Contraintes et programmation, une histoire triste, une histoire gaie

Séminaire Code Source 2025

Charlotte Truchet

STMS, UMR 9912, Sorbonne Université

Crédits

Certains slides (les jolis) sont empruntés à mes ancien.ne.s doctorant.e.s :

Marie Pelleau

Ghiles Ziat

Bruno Belin

Giovanni Lo Bianco

Mathieu Vavrille

Urban planning

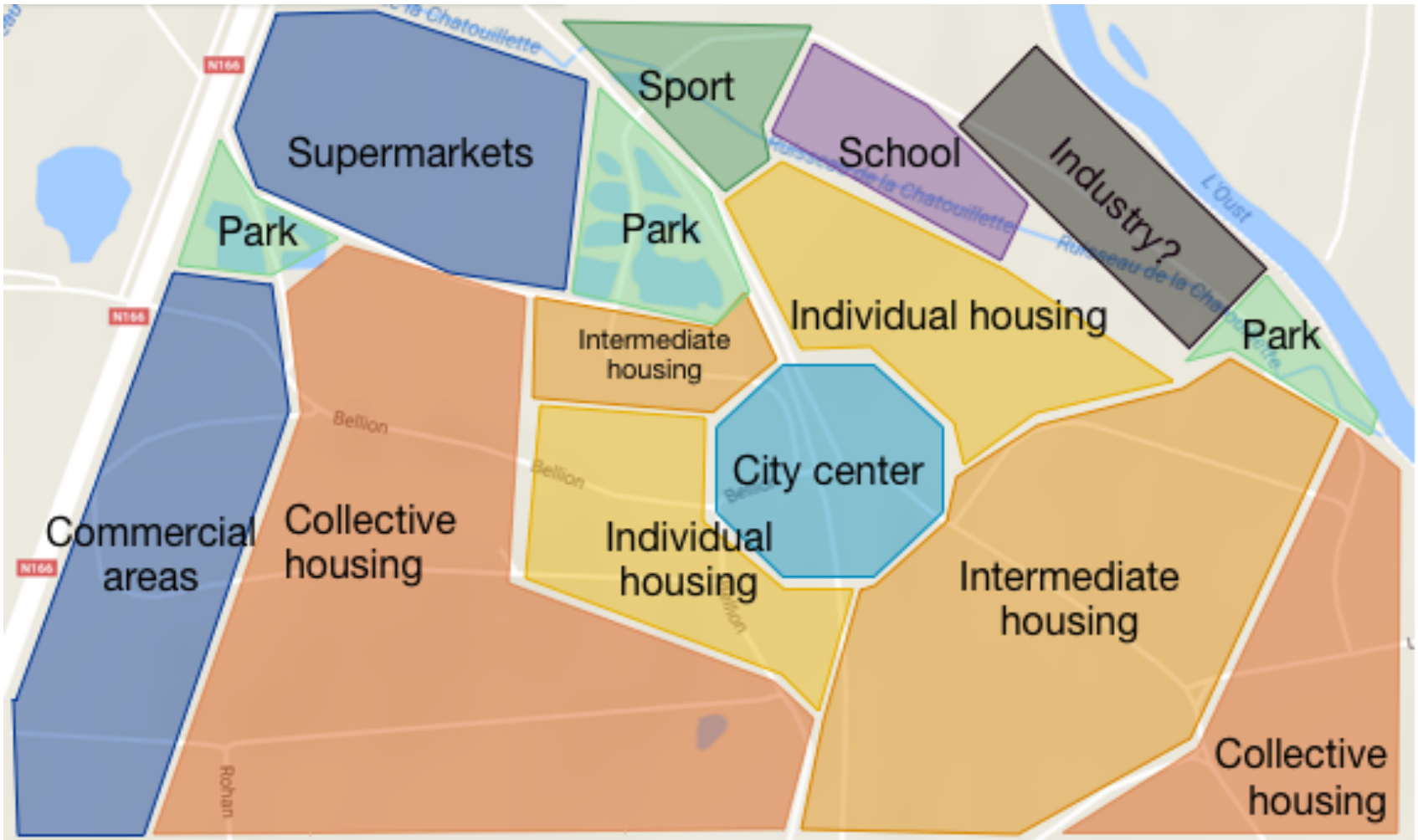
Urban planning

Urban planning

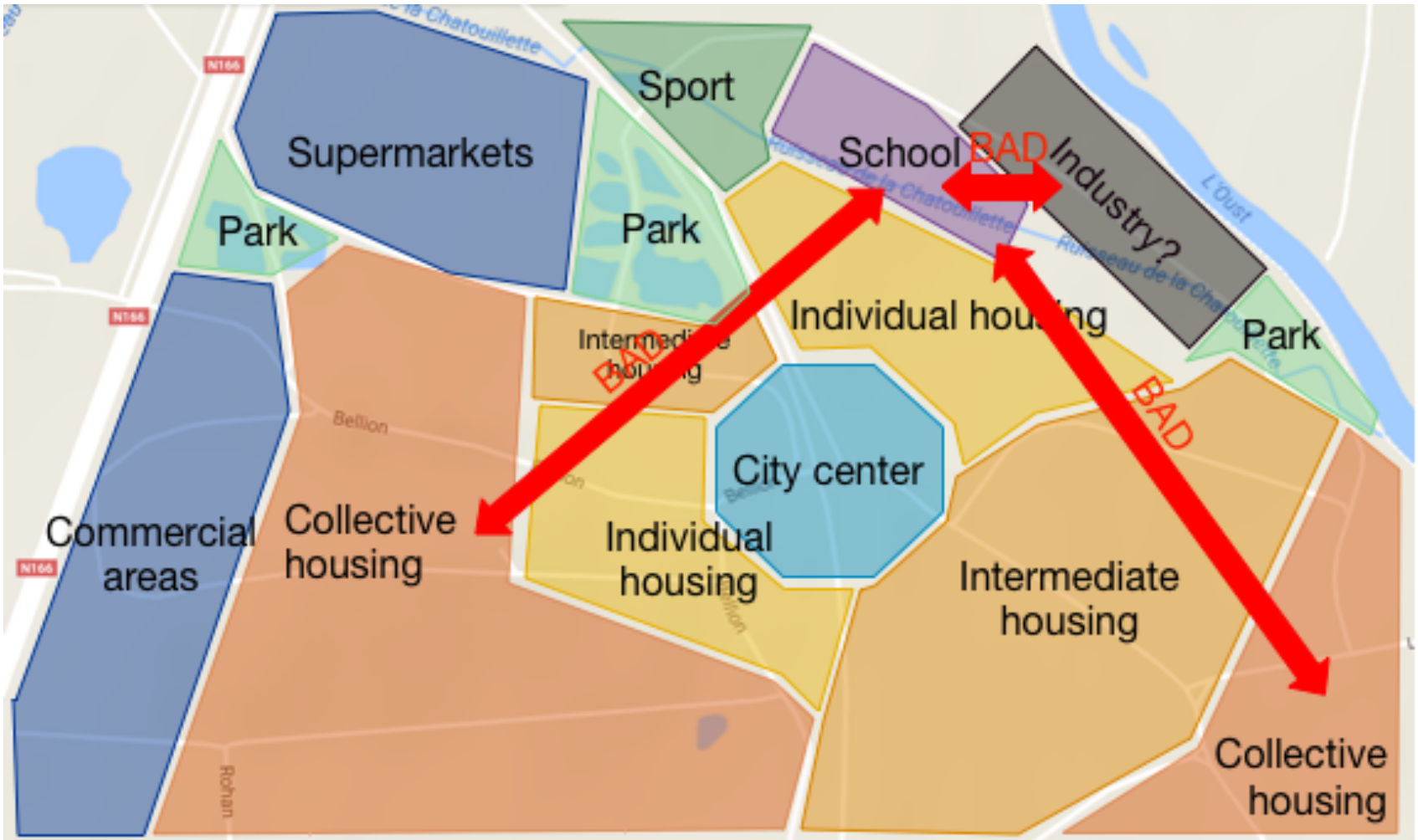
Urban planning

Urban planning

Urban planning



Urban planning



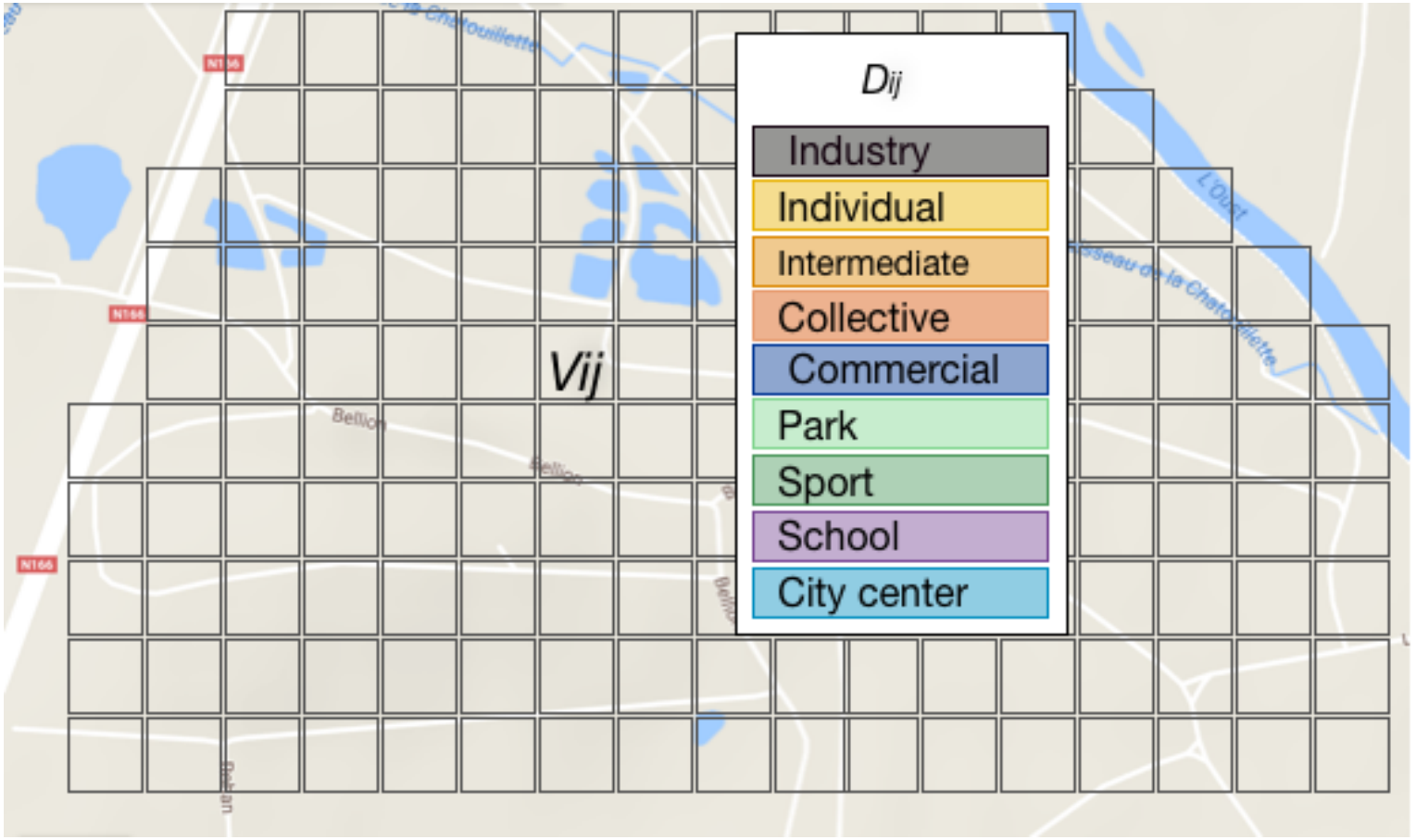
Constraint Programming

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,
  - building is difficult (takes a long time).

