Programmation par Contraintes

1 - Notion de contrainte

Une contrainte est une relation logique établie entre différentes variables, chacune prenant sa valeur dans un ensemble qui lui est propre, appelé domaine.

Une contrainte sur un ensemble de variables restreint les valeurs que peuvent prendre simultanément ses variables. Elle est déclarative et relationnelle puisqu'elle définit une relation entre les variables sans spécifier de procédure opérationnelle pour assurer cette relation.

Ainsi, lorsqu'on pose la contrainte « $2x_1 - x_2 = x_3$ », on ne s'occupe pas de donner un algorithme permettant de la résoudre. La connaissance de x_1 et x_2 nous donne la valeur de x_3, de même la connaissance de x_2 et x_3 nous donne la valeur de x_1, et même encore la connaissance de x_1 et x_3 nous donne la valeur de x_2.

Les contraintes varient en fonction des domaines de valeurs des variables et peuvent être définies en extension ou en intension :

- Pour définir une contrainte en extension, on énumère les tuples de valeurs appartenant à la relation. Par exemple, si les domaines des variables *x* et *y* contiennent les valeurs *0, 1* et *2*, alors on peut définir la contrainte « *x est plus petit que y* » en extension par « $(x,y) \in \{(0,1), (0,2), (1,2)\}$ »

- Une contrainte peut également être définie en intension, on utilise des propriétés mathématiques connues. Par exemple : « $x+y \leq 4$ » ou encore « $x \leq 1 \Rightarrow y \neq 2$ » ou même par un prédicat portant sur *n* variables « allDifferent (x_1, …, x_n) », appelée contrainte globale.

1.1 - Arité d'une contrainte

L'arité d'une contrainte est le nombre de variables sur les quelles elle porte. On dira que la contrainte est :

- ***unaire*** si son arité est égale à 1, elle porte sur une seule variable, par exemple « $x*x = 4$ » ou encore « *est-un-triangle(y)* »

- ***binaire*** si son arité est égale à 2, elle met en relation 2 variables), par exemple « $x \neq y$ » ou encore « *A U B = C* »

- ***ternaire*** si son arité est égale à 3 (elle met en relation 3 variables, par exemple « $x+y < 3*z-4$ » ou encore « *(non x) ou y ou z = vrai* »

- ...

- ***n-aire*** si son arité est égale à *n*, elle met en relation un ensemble de *n* variables. Si la contrainte porte sur toutes les variables on dira que la contrainte est globale.

1.2 - Binarisation des contraintes

D'une façon générale, les contraintes d'un *CSP* peuvent être *n-aires*, mais on peut toujours les exprimer en termes de contraintes binaires. Ainsi tout *CSP* peut se représenter comme un *CSP* binaire. Un tel *CSP* peut être représenté comme un graphe de contraintes

2 - Problèmes de satisfaction de contraintes

Un CSP (Problème de Satisfaction de Contraintes) est un problème modélisé sous la forme d'un ensemble de contraintes posées sur des variables, chacune de ces variables prenant ses valeurs dans un domaine. De façon plus formelle, on définira un CSP par un triplet *(X, D, C)* tel que :

- $X = \{X_1, X_2, ..., X_n\}$ est l'ensemble des variables (les inconnues) du problème ;

- *D* est la fonction qui associe à chaque variable X_i son domaine $D(X_i)$, c'est-à-dire l'ensemble des valeurs que peut prendre X_i.

- $C = \{C_1, C_2, ..., C_k\}$ est l'ensemble des contraintes. Chaque contrainte C_j est une relation entre certaines variables de *X*, restreignant les valeurs que peuvent prendre simultanément ces variables. Par exemple, on peut définir le CSP *(X, D, C)* suivant :

  - ✓ $X = \{a, b, c, d\}$
  - ✓ $D(a) = D(b) = D(c) = D(d) = \{0, 1\}$
  - ✓ $C = \{a \neq b, c \neq d, a+c < b\}$

  Ce CSP comporte 4 variables *a*, *b*, *c* et *d*, chacune pouvant prendre 2 valeurs (*0* ou *1*). Ces variables doivent respecter les contraintes suivantes : *a* doit être différente de *b*, *c* doit être différente de *d* et la somme de *a* et *c* doit être inférieure à *b*.

2.1 - Solution d'un CSP

Etant donné un CSP *(X, D, C)*, sa résolution consiste à affecter des valeurs aux variables, de telle sorte que toutes les contraintes soient satisfaites. On introduit pour cela les notations et définitions suivantes :

- On appelle ***affectation*** le fait d'instancier certaines variables par des valeurs prises dans leurs domaines. On notera $A = \{ (X_1, V_1), (X_2, V_2), ..., (X_r, V_r) \}$ l'affectation qui instancie la variable X_1 par la valeur V_1, la variable X_2 par la valeur V_2, ..., et la variable X_r par la valeur V_r.

- Une affectation est dite ***totale*** si elle instancie toutes les variables du problème, elle est dite ***partielle*** si elle n'en instancie qu'une partie.

- Une affectation *A* ***viole*** une contrainte C_k si toutes les variables de C_k sont instanciées dans *A*, et si la relation définie par C_k n'est pas vérifiée pour les valeurs des variables de C_k définies dans *A*.

- Une affectation (totale ou partielle) est ***consistante*** si elle ne viole aucune contrainte, et ***inconsistante*** si elle viole une ou plusieurs contraintes.

- Une ***solution*** est une affectation totale consistante, c'est-à-dire une instanciation de toutes les variables du problème qui ne viole aucune contrainte.

Exemple

$X = \{X_1, X_2, X_3, X_4\}$
$D = \{D_1, D_2, D_3, D_4\}$ avec $D_1 = D_2 = D_3 = D_4 = \{0, 1\}$
$C = \{X_1 \neq X_2, X_3 \neq X_4, X_1 + X_3 < X_2\}$

Une affectation partielle : $A = \{(X_1, 0), (X_2, 0)\}$
Une affectation totale : $A = \{(X_1, 0), (X_2, 0), (X_3, 0), (X_4, 0)\}$
Une affectation consistante : $A = \{(X_3, 0), (X_4, 1)\}$
Une affectation inconsistante : $A = \{(X_1, 0), (X_2, 0)\}$
Une solution : $A = \{(X_1, 0), (X_2, 1), (X_3, 0), (X_4, 1)\}$

2.2 - CSP surcontraint ou souscontraint

Lorsqu'un CSP n'a pas de solution, on dit qu'il est *surcontraint* : il y a trop de contraintes et on ne peut pas toutes les satisfaire. Dans ce cas, on peut souhaiter trouver l'affectation totale qui viole le moins de contraintes possibles. Un tel CSP est appelé *max-CSP* (max car on cherche à maximiser le nombre de contraintes satisfaites).

Une autre possibilité est d'affecter un poids à chaque contrainte (une valeur proportionnelle à l'importance de cette contrainte, et de chercher l'affectation totale qui minimise la somme des poids des contraintes violées. Un tel CSP est appelé *CSP valué* (*VCSP*).

Il existe encore d'autre types de CSPs, appelés *CSPs basés sur les semi-anneaux* (semiring based CSPs), permettant de définir plus finement des préférences entre les contraintes.

Inversement, lorsqu'un CSP admet beaucoup de solutions différentes, on dit qu'il est *sous-contraint*. Si les différentes solutions ne sont pas toutes équivalentes, dans le sens où certaines sont mieux que d'autres, on peut exprimer des préférences entre les différentes solutions. Pour cela, on ajoute une fonction qui associe une valeur numérique à chaque solution, valeur dépendante de la qualité de cette solution. L'objectif est alors de trouver la solution du CSP qui maximise cette fonction. Un tel CSP est appelé *CSOP* (Constraint Satisfaction Optimisation Problem).

3 - Exemple

3.1 - Description du problème

Il s'agit de placer 4 reines sur un échiquier comportant 4 lignes et 4 colonnes, de manière à ce qu'aucune reine ne soit en prise. On rappelle que 2 reines sont en prise si elles se trouvent sur une même diagonale, une même ligne ou une même colonne de l'échiquier.

3.2 - Modélisation sous la forme d'un CSP

Pour modéliser un problème sous la forme d'un CSP, il faut identifier :

- ✓ L'ensemble des variables *X* (les inconnues du problème),

- ✓ La fonction *D* qui associe à chaque variable de *X* son domaine (les valeurs que la variable peut prendre),
- ✓ Les contraintes *C* entre les variables.

Notons qu'à ce niveau, on cherche simplement une spécification formelle du problème, sans savoir comment le résoudre. Un même problème peut généralement être modélisé par différents CSPs. Le choix d'une modélisation peut avoir une influence sur l'efficacité de la résolution.

Première modélisation

Les "inconnues" du problème sont les positions des reines sur l'échiquier. En numérotant les lignes et les colonnes de l'échiquier de la façon suivante :

On peut déterminer la position d'une reine par un numéro de ligne et un numéro de colonne. Ainsi, une première modélisation consiste à associer à chaque reine *i* deux variables *Li* et *Ci* correspondant respectivement à la ligne et la colonne sur laquelle placer la reine. Les contraintes spécifient alors que les reines doivent être sur des lignes différentes, des colonnes différentes et des diagonales différentes. Notons pour cela que lorsque 2 reines sont sur une même diagonale montante, la somme de leurs numéros de ligne et de colonne est égale, tandis que lorsqu'elles sont sur une même diagonale descendante, la différence de leurs numéros de ligne et de colonne est égale. On en déduit le CSP suivant :

- Variables :
  $X = \{L_1, L_2, L_3, L_4, C_1, C_2, C_3, C_4\}$
- Domaines :
  $D(L_1) = D(L_2) = D(L_3) = D(L_4) = D(C_1) = D(C_2) = D(C_3) = D(C_4) = \{1, 2, 3, 4\}$
- Contraintes : on identifie 4 types de contraintes
  - Les reines doivent être sur des lignes différentes :
    $C_{lig} = \{L_1 \neq L_2,\ L_1 \neq L_3,\ L_1 \neq L_4,\ L_2 \neq L_3,\ L_2 \neq L_4,\ L_3 \neq L_4\}$
  - Les reines doivent être sur des colonnes différentes :
    $C_{col} = \{C_1 \neq C_2, C_1 \neq C_3, C_1 \neq C_4, C_2 \neq C_3, C_2 \neq C_4, C_3 \neq C_4\}$
  - Les reines doivent être sur des diagonales montantes différentes :
    $C_{dm} = \{C_1 + L_1 \neq C_2 + L_2,\ C_1 + L_1 \neq C_3 + L_3,\ C_1 + L_1 \neq C_4 + L_4,$
    $C_2 + L_2 \neq C_3 + L_3, C_2 + L_2 \neq C_4 + L_4, C_3 + L_3 \neq C_4 + L_4\}$
  - Les reines doivent être sur des diagonales descendantes différentes :
    $C_{dd} = \{C_1 - L_1 \neq C_2 - L_2,\ C_1 - L_1 \neq C_3 - L_3,\ C_1 - L_1 \neq C_4 - L_4,$
    $C_2 - L_2 \neq C_3 - L_3,\ C_2 - L_2 \neq C_4 - L_4,\ C_3 - L_3 \neq C_4 - L_4\}$

L'ensemble des contraintes est défini par l'union de ces 4 ensembles :

$C = C_{lig} \cup C_{col} \cup C_{dm} \cup C_{dd}$

Les contraintes C_{lig} et C_{col} sont des contraintes binaires ; les contraintes C_{dm} et C_{dd} sont des contraintes quaternaires. L'énumération de toutes ces contraintes est ici un peu fastidieuse. On peut tout aussi bien les définir de la façon suivante :

- Contraintes :
  - les reines doivent être sur des lignes différentes :
    $C_{lig} = \{L_i \neq L_j / i \in \{1,2,3,4\}, j \in \{1,2,3,4\} \text{ et } i \neq j\}$
  - les reines doivent être sur des colonnes différentes :
    $C_{col} = \{C_i \neq C_j / i \in \{1,2,3,4\}, j \in \{1,2,3,4\} \text{ et } i \neq j\}$
  - les reines doivent être sur des diagonales montantes différentes :
    $C_{dm} = \{C_i+L_i \neq C_j+L_j / i \in \{1,2,3,4\}, j \in \{1,2,3,4\} \text{ et } i \neq j\}$
  - les reines doivent être sur des diagonales descendantes différentes :
    $C_{dd} = \{C_i-L_i \neq C_j-L_j / i \in \{1,2,3,4\}, j \in \{1,2,3,4\} \text{ et } i \neq j\}$

On aurait également pu utiliser une contrainte globale pour exprimer le fait que toutes les variables d'un ensemble doivent avoir des valeurs différentes :

$C_{lig} = toutesDiff(\{L_1, L_2, L_3, L_4\})$ et $C_{col} = toutesDiff(\{C_1, C_2, C_3, C_4\})$.

Une solution du problème des 4 reines, pour cette première modélisation, est
$A = \{(C_1, 1), (L_1, 2), (C_2, 2), (L_2, 4), (C_3, 3), (L_3, 1), (C_4, 4), (L_4, 3)\}$, autrement dit, la première reine est placée colonne 1 ligne 2, la deuxième, colonne 2 ligne 4, la troisième, colonne 3 ligne 1 et la quatrième, colonne 4 ligne 3.

Deuxième modélisation

Dans la mesure où l'on sait dès le départ qu'il y aura une reine et une seule sur chaque colonne de l'échiquier, le problème peut se résumer à déterminer sur quelle ligne se trouve la reine placée sur la colonne *i*. Par conséquent, une deuxième modélisation consiste à associer une variable X_i à chaque colonne *i* de telle sorte que X_i désigne le numéro de ligne sur laquelle placer la reine de la colonne *i*. Notons que pour cette deuxième modélisation, on a été obligé de « réfléchir » un peu pour introduire dans la modélisation une déduction (il y a une seule reine par colonne) qui, on l'espère, va faciliter le travail de la machine. Le CSP correspondant à cette deuxième modélisation est le suivant :

- Variables :
  $X = \{X_1, X_2, X_3, X_4\}$
- Domaines :
  $D(X_1) = D(X_2) = D(X_3) = D(X_4) = \{1, 2, 3, 4\}$
- Contraintes :
  - les reines doivent être sur des lignes différentes :
    $C_{lig} = \{X_i \neq X_j / i \in \{1,2,3,4\}, j \in \{1,2,3,4\} \text{ et } i \neq j\}$
  - les reines doivent être sur des diagonales montantes différentes :
    $C_{dm} = \{X_i + i \neq X_j + j / i \in \{1,2,3,4\}, j \in \{1,2,3,4\} \text{ et } i \neq j\}$
  - les reines doivent être sur des diagonales descendantes différentes :
    $C_{dd} = \{X_i - i \neq X_j - j / i \in \{1,2,3,4\}, j \in \{1,2,3,4\} \text{ et } i \neq j\}$

L'ensemble des contraintes est défini par l'union de ces 3 ensembles :
$C = C_{lig} \cup C_{dm} \cup C_{dd}$

Une solution du problème des 4 reines, pour cette deuxième modélisation, est :
$A = \{(X_1, 2), (X_2, 4), (X_3, 1), (X_4, 3)\}$

C'est-à-dire, la reine de la colonne 1 est placée sur la ligne 2, celle de la colonne 2, ligne 4, celle de la colonne 3, ligne 1 et celle de la colonne 4, ligne 3.

Troisième modélisation

Une autre façon, radicalement opposée, de modéliser le problème consiste à choisir comme variables non pas les positions des reines, mais les états des cases de l'échiquier : on associe une variable à chacune des 16 cases de l'échiquier (on notera X_{ij} la variable associée à la case située ligne *i* et colonne *j*), chaque variable peut prendre pour valeur *0* (absence de reine sur la case) ou *1* (présence de reine sur la case), les contraintes spécifient qu'il ne peut y avoir plusieurs reines sur une même ligne, une même colonne ou une même diagonale. Le CSP correspondant à cette troisième modélisation est le suivant :

- Variables :
  $X = \{X_{11}, X_{12}, X_{13}, X_{14}, X_{21}, X_{22}, X_{23}, X_{24}, X_{31}, X_{32}, X_{33}, X_{34}, X_{41}, X_{42}, X_{43}, X_{44}\}$
- Domaines :
  $D(X_{ij}) = \{0, 1\}$ *pour tout i et tout j compris entre 1 et 4*
- Contraintes :
  - Il y a une reine par ligne
    $C_{lig} = \{X_{i1} + X_{i2} + X_{i3} + X_{i4} = 1 / i \in \{1, 2, 3,4\}\}$
  - Il y a une reine par colonne
    $C_{col} = \{X_{1i} + X_{2i} + X_{3i} + X_{4i} = 1 / i \in \{1, 2, 3,4\}\}$
  - Les reines doivent être sur des diagonales montantes différentes
    C_{dm} = *pour tout couple de variables différentes* X_{ij} *et* X_{kl}, $i + j = k+l \Rightarrow X_{ij} + X_{kl} \leq 1$
  - Les reines doivent être sur des diagonales descendantes différentes
    C_{dd} = *pour tout couple de variables différentes* X_{ij} *et* X_{kl}, $i - j=k - l \Rightarrow X_{ij} + X_{kl} \leq 1$

  L'ensemble des contraintes est défini par l'union de ces 4 ensembles
  $C = C_{lig} \cup C_{col} \cup C_{dm} \cup C_{dd}$

Une solution du problème des 4 reines, pour cette troisième modélisation, est :
$A = \{(X_{11}, 0), (X_{12}, 1), (X_{13}, 0), (X_{14}, 0), (X_{21}, 0), (X_{22}, 0), (X_{23}, 0), (X_{24}, 1), (X_{31}, 1), (X_{32}, 0), (X_{33}, 0), (X_{34}, 0), (X_{41}, 0), (X_{42}, 0), (X_{43}, 1), (X_{44}, 0)\}$

C'est-à-dire, la case ligne 1 colonne 1 (X_{11}) est vide, la case ligne 1 colonne 2 (X_{12}) est occupée, ...

Choix d'une modélisation

La question (légitime) que l'on peut maintenant se poser est la suivante: « Quelle est la meilleure modélisation ? »
Pour répondre à cette question on peut envisager plusieurs cas :

1. Celle qui modélise le mieux la réalité du problème. De ce point de vue, les 3 modélisations sont équivalentes.

2. Celle qui est la plus facile à trouver. De ce point de vue, la première modélisation est probablement plus "simple"... même si cela est subjectif !
3. Celle qui permettra de résoudre le problème le plus efficacement. On ne peut vraiment répondre à cette question qu'à partir du moment où l'on sait comment un CSP est résolu. Intuitivement, on se doute que la deuxième modélisation devrait être meilleure que la première dans la mesure où elle prend en compte le fait que les reines sont sur des colonnes différentes par la définition même des variables, sans avoir à poser de contrainte. On verra que « l'espace de recherche » de cette deuxième modélisation est plus petit que celui de la première.

Généralisation à n reines

On peut généraliser le problème au placement de *n* reines sur un échiquier comportant *n* colonnes et *n* lignes. Par exemple, la deuxième modélisation devient :

- Variables :
  $X = \{X_i / i$ est un entier compris entre 1 et $n\}$
- Domaines :
  Quelque soit $X_i \in X$, $D(X_i) = \{j / j$ est un entier compris entre 1 et $n\}$
- Contraintes :
  - les reines doivent être sur des lignes différentes :
    $C_{lig} = \{X_i \neq X_j / i$ *et* j sont 2 entiers différents compris entre 1 et $n\}$
  - les reines doivent être sur des diagonales montantes différentes :
    $C_{dm} = \{X_i + i \neq X_j + j / i$ *et* j sont 2 entiers différents compris entre 1 et $n\}$
  - les reines doivent être sur des diagonales descendantes différentes :
    $C_{dd} = \{X_i - i \neq X_j - j / i$ *et* j sont 2 entiers différents compris entre 1 et $n\}$

L'ensemble des contraintes est défini par l'union de ces 3 ensembles :
$C = C_{lig} \cup C_{dm} \cup C_{dd}$

4 - Résolution des CSPs

Après la phase de modélisation, on va maintenant étudier quelques algorithmes permettant de résoudre, de façon générique, certains de ces CSPs. On se restreindra aux *CSPs sur les domaines finis*, c'est-à-dire, les CSPs dont les domaines des variables sont des ensembles finis de valeurs. Le point commun à tous les algorithmes que nous allons étudier est d'explorer méthodiquement l'ensemble des affectations possibles jusqu'à, soit trouver une solution (quand le CSP est consistant), soit démontrer qu'il n'existe pas de solution (quand le CSP est inconsistant).

4.1 - L'algorithme « génère et teste »

- **Principe de l'algorithme**

Il s'agit d'une recherche systématique d'une solution :
- ✓ Génération d'une affectation totale
- ✓ Test de la satisfaction de toutes les contraintes

La façon la plus simple (très naïve !) de résoudre un CSP sur les domaines finis consiste à énumérer toutes les affectations totales possibles jusqu'à en trouver une qui satisfasse toutes les contraintes.

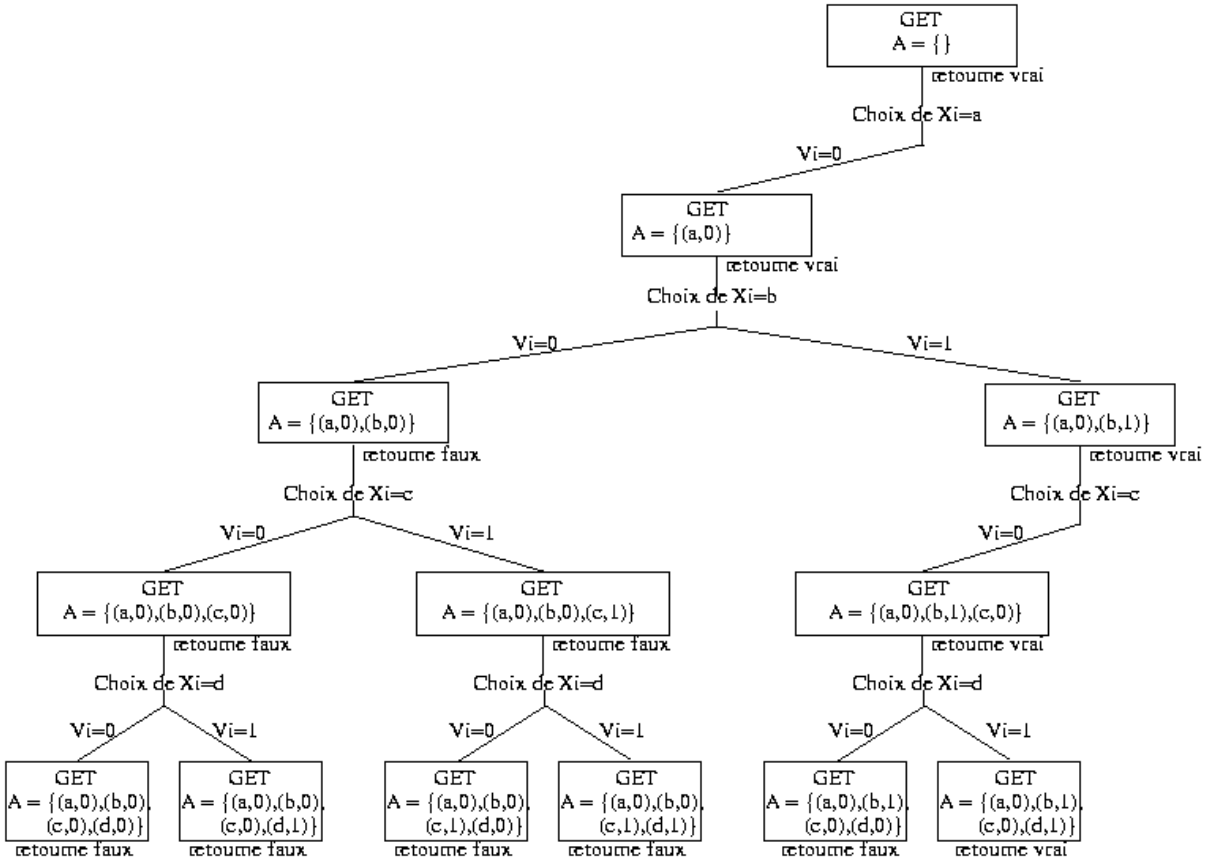
Ce principe est repris dans la fonction récursive « G*enereEtTeste(A, (X, D, C))* » décrite ci-dessous. Dans cette fonction, *A* contient une affectation partielle et *(X, D, C)* décrit le CSP à résoudre. Au départ, l'affectation partielle *A* sera vide, la fonction retourne *vrai* si on peut étendre l'affectation partielle *A* en une affectation totale consistante (une solution) et *faux* sinon.

Fonction GenereEtTeste(*A, (X, D, C)*) : **Booléen**
/* *(X, D, C)* = un *CSP* sur les domaines finis */
/* *A* = une affectation partielle pour *(X, D, C)* */
Début
Si (toutes les variables de *X se trouvent dans A*) **Alors** */* A est une affectation totale*/*
Si (*A* est consistante) **Alors** /* *A est une solution* */
Retourner *vrai*
Sinon
Retourner *faux*
Fin Si
Sinon **/*** *A est une affectation partielle* */*
Choisir une variable X_i de *X*, non affectée à une valeur dans *A*
Pour toute valeur $V_j \in D(X_i)$ ***Faire***
Si (GénèreEtTeste($A \cup (X_i , V_j)$, *(X, D, C))* = vrai) **Alors**
Retourner *vrai*
Fin Si
Fin Pour
Retourner *faux*
Fin Si
Fin

- **Exemple de trace d'exécution de « GenereEtTeste»**

Considérons par exemple le *CSP (X, D, C)* suivant :
- ✓ $X = \{a, b, c, d\}$
- ✓ $D(a) = D(b) = D(c) = D(d) = \{0, 1\}$
- ✓ $C = \{a \neq b, c \neq d, a + c < b\}$

L'enchainement des appels successifs à la fonction GenereEtTeste (abrégée par GET) est représenté ci-dessous (chaque rectangle correspond à un appel de la fonction, et précise la valeur de l'affectation partielle en cours de construction *A*).



4.2 - Espace de recherche d'un CSP

On appelle « *Espace de recherche d'un CSP* », L'ensemble de toutes les affectations complètes possibles.

Pour un *CSP (X, D, C)* où $X = \{X_1, X_2, ..., X_n\}$ et $D = \{D(X_1), D(X_2), ..., D(X_n)\}$, la taille de l'espace de recherche est défini par :

$|E| = |D(X_1)| * |D(X_2)| * ... * |D(X_n)|$

Ainsi, si tous les domaines des variables sont de taille k ($|D(Xi)| = k$), alors la taille de l'espace de recherche devient : $|E| = k^n$, Donc, le nombre d'affectations que « GenereEtTeste» génère croit de façon exponentielle en fonction du nombre de variables du problème. Dans le pire des cas, si *n* est grand c'est l'explosion combinatoire !,

- ***Quelques idées pour améliorer « génère et teste »***

Dans le cas où le nombre de variables est élevé, il est déconseillé d'appliquer bêtement l'algorithme « génère et teste ». Il faut donc chercher à réduire au tant que possible l'espace de recherche:

- Ne développer que les affectations partielles consistantes : dès lors qu'une affectation partielle est inconsistante, il est inutile de chercher à l'étendre en une affectation totale puisque celle-ci sera nécessairement inconsistante.

- Réduire les tailles des domaines des variables en leur enlevant les valeurs « incompatibles » : pendant la génération d'affectations, on filtre le domaine des variables pour ne garder que les valeurs « localement consistantes » avec l'affectation en cours de construction, et dès lors que le domaine d'une variable devient vide, on arrête l'énumération pour cette affectation partielle.

- Introduire des « heuristiques » pour guider la recherche : lorsqu'on énumère les affectations possibles, on peut essayer d'énumérer en premier celles qui sont les plus « prometteuses », en espérant ainsi tomber rapidement sur une solution.

- Lors d'un échec, on peut essayer d'identifier la cause de l'échec (quelle est la variable qui viole une contrainte) pour ensuite « retourner en arrière » directement là où cette variable a été instanciée afin de remettre en cause plus rapidement la variable à l'origine de l'échec. C'est ce que l'on appelle le « retour arrière intelligent » (intelligent backtracking).

- Une autre approche particulièrement séduisante consiste à exploiter des connaissances sur les types de contraintes utilisées pour réduire l'espace de recherche. considérons par exemple le CSP (X, D, C) suivant :

  - *X={a, b, c},*
  - *D(a)=D(b)=D(c)={0,1,2,3,4, ..., 10000},*
  - *C={4*a - 2*b = 6*c + 3}*

  L'espace de recherche de ce CSP comporte 1000 milliard d'affectations. Pour résoudre ce CSP, on peut énumérer toutes ces combinaisons, en espérant de trouver une qui satisfasse la contrainte 4*a - 2*b = 6*c + 3. En revanche un simple raisonnement permet de conclure très rapidement que ce CSP n'a pas de solution. En effet, la partie gauche de l'équation donne toujours un nombre pair, et celle de la droite donne toujours un nombre impair !.

4.3 - L'algorithme « simple retour-arrière»

- **Principe de l'algorithme**

Une première idée d'améliorer l'algorithme « génère et teste» c'est de tester la consistance de l'affectation partielle au fur et à mesure de sa construction. En effet si une affectation partielle est inconsistante, il est impossible de trouver une solution dans cette branche. Dans ce cas, pour continuer la recherche on fait un « backtrack », c'est-à-dire un retour en arrière jusqu'à la plus récente instanciation partielle consistante.

Par exemple, sur la trace d'exécution de « GenereEtTeste», décrite ci-dessus, on remarque que l'algorithme génère toutes les affectations totales inconsistantes après l'affectation partielle $A = \{(a, 0), (b, 0)\}$ qui viole la contrainte $\{a \neq b\}$. L'algorithme « simple retour-arrière» va faire un « backtrack » pour choisir une autre valeur pour *b*.

Ce principe est repris dans la fonction récursive « SimpleRetourArrière(*A*, (*X*, *D*, *C*)) » décrite ci-dessous.

Fonction SimpleRetourArrière(*A*, (*X*, *D*, *C*)) : **Booléen**
Début
Si (*A* n'est pas consistante) **Alors**
Retourner *faux*
Fin Si
Si (toutes les variables de *X* sont instanciées dans *A*) **Alors**
/* *A est une affectation totale et consistante « une solution »* */
Retourner *vrai*
Sinon /* *A est une affectation partielle consistante* */
Choisir une variable X_i de *X* qui n'est pas encore instanciée dans *A*
Pour toute valeur $V_j \in D(X_i)$ **Faire**
Si (simpleRetourArrière($A \cup (X_i, V_j)$, (*X*, *D*, *C*)) = vrai) **Alors**
Retourner *vrai*
Fin Si
Fin Pour
Retourner *faux*
Fin Si
Fin

- **Exemple de trace d'exécution de « SimpleRetourArrière »**

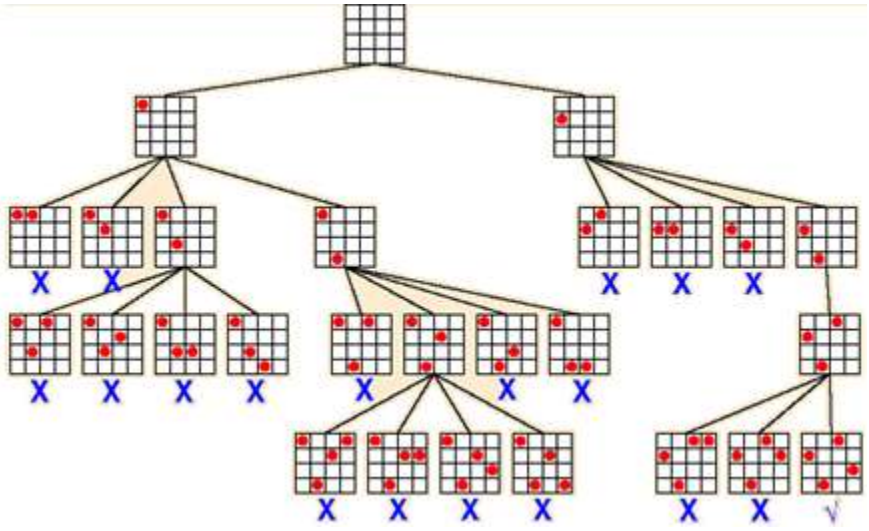
Reprenons le problème des 4 reines :

Variables : $X = \{X_1, X_2 X_3, X_4\}$

Domaines : $D(X_1) = D(X_2) = D(X_3) = D(X_4) = \{1, 2, 3, 4\}$

Contraintes : $C = \{X_i \neq X_j \mid i \in \{1, 2,3, 4\}, j \in \{1, 2,3, 4\}$ et $i \neq j\} \cup \{X_i + i \neq X_j + j \mid i \in \{1,2,3,4\}, j \in \{1,2,3,4\}$ et $i \neq j\} \cup \{X_i - i \neq X_j - j \mid i \in \{1,2,3,4\}, j \in \{1,2,3,4\}$ et $i \neq j\}$

L'enchainement des appels successifs à la fonction SimpleRetourArrière peut être représenté par l'arbre ci-dessous (chaque nœud correspond à un appel de la fonction, l'échiquier dessiné à chaque nœud décrit l'affectation partielle en cours)



4.4 - L'algorithme « anticipation»

- **Notions de filtrage et de consistance locale**

Pour améliorer l'algorithme « simple retour-arrière», dès qu'une nouvelle variable est instanciée, on va supprimer des domaines des variables non encore instanciées les valeurs qui ne sont pas compatibles avec la valeur choisie, il s'agit d'une anticipation.

Pour mettre ce principe en œuvre, on va, à chaque étape de la recherche, filtrer les domaines des variables non affectées en enlevant les valeurs « localement inconsistantes», c'est-à-dire celles dont on peut inférer qu'elles n'appartiendront à aucune solution.

Considérons un CSP *(X, D, C),* et une affectation partielle consistante *A.*

Le filtrage le plus simple consiste à anticiper d'une étape l'énumération: pour chaque variable X non affectée dans A, on enlève de $D(X)$ toute valeur v telle que l'affectation $A \cup \{(X, v)\}$ soit inconsistante.

Par exemple pour le problème des 4 reines, après avoir instancié X_1 à *1*, on peut enlever du domaine de X_2 la valeur *1* (qui viole la contrainte : $X_1 \neq X_2$) et la valeur *2* (qui viole la contrainte : $1 - X_1 \neq 2 - X_2$).

Un tel filtrage permet d'établir ce qu'on appelle la *consistance de nœud* **«** node-consistency »**,** appelée aussi «*1-consistance* **»**. Un CSP (*X, D, C*) est consistant de nœud si pour toute variable X_i de X, et pour toute valeur v de D_i, l'affectation partielle $\{(X_i, v)\}$ satisfait toutes les contraintes unaires de *C*.

Un autre filtrage plus fort, mais aussi plus long à effectuer, consiste à anticiper de deux étapes l'énumération: pour chaque variable X non affectée dans A, on enlève de $D(X)$ toute valeur v telle qu'il existe une variable X_j non affectée pour laquelle, pour toute valeur w de $D(X_j)$, l'affectation $A \cup \{(X, v), (X_j, w)\}$ soit inconsistante.

Par exemple pour le problème des 4 reines, après avoir instancié X_1 à *1*, on peut enlever la valeur *3* du domaine de X_2 car si $X_1 = 1$ et $X_2 = 3$, alors la variable X_3 ne peut plus prendre de valeurs : si $X_3 = 1$, on viole la contrainte $X_3 \neq X_1$, si $X_3 = 2$, on viole la contrainte $X_3 + 3 \neq X_2 + 2$, si $X_3 = 3$, on viole la contrainte $X_3 \neq X_2$, et si $X_3 = 4$, on viole la contrainte $X_3 - 3 \neq X_2 - 2$.

Ce filtrage permet d'établir ce qu'on appelle la *consistance d'arc* «arc-consistency», aussi appelée «*2-consistance»***.** Un CSP (*X, D, C*) est consistant d'arc si tout couple de variables (X_i, X_j) de X, et pour toute valeur v_i de D_i, il existe une valeur v_j appartenant D_j telle que l'affectation partielle $\{(X_i, v_i), (X_j, v_j)\}$ satisfasse toutes les contraintes binaires de *C*.

Un filtrage encore plus fort, mais aussi encore plus long à effectuer, consiste à anticiper de trois étapes l'énumération. Ce filtrage permet d'établir ce qu'on appelle la *consistance de chemin* « *path-consistency* », appelée aussi « *3-consistance* **»** et ainsi de suite. Notons que s'il reste k variables à affecter, et si l'on anticipe de k étapes l'énumération pour établir la « *k-consistance* », l'opération de filtrage revient à résoudre le CSP, c'est-à-dire que toutes les valeurs restant dans les domaines des variables après un tel filtrage appartiennent à une solution.

- **Principe de l'algorithme « anticipation»**

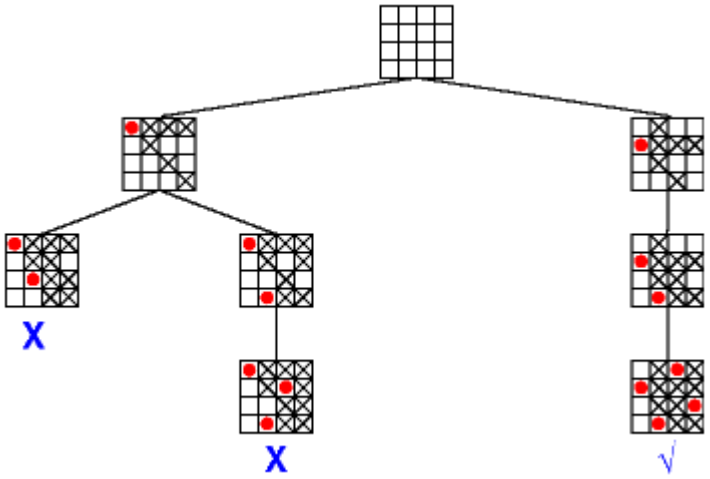
Le principe général de l'algorithme « anticipation» reