \in \{>, \geq, <, \leq, =, \neq\}$
| $\neg$ <bool-expr>
| <bool-expr> $\vee$ <bool-expr>
| <bool-expr> $\wedge$ <bool-expr>

⟨*arith-expr*⟩ ::= c — $c \in \mathcal{R}$
| $\mathcal{V}$ — variable
| <arith-expr> $\oplus$ <arith-expr>
$\oplus \in \{+, -, *, /, \%\}$
| - <arith-expr>
| *ident* '('<args>')' — appel de fonction

⟨*args*⟩ ::= <arith-expr> { "," <arith-expr> }

L'architecture du solveur

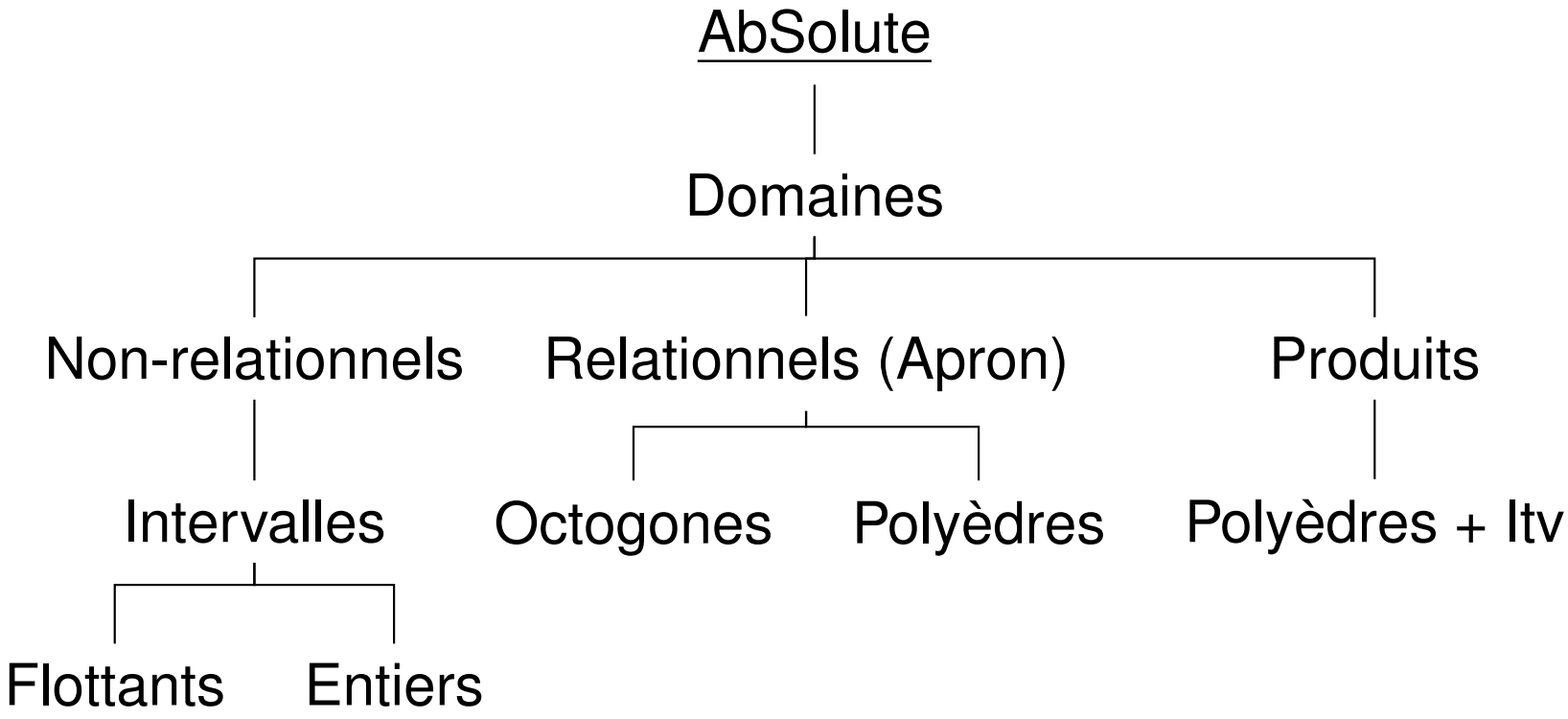
Un solveur basé sur des domaines abstraits

domaine abstrait

```
type domain

val init : prob -> domain
val propagate : domain -> constraints ->
   domain
val split : domain -> domain list
val size : domain -> bool
```

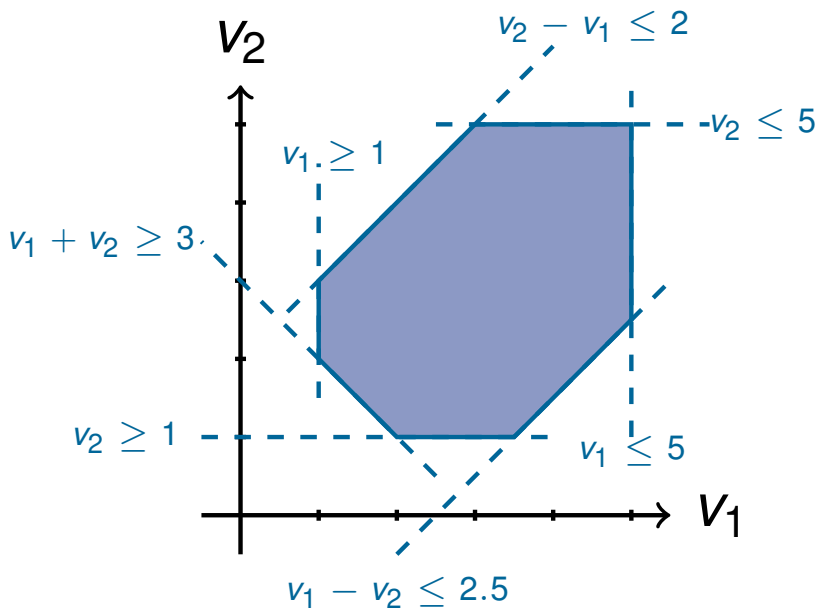
Les domaines abstraits



Octagons

Definition (Octagon [Miné, 2006])

Set of points satisfying a conjunction of constraints of the form $\pm v_i \pm v_j \leq c$, called octagonal constraints



- In dimension n, an octagon has at most $2n^2$ faces
- An octagon can be unbounded

Octagons

A Difference Bound Matrix (DBM) to represent an octagon

$$\begin{array}{c} \\ v_i \end{array} \begin{array}{c} \begin{array}{cc} & v_j \end{array} \\ \begin{pmatrix} \cdots & c \\ & \end{pmatrix} \end{array} \qquad v_j - v_i \leq c$$

Octagons

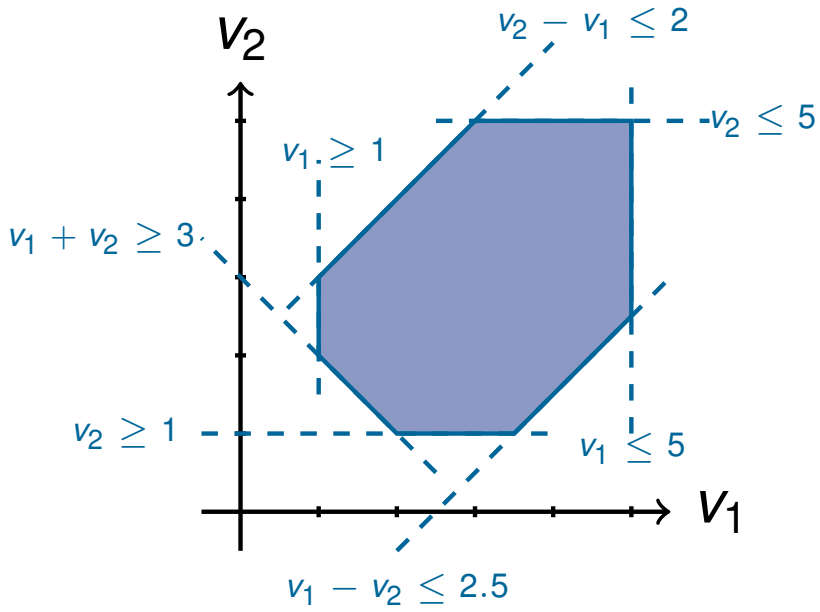
A Difference Bound Matrix (DBM) to represent an octagon

$$\begin{array}{c} \\ v_i \end{array} \begin{array}{c} \begin{array}{cc} & v_j \end{array} \\ \begin{pmatrix} \cdots & \vdots \\ & c \\ & \end{pmatrix} \end{array} \qquad v_j - v_i \leq c$$

$v_1 + v_2 \leq c \quad \rightarrow \quad v_1 - (-v_2) \leq c \quad$ and $\quad v_2 - (-v_1) \leq c$

Need of two rows and columns for each variable $(v_i, -v_i)$

Octagons

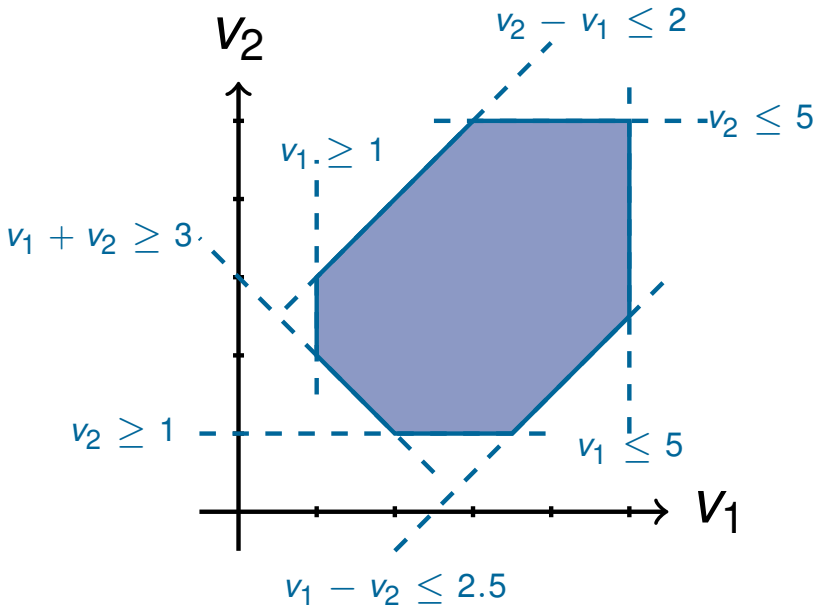


$$
\begin{array}{c} \\ v_1 \\ -v_1 \\ v_2 \\ -v_2 \end{array}
\begin{array}{c}
\begin{array}{cccc} v_1 & -v_1 & v_2 & -v_2 \end{array} \\
\begin{pmatrix} 0 & -2 & 2 & -3 \\ 10 & 0 & +\infty & 2.5 \\ 2.5 & -3 & 0 & -2 \\ +\infty & 2 & 10 & 0 \end{pmatrix}
\end{array}
$$

Canonical Representation

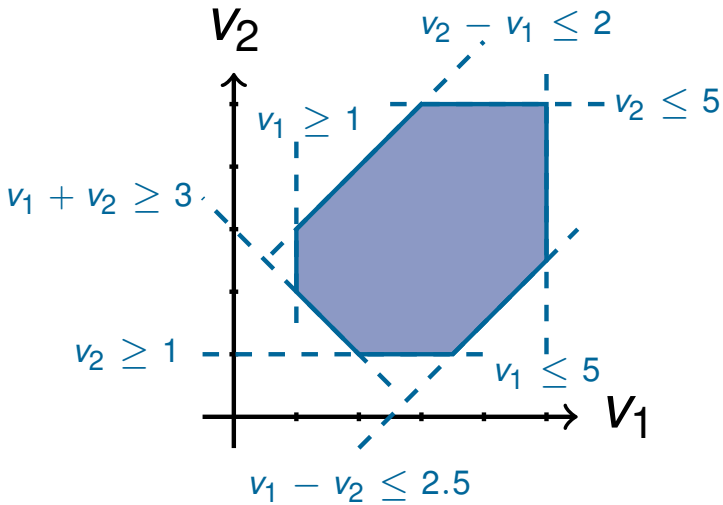
- Different DBMs can correspond to the same octagon $\Rightarrow$ need for a canonical form
- A modified version of Floyd-Warshall shortest path algorithm computes in $\mathcal{O}(n^3)$ the smallest DBM representing an octagon of dimension n [Miné, 2006]
- This canonical form corresponds to the consistency of the difference constraints [Dechter *et al.*, 1991]

Octagons

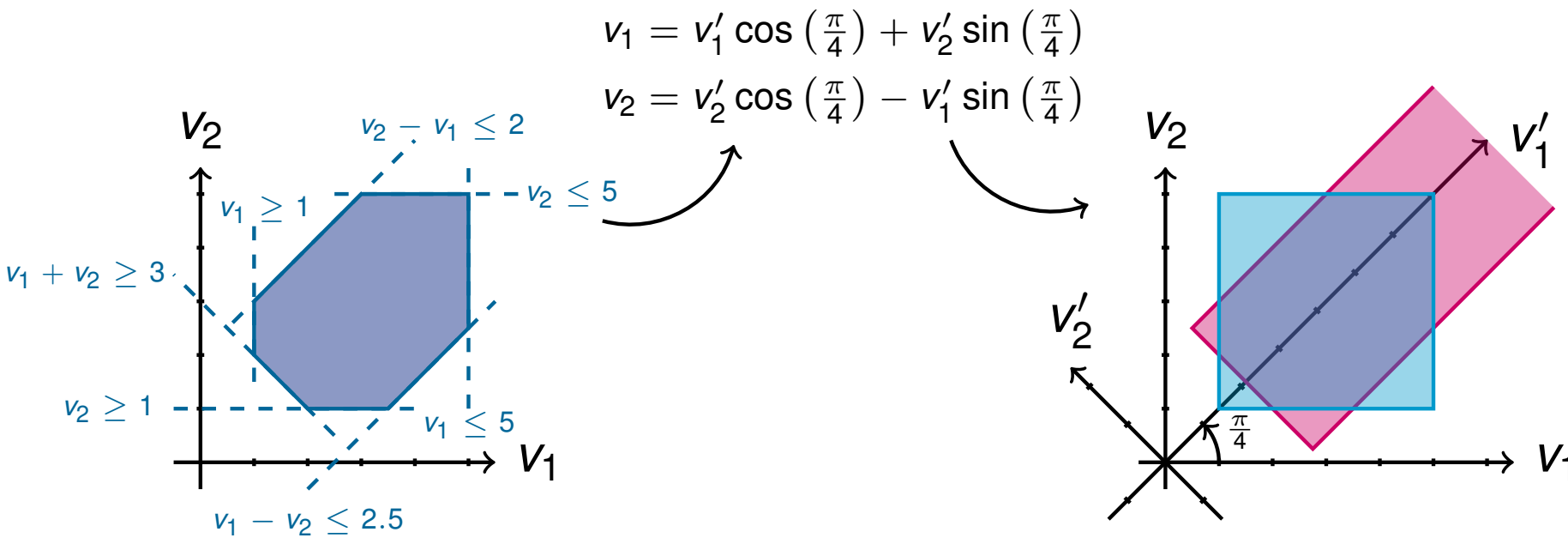


$$\begin{array}{c} \\ v_1 \\ -v_1 \\ v_2 \\ -v_2 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & -v_1 & v_2 & -v_2 \end{array} \\ \begin{pmatrix} 0 & -2 & 2 & -3 \\ 10 & 0 & 10 & 2.5 \\ 2.5 & -3 & 0 & -2 \\ 10 & 2 & 10 & 0 \end{pmatrix} \end{array}$$

