Introduction à la Programmation Par Contraintes

Lebbah Yahia

Université Oran1, Laboratoire LITIO
Faculté des Sciences Exactes et Appliquées, Département Informatique
B.P. 1524 El-M'Naouer, 31000 Oran, Algérie
email: ylebbah@gmail.com

2022

Origine

La programmation par contraintes (PPC) est issue d'un rapprochement entre :

- Satisfaction de contraintes : Le système Alice - [*J.L. Laurière, A Language and a Program for Stating and Solving Combinatorial Problems, AIJ 1978*]

Origine

La programmation par contraintes (PPC) est issue d'un rapprochement entre :

- Satisfaction de contraintes : Le système Alice - [*J.L. Laurière, A Language and a Program for Stating and Solving Combinatorial Problems, AIJ 1978*]
  - Absence de résultats pratiques de la PNE !

Origine

La programmation par contraintes (PPC) est issue d'un rapprochement entre :

- Satisfaction de contraintes : Le système Alice - [*J.L. Laurière, A Language and a Program for Stating and Solving Combinatorial Problems, AIJ 1978*]
  - Absence de résultats pratiques de la PNE !
  - Idée : Recours aux démarches heuristiques et aux raisonnements locaux, tout en guarantissant la globalité !

Origine

La programmation par contraintes (PPC) est issue d'un rapprochement entre :

- Satisfaction de contraintes : Le système Alice - [*J.L. Laurière, A Language and a Program for Stating and Solving Combinatorial Problems, AIJ 1978*]
  - Absence de résultats pratiques de la PNE !
  - Idée : Recours aux démarches heuristiques et aux raisonnements locaux, tout en guarantissant la globalité !
- Recherche opérationnelle : PL (198?, ...), Graphes (1994, ...), PLNE (199?, ...), PNLNE (200?, ...), ...

Origine

La programmation par contraintes (PPC) est issue d'un rapprochement entre :

- Satisfaction de contraintes : Le système Alice - [*J.L. Laurière, A Language and a Program for Stating and Solving Combinatorial Problems, AIJ 1978*]
  - Absence de résultats pratiques de la PNE !
  - Idée : Recours aux démarches heuristiques et aux raisonnements locaux, tout en guarantissant la globalité !
- Recherche opérationnelle : PL (198?, ...), Graphes (1994, ...), PLNE (199?, ...), PNLNE (200?, ...), ...
  - 1ère Idée : RO pour améliorer la PPC

Origine

La programmation par contraintes (PPC) est issue d'un rapprochement entre :

- Satisfaction de contraintes : Le système Alice - [*J.L. Laurière, A Language and a Program for Stating and Solving Combinatorial Problems, AIJ 1978*]
  - Absence de résultats pratiques de la PNE !
  - Idée : Recours aux démarches heuristiques et aux raisonnements locaux, tout en guarantissant la globalité !
- Recherche opérationnelle : PL (198?, ...), Graphes (1994, ...), PLNE (199?, ...), PNLNE (200?, ...), ...
  - 1ère Idée : RO pour améliorer la PPC
  - 2ème Idée : PPC pour améliorer la RO

$$\text{PPC} = \text{Contraintes} + \text{Résolution}$$

La notation CSP

Un CSP (Constraint Satisfaction Problem) $\mathcal{P}$ est un triplet $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, avec :

- $\mathcal{X}$ un ensemble de n variables $x_1, \ldots, x_n$.

La notation CSP

Un CSP (Constraint Satisfaction Problem) $\mathcal{P}$ est un triplet $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, avec :

- $\mathcal{X}$ un ensemble de n variables $x_1, \ldots, x_n$.
- $\mathcal{D}$ le n-uplet $\langle D_1, \ldots, D_n \rangle$ des domaines des variables. D_i est l'ensemble contenant les valeurs de la variable x_i.

$$x_i \in D_i, i = 1..n.$$

La notation CSP

Un CSP (Constraint Satisfaction Problem) $\mathcal{P}$ est un triplet $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, avec :

- $\mathcal{X}$ un ensemble de n variables $x_1, \ldots, x_n$.
- $\mathcal{D}$ le n-uplet $\langle D_1, \ldots, D_n \rangle$ des domaines des variables. D_i est l'ensemble contenant les valeurs de la variable x_i.

$$x_i \in D_i, i = 1..n.$$

- $\mathcal{C} = \{C_1, \ldots, C_m\}$ l'ensemble des contraintes. Sémantiquement

$$C_i(x_{i_1}, \ldots, x_{i_l}) \subseteq D_{i_1} \times \ldots \times D_{i_l}.$$

Exemple : cryptarithmétique

Remplacer chaque lettre par une décimale telle que la somme

$$
\begin{array}{rr}
 & S\ E\ N\ D \\
+ & \\
 & M\ O\ R\ E \\
\hline
= & M\ O\ N\ E\ Y
\end{array}
$$

soit correcte.

Exemple : cryptarithmétique

Remplacer chaque lettre par une décimale telle que la somme

$$\begin{array}{rr} & S\;E\;N\;D \\ + & \\ & M\;O\;R\;E \\ \hline = & M\;O\;N\;E\;Y \end{array}$$

soit correcte.
Il existe une solution unique $9567 + 1085 = 10652$.

Exemple : cryptarithmétique - CSP

$$\mathcal{X} = \{S, E, N, D, M, O, R, Y, R_1, R_2, R_3, R_4\}$$

Exemple : cryptarithmétique - CSP

$$
\begin{aligned}
\mathcal{X} = \quad & \{S, E, N, D, M, O, R, Y, R_1, R_2, R_3, R_4\} \\
\mathcal{D} = \quad & \{D_S, D_E, D_N, D_D, D_M, D_O, D_R, D_Y, D_{R_1}, D_{R_2}, D_{R_3}\} \\
& D_S = D_E = D_N = D_D = D_M = D_O = D_R = D_Y = [0...9], \\
& D_{R_1} = D_{R_2} = D_{R_3} = [0...1]
\end{aligned}
$$

Exemple : cryptarithmétique - CSP

$$
\begin{array}{ll}
\mathcal{X} = & \{S, E, N, D, M, O, R, Y, R_1, R_2, R_3, R_4\} \\
\mathcal{D} = & \{D_S, D_E, D_N, D_D, D_M, D_O, D_R, D_Y, D_{R_1}, D_{R_2}, D_{R_3}\} \\
 & D_S = D_E = D_N = D_D = D_M = D_O = D_R = D_Y = [0...9], \\
 & D_{R_1} = D_{R_2} = D_{R_3} = [0...1] \\
\mathcal{C} = & \{C_1, ..., C_8\} \\
C_1 : & S \neq E \neq N \neq D \neq M \neq O \neq R \neq Y
\end{array}
$$

Exemple : cryptarithmétique - CSP

$$
\begin{array}{ll}
\mathcal{X} = & \{S, E, N, D, M, O, R, Y, R_1, R_2, R_3, R_4\} \\
\mathcal{D} = & \{D_S, D_E, D_N, D_D, D_M, D_O, D_R, D_Y, D_{R_1}, D_{R_2}, D_{R_3}\} \\
 & D_S = D_E = D_N = D_D = D_M = D_O = D_R = D_Y = [0...9], \\
 & D_{R_1} = D_{R_2} = D_{R_3} = [0...1] \\
\mathcal{C} = & \{C_1, ..., C_8\} \\
C_1 : & S \neq E \neq N \neq D \neq M \neq O \neq R \neq Y \\
C_2 : & S \neq 0; \quad C_3 : M \neq 0 \quad C_4 : R_1 = M
\end{array}
$$

Exemple : cryptarithmétique - CSP

$$\begin{array}{ll}
\mathcal{X} = & \{S, E, N, D, M, O, R, Y, R_1, R_2, R_3, R_4\} \\
\mathcal{D} = & \{D_S, D_E, D_N, D_D, D_M, D_O, D_R, D_Y, D_{R_1}, D_{R_2}, D_{R_3}\} \\
 & D_S = D_E = D_N = D_D = D_M = D_O = D_R = D_Y = [0...9], \\
 & D_{R_1} = D_{R_2} = D_{R_3} = [0...1] \\
\mathcal{C} = & \{C_1, ..., C_8\} \\
C_1 : & S \neq E \neq N \neq D \neq M \neq O \neq R \neq Y \\
C_2 : & S \neq 0; \quad C_3 : M \neq 0 \quad C_4 : R_1 = M \\
C_5 : & R_2 + S + M = O + 10 \times R_1
\end{array}$$

Exemple : cryptarithmétique - CSP

$$\begin{array}{ll}
\mathcal{X} = & \{S, E, N, D, M, O, R, Y, R_1, R_2, R_3, R_4\} \\
\mathcal{D} = & \{D_S, D_E, D_N, D_D, D_M, D_O, D_R, D_Y, D_{R_1}, D_{R_2}, D_{R_3}\} \\
 & D_S = D_E = D_N = D_D = D_M = D_O = D_R = D_Y = [0...9], \\
 & D_{R_1} = D_{R_2} = D_{R_3} = [0...1] \\
\mathcal{C} = & \{C_1, ..., C_8\} \\
C_1 : & S \neq E \neq N \neq D \neq M \neq O \neq R \neq Y \\
C_2 : & S \neq 0; \quad C_3 : M \neq 0 \quad C_4 : R_1 = M \\
C_5 : & R_2 + S + M = O + 10 \times R_1 \\
C_6 : & R_3 + E + O = N + 10 \times R_2
\end{array}$$

Exemple : cryptarithmétique - CSP

$$
\begin{array}{ll}
\mathcal{X} = & \{S, E, N, D, M, O, R, Y, R_1, R_2, R_3, R_4\} \\
\mathcal{D} = & \{D_S, D_E, D_N, D_D, D_M, D_O, D_R, D_Y, D_{R_1}, D_{R_2}, D_{R_3}\} \\
 & D_S = D_E = D_N = D_D = D_M = D_O = D_R = D_Y = [0...9], \\
 & D_{R_1} = D_{R_2} = D_{R_3} = [0...1] \\
\mathcal{C} = & \{C_1, ..., C_8\} \\
C_1 : & S \neq E \neq N \neq D \neq M \neq O \neq R \neq Y \\
C_2 : & S \neq 0; \quad C_3 : M \neq 0 \quad C_4 : R_1 = M \\
C_5 : & R_2 + S + M = O + 10 \times R_1 \\
C_6 : & R_3 + E + O = N + 10 \times R_2 \\
C_7 : & R_4 + N + R = E + 10 \times R_3
\end{array}
$$

Exemple : cryptarithmétique - CSP

$$
\begin{array}{ll}
\mathcal{X} = & \{S, E, N, D, M, O, R, Y, R_1, R_2, R_3, R_4\} \\
\mathcal{D} = & \{D_S, D_E, D_N, D_D, D_M, D_O, D_R, D_Y, D_{R_1}, D_{R_2}, D_{R_3}\} \\
 & D_S = D_E = D_N = D_D = D_M = D_O = D_R = D_Y = [0...9], \\
 & D_{R_1} = D_{R_2} = D_{R_3} = [0...1] \\
\mathcal{C} = & \{C_1, ..., C_8\} \\
C_1 : & S \neq E \neq N \neq D \neq M \neq O \neq R \neq Y \\
C_2 : & S \neq 0; \quad C_3 : M \neq 0 \quad C_4 : R_1 = M \\
C_5 : & R_2 + S + M = O + 10 \times R_1 \\
C_6 : & R_3 + E + O = N + 10 \times R_2 \\
C_7 : & R_4 + N + R = E + 10 \times R_3 \\
C_8 : & D + E = Y + 10 \times R_4
\end{array}
$$

Exemple : emploi du temps

Définir l'emploi du temps de managers d'un centre d'assistance comportant 5 activités, représentés par un ensemble $A = \{$**M***atin*, **J***our*, **A***près-midi*, **S***oir*, **W***eek-end*$\}$, 7 personnes, représentés par l'ensemble $P = \{Hassen, Yassine, Ahmed, Tarik, Omar, Saïd, Ali\}$.
Une personne de P doit effectuer au moins une des activiés de A.
On veut affecter les activités aux managers en respectant le fait que :

- Au moins une personne doit faire M, et au plus deux.
- Au moins une personne doit faire J, et au plus deux.
- Au moins une personne doit faire A, et pas plus d'une.
- Au plus deux personnes peuvent faire S.
- Au plus deux personnes peuvent faire W.

Exemple : emploi du temps

Définir l'emploi du temps de managers d'un centre d'assistance comportant 5 activités, représentés par un ensemble $A = \{$**M***atin*, **J***our*, **A***près-midi*, **S***oir*, **W***eek-end*$\}$, 7 personnes, représentés par l'ensemble $P = \{Hassen, Yassine, Ahmed, Tarik, Omar, Saïd, Ali\}$. Une personne de P doit effectuer au moins une des activiés de A. On veut affecter les activités aux managers en respectant le fait que :

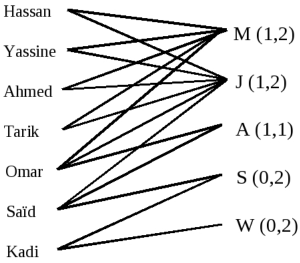
- Au moins une personne doit faire M, et au plus deux.
- Au moins une personne doit faire J, et au plus deux.
- Au moins une personne doit faire A, et pas plus d'une.
- Au plus deux personnes peuvent faire S.
- Au plus deux personnes peuvent faire W.
- *Hassan* est intéressé par M et J; *Yassine* par M et J; *Ahmed* par M et J; *Tarik* par M et J; *Omar* par M, J et A; *Sad* par J, A et S; *Ali* par S et W;

Exemple : emploi du temps - CSP

$$\begin{array}{lll} \mathcal{X} = & \{H, Y, A, T, O, S, K\} & \mathcal{D} = \{D_H, D_Y, D_A, D_T, D_O, D_S, D_K\} \\ & D_H = \{M, J\}, ..., D_A = \{S, W\} & \mathcal{C} = \{C_1\} \end{array}$$

Exemple : emploi du temps - CSP

$$
\begin{array}{ll}
\mathcal{X} = & \{H, Y, A, T, O, S, K\} \quad \mathcal{D} = \{D_H, D_Y, D_A, D_T, D_O, D_S, D_K\} \\
 & D_H = \{M, J\}, ..., D_A = \{S, W\} \qquad \mathcal{C} = \{C_1\} \\
C_1 : & gcc(\{H, Y, A, T, O, S, K\}, \{M, J, A, S, W\}, \\
 & [1, 1, 1, 0, 0], [2, 2, 1, 2, 2])
\end{array}
$$



Exemple : ordonnancement

Soit un problème d'odonnancement :

- 7 tâches.
- Chaque tâche a une durée et une consommation de ressources.

tâche	t_1	t_2	t_3	t_4	t_5	t_6	t_7
durée	16	6	13	7	5	18	4
ressource	2	9	3	7	10	1	11

Ordonnancer les tâches sans dépasser la capacité en ressource 13 à n'importe quel instant. (Toute les tâches sont effectuées en moins de 30 minutes.)

Exemple : ordonnancement - CSP

$\mathcal{X} = LO \cup LE$
$LO = \{O_1, ..., O_7\}$
$LE = \{E_1, ..., E_7\}$
$D_{O_1} = ... = D_{O_7} = D_{E_1} = ... = D_{E_7} = [1, ..., 30]$
$C_1 : O_1 + 16 = E_1 \quad C_2 : O_2 + 6 = E_2 \quad C_3 : O_3 + 13 = E_3$
$C_4 : O_4 + 7 = E_4 \quad C_5 : O_5 + 5 = E_5 \quad C_6 : O_6 + 18 = E_6$
$C_7 : O_7 + 4 = E_7$
Soit $D = (16, 6, 13, 7, 5, 18, 4), R = (2, 9, 3, 7, 10, 1, 11), L = 13$
$C_8 : \forall i \in \boldsymbol{N}, \Sigma_{j|O_j \leq i \leq O_j + D_j - 1} R_j \leq 13$

Exemple : mélange chimique

Nous avons quatre réservoires : $R1, R2, R3$ et $R4$. Les deux premiers reçoivent trois produits de sources distinctes, puis leur contenu est combiné dans les deux autres afin de créer les mélanges désirés. La question est de déterminer la quantité de chaque produit à acheter afin de maximiser les profits.

$$
\begin{array}{ll}
 & x_1 \geq 0, \leq 100, x_2 \geq 0, \leq 100, \\
 & x_3 \geq 0, \leq 100, x_4 \geq 0, \leq 100, x_5 \geq 1, \leq 3. \\
\textit{minimize} & 120x_1 + 60x_2 + 10x_3 - 50x_4 - 50x_1x5 - 50x_2x_5 \\
\textit{subject to} & c_1 : 2.5x_1 + 0.5x_3 - x_1x_5 \geq 0 \\
 & c_2 : 1.5x_2 - 0.5x_4 - x_2x_5 \geq 0 \\
 & c_3 : x_1 + x_3 \leq 10 \\
 & c_4 : x_2 + x_4 \leq 20
\end{array}
$$

Exemple : PPC et test logiciel

- Test structurel :
  Trouver un jeu de cas de test dont le flot d'exécution passe par un point du programme !
- Démarches classiques : génération aléatoire !
  → un fruit du hasard !
- Avec la PPC :
  - Transformation du programme en un système de contraintes
  - Transformation du critère structurel en une contrainte
  - Résolution du système résultant
    - Pas de solution : du code mort
    - Des solutions : toutes sont des cas de test

```
int proc(int i)
  int j;
  j := 2;
  if (I ≤ 16) {
      j := j*I;
  }
  if (j > 8) {
      j := 0
  }
  return (j)
```

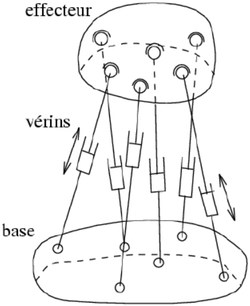
Exemple : PPC et test logiciel

```
int proc(int i)
  int j;
  j := 2;
  if (I ≤ 16) {
     j := j*i;
  }
  if (j > 8) {
     j := 0
  }
  return (j)
```

```
I0∈[0, 2^16-1]
J1,J2,J3,J4,J5∈[0, 2^16-1]
J1=2
ite(J1 ≤16,
   J2=J1*I0∧J3=J2,
   J3=J1)
ite(J3≤8,
   J4=0∧J5=J4,
   J5=J3)
RET = J5
J3 > 8
```

- Après un filtrage par *arc-consistance* : $i \in [5, 16]$
- Après une *énumération* : parmi les solutions $i=5$

Exemple : PPC et robotique



Problem 1 (Gough-Stewart [11]).

$$\begin{cases} x_1^2 + y_1^2 + z_1^2 = 31;\ x_2^2 + y_2^2 + z_2^2 = 39;\ x_3^2 + y_3^2 + z_3^2 = 29; \\ x_1x_2 + y_1y_2 + z_1z_2 + 6x_1 - 6x_2 = 51; \\ x_1x_3 + y_1y_3 + z_1z_3 + 7x_1 - 2y_1 - 7x_3 + 2y_3 = 50; \\ x_2x_3 + y_2y_3 + z_1z_3 + x_2 - 2y_2 - x_3 + 2y_3 = 34; \\ -12x_1 + 15y_1 - 10x_2 - 25y_2 + 18x_3 + 18y_3 = -32; \\ -14x_1 + 35y_1 - 36x_2 - 45y_2 + 30x_3 + 18y_3 = 8; \\ 2x_1 + 2y_1 - 14x_2 - 2y_2 + 8x_3 - y_3 = 20; \\ x_1 \in [-2.00, 5.57];\ y_1 \in [-5.57, 2.70];\ z_1 \in [0, 5.57] \\ x_2 \in [-6.25, 1.30];\ y_2 \in [-6.25, 2.70];\ z_2 \in [-2.00, 6.25] \\ x_3 \in [-5.39, 0.70];\ y_3 \in [-5.39, 3.11];\ z_3 \in [-3.61, 5.39] \end{cases}$$

Exemple : vérification et SAT

- Modélisation des problèmes de vérification sous forme de formules booléennes

$$(a_{11} \vee \ldots \vee a_{1l_1}) \wedge (a_{21} \vee \ldots \vee a_{2l_2}) \wedge \ldots \wedge (a_{n1} \vee \ldots \vee a_{nl_n})$$

- Résolution de ces formules avec les solveurs SAT

Résolution = Filtrage + Recherche

Résolution = Filtrage + Recherche

- Problèmes NP-complets, NP-difficiles, ... problèmes indécidables, ...

Résolution = Filtrage + Recherche

- Problèmes NP-complets, NP-difficiles, ... problèmes indécidables, ...
- ... difficulté dans le cas général, ...
- ... mais les instances peuvent être résolues

"r a p i d e m e n t"

Résolution = Filtrage + Recherche

- Problèmes NP-complets, NP-difficiles, ... problèmes indécidables, ...
- ... difficulté dans le cas général, ...
- ... mais les instances peuvent être résolues
  "r a p i d e m e n t"
- Résolution = Filtrage + Recherche

Résolution = Filtrage + Recherche

- Problèmes NP-complets, NP-difficiles, ... problèmes indécidables, ...
- ... difficulté dans le cas général, ...
- ... mais les instances peuvent être résolues

  "r a p i d e m e n t"
- Résolution = Filtrage + Recherche
- Filtrage :
  - étant donné les propriétés mathématiques d'une contrainte C_i,

Résolution = Filtrage + Recherche

- Problèmes NP-complets, NP-difficiles, ... problèmes indécidables, ...
- ... difficulté dans le cas général, ...
- ... mais les instances peuvent être résolues

  "r a p i d e m e n t"
- Résolution = Filtrage + Recherche
- Filtrage :
  - étant donné les propriétés mathématiques d'une contrainte C_i,
  - exploitation de ces propriétés pour supprimer les valeurs "inconsistantes" ne participant pas aux solutions de C_i.

Résolution = Filtrage + Recherche

- Problèmes NP-complets, NP-difficiles, ... problèmes indécidables, ...
- ... difficulté dans le cas général, ...
- ... mais les instances peuvent être résolues

  "r a p i d e m e n t"
- Résolution = Filtrage + Recherche
- Filtrage :
  - étant donné les propriétés mathématiques d'une contrainte C_i,
  - exploitation de ces propriétés pour supprimer les valeurs "inconsistantes" ne participant pas aux solutions de C_i.
- Recherche : Diviser le problème en sous-problèmes en décomposant le domaine des variables.

Filtrage - Exemple illustratif

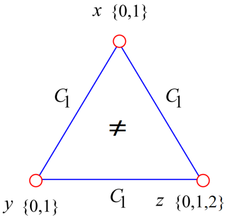
Soit un CSP

$$\mathcal{X} = \{x, y, z, t, r\}$$
$$D_x = D_y = \{0, 1\} \quad D_z = \{0, 1, 2\} \quad D_t = \{2, 3, 4\} \quad D_r = \{0, 1, 2, 3, 4\}$$
$$C_1 : x \neq y \neq z \quad C_2 : x^2 + t = z \quad C_3 : r^3 + x \geq 0$$

Filtrage - Exemple illustratif

Soit un CSP

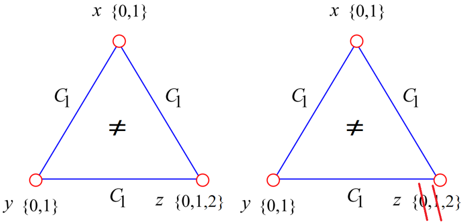
$$\mathcal{X} = \{x, y, z, t, r\}$$
$$D_x = D_y = \{0, 1\} \quad D_z = \{0, 1, 2\} \quad D_t = \{2, 3, 4\} \quad D_r = \{0, 1, 2, 3, 4\}$$
$$C_1 : x \neq y \neq z \quad C_2 : x^2 + t = z \quad C_3 : r^3 + x \geq 0$$



Filtrage - Exemple illustratif

Soit un CSP

$\mathcal{X} = \{x, y, z, t, r\}$
$D_x = D_y = \{0, 1\}$ $D_z = \{0, 1, 2\}$ $D_t = \{2, 3, 4\}$ $D_r = \{0, 1, 2, 3, 4\}$
$C_1 : x \neq y \neq z$ $C_2 : x^2 + t = z$ $C_3 : r^3 + x \geq 0$



Avec C_1, on peut déduire

$$D_z = \{2\}.$$

Filtrage - Exemple illustratif

Soit un CSP

$\mathcal{X} = \{x, y, z, t, r\}$
$D_x = D_y = \{0, 1\}$ $\quad D_z = \{2\}$ $\quad D_t = \{2, 3, 4\}$ $\quad D_r = \{0, 1, 2, 3, 4\}$
$C_1 : x \neq y \neq z$ $\quad C_2 : x^2 + t = z$ $\quad C_3 : r^3 + x \geq 0$

C_2 prend la forme $x^2 + t = 2$, d'où :

$$D_x = \{0\} \quad D_t = \{2\}.$$

Filtrage - Exemple illustratif

Soit un CSP

$$\mathcal{X} = \{x, y, z, t, r\}$$
$$D_x = D_y = \{0, 1\} \quad D_z = \{2\} \quad D_t = \{2, 3, 4\} \quad D_r = \{0, 1, 2, 3, 4\}$$
$$C_1 : x \neq y \neq z \quad C_2 : x^2 + t = z \quad C_3 : r^3 + x \geq 0$$

C_2 prend la forme $x^2 + t = 2$, d'où :

$$D_x = \{0\} \quad D_t = \{2\}.$$

Puis, on revient à C_1 qui prend désormais la forme $0 \neq y \neq 2$, d'où :

$$D_y = \{1\}.$$

Filtrage - Exemple illustratif

Soit un CSP

$\mathcal{X} = \{x, y, z, t, r\}$
$D_x = D_y = \{0, 1\} \quad D_z = \{2\} \quad D_t = \{2, 3, 4\} \quad D_r = \{0, 1, 2, 3, 4\}$
$C_1 : x \neq y \neq z \quad C_2 : x^2 + t = z \quad C_3 : r^3 + x \geq 0$

C_2 prend la forme $x^2 + t = 2$, d'où :

$$D_x = \{0\} \quad D_t = \{2\}.$$

Puis, on revient à C_1 qui prend désormais la forme $0 \neq y \neq 2$, d'où :

$$D_y = \{1\}.$$

Enfin :

$$D_x = \{0\}, D_y = \{1\}, D_z = \{2\} \quad D_t = \{2\} \quad D_r = \{0, 1, 2, 3, 4\}.$$

Définitions graphiques de base

hypergraphes des contraintes

A chaque CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ est associé un hypergraphe des contraintes obtenu en représentant chaque variable du CSP par un sommet et chaque contrainte $C_j(x_{j_1}, \ldots, x_{i_l})$ par une hyperarête entre $x_{j_1}, \ldots,$ et x_{i_l}.

hypergraphe de consistance

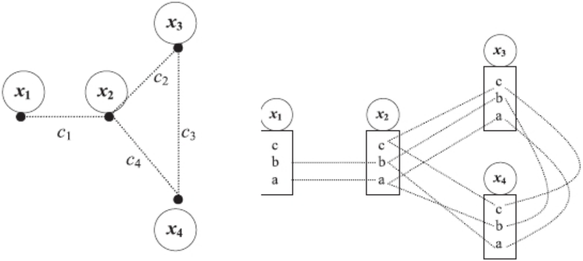
L'hypergraphe de consistance ou microstructure d'un CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ est l'hypergraphe dont les sommets sont le éléments de domaines d_i, et qui comporte une hyperarête $d_{i_1}, \ldots, d_{i_l}$ ssi le l-uplet est autorisé par la contrainte $C(x_{i_1}, \ldots, x_{i_l})$.

Définitions graphiques de base - exemples

Soit le CSP

- $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$
- $\mathcal{D} = \{D_1, D_2, D_3, D_4\}$, avec $D_1 = D_2 = D_3 = D_4 = \{a, b, c\}$
- $\mathcal{C} = \{C_1(x_2, x_3), C_2(x_2, x_3), C_3(x_3, x_4), C_4(x_2, x_4)\}$ où :
  - $C_1(x_1, x_2) = \{(a, a), (b, b)\}$
  - $C_2(x_2, x_3) = C_3(x_3, x_4) = \{(a, a), (b, b), (c, c)\}$
  - $C_4(x_2, x_4) = \{(a, b), (b, a), (c, c)\}$

Définitions graphiques de base - exemples



Définitions de base

Instanciation partielle ou locale

Une instanciation partielle I de V est de la forme $\{x_{i_1} \rightarrow d_{i_1}, \ldots, x_{i_k} \rightarrow d_{i_k}\}$. I est un élément de $D_{i_1} \times \ldots \times D_{i_k}$.

Définitions de base

Instanciation partielle ou locale

Une instanciation partielle I de V est de la forme $\{x_{i_1} \to d_{i_1}, \ldots, x_{i_k} \to d_{i_k}\}$. I est un élément de $D_{i_1} \times \ldots \times D_{i_k}$.

Satisfiabilité d'une instanciation partielle

Une instanciation partielle $\langle d_{j_1}, \ldots, d_{j_l} \rangle$ vérifie une contrainte $C_j(x_{j_1}, \ldots, x_{j_l} \rangle)$ si et seulement si elle appartient à cette contrainte.

Définitions de base

Instanciation partielle ou locale

Une instanciation partielle I de V est de la forme $\{x_{i_1} \rightarrow d_{i_1}, \ldots, x_{i_k} \rightarrow d_{i_k}\}$. I est un élément de $D_{i_1} \times \ldots \times D_{i_k}$.

Satisfiabilité d'une instanciation partielle

Une instanciation partielle $\langle d_{j_1}, \ldots, d_{j_l} \rangle$ vérifie une contrainte $C_j(x_{j_1}, \ldots, x_{j_l} \rangle)$ si et seulement si elle appartient à cette contrainte.

Consistance locale d'une instanciation

Une instanciation partielle est dite localement consistante ssi elle satisfait toutes les contraintes se portant sur les variables de cette instanciation.

Définitions de base

Valeur compatible

Une valeur (x_i, d_i), avec $d_i \in D_i$, est dite compatible avec une instanciation localement consistante I, si $I \cup \{x_i \rightarrow d_i\}$ est localement consistante.

Définitions de base

Valeur compatible

Une valeur (x_i, d_i), avec $d_i \in D_i$, est dite compatible avec une instanciation localement consistante I, si $I \cup \{x_i \to d_i\}$ est localement consistante.

Instanciation globalement consistante

Une instanciation globalement consistante ou une solution d'un CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ est une instanciation complète de $\mathcal{X}$ localement consistante.

Définitions de base

Valeur compatible

Une valeur (x_i, d_i), avec $d_i \in D_i$, est dite compatible avec une instanciation localement consistante I, si $I \cup \{x_i \rightarrow d_i\}$ est localement consistante.

Instanciation globalement consistante

Une instanciation globalement consistante ou une solution d'un CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ est une instanciation complète de $\mathcal{X}$ localement consistante.

Instanciation partielle globalement consistante

Une instanciation partielle est globalement consistante ssi elle peut être étendue à une solution.

Définitions de base

Propriété

Si toutes les contraintes sont d'arité n, on aura $Sol(\mathcal{P}) = \mathcal{D} \cap C_1 \cap \ldots \cap C_m$.

Définitions de base

Propriété

Si toutes les contraintes sont d'arité n, on aura $Sol(\mathcal{P}) = \mathcal{D} \cap C_1 \cap \ldots \cap C_m$.

Equivalence de CSPs

Nous dirons que les deux CSP $\mathcal{P}_1$ et $\mathcal{P}_2$ sont équivalents si et seulement si $Sol(\mathcal{P}_1) = Sol(\mathcal{P}_2)$.

Filtrage et *arc*-consistance

Etant donné un CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$:

- $C_i(x_{i_1}, \ldots, x_{i_l}) \subseteq D_{i_1} \times \ldots \times D_{i_l}$

Filtrage et *arc*-consistance

Etant donné un CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$:

- $C_i(x_{i_1}, \ldots, x_{i_l}) \subseteq D_{i_1} \times \ldots \times D_{i_l}$
- Un tuple $(d_{i_1}, \ldots, d_{i_l}) \in C_i$ est dit Solution de C_i.

Filtrage et *arc*-consistance

Etant donné un CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$:

- $C_i(x_{i_1}, \ldots, x_{i_l}) \subseteq D_{i_1} \times \ldots \times D_{i_l}$
- Un tuple $(d_{i_1}, \ldots, d_{i_l}) \in C_i$ est dit Solution de C_i.
- On dit aussi que ce tuple satisfait C_i.

Filtrage et *arc*-consistance

Etant donné un CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$:

- $C_i(x_{i_1}, \ldots, x_{i_l}) \subseteq D_{i_1} \times \ldots \times D_{i_l}$
- Un tuple $(d_{i_1}, \ldots, d_{i_l}) \in C_i$ est dit Solution de C_i.
- On dit aussi que ce tuple satisfait C_i.

arc-consistance

Une valeur $v \in D_{x_i}$ est *arc*-consistante si et seulement si elle est localement consistante. Sinon, on dit que v est non-*arc*-consistante.

Filtrage et *arc*-consistance : formellement

Etant donné un CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$:

- Ce CSP est *arc*-consistant si et seulement si pour toute variable x_i et toute valeur $d_i \in D_{x_i}$, d_i est *arc*-consistante.
- d_i est *arc*-consistante si et seulement si

Filtrage et *arc*-consistance : formellement

Etant donné un CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$:

- Ce CSP est *arc*-consistant si et seulement si pour toute variable x_i et toute valeur $d_i \in D_{x_i}$, d_i est *arc*-consistante.
- d_i est *arc*-consistante si et seulement si

$$
\begin{array}{l}
\forall C_j(x_{j_1}, \ldots, x_{j_k} \ldots, x_{j_l}), \\
\quad x_i = x_{j_k}, \\
\quad \exists d_{i_1}, \ldots, \exists d_{j_{k-1}}, \exists d_{j_{k+1}}, \ldots, \exists d_{j_l} \\
\quad (d_{i_1} \in D_{i_1}, \ldots, d_{j_{k-1}} \in D_{j_{k-1}}, d_{j_{k+1}} \in D_{j_{k+1}}, \ldots, d_{j_l} \in D_{j_l}) \\
(d_{i_1}, \ldots, d_i, \ldots, d_{j_l}) \in C_j
\end{array}
$$

Filtrage : propagation

- Tant qu'il existe des valeurs non-*arc*-consistantes, les supprimer, et continuer à détecter de nouvelles valeurs inconsistantes.
- Soit $\pi_{j,k}(\mathcal{D})$ l'ensemble des valeurs de la variable x_k qui sont *arc*-consistantes relativement à la contrainte C_j :

$$
\begin{array}{l}
\pi_{j,i}(\mathcal{D}) = \\
\{d_i | d_i \in D_i, \\
\quad x_i = x_{j_k}, \\
\quad \exists d_{i_1}, \ldots, \exists d_{j_{k-1}}, \exists d_{j_{k+1}}, \ldots, \exists d_{j_l} \\
\quad (d_{i_1} \in D_{i_1}, \ldots, d_{j_{k-1}} \in D_{j_{k-1}}, d_{j_{k+1}} \in D_{j_{k+1}}, \ldots, d_{j_l} \in D_{j_l}) \\
(d_{i_1}, \ldots, d_i, \ldots, d_{j_l}) \in C_j\}.
\end{array}
$$

Filtrage - AC3

1: $L := \{C_1, ..., C_m\}$; % toutes les contraintes

Filtrage - AC3

$L := \{C_1, ..., C_m\}$; % toutes les contraintes
while L est non vide **do**
    Choisir et supprimer dans L une contrainte $C_j(x_{j_1}, \ldots, x_{j_l})$;

Filtrage - AC3

$L := \{C_1, ..., C_m\}$; % toutes les contraintes
while L est non vide **do**
    Choisir et supprimer dans L une contrainte $C_j(x_{j_1}, \ldots, x_{j_l})$;
    for $x_k \in \{x_{j_1}, \ldots, x_{j_l}\}$ **do**
        Réduire le domaine de x_k avec la contrainte C_j

Filtrage - AC3

```
L := {C_1, ..., C_m}; % toutes les contraintes
while L est non vide do
    Choisir et supprimer dans L une contrainte C_j(x_{j_1}, ..., x_{j_l});
    for x_k ∈ {x_{j_1}, ..., x_{j_l}} do
        Réduire le domaine de x_k avec la contrainte C_j
        Si réduction alors L := L ∪ {C_p | x_k ∈ var(C_p)};
    end for
end while
```

Filtrage - AC3

$L := \mathcal{C}$;
while L est non vide **do**
  Choisir et supprimer dans L une contrainte $C_j(x_{j_1}, \ldots, x_{j_l})$;
  for $x_k \in \{x_{j_1}, \ldots, x_{j_l}\}$ **do**
    if $\pi_{j,k}(\mathcal{D}) \neq D_k$ **then**
      $D_k := \pi_{j,k}(\mathcal{D})$;
      $L := L \cup \{C_p | x_k \in var(C_p)\}$;
    end if
  end for
end while

Filtrage - Contraintes globales

- *alldifferent*$([x_1, ..., x_n])$: $x_1, ..., x_n$ doivent prendre des valeurs différentes deux à deux.

Filtrage - Contraintes globales

- *alldifferent*$([x_1, ..., x_n])$: $x_1, ..., x_n$ doivent prendre des valeurs différentes deux à deux.
- *element*(i, l, v) : exprime que la *i*ème variable dans une liste de variables $l = [x_1, ..., x_n]$ prenne la valeur v, i.e. $x_i = v$.

Filtrage - Contraintes globales

- *alldifferent*$([x_1, ..., x_n])$: $x_1, ..., x_n$ doivent prendre des valeurs différentes deux à deux.
- *element*(i, l, v) : exprime que la $i$ème variable dans une liste de variables $l = [x_1, ..., x_n]$ prenne la valeur v, i.e. $x_i = v$.
- *cumulative*(S, D, R, l, e) : Soient n tâches. La tâche j commence à S_j de durée D_j et a besoin de R_j unités d'une ressource donnée. Elle impose que les tâches doivent être exécutées sans dépasser l unités et la fin du temps e.

Filtrage - Contraintes globales

- *alldifferent*$([x_1, ..., x_n])$: $x_1, ..., x_n$ doivent prendre des valeurs différentes deux à deux.
- *element*(i, l, v) : exprime que la $i$ème variable dans une liste de variables $l = [x_1, ..., x_n]$ prenne la valeur v, i.e. $x_i = v$.
- *cumulative*(S, D, R, l, e) : Soient n tâches. La tâche j commence à S_j de durée D_j et a besoin de R_j unités d'une ressource donnée. Elle impose que les tâches doivent être exécutées sans dépasser l unités et la fin du temps e.
- *sort*$([x_1, ..., x_n], [y_1, ..., y_n])$: exprime que $(y_1, ..., y_n)$ est obtenu à partir de $(x_1, ..., x_n)$ en triant les éléments par ordre croissant.

Filtrage - Contraintes globales

- *alldifferent*$([x_1, ..., x_n])$: $x_1, ..., x_n$ doivent prendre des valeurs différentes deux à deux.
- *element*(i, l, v) : exprime que la $i$ème variable dans une liste de variables $l = [x_1, ..., x_n]$ prenne la valeur v, i.e. $x_i = v$.
- *cumulative*(S, D, R, l, e) : Soient n tâches. La tâche j commence à S_j de durée D_j et a besoin de R_j unités d'une ressource donnée. Elle impose que les tâches doivent être exécutées sans dépasser l unités et la fin du temps e.
- *sort*$([x_1, ..., x_n], [y_1, ..., y_n])$: exprime que $(y_1, ..., y_n)$ est obtenu à partir de $(x_1, ..., x_n)$ en triant les éléments par ordre croissant.
- *gcc*(X, V, l, u) : la cardinalité d'une valeur V_i doit être dans $[l_i, u_i]$ parmi les variables X.

Filtrage - Contraintes globales

- *alldifferent*$([x_1, ..., x_n])$: $x_1, ..., x_n$ doivent prendre des valeurs différentes deux à deux.
- *element*(i, l, v) : exprime que la $i$ème variable dans une liste de variables $l = [x_1, ..., x_n]$ prenne la valeur v, i.e. $x_i = v$.
- *cumulative*(S, D, R, l, e) : Soient n tâches. La tâche j commence à S_j de durée D_j et a besoin de R_j unités d'une ressource donnée. Elle impose que les tâches doivent être exécutées sans dépasser l unités et la fin du temps e.
- *sort*$([x_1, ..., x_n], [y_1, ..., y_n])$: exprime que $(y_1, ..., y_n)$ est obtenu à partir de $(x_1, ..., x_n)$ en triant les éléments par ordre croissant.
- *gcc*(X, V, l, u) : la cardinalité d'une valeur V_i doit être dans $[l_i, u_i]$ parmi les variables X.
- ...

Contraintes globales

- Le calcul de la projection $\pi_{j,k}(\mathcal{D}) \neq D_k$ est de complexité exponentielle d^n où d est la cardinalité *max* d'un domaine.
- Si c_i est d'arité 2, dite contrainte binaire, $\pi_{j,k}(\mathcal{D})$ est polynomiale.
- Comment faire avec les contraintes n-aires ?

Contraintes globales

- Le calcul de la projection $\pi_{j,k}(\mathcal{D}) \neq D_k$ est de complexité exponentielle d^n où d est la cardinalité *max* d'un domaine.
- Si c_i est d'arité 2, dite contrainte binaire, $\pi_{j,k}(\mathcal{D})$ est polynomiale.
- Comment faire avec les contraintes n-aires ?
  - La projection est exponentielle dans le cas général.

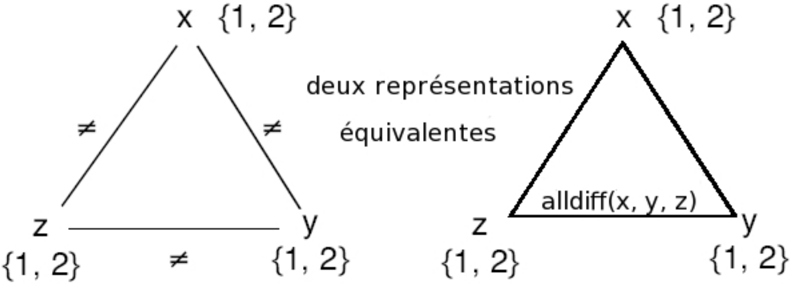
Contraintes globales

- Le calcul de la projection $\pi_{j,k}(\mathcal{D}) \neq D_k$ est de complexité exponentielle d^n où d est la cardinalité *max* d'un domaine.
- Si c_i est d'arité 2, dite contrainte binaire, $\pi_{j,k}(\mathcal{D})$ est polynomiale.
- Comment faire avec les contraintes n-aires ?
  - La projection est exponentielle dans le cas général.
  - La binarisation des contraintes n-aires ne filtre pas assez !

Contraintes globales

- Le calcul de la projection $\pi_{j,k}(\mathcal{D}) \neq D_k$ est de complexité exponentielle d^n où d est la cardinalité *max* d'un domaine.
- Si c_i est d'arité 2, dite contrainte binaire, $\pi_{j,k}(\mathcal{D})$ est polynomiale.
- Comment faire avec les contraintes n-aires ?
  - La projection est exponentielle dans le cas général.
  - La binarisation des contraintes n-aires ne filtre pas assez !
  - Idée : Proposer une projection dédiée à chaque contrainte ! Terrain d'intégration des techniques RO

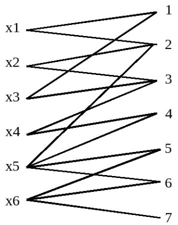
Consistance partielle : faiblesse binarisation



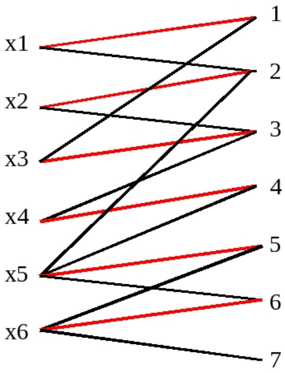
Filtrage - alldifferent - Contrainte de différence

La contrainte *alldif* impose que les valeurs prises par un ensemble de variables soient différentes deux à deux. Soit un CSP

$$\begin{aligned} \mathcal{X} = \quad & \{x_1, ..., x_6\} \\ & D_{x_1} = \{1, 2\}, ..., D_{x_6} = \{5, 6, 7\} \\ C_1 : \quad & x_1 \neq x_2 \neq x_3 \neq x_4 \neq x_5 \neq x_6 \end{aligned}$$



Filtrage - alldifferent - couplage

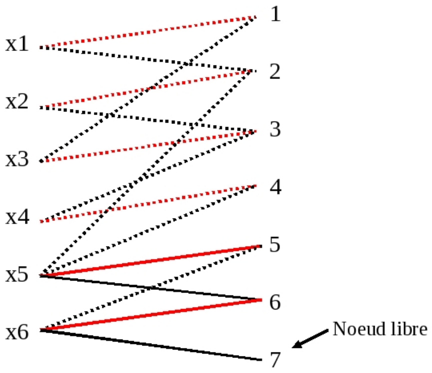


Filtrage - alldifferent - exploitation du couplage

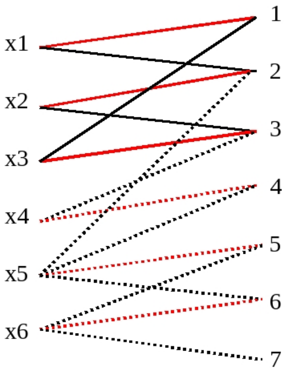
Propriété de Berge

Une arête appartient à des couplages maximum mais non à tous si et seulement si pour un couplage maximum arbitraire M, cette arête appartient soit à une chaîne alternée paire qui commence à un noeud libre, soit à un cycle alterné pair.

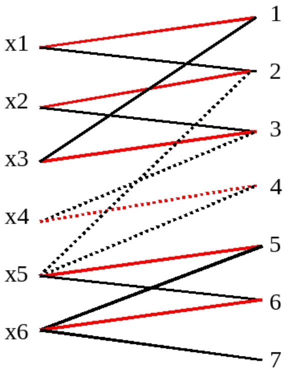
Filtrage - alldifferent - chaîne alternée paire



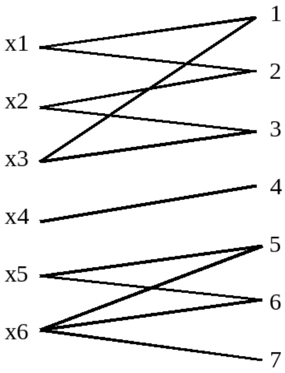
Filtrage - alldifferent - cycle alterné pair



Filtrage - alldifferent - arêtes consistantes



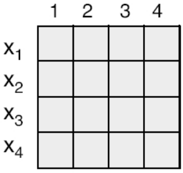
Filtrage - alldifferent



Résolution : exemple

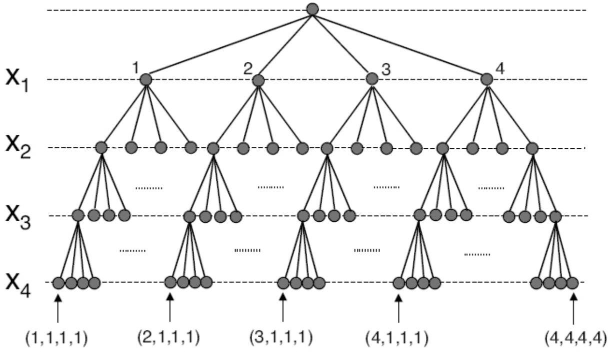
Soit le problème des quatres reines.

Soit $n = 4$ la taille de l'échiquier
$\mathcal{X} = \{x_1, ..., x_n\}$
$\mathcal{D} = \{D_1, ..., D_n\}, avec D_1 = ... D_N = [1, N]$
$\mathcal{C} = \{C_1, C_2\}$
$C_1 : x_1 \neq x_2 \neq ... \neq x_n$
$C_2 : |x_i - x_j| \neq |i - j|, 1 \leq i < j \leq n$



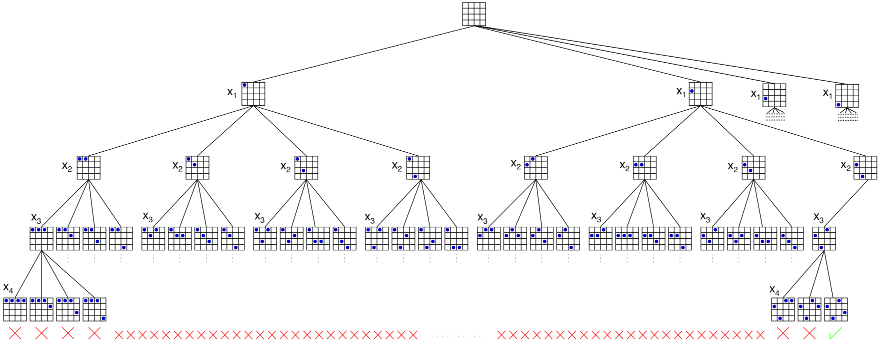
Résolution : espace de recherche

- Espace de recherche



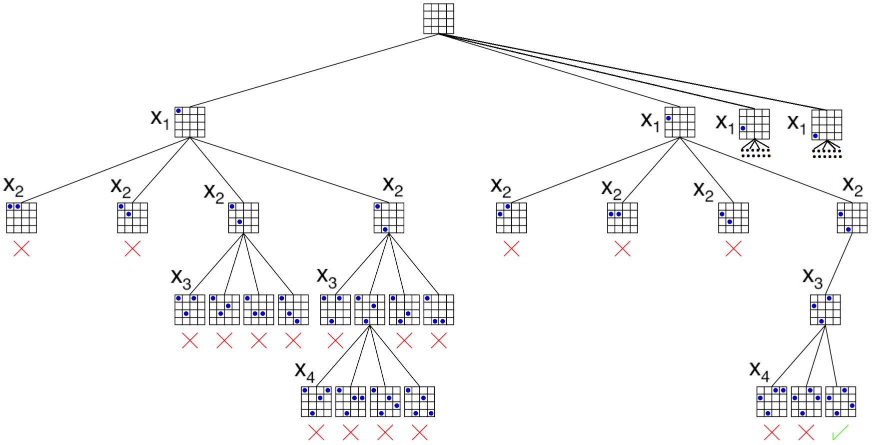
Résolution : recherche naïve GT (generate and test)

- GT (generate and test) : générer toutes les combinaisons



Résolution : recherche incrémentale BT (Backtrack)

- BT (Backtrack) : vérifier incrémentalement les contraintes



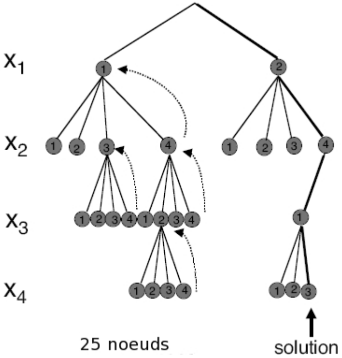
Résolution : recherche incrémentale BT (Backtrack)

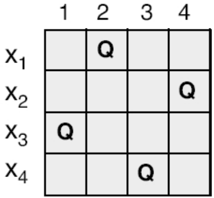
- Backtrack : un algorithme sans filtrage, décomposition par énumération, et exploration par retour arrière.

Procédure Backtrack(in I)
$k := |I| + 1$
for $v_k \in D_k$ **do**
    $Iloc := I \cup \{x_k \rightarrow d_k\}$
    if *Iloc* consistant **then**
        if $k = n$ **then**
            afficher la solution *Iloc*
        else
            Backtrack(*Iloc*)
        end if
    end if
end for

Résolution : l'algorithme simple Backtrack

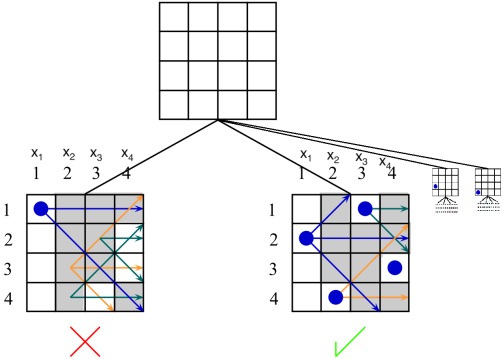
- Comportement de Backtrack





Résolution : recherche incrémentale avec filtrage (BP/MAC)

- BP (Branch & Prune) : filtrage avec les contraintes



Résolution : recherche incrémentale avec filtrage (BP/MAC)

- BP : filtrage+énumération+backtrack.
- MAC : Maintaining Arc-Consistency (AC3).

```
Procédure Backtrack(in I)
k := |I| + 1
for v_k ∈ D_k do
    Iloc := I ∪ {x_k → d_k}
    Filtering(Iloc) /
    if Iloc consistant then
        if k = n then
            afficher la solution Iloc
        else
            Backtrack(Iloc)
        end if
    end if
end for
```

Résolution : l'algorithme simple Backtrack

Résolution : l'algorithme simple Backtrack

- L'algorithme Backtrack représente le comportement le plus naïf en $O(d^n)$ d'un algorithme de résolution.
- Améliorations :
  - Filtrage : réduction des domaines

Résolution : l'algorithme simple Backtrack

- L'algorithme Backtrack représente le comportement le plus naïf en $O(d^n)$ d'un algorithme de résolution.
- Améliorations :
  - Filtrage : réduction des domaines
  - Décomposition : heuristiques de choix des variables

Résolution : l'algorithme simple Backtrack

- L'algorithme Backtrack représente le comportement le plus naïf en $O(d^n)$ d'un algorithme de résolution.
- Améliorations :
  - Filtrage : réduction des domaines
  - Décomposition : heuristiques de choix des variables
    - mindomain : Choisir la variable avec le plus petit domaine
    - degree : Choisir la variable impliquée dans le plus de contraintes
    - combination : minimum des (mindomain/degree)

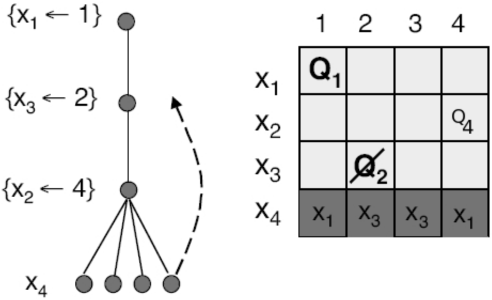
Résolution : l'algorithme simple Backtrack

- L'algorithme Backtrack représente le comportement le plus naïf en $O(d^n)$ d'un algorithme de résolution.
- Améliorations :
  - Filtrage : réduction des domaines
  - Décomposition : heuristiques de choix des variables
    - mindomain : Choisir la variable avec le plus petit domaine
    - degree : Choisir la variable impliquée dans le plus de contraintes
    - combination : minimum des (mindomain/degree)
  - Exploration :
    - "backtrack intelligent" pour remonter vers les noeuds parents responsables de l'échec,

Résolution : l'algorithme simple Backtrack

- L'algorithme Backtrack représente le comportement le plus naïf en $O(d^n)$ d'un algorithme de résolution.
- Améliorations :
  - Filtrage : réduction des domaines
  - Décomposition : heuristiques de choix des variables
    - mindomain : Choisir la variable avec le plus petit domaine
    - degree : Choisir la variable impliquée dans le plus de contraintes
    - combination : minimum des (mindomain/degree)
  - Exploration :
    - "backtrack intelligent" pour remonter vers les noeuds parents responsables de l'échec,
    - "branch& bound" en présence d'une fonction objectif.

Exploration : Backtrack intelligent



Exploration : Thrashing

- Thrashing : le même échec peut être redécouvert plusieurs fois, ... un nombre exponentiel de fois, ...
- Idée : remonter plus haut dans l'arbre de recherche
  - Backjumping
  - Dynamic Backtracking

Programmation par contraintes versus Programmation en nombres entiers/RO

- Programmation en Nombres Entiers (PNE) : Branch & Bound (Séparation & Evaluation)
  - Branch : énumération du domaine des variables
  - Bound : évaluation d'une borne en considérant (relaxation) toutes les contraintes

Programmation par contraintes versus Programmation en nombres entiers/RO

- Programmation en Nombres Entiers (PNE) : Branch & Bound (Séparation & Evaluation)
  - Branch : énumération du domaine des variables
  - Bound : évaluation d'une borne en considérant (relaxation) toutes les contraintes
- Programmation Par Contraintes (PPC) : Branch & Prune (Séparation & Filtrage)
  - Branch : énumération du domaine des variables
  - Filtrage : réduction du domaine des variables en considérant chaque contrainte individuelle

Programmation par contraintes versus Programmation en nombres entiers/RO

- Programmation en Nombres Entiers (PNE) : Branch & Bound (Séparation & Evaluation)
  - Branch : énumération du domaine des variables
  - Bound : évaluation d'une borne en considérant (relaxation) toutes les contraintes
- Programmation Par Contraintes (PPC) : Branch & Prune (Séparation & Filtrage)
  - Branch : énumération du domaine des variables
  - Filtrage : réduction du domaine des variables en considérant chaque contrainte individuelle
- PNE : efficacité de "bound" servie par la rigidité du modèle

Programmation par contraintes versus Programmation en nombres entiers/RO

- Programmation en Nombres Entiers (PNE) : Branch & Bound (Séparation & Evaluation)
  - Branch : énumération du domaine des variables
  - Bound : évaluation d'une borne en considérant (relaxation) toutes les contraintes
- Programmation Par Contraintes (PPC) : Branch & Prune (Séparation & Filtrage)
  - Branch : énumération du domaine des variables
  - Filtrage : réduction du domaine des variables en considérant chaque contrainte individuelle
- PNE : efficacité de "bound" servie par la rigidité du modèle
- PPC : efficacité du "filtrage" versus flexibilité du modèle