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

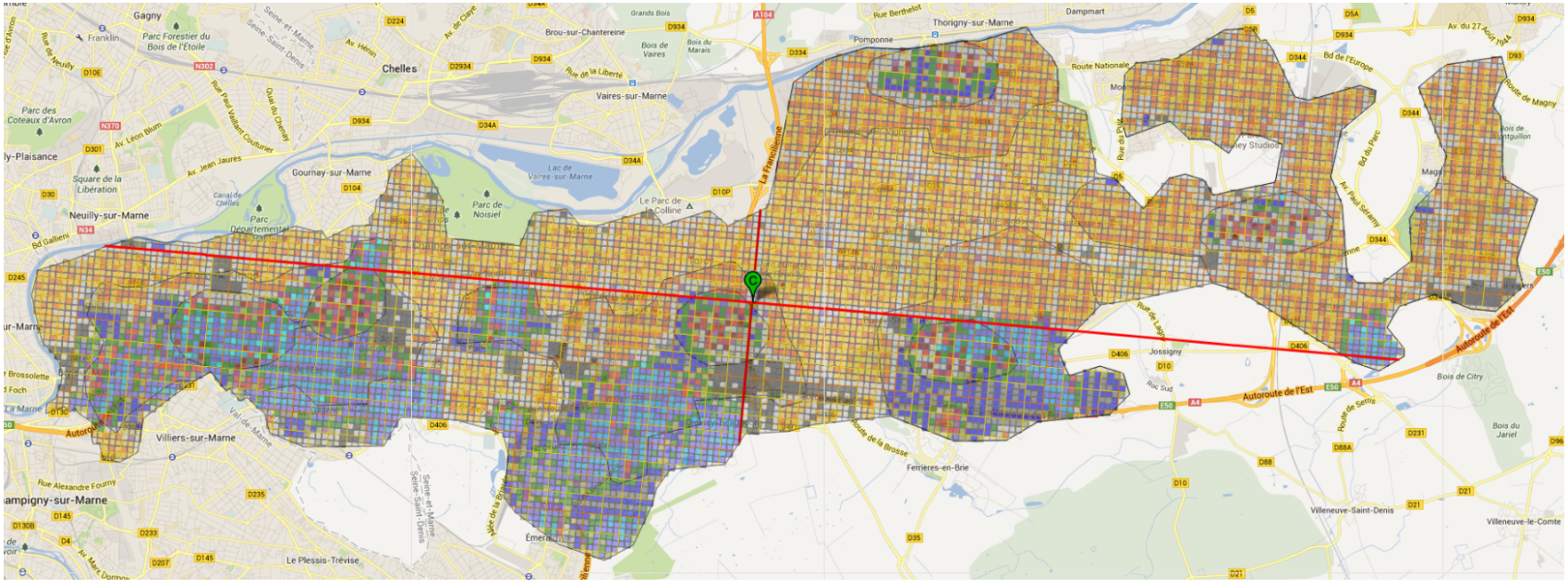
A variable is an unknown of the problem. It has a given domain, set of values the variable can take.

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



Simulation on Marne-la-Vallée, a French city of 8728 hectares,
with $\sim$ 230 000 inhabitants, $\sim$ 10 000 cells.
PhD of Bruno Belin, with Marc Christie, Frédéric Benhamou

CP on an example

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

Remarques :

- 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

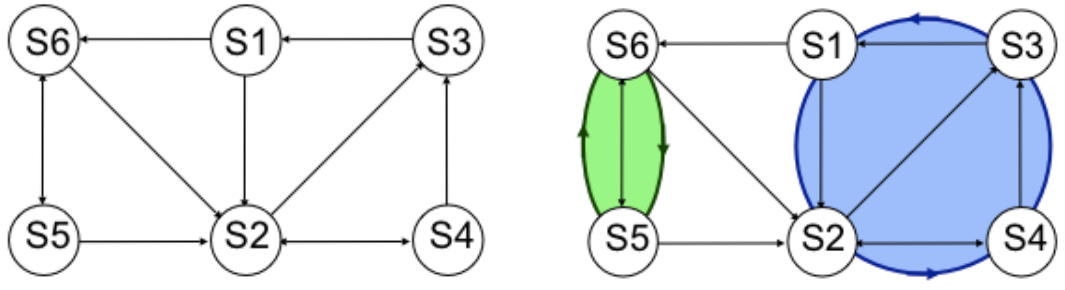
Les langages de contraintes incluent en général

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

Contraintes

Les langages de contraintes incluent en général

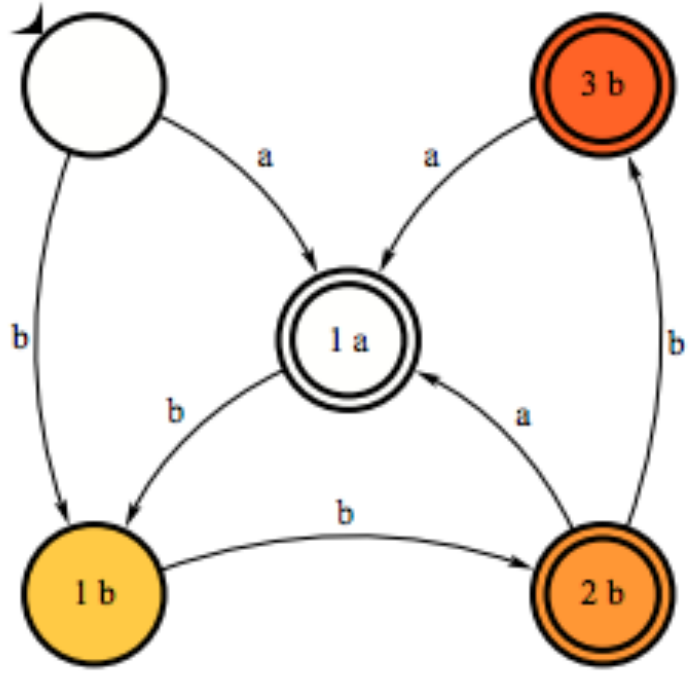
- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...



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

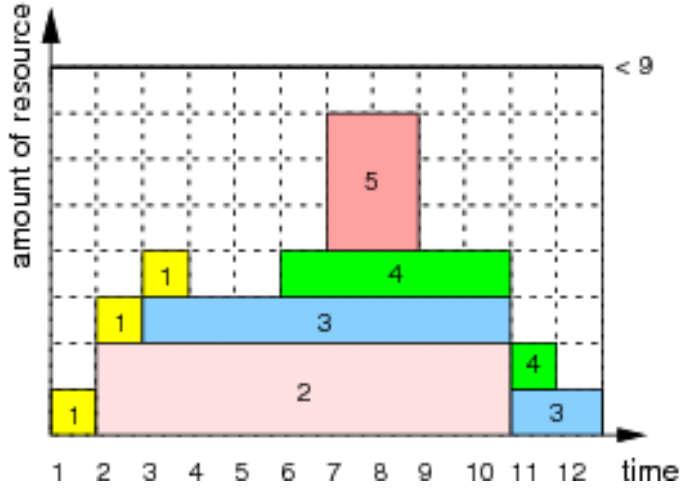
Les langages de contraintes incluent en général

- 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

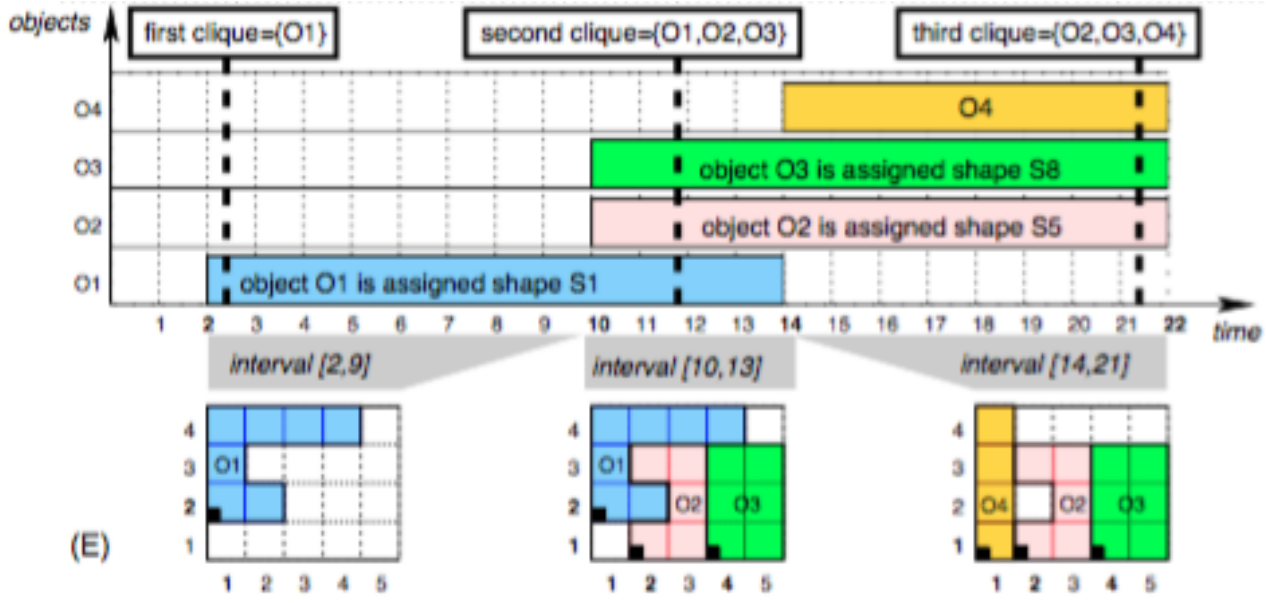
- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...
  - pratiques : `element`, `table`...
  - spécifiques : `cumulative`,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

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.

NB : si une série d'hypothèse aboutit à des domaines vides, c'est qu'il n'y a pas de solution dans cette branche, on abandonne l'hypothèse et on en fait une autre (backtrack).

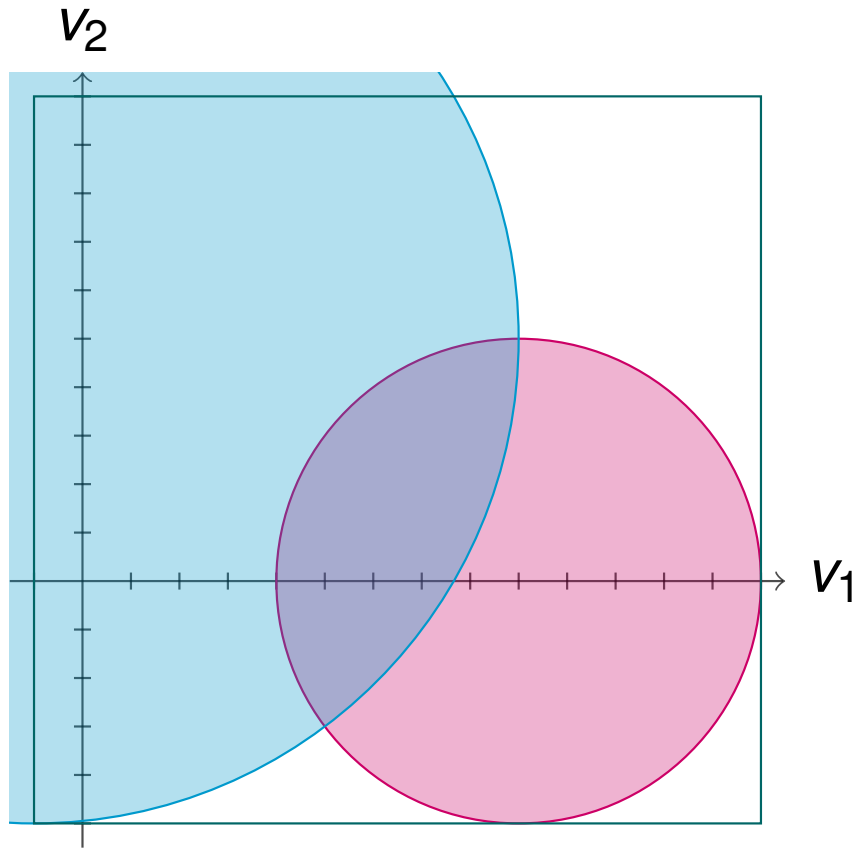
Continuous Solving Method

```
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
```



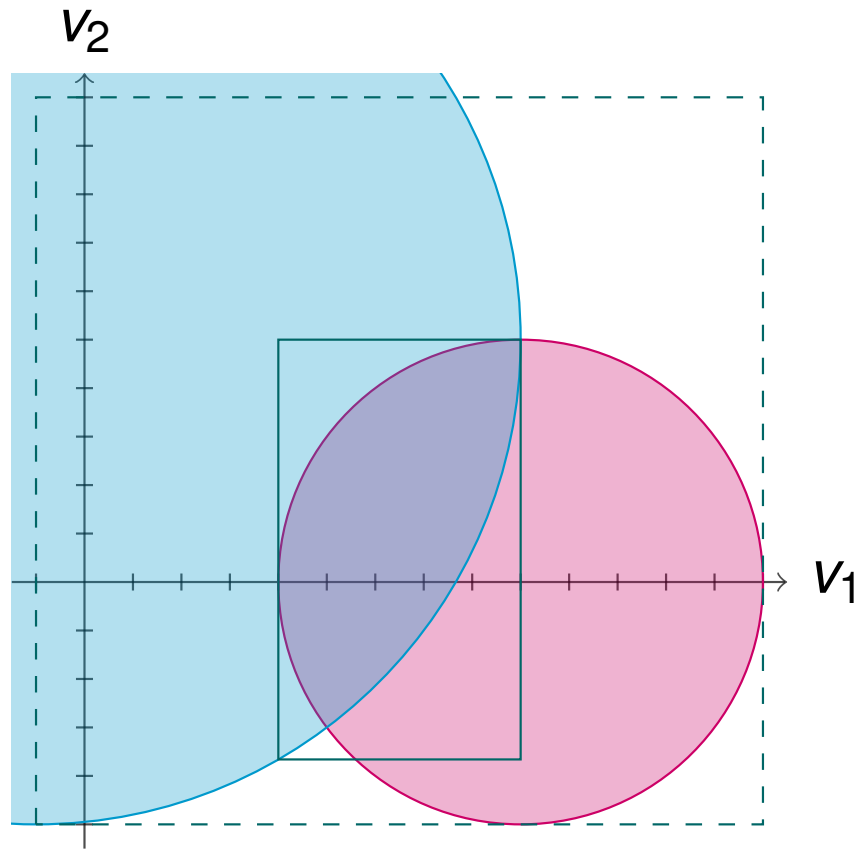
Continuous Solving Method

```
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
```



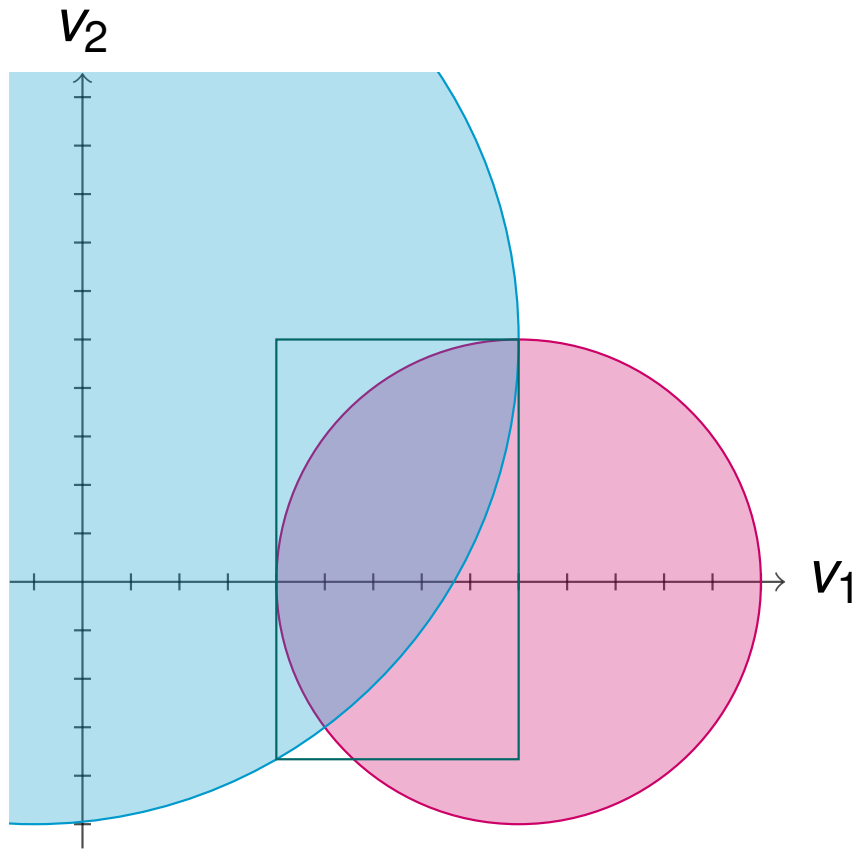
Continuous Solving Method

```
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
```



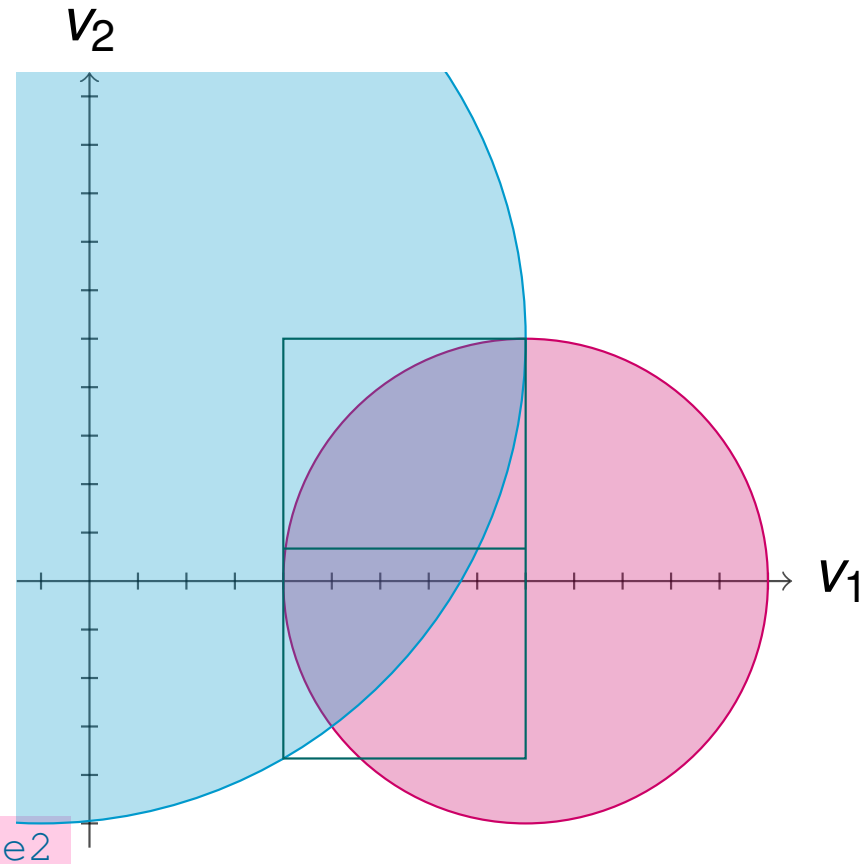
Continuous Solving Method

```
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
```



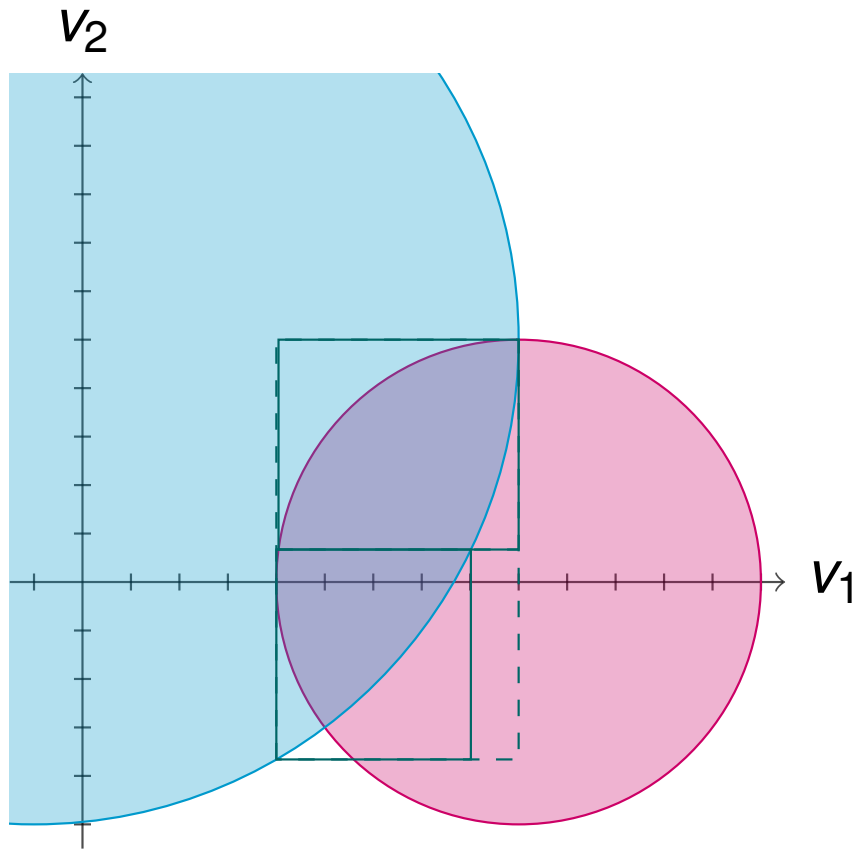
Continuous Solving Method

```
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
```



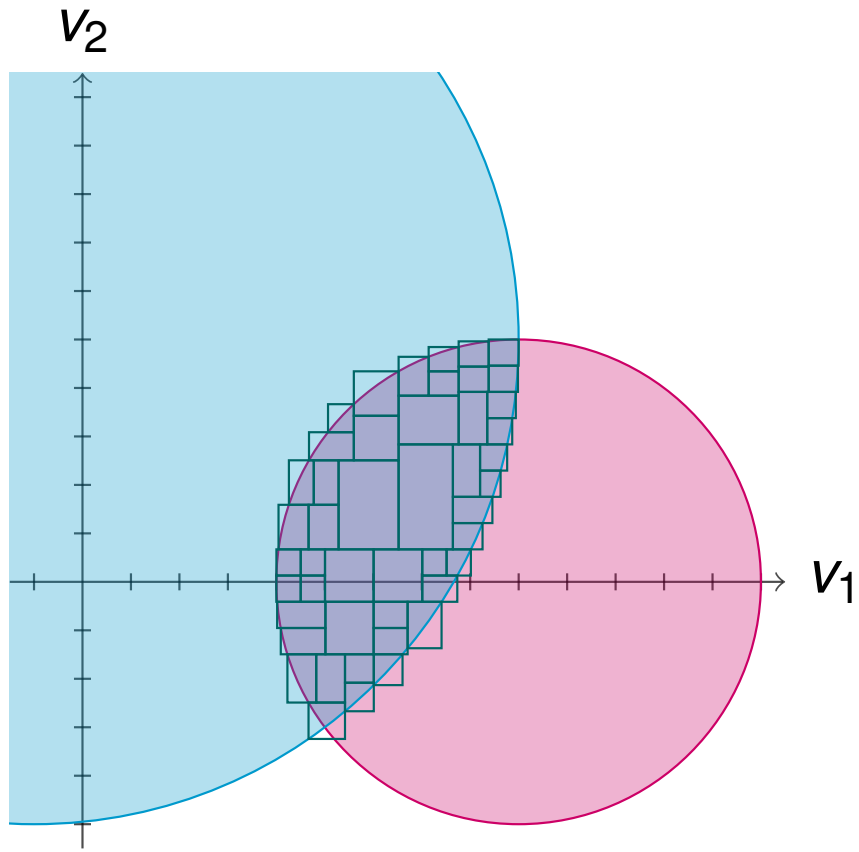
Continuous Solving Method

```
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
```



L'étape clef

La propagation des contraintes permet d'enlever au plus tôt des domaines les valeurs inutiles.

- les contraintes globales viennent avec des algorithmes de propagations dédiés,
- on peut avoir des niveaux plus ou moins forts/coûteux de propagation.

C'est la clef d'une résolution efficace.

Exemple : la contrainte `alldifferent`

La contrainte `alldifferent`$(X_1, ..., X_n)$ peut se réécrire en $\frac{n(n-1)}{2}$ contraintes élémentaires $X_i \neq X_j$.

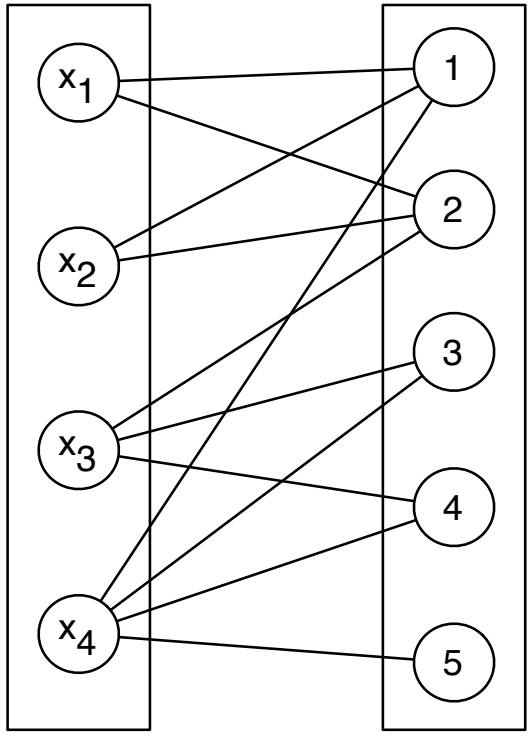
Mais si on voit l'ensemble, on voit plus de propagation.
Exemple pour trois variables X_1, X_2, X_3 :
$D_1 = \{2, 3\}$
$D_2 = \{2, 3\}$
$D_3 = \{2, 3\}$

Propagation de `alldifferent`

Soit une contrainte `alldifferent`$(X_1, ..., X_n)$ dont les variables X_i prennent leurs valeurs dans des domaines D_i.

Construction d'un graphe biparti

- On range d'un côté les variables, de l'autre toutes les valeurs disponibles pour toutes les variables, $D = \cup_i D_i$.
- On trace une arête entre X_i et v_j ssi $v_j \in D_i$.

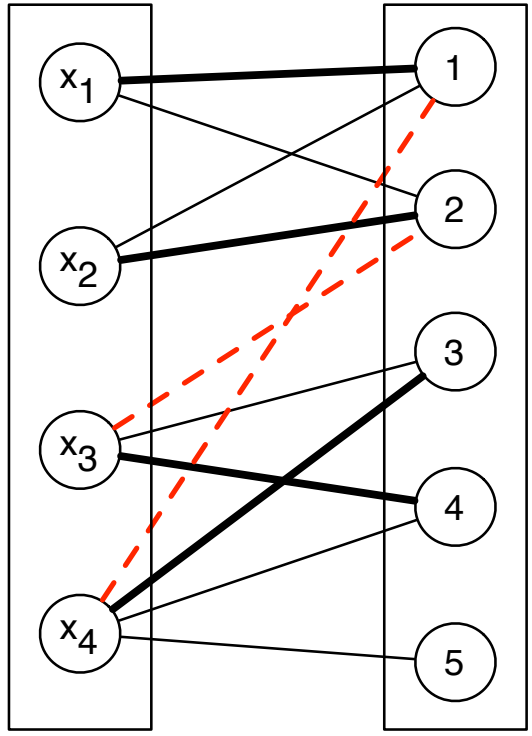


Une affectation des variables aux valeurs est un couplage maximal : toute variable a une valeur.

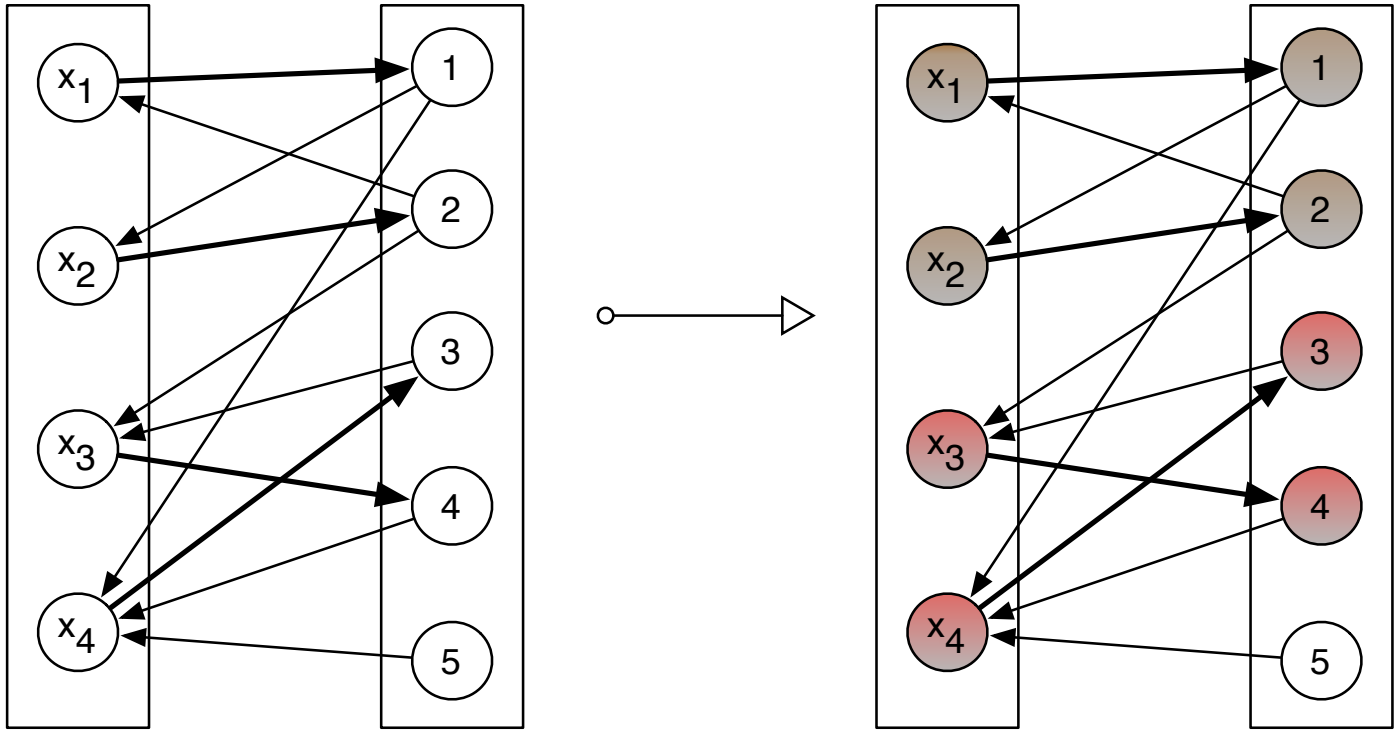
Propagation de `alldifferent`

Arêtes incompatibles

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



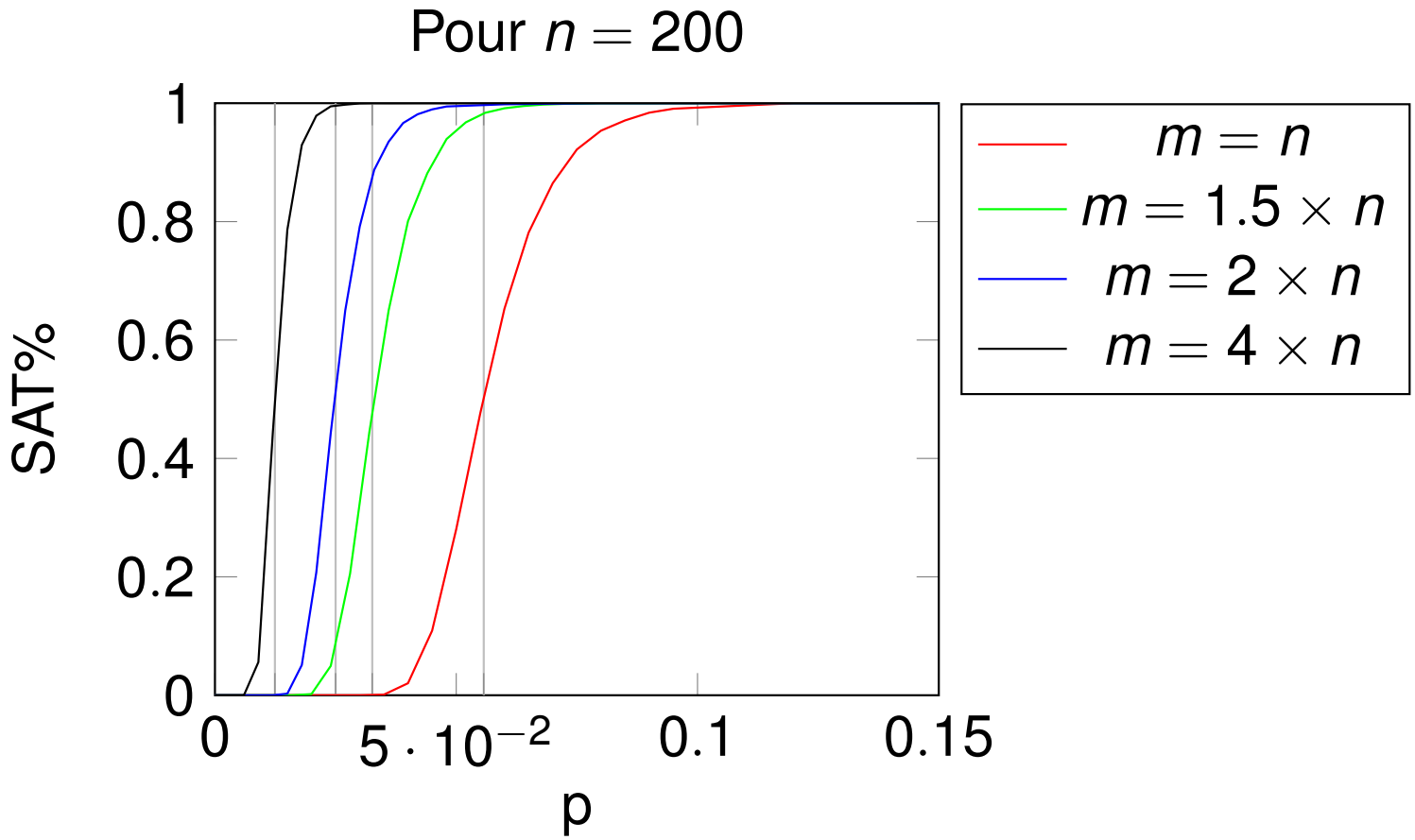
Propagation de `alldifferent`



Complexité:
$O(n * log(n))$ pour les bornes,
$O(m * \sqrt{n})$ pour toutes les valeurs,
où n est le nombre de variables et $m = |Dl$ le nombre de valeurs.

Quelques propriétés : nombre de solutions

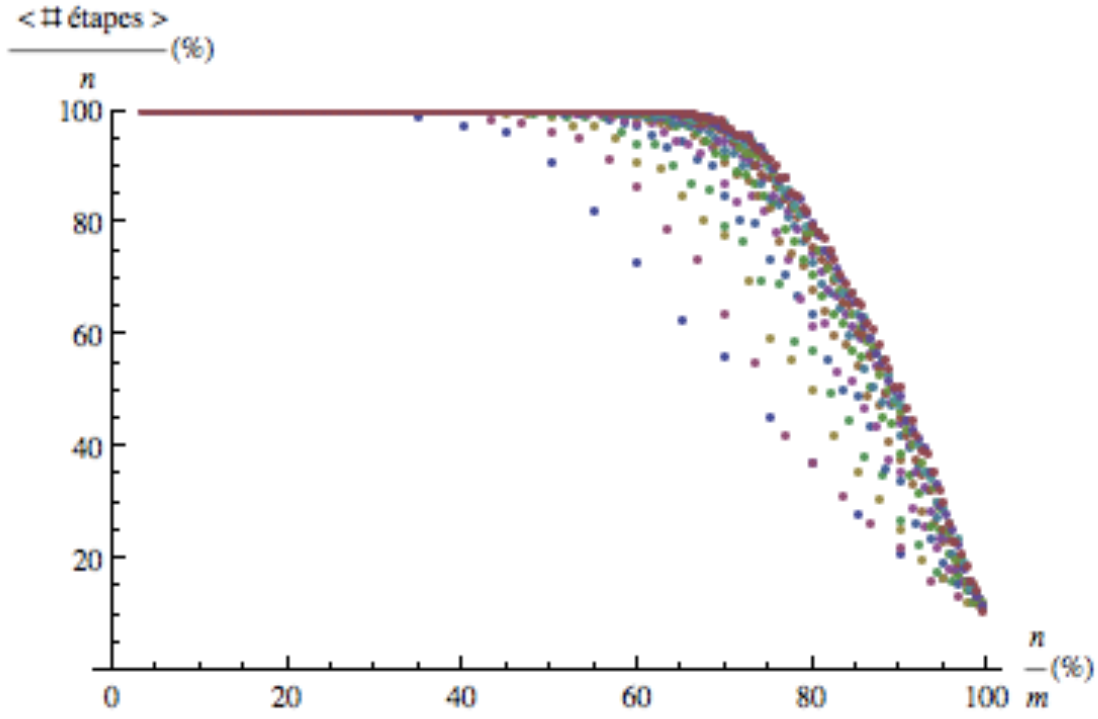
Etude de la satisfiabilité d'une l'instance `alldifferent` en fonction de p sa densité d'arêtes.



A quoi sert la propagation ?

En théorie : à tout.
En pratique : souvent à rien.

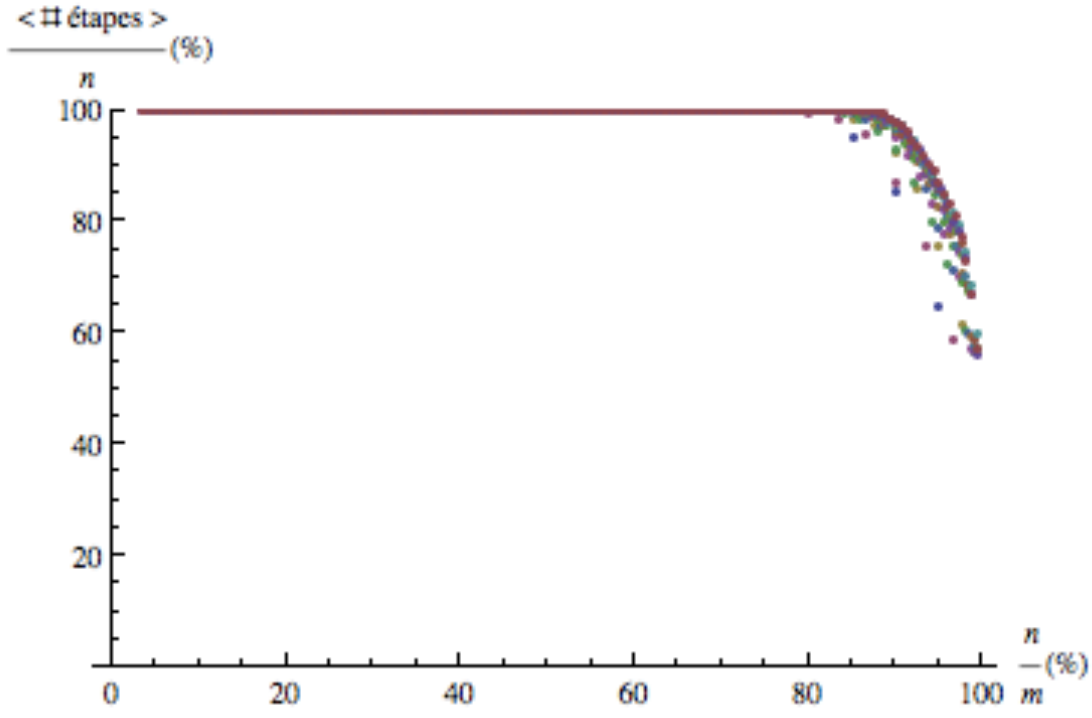
Exemple : Contraint alldifferent (seule) sur des domaines de taille $m/4$, où m est la taille de l'union des domaines.



A quoi sert la propagation ?

En théorie : à tout.
En pratique : souvent à rien.

Exemple : Contraint alldifferent (seule) sur des domaines de taille $3m/4$, où m est la taille de l'union des domaines.



A quoi sert la propagation ?

En théorie : à tout.
En pratique : souvent à rien.

Constat : on appelle à chaque noeud de l'arbre de recherche un algorithme qui, souvent, ne sert à rien, et qui manipule des structures de données lourdes.

NB : ce n'est pas trop grave, car l'algorithme qui propage la consistance est en $O(n * ln(n))$.

Probabilités ?

La résolution de CSP n'est *pas* un processus aléatoire. **Mais...**

- structures combinatoires sous-jacentes,
- notion intuitive de *chance* :

Probabilités ?

La résolution de CSP n'est *pas* un processus aléatoire. **Mais...**

- structures combinatoires sous-jacentes,
- notion intuitive de *chance* :
  - Soit un `alldifferent`sur des domaines de taille 2, 3 et 3 : aucune chance d'être insatisfiable !

Probabilités ?

La résolution de CSP n'est *pas* un processus aléatoire. **Mais...**

- structures combinatoires sous-jacentes,
- notion intuitive de *chance* :
  - Soit un `alldifferent`sur des domaines de taille 2, 3 et 3 : aucune chance d'être insatisfiable !
  - Soit un `alldifferent`sur des domaines de taille 2, 2 et 3 : aucune chance d'être insatisfiable !

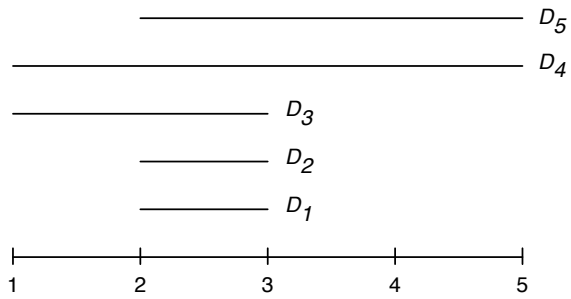
Probabilités ?

La résolution de CSP n'est *pas* un processus aléatoire. **Mais...**

- structures combinatoires sous-jacentes,
- notion intuitive de *chance* :
  - Soit un `alldifferent`sur des domaines de taille 2, 3 et 3 : aucune chance d'être insatisfiable !
  - Soit un `alldifferent`sur des domaines de taille 2, 2 et 3 : aucune chance d'être insatisfiable !
  - Soit un `alldifferent`sur des domaines de taille 2, 2 et 2 : il faut voir...

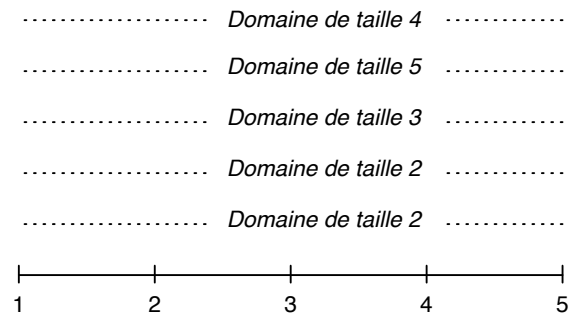
Idée

Microscopique
On connaît tout : les domaines, leur placement, etc... Tout est déterminé.



La contrainte est consistante, ou ne l'est pas.

Macroscopique
On ne connaît que certaines caractéristiques : tailles des domaines par exemple.
On probabilise le reste



Quelle est la probabilité pour que la contrainte soit consistante ?

Main result

Proposition

For a given interval $I \subset E$ and a domain $\mathcal{D}$ drawn with a uniform distribution on $\mathcal{I}(E)$, with $m = |E|$ and $l = |I|$, let p_l and q_l be the respective probabilities that $\mathcal{D} \subset I$, and that either $\mathcal{D} \cap I = \emptyset$, or $\underline{\mathcal{D}} < \underline{I} < \overline{I} < \overline{\mathcal{D}}$. Then

$$p_l = \frac{l(l-1)}{m(m-1)}; \qquad q_l = \frac{(m-l)(m-l-1)}{m(m-1)}.$$

Main result

Theorem

Consider an `alldifferent` *constraint on variables* $V_1, ..., V_n$, *initially BC. Let* E *and the domains* $\mathcal{D}_i$, $1 \leq i < n$, *satisfy the assumptions* **A1** *and* **A2**. *Furthermore, assume that we know the domain* $D_n = [a...b]$. *Then the probability* $P_{m,n,x,a,b}$ *that the constraint remains BC after the instanciation of the variable* V_n *to a value* x *is*

$$P_{m,n,x,a,b} = \prod_{l=1}^{n-2} (1 - P^{(1)}_{m,n,l} - P^{(2)}_{m,n,l})^{\Phi(m,l,x,a,b)}$$

where $\Phi(m, l, x, a, b)$ *is defined as* $\min(x, m-l) - \max(1, x-l) + 1$, *if* $l < b - a$ *and as* $\min(x, m-l) - \max(1, x-l) - \min(a, m-l) + \max(1, b-l)$, *otherwise, and where*

$$\begin{aligned} P^{(1)}_{m,n,l} &= \binom{n-1}{l+1} p_{l+1}^{l+1} (1 - p_{l+1})^{n-l-2}, \\ P^{(2)}_{m,n,l} &= \binom{n-1}{l} p_{l+1}^{l} \left((1 - p_{l+1})^{n-l-1} - q_{l+1}^{n-l-1} \right), \end{aligned}$$

with p_i *and* q_i *given by the previous proposition*

Main result

Theorem

Define a function $\Psi(m, x, a, b)$ *for* $1 \leq a \leq x \leq b \leq m$ *and* $a \neq b$ *by*

- $\Psi(m, x, a, a+1) = 1$, *except*
  $\Psi(m, 1, 1, 2) = \Psi(m, m, m-1, m) = 0$;
- *If* $b > a+1$ *then* $\Psi(m, x, a, b) = 2$, *except*
  $\Psi(m, 1, 1, b) = \Psi(m, m, a, m) = 1$.

Then the probability $P_{m,n,x,a,b}$ *that the constraint remains BC after the instanciation of* V_n *to the value* x *has asymptotic value*

- *if* $n = \rho m,\ \rho < 1$, $1 - \Psi(m, x, a, b)\frac{2\rho(1-e^{-4\rho})}{m} + O\left(\frac{1}{m^2}\right)$;
- *if* $n = m - i,\ i = o(m)$,
  $e^{C_i}\left(1 - \Psi(m, x, a, b)\frac{2(1-e^{-4})+D_i}{m} + O\left(\frac{1}{m^2}\right)\right)$.

[...]

Main result, relifted

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

More in [du Boisberranger, Gardy, Lorca, Truchet, Analco 2013].

Qu'en conclure ?

- On a un indicateur probabiliste pour la consistance d'un `alldifferent`.
- Plus intéressant, on identifie deux comportements pour la contraintes:
  - si n est constamment inférieur à m, la probabilité de rester consistante après instanciation tend vers 1,
  - si n est proche de m, elle tend vers une constante strictement plus petite que 1.

Implémentation

Facile !

D=propagate(D)

devient

if n/m > α **then**
    D=propagate(D)
else
    do nothing

Si éventuellement on se trompe (on ne propage pas alors qu'il aurait fallu), on récupère la propagation quand elle se déclence.

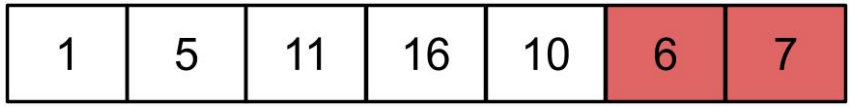
Problème...

Structures backtrackables : on se débrouille pour pouvoir récupérer facilement/rapidement des états antérieurs des domaines.

En Choco :

- un domaine est un tableau d'entiers,

- quand on supprime des valeurs, on ne modifie pas le tableau, on les met à la fin et on garde un pointeur sur la zone inutilisée.



Efficacité : on ne passe pas son temps à allouer et copier des tableaux. Il est facile de revenir à un état antérieur des domaines pendant les backtracks.

Histoire triste

- pour calculer l'oracle, il faut calculer $m = \cup_i D_i$,
- le calcul doit être backtrackable,
- il est refait à chaque noeud (très souvent).

En version backtrackable, le calcul de $m = \cup_i D_i$ est trop coûteux pour rentabiliser le gain.

Histoire triste

- pour calculer l'oracle, il faut calculer $m = \cup_i D_i$,
- le calcul doit être backtrackable,
- il est refait à chaque noeud (très souvent).

En version backtrackable, le calcul de $m = \cup_i D_i$ est trop coûteux pour rentabiliser le gain.

Histoire gaie

Mais si on sait compter, on sait rendre aléatoire.

- thèse de Giovanni Lo Bianco : les outils proposés se généralisent et permettent d'estimer le nombre de solutions de `alldifferent`, et même de toutes les autres contraintes de cardinalité
- thèse de Mathieu Vavrille : on rend le solveur aléatoire avec des bonnes propriétés, quasiment uniformément dans l'ensemble des solutions, pour un surcoût faible.

Idée

Algorithm

Algorithm to :

- sample solutions (inspired by Kuldeep S. Meel [Meel, 2018]),
- by adding table constraints,
- and using a CP solver as a black box.