Representation for CP



Representation for CP



Octagons for CP

Rotation

$$v_1 = v_1' \cos\left(\frac{\pi}{4}\right) + v_2' \sin\left(\frac{\pi}{4}\right)$$
$$v_2 = v_2' \cos\left(\frac{\pi}{4}\right) - v_1' \sin\left(\frac{\pi}{4}\right)$$

Constraint Translation

$$2v_1 - v_2 \leq 4$$

Octagons for CP

Rotation

$$v_1 = v_1' \cos\left(\frac{\pi}{4}\right) + v_2' \sin\left(\frac{\pi}{4}\right)$$
$$v_2 = v_2' \cos\left(\frac{\pi}{4}\right) - v_1' \sin\left(\frac{\pi}{4}\right)$$

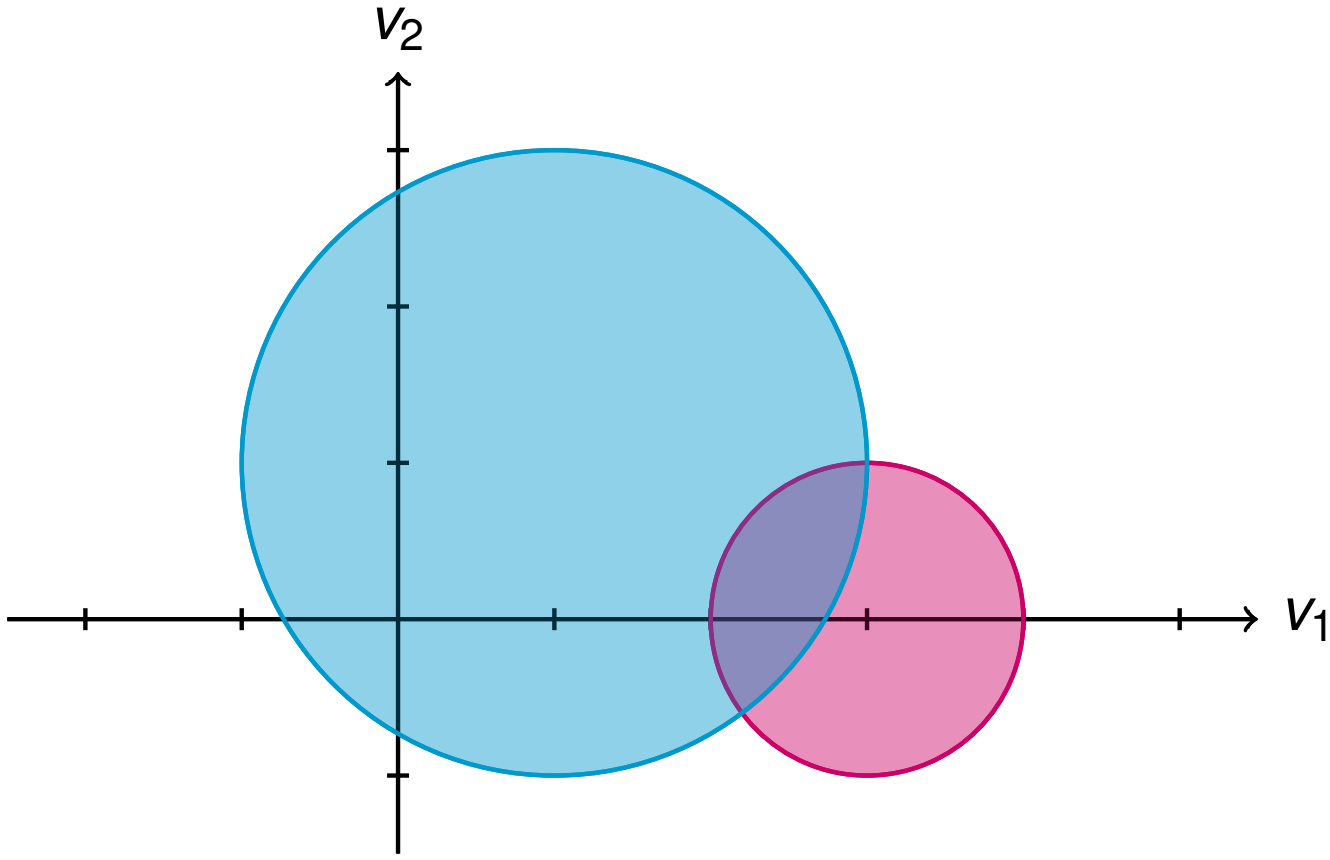
Constraint Translation

$$\begin{aligned} & 2v_1 - v_2 \leq 4 \\ \Leftrightarrow \quad & 2(v_1' \cos\left(\tfrac{\pi}{4}\right) + v_2' \sin\left(\tfrac{\pi}{4}\right)) - v_2' \cos\left(\tfrac{\pi}{4}\right) + v_1' \sin\left(\tfrac{\pi}{4}\right) \leq 4 \\ \Leftrightarrow \quad & 3\tfrac{\sqrt{2}}{2} v_1' + \tfrac{\sqrt{2}}{2} v_2' \leq 4 \end{aligned}$$

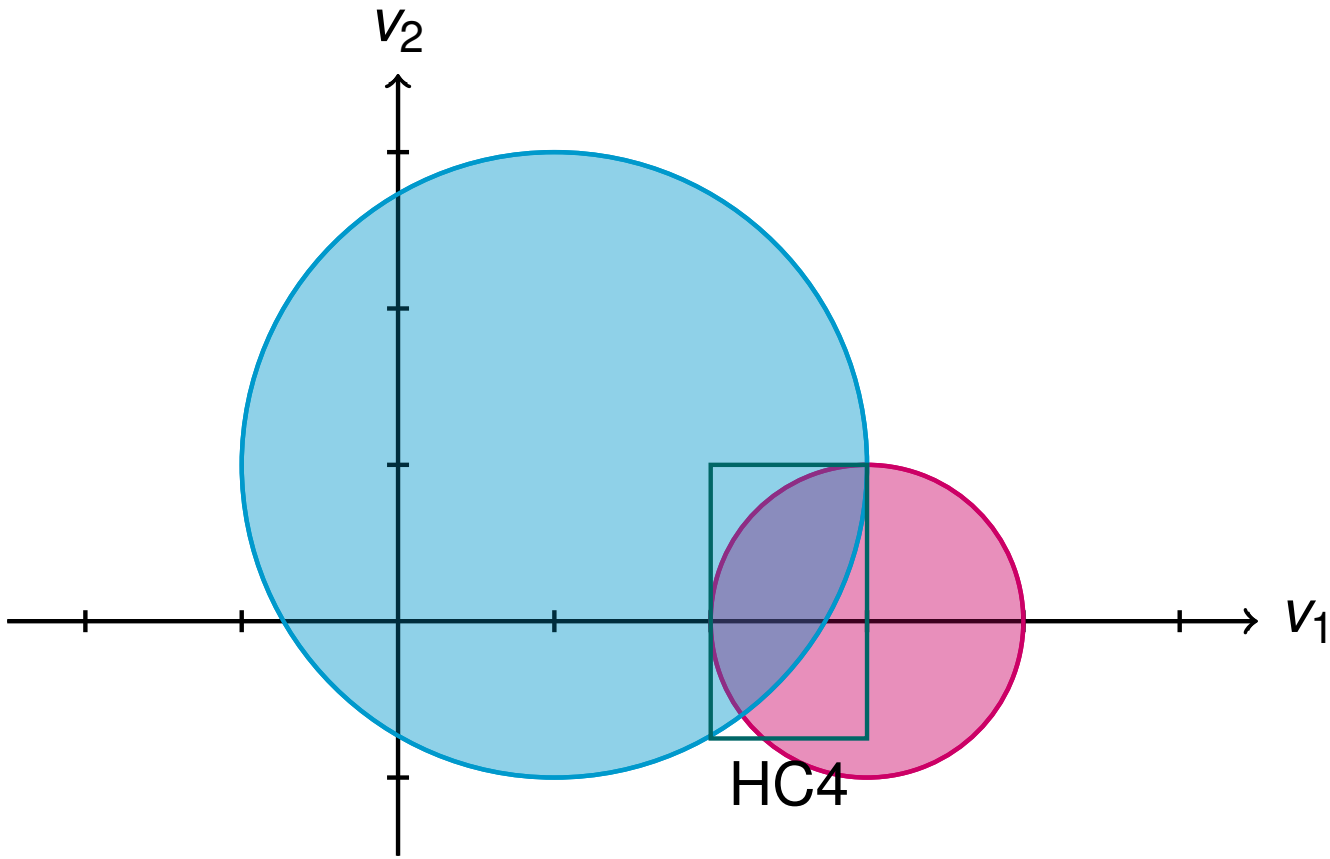
Representation for CP

- Representation in $\mathcal{O}(n^2)$ for a CSP with n variables and p constraints
  - n^2 variables
  - $p(n(n-1)+2)/2$ constraints
- Back to the boxes
- Direct definition of the needed tools for the resolution