Experimentally

- Implemented in Java[1], using the constraint solver Choco-solver,
- improvement of the randomness compared to RandomVarDom,
- running time in the same order of magnitude.

[1] https://github.com/MathieuVavrille/tableSampling

Solution Sampling

Return a solution **randomly** and **uniformly**.

Useful for:

- a user feedback
- equity
- coverage of the solutions

Property:

- non-deterministic solution
- no solving bias
- representative solutions

Sampling algorithm

We choose a pivot value κ.

FINDSOLUTIONS$(\mathcal{P}, \kappa)$

Enumerates at most κ solutions.

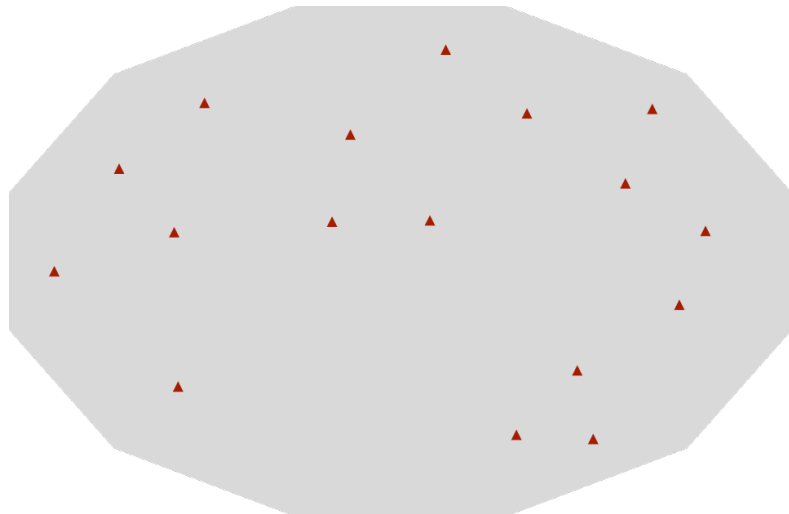
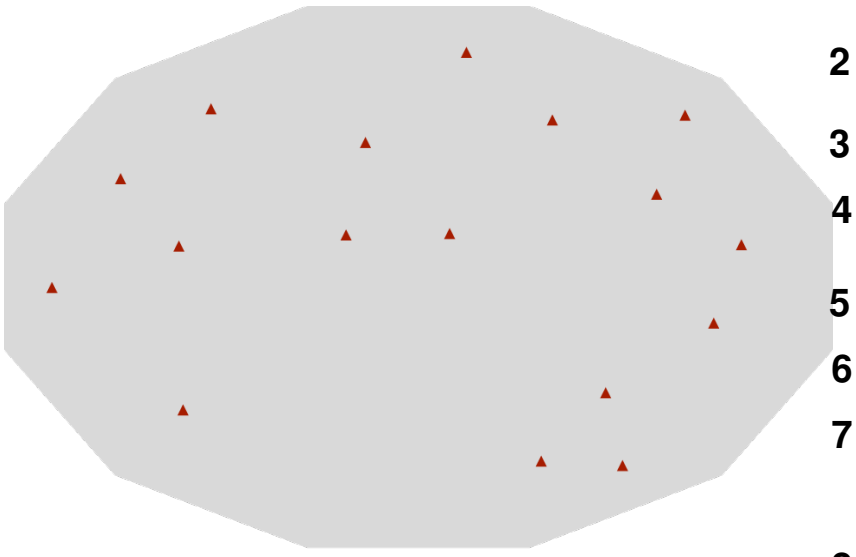


Figure: Search space. Triangles are solutions.

Sampling algorithm

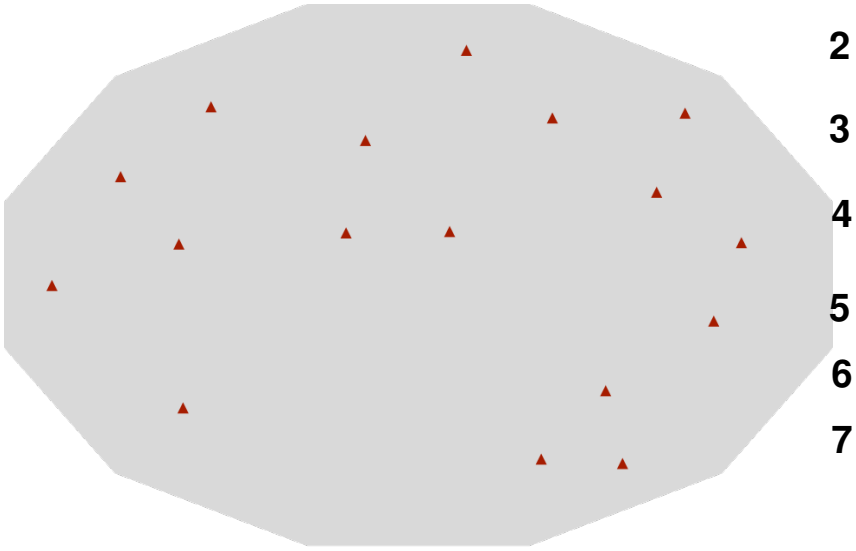
Example with $\kappa = 4$



1 **Function** TABLESAMPLING($\mathcal{P}, \kappa, v, p$)
  Data: A CSP $\mathcal{P}$, $\kappa \geq 2$, $v > 0$, $0 < p < 1$
2 $S \leftarrow$ FINDSOLUTIONS($\mathcal{P}, \kappa$) ;
3 **if** $|S| = 0$ **then**
4 **return** "No solution";
5 **while** $|S| = 0 \vee |S| = \kappa$ **do**
6 $T \leftarrow$ RANDOMTABLE($\mathcal{P}, v, p$);
7 $S \leftarrow$ FINDSOLUTIONS($\mathcal{P} \wedge T, \kappa$);
8 **if** $|S| \neq 0$ **then**
9 $\mathcal{P} \leftarrow \mathcal{P} \wedge T$;
10 **return** RANDOMELEMENT(S);

Sampling algorithm

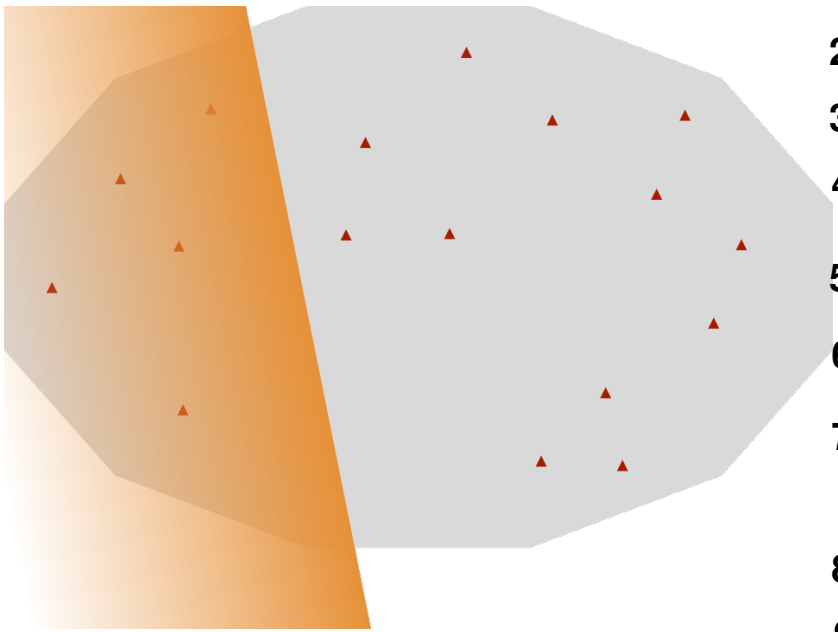
Example with $\kappa = 4$



```
1  Function TableSampling(𝒫, κ, v, p)
       Data: A CSP 𝒫, κ ≥ 2, v > 0,
             0 < p < 1
2      S ← FindSolutions(𝒫, κ);
3      if |S| = 0 then
4          return "No solution";
5      while |S| = 0 ∨ |S| = κ do
6          T ← RandomTable(𝒫, v, p);
7          S ←
             FindSolutions(𝒫 ∧ T, κ);
8          if |S| ≠ 0 then
9              𝒫 ← 𝒫 ∧ T;
10     return RandomElement(S);
```

Sampling algorithm

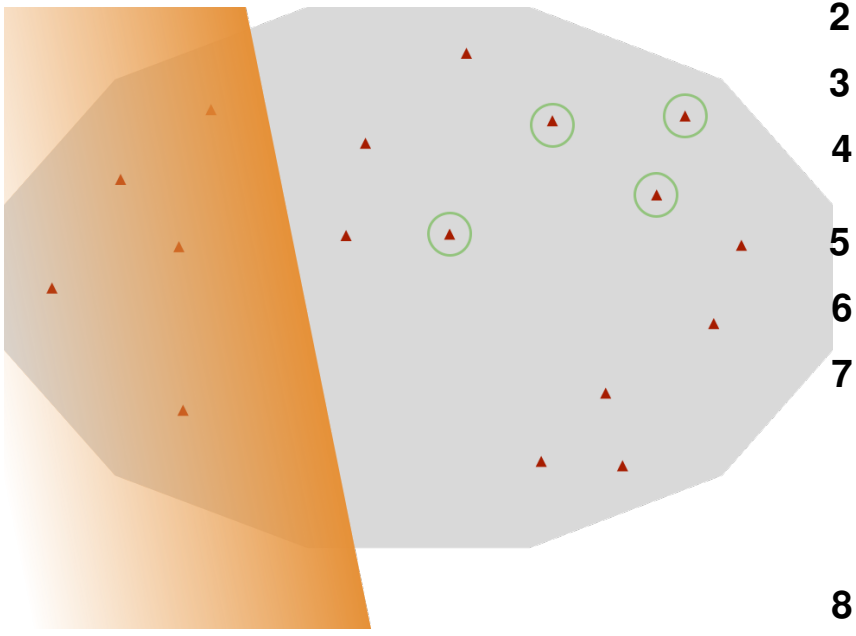
Example with $\kappa = 4$



1 **Function** TABLESAMPLING($\mathcal{P}, \kappa, v, p$)
  Data: A CSP $\mathcal{P}$, $\kappa \geq 2$, $v > 0$, $0 < p < 1$
2 $S \leftarrow$ FINDSOLUTIONS($\mathcal{P}, \kappa$);
3 **if** $|S| = 0$ **then**
4 **return** "No solution";
5 **while** $|S| = 0 \vee |S| = \kappa$ **do**
6 $T \leftarrow$ RANDOMTABLE($\mathcal{P}, v, p$);
7 $S \leftarrow$ FINDSOLUTIONS($\mathcal{P} \wedge T, \kappa$);
8 **if** $|S| \neq 0$ **then**
9 $\mathcal{P} \leftarrow \mathcal{P} \wedge T$;
10 **return** RANDOMELEMENT(S);

Sampling algorithm

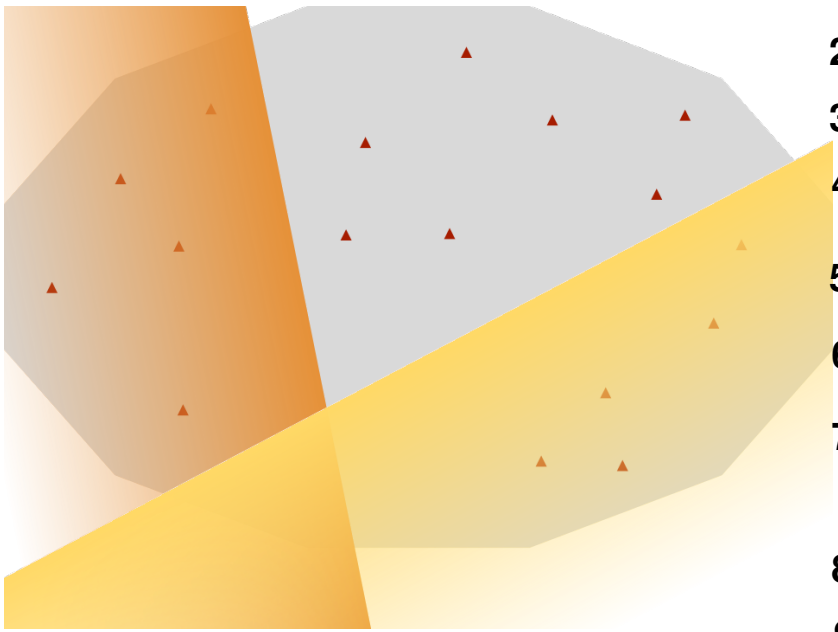
Example with $\kappa = 4$



1 **Function** TABLESAMPLING($\mathcal{P}, \kappa, v, p$)
  Data: A CSP $\mathcal{P}$, $\kappa \geq 2$, $v > 0$,
  $0 < p < 1$
2 $S \leftarrow$ FINDSOLUTIONS($\mathcal{P}, \kappa$);
3 **if** $|S| = 0$ **then**
4 **return** "No solution";
5 **while** $|S| = 0 \vee |S| = \kappa$ **do**
6 $T \leftarrow$ RANDOMTABLE($\mathcal{P}, v, p$);
7 $S \leftarrow$ FINDSOLUTIONS($\mathcal{P} \wedge T, \kappa$)
  ;
8 **if** $|S| \neq 0$ **then**
9 $\mathcal{P} \leftarrow \mathcal{P} \wedge T$;
10 **return** RANDOMELEMENT(S);

Sampling algorithm

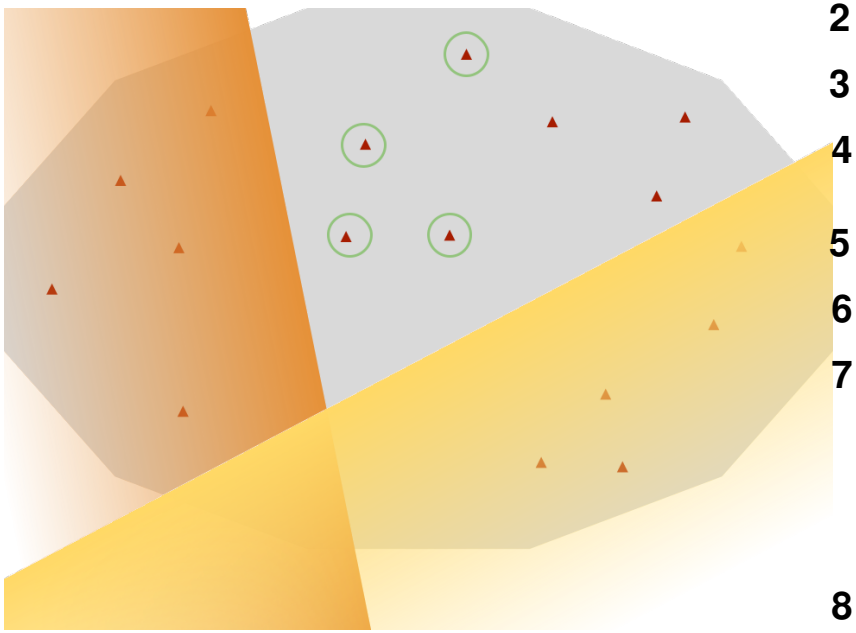
Example with $\kappa = 4$



```
1  Function TableSampling(𝒫, κ, v, p)
       Data: A CSP 𝒫, κ ≥ 2, v > 0,
             0 < p < 1
2      S ← FindSolutions(𝒫, κ);
3      if |S| = 0 then
4          return "No solution";
5      while |S| = 0 ∨ |S| = κ do
6          T ← RandomTable(𝒫, v, p);
7          S ←
             FindSolutions(𝒫 ∧ T, κ);
8          if |S| ≠ 0 then
9              𝒫 ← 𝒫 ∧ T;
10     return RandomElement(S);
```

Sampling algorithm

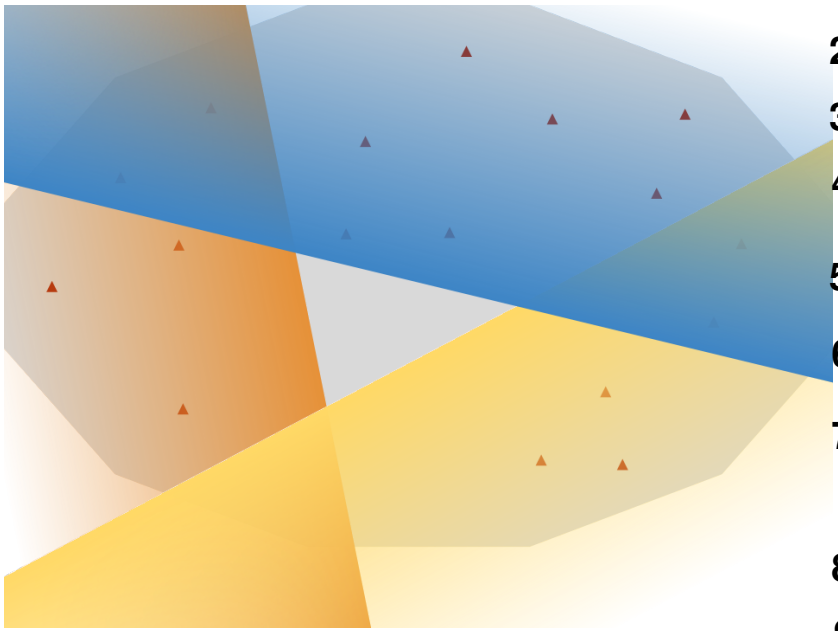
Example with $\kappa = 4$



1 **Function** TABLESAMPLING($\mathcal{P}, \kappa, v, p$)
  Data: A CSP $\mathcal{P}$, $\kappa \geq 2$, $v > 0$,
  $0 < p < 1$
2 $S \leftarrow$ FINDSOLUTIONS($\mathcal{P}, \kappa$);
3 **if** $|S| = 0$ **then**
4 **return** "No solution";
5 **while** $|S| = 0 \vee |S| = \kappa$ **do**
6 $T \leftarrow$ RANDOMTABLE($\mathcal{P}, v, p$);
7 $S \leftarrow$ FINDSOLUTIONS($\mathcal{P} \wedge T, \kappa$)
  ;
8 **if** $|S| \neq 0$ **then**
9 $\mathcal{P} \leftarrow \mathcal{P} \wedge T$;
10 **return** RANDOMELEMENT(S);

Sampling algorithm

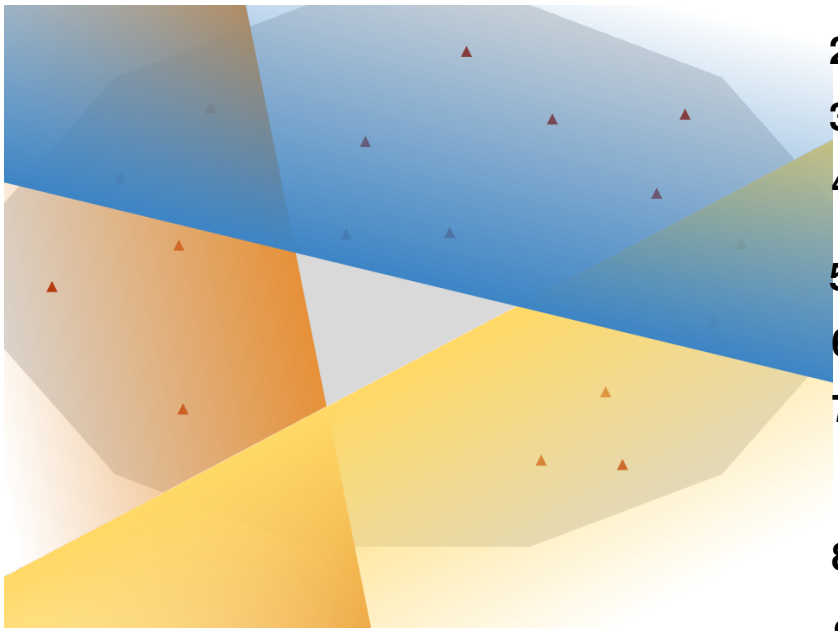
Example with $\kappa = 4$



```
1  Function TableSampling(𝒫, κ, v, p)
       Data: A CSP 𝒫, κ ≥ 2, v > 0,
             0 < p < 1
2      S ← FindSolutions(𝒫, κ);
3      if |S| = 0 then
4          return "No solution";
5      while |S| = 0 ∨ |S| = κ do
6          T ← RandomTable(𝒫, v, p);
7          S ←
             FindSolutions(𝒫 ∧ T, κ);
8          if |S| ≠ 0 then
9              𝒫 ← 𝒫 ∧ T;
10     return RandomElement(S);
```

Sampling algorithm

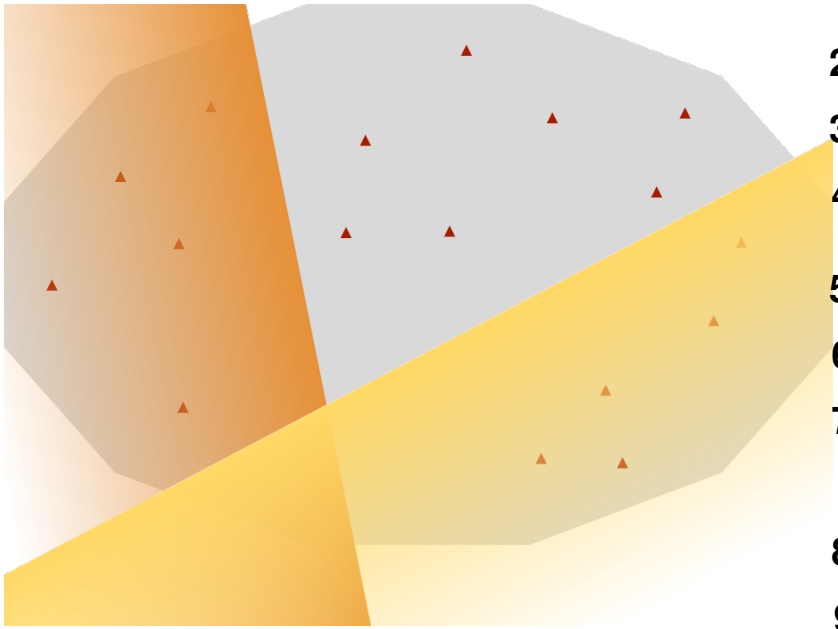
Example with $\kappa = 4$



```
1  Function TableSampling(𝒫, κ, v, p)
       Data: A CSP 𝒫, κ ≥ 2, v > 0,
             0 < p < 1
2      S ← FindSolutions(𝒫, κ);
3      if |S| = 0 then
4          return "No solution";
5      while |S| = 0 ∨ |S| = κ do
6          T ← RandomTable(𝒫, v, p);
7          S ←
             FindSolutions(𝒫 ∧ T, κ);
8          if |S| ≠ 0 then
9              𝒫 ← 𝒫 ∧ T;
10     return RandomElement(S);
```

Sampling algorithm

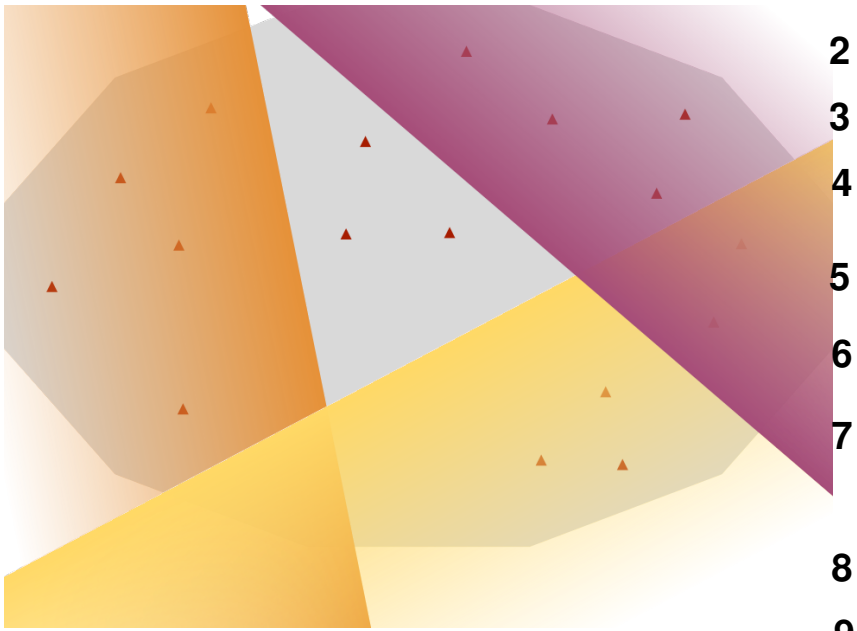
Example with $\kappa = 4$



1 **Function** TABLESAMPLING($\mathcal{P}, \kappa, v, p$)
  Data: A CSP $\mathcal{P}$, $\kappa \geq 2$, $v > 0$, $0 < p < 1$
2 $S \leftarrow$ FINDSOLUTIONS($\mathcal{P}, \kappa$);
3 **if** $|S| = 0$ **then**
4 **return** "No solution";
5 **while** $|S| = 0 \vee |S| = \kappa$ **do**
6 $T \leftarrow$ RANDOMTABLE($\mathcal{P}, v, p$);
7 $S \leftarrow$ FINDSOLUTIONS($\mathcal{P} \wedge T, \kappa$);
8 **if** $|S| \neq 0$ **then**
9 $\mathcal{P} \leftarrow \mathcal{P} \wedge T$;
10 **return** RANDOMELEMENT(S);

Sampling algorithm

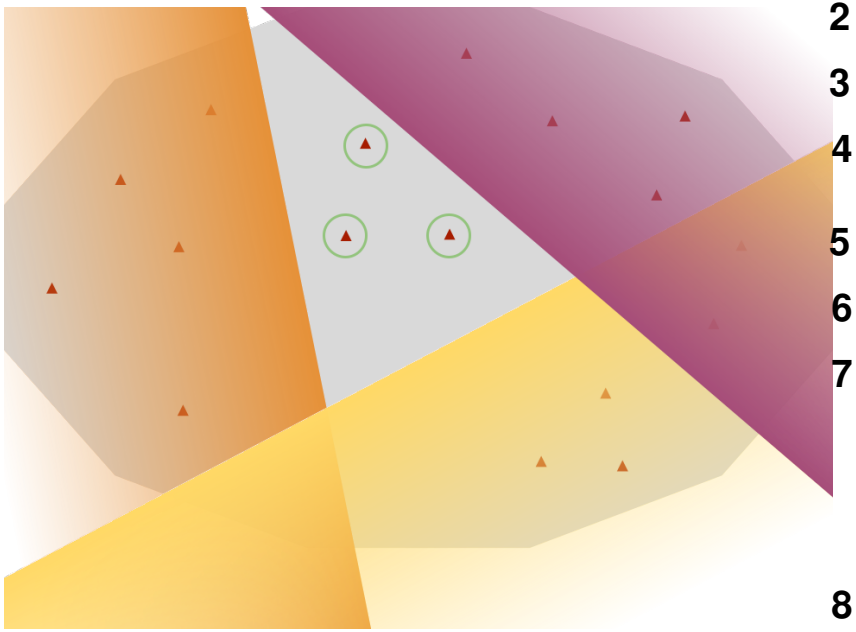
Example with $\kappa = 4$



```
1  Function TableSampling(𝒫, κ, v, p)
       Data: A CSP 𝒫, κ ≥ 2, v > 0,
             0 < p < 1
2      S ← FindSolutions(𝒫, κ);
3      if |S| = 0 then
4          return "No solution";
5      while |S| = 0 ∨ |S| = κ do
6          T ← RandomTable(𝒫, v, p);
7          S ←
             FindSolutions(𝒫 ∧ T, κ);
8          if |S| ≠ 0 then
9              𝒫 ← 𝒫 ∧ T;
10     return RandomElement(S);
```

Sampling algorithm

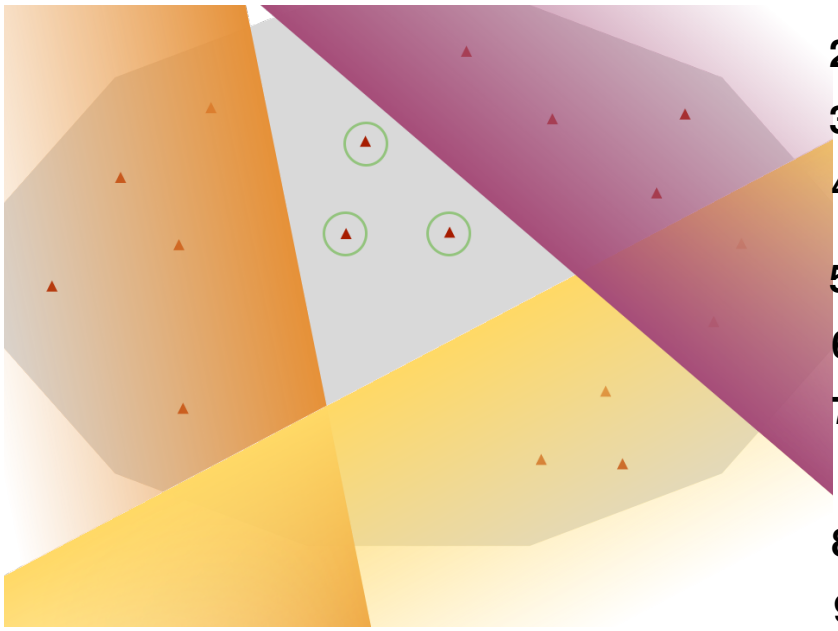
Example with $\kappa = 4$



1 **Function** TABLESAMPLING($\mathcal{P}, \kappa, v, p$)
  Data: A CSP $\mathcal{P}$, $\kappa \geq 2$, $v > 0$,
  $0 < p < 1$
2 $S \leftarrow$ FINDSOLUTIONS($\mathcal{P}, \kappa$);
3 **if** $|S| = 0$ **then**
4 **return** "No solution";
5 **while** $|S| = 0 \vee |S| = \kappa$ **do**
6 $T \leftarrow$ RANDOMTABLE($\mathcal{P}, v, p$);
7 $S \leftarrow$ FINDSOLUTIONS($\mathcal{P} \wedge T, \kappa$)
  ;
8 **if** $|S| \neq 0$ **then**
9 $\mathcal{P} \leftarrow \mathcal{P} \wedge T$;
10 **return** RANDOMELEMENT(S);

Sampling algorithm

Example with $\kappa = 4$



1 **Function** TABLESAMPLING($\mathcal{P}, \kappa, v, p$)
  Data: A CSP $\mathcal{P}$, $\kappa \geq 2$, $v > 0$,
    $0 < p < 1$
2 $S \leftarrow$ FINDSOLUTIONS($\mathcal{P}, \kappa$);
3 **if** $|S| = 0$ **then**
4 **return** "No solution";
5 **while** $|S| = 0 \vee |S| = \kappa$ **do**
6 $T \leftarrow$ RANDOMTABLE($\mathcal{P}, v, p$);
7 $S \leftarrow$
      FINDSOLUTIONS($\mathcal{P} \wedge T, \kappa$);
8 **if** $|S| \neq 0$ **then**
9 $\mathcal{P} \leftarrow \mathcal{P} \wedge T$;
10 **return** RANDOMELEMENT(S);

Improvement : propagation before table creation

1 **Function** TABLESAMPLING($\mathcal{P}, \kappa, v, p$)
  Data: A CSP $\mathcal{P}$, $\kappa \geq 2$, $v > 0$, $0 < p < 1$
2 $S \leftarrow$ FINDSOLUTIONS($\mathcal{P}, \kappa$);
3 **if** $|S| = 0$ **then**
4 **return** "No solution";
5 **while** $|S| = 0 \vee |S| = \kappa$ **do**
6 PROPAGATE($\mathcal{P}$);
7 $T \leftarrow$ RANDOMTABLE($\mathcal{P}, v, p$)
  $S \leftarrow$ FINDSOLUTIONS($\mathcal{P} \wedge T, \kappa$);
8 **if** $|S| \neq 0$ **then**
9 $\mathcal{P} \leftarrow \mathcal{P} \wedge T$;
10 **return** RANDOMELEMENT(S);

Improvement : dichotomic addition of tables

Idea : add twice more tables at each step

```
1  Function TableSampling(𝒫, κ, v, p)
       Data: A CSP 𝒫, κ ≥ 2, v > 0, 0 < p < 1
2      S ← FindSolutions(𝒫, κ);
3      if |S| = 0 then
4          return "No solution";
5      while |S| = 0 ∨ |S| = κ do
6          T_1, ..., T_k ← RandomTable(𝒫, v, p);
7          S ← FindSolutions(𝒫 ∧ T_1, ..., T_k , κ);
8          if |S| ≠ 0 then
9              𝒫 ← 𝒫 ∧ T_1, ..., T_k;
       /* k evolves as a power of 2 */
10     return RandomElement(S);
```

Implementation

Implementation

- Java 11
- Using Choco-Solver

Instances used

- On Call Rostering
- Feature Models
- Renault Mégane configuration
- *N*-queens

RandomVarDom

Definition

This search strategy randomly chooses

- a variable X such that $|\mathcal{D}(X)| \neq 1$
- a value x in $\mathcal{D}(X)$

And makes the decision $X = x$.

Non uniformity

Consider the problem $X, Y \in \{0, 1\}, X \vee Y = 1$, then with RandomVarDom the probability to sample a solution s is

$$\begin{array}{lcccc} \mathbb{P}(s = \{X \mapsto 0, Y \mapsto 1\}) & = & 3/8 & & 1/3 \\ \mathbb{P}(s = \{X \mapsto 1, Y \mapsto 0\}) & = & 3/8 & \textit{versus} & 1/3 \\ \mathbb{P}(s = \{X \mapsto 1, Y \mapsto 1\}) & = & 1/4 & & 1/3 \end{array}$$

How to evaluate the approach ?

Experimentally !

Evaluate:

- impact of parameters
- and versus RANDOMVARDOM

on:

- quality of randomness
- running time

Result:

- better randomness
- but no uniformity guarantee
- reasonable running time

Random Table constraints

Parameters

- Number of variables in the table : v
- Probability to keep a tuple : p

Example

$X_1, X_2 \in \{0, 1, 2\}$
$X_3, X_4 \in \{0, 1\}$
$p = 1/2$

X_1	X_2	X_3	X_4
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
0	2	0	0
0	2	0	1
0	2	1	0
0	2	1	1
1	0	0	0
⋮	⋮	⋮	⋮

Random Table constraints

Parameters

- Number of variables in the table : v
- Probability to keep a tuple : p

Example

$X_1, X_2 \in \{0, 1, 2\}$
$X_3, X_4 \in \{0, 1\}$
$p = 1/2$

X_1	X_2	X_3	X_4
0	0	0	0
0	0	1	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
0	2	0	0
0	2	0	1
0	2	1	0
0	2	1	1
1	0	0	0
⋮	⋮	⋮	⋮

Division of the space

Let $\mathcal{P}$ be a CSP, T a random table constraint with probability p, then

$$\mathbb{E}\left(|Sols(\mathcal{P} \wedge T)|\right) = p|Sols(\mathcal{P})|$$

Comparison of running time

Problem	RANDOM-VARDOM	Table Sampling				Ratio
		v	κ	p	Time	
Mégane	32 ms	2	16	1/16	55 ms	1.7
				1/32	59 ms	1.8
			32	1/16	89 ms	2.8
				1/32	86 ms	2.7
		3	16	1/16	74 ms	2.3
				1/32	67 ms	2.1
			32	1/16	83 ms	2.6
				1/32	65 ms	2.0
On Call Rostering	38 ms	2	16	1/16	14 ms	0.36
				1/32	14 ms	0.38
			32	1/16	15 ms	0.4
				1/32	18 ms	0.46
		3	16	1/16	18 ms	0.47
				1/32	15 ms	0.4
			32	1/16	19 ms	0.5
				1/32	17 ms	0.44
12-queens	2 ms	2	16	1/16	4 ms	2.4
				1/32	4 ms	2.3
			32	1/16	7 ms	4.1
				1/32	7 ms	3.9
		3	16	1/16	10 ms	5.6
				1/32	8 ms	4.3
			32	1/16	12 ms	6.9
				1/32	11 ms	6.0

Table: Comparison of the time to sample one solution between RANDOMVARDOM and table sampling. Times are averaged over 50 samples.

Conclusion

- on ne réfléchit pas en math comme en info : attention aux complexités cachées (ici m) !
- les optimisations sont toujours des opérations risquées : en CP, même un calcul minuscule finit par coûter cher parce qu'il est répété à chaque étape de la recherche.

Credo :
les solveurs de contraintes sont des machines très sophistiquées mais largement artisanales. Il faut les étudier comme un système physique : les mesurer, les quantifier, les modéliser pour comprendre ce qui s'y passe.