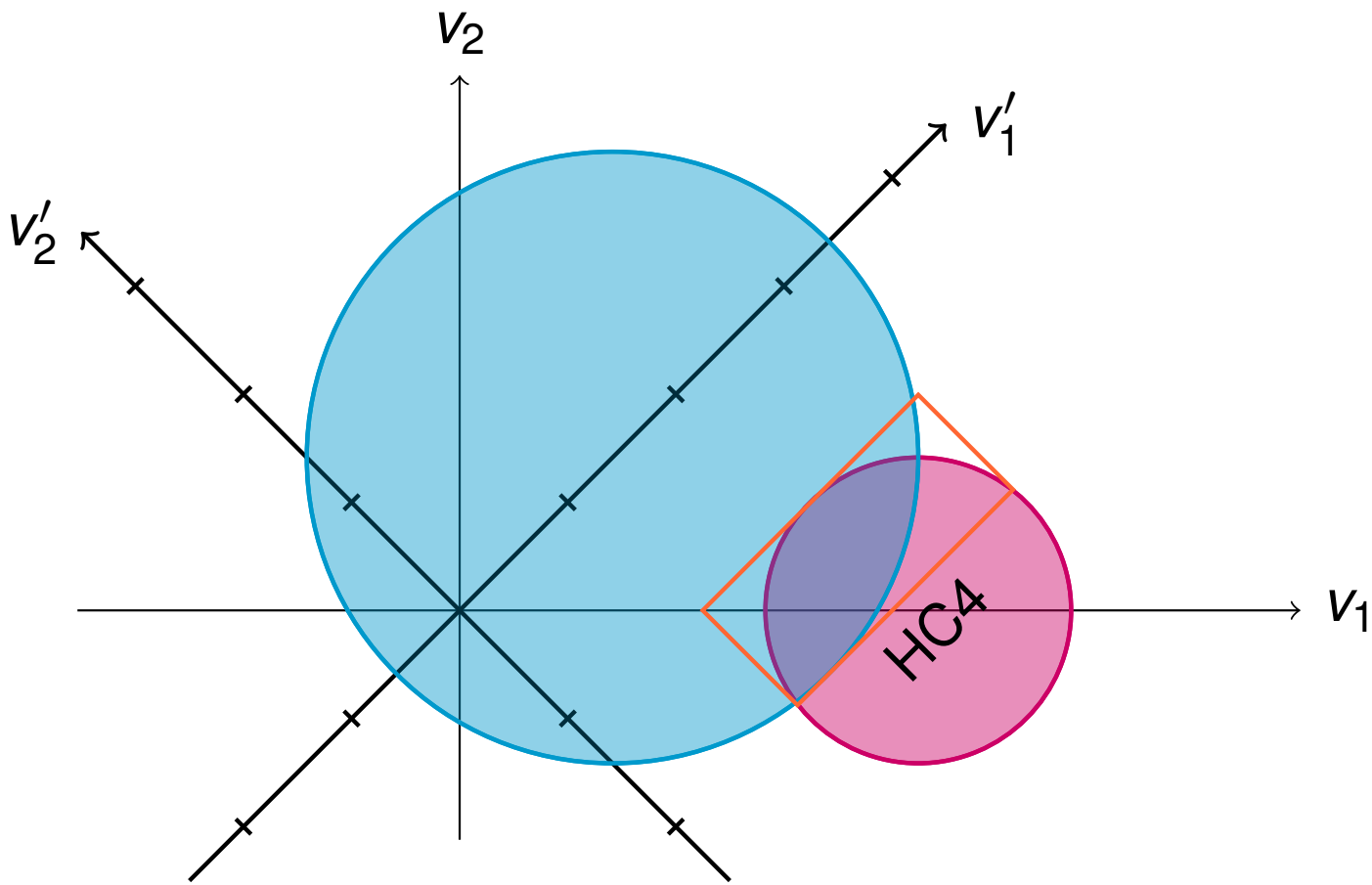
Octagonal Consistency



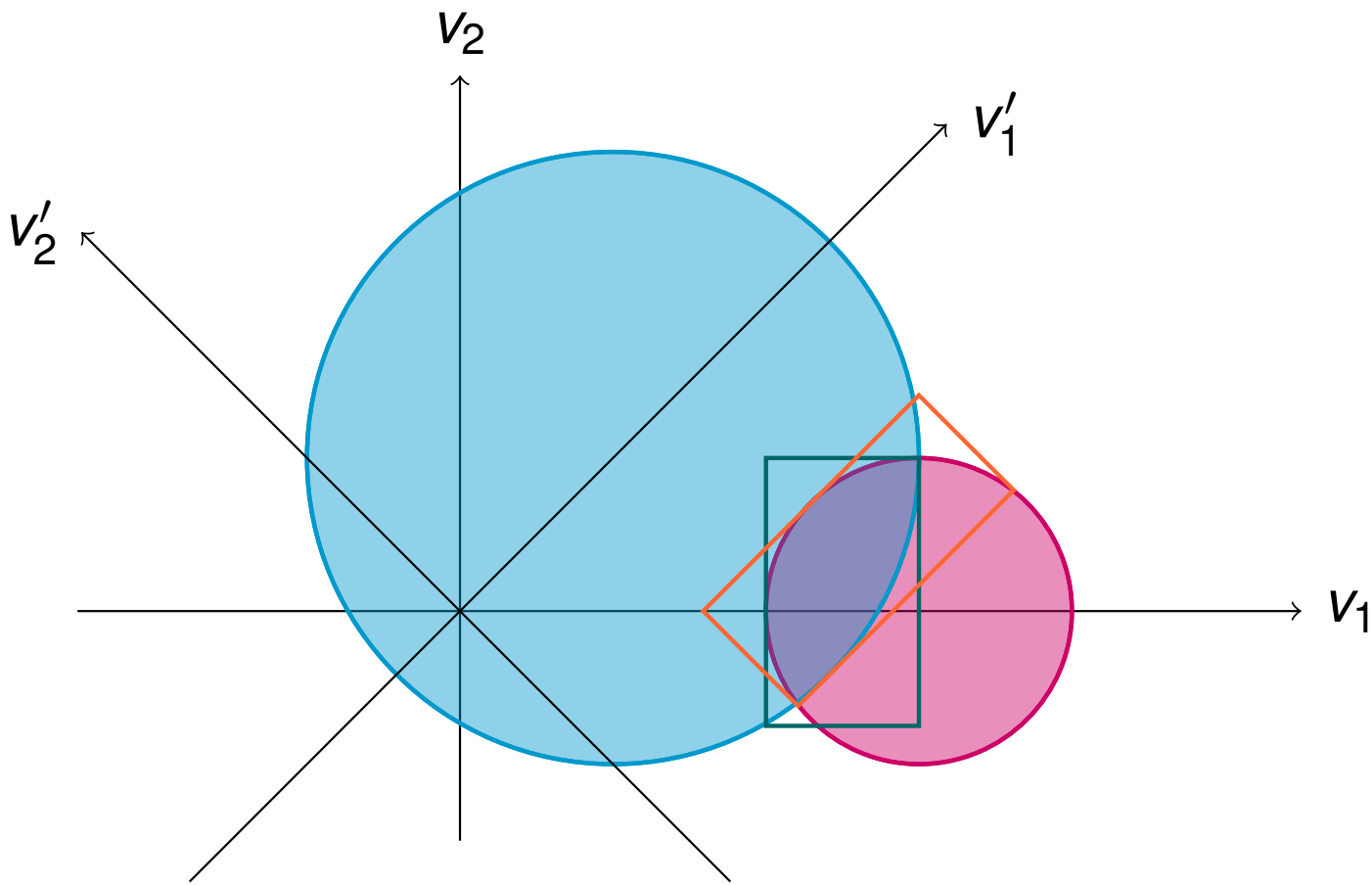
Octagonal Consistency



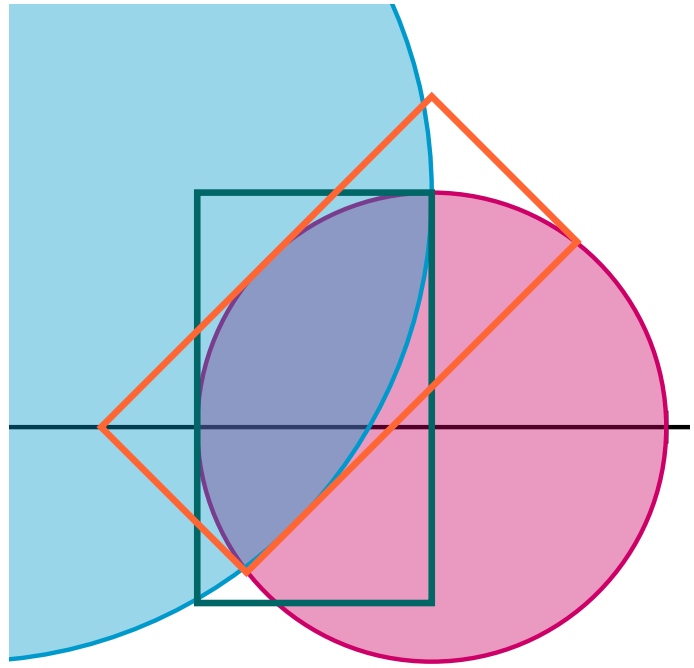
Octagonal Consistency



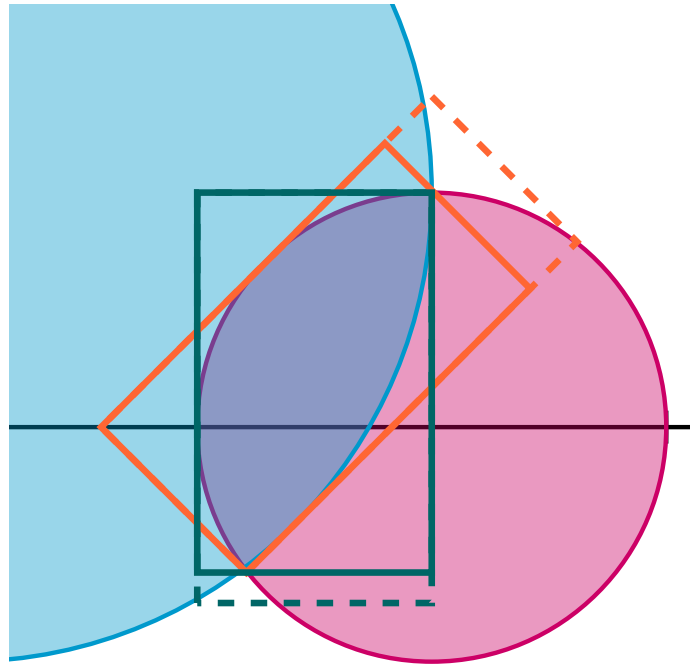
Octagonal Consistency



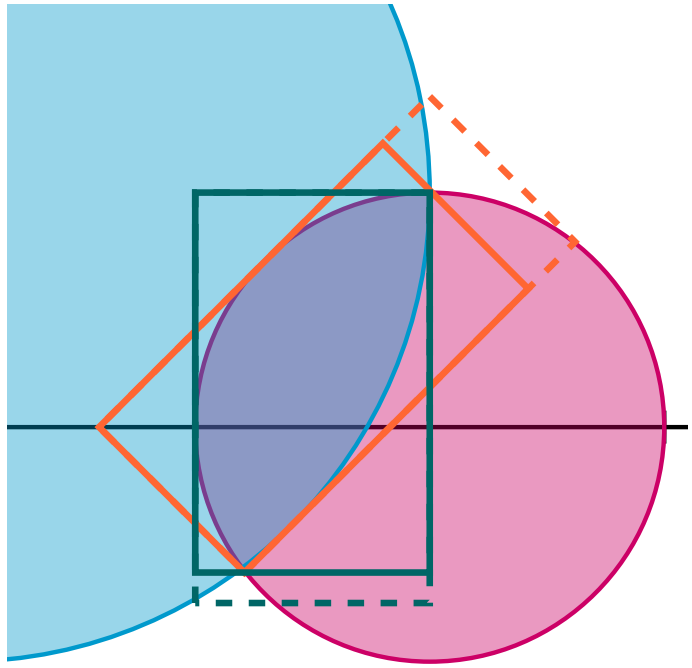
Octagonal Hull Consistency



Octagonal Hull Consistency

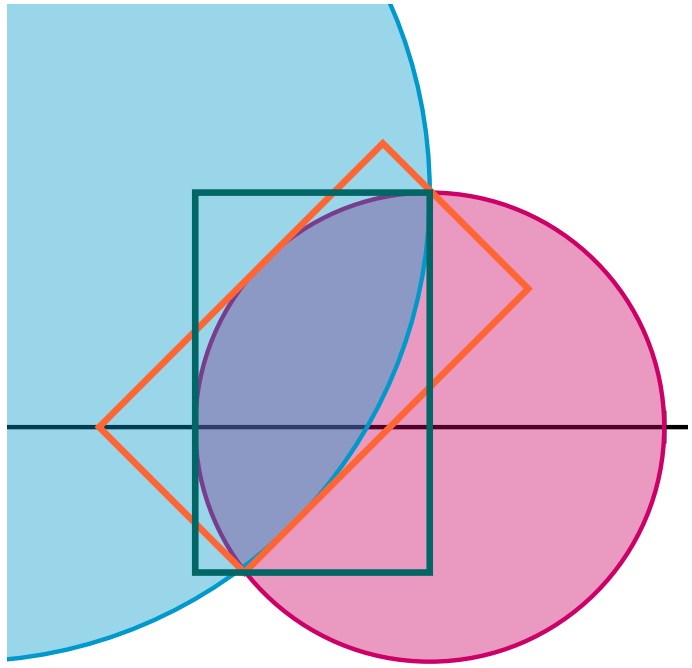


Octagonal Hull Consistency



Use the DBM and apply the consistency

Octagonal Hull Consistency

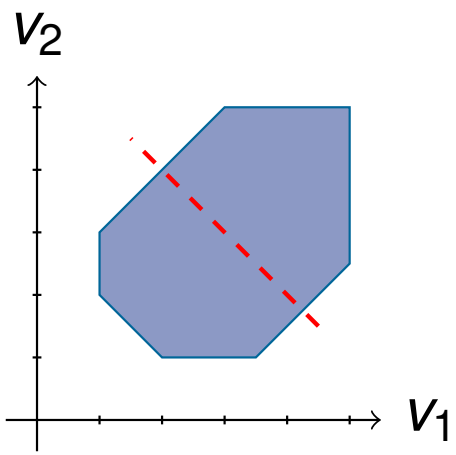


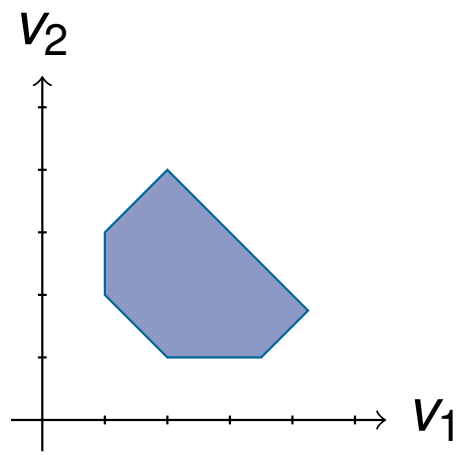
Use the DBM and apply the consistency

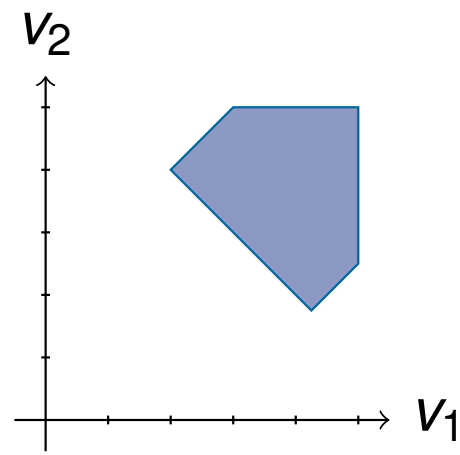
The canonical DBM corresponds to the smallest octagon

Octagonal Split

A splitting operator, splits a variable domain

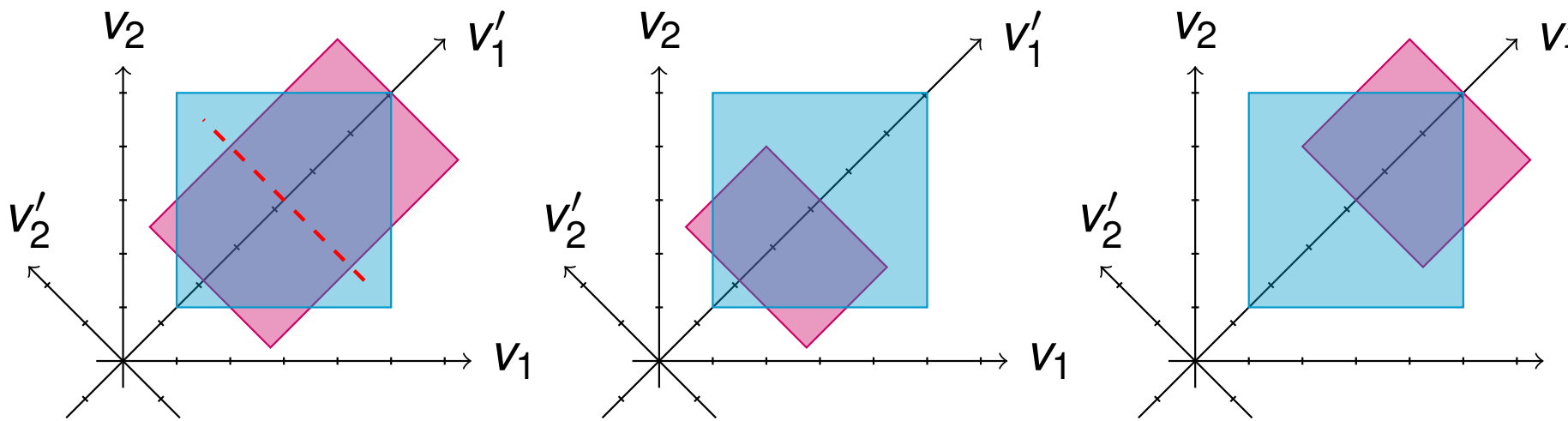




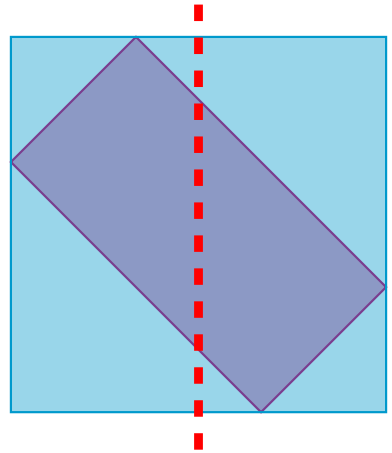


Octagonal Split

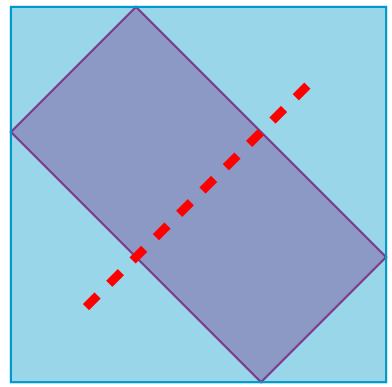
A splitting operator, splits a variable domain



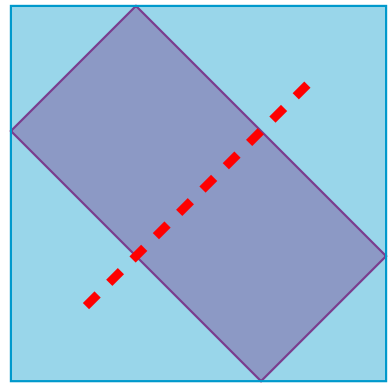
Octagonal Heuristic



Octagonal Heuristic



Octagonal Heuristic

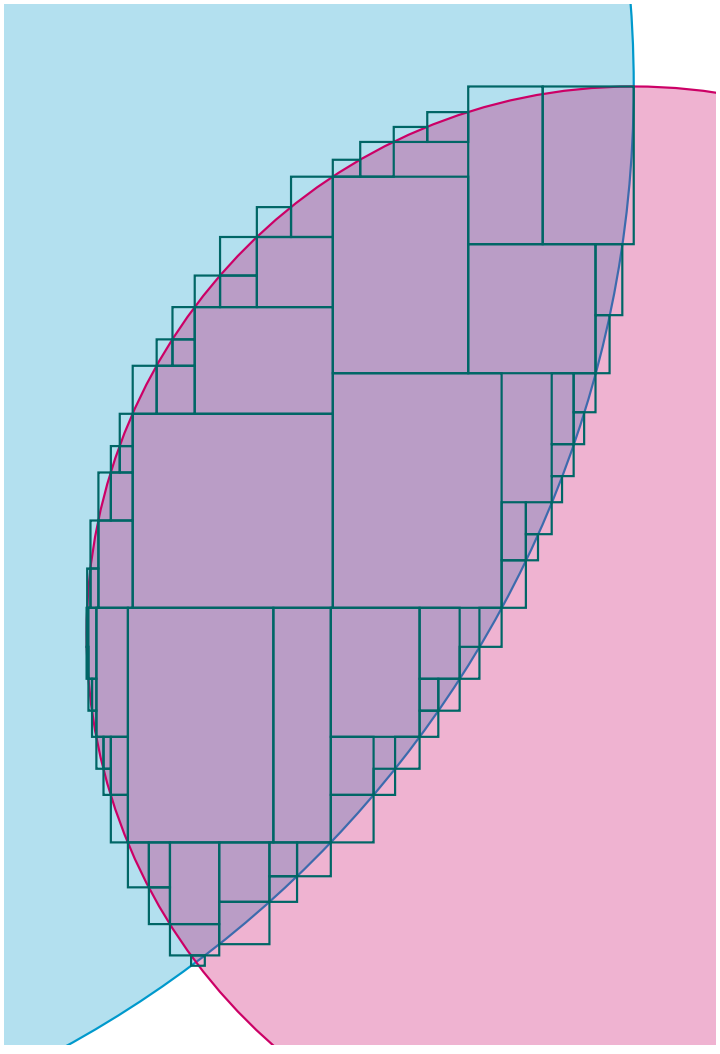


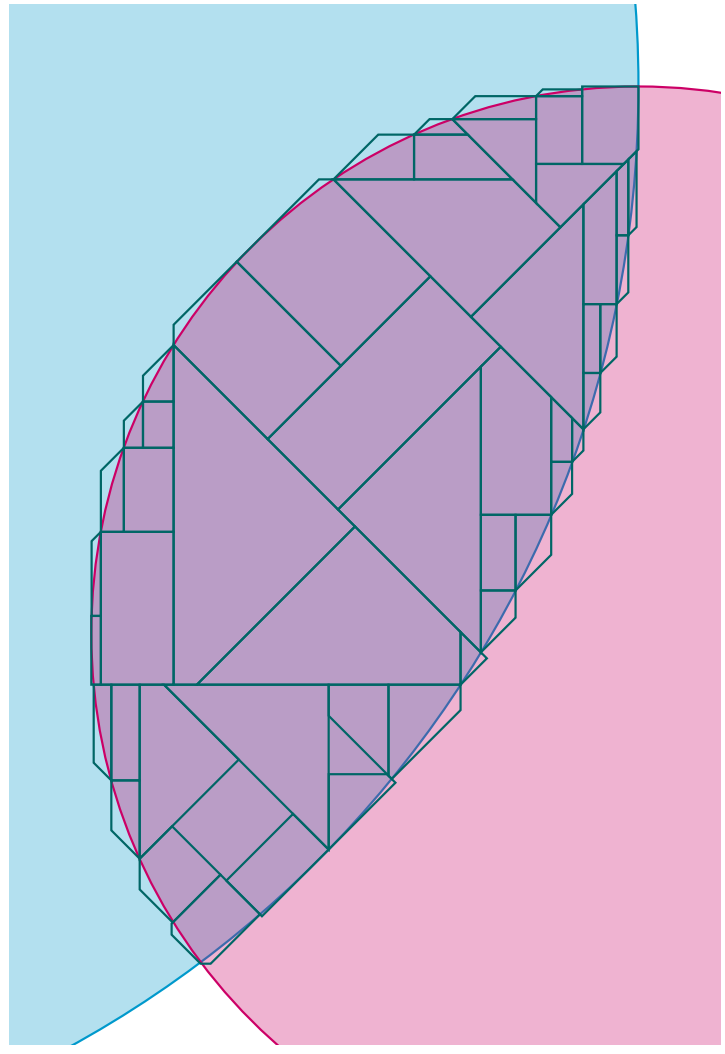
Take the "best" basis, *the box with the minimum of the maximum width*
Split the largest domain in this basis, *the domain with the maximum width*

Octagonal Solving

- We have:
  - an octagonal consistency
  - a splitting operator
  - a choice heuristic
  - a precision.
- We obtain an Octagonal Solver

Output





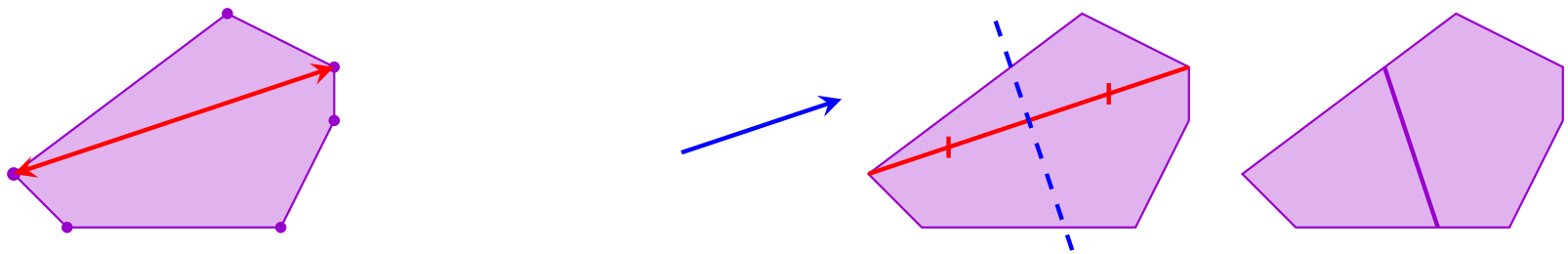
Same problem with the same time limit.

Polyhedra

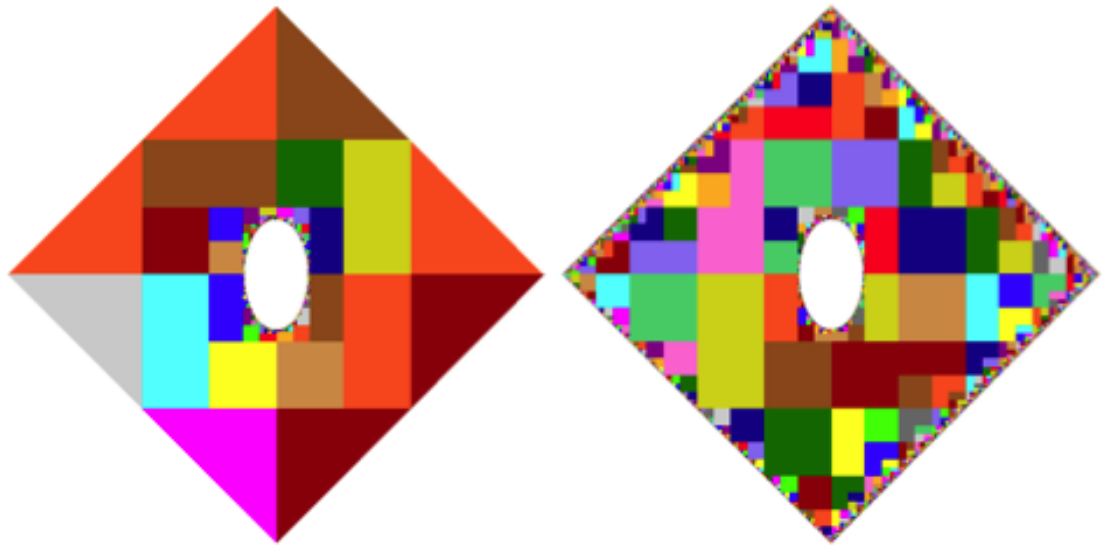
Polyhedra abstract domain $\mathcal{P}^\sharp$

$$\tau_p(X^\sharp) = \max_{g_i, g_j \in X^\sharp} ||g_i - g_j||$$

$$\oplus_p(X^\sharp) = \left\{ X^\sharp \cup \left\{ \sum_i \beta_i v_i \leq h \right\}, X^\sharp \cup \left\{ \sum_i \beta_i v_i \geq h \right\} \right\}$$



Polyhedra

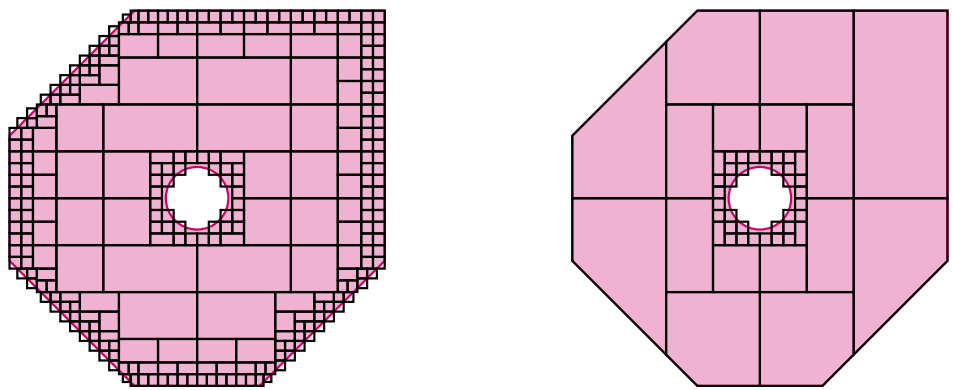


Les produits réduits

Utiliser uniquement les domaines les plus couteux pour les contraintes qu'ils peuvent encoder sans perte de précision

Le produit Intervalle-Polyèdre est intéressant

- Chaque contrainte est attribuée à un domaine
- Résolution séparée des sous-problèmes
- Un opérateur de réduction pour réduire les éléments obtenus



Conclusion

Links between CP and Verification are useful in both ways!

Outline

Conclusion

La CP offre un cadre générique pour utiliser de nombreux algorithmes d'optimisation combinatoire avec :

- des modèles faciles à modifier / améliorer incrémentalement,
- des outils efficaces en pratique sur beaucoup de problèmes,
- des liens très naturels avec l'analyse de programmes.

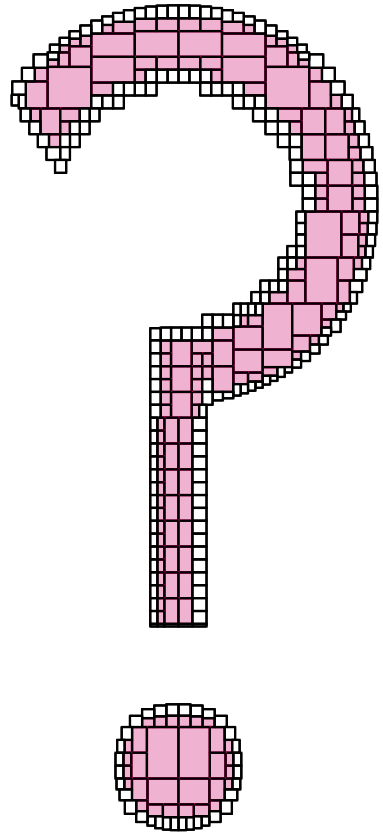
Trends in CP

- modeling, reformulation, constraint acquisition (ModRef workshop @CP, IJCAI modeling challenge...)
- links with other domains: CP & OR (CPAIOR conference), CP & Planning (first workshop @CP this year),
- CP with uncertainty,
- CP & verification (CP meets Verification workshop @CP).

Publicité

Offre de post-doc à Nantes !

- 18 mois
- début n'importe quand avant mars 2019
- projet ANR Coverif : CP & AI




Apt, K. R. (1999).
The essence of constraint propagation.
Theoretical Computer Science, 221.


Benhamou, F. (1996).
Heterogeneous constraint solvings.
In Proceedings of the 5th International Conference on Algebraic and Logic Programming, pages 62–76.


Cousot, P. and Cousot, R. (1976).
Static determination of dynamic properties of programs.
In Proceedings of the 2nd International Symposium on Programming, pages 106–130.


Cousot, P. and Cousot, R. (1977).
Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints.
In Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 238–252.


Cousot, P. and Cousot, R. (1992).
Abstract interpretation frameworks.
Journal of Logic and Computation, 2(4):511–547.


Granger, P. (1992).
Improving the results of static analyses of programs by local decreasing iterations.
In Proceedings of the 12th Conference on Foundations of Software Technology and Theoretical Computer Science.


Katriel, I. (2006).
Expected-case analysis for delayed filtering.
In CPAIOR, Lecture Notes in Computer Science, pages 119–125. Springer.


Mackworth, A. K. (1977).
Consistency in networks of relations.
Artificial Intelligence, 8(1):99–118.


Régin, J. (1994).
A filtering algorithm for constraints of difference in csps.

In Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1., pages 362–367.

Schulte, C. and Tack, G. (2001).
Implementing efficient propagation control.
In Proceedings of the 3rd workshop on Techniques for Implementing Constraint Programming Systems.

van Hoeve, W. J. (2001).
The alldifferent constraint: A survey.
CoRR, cs.PL/0105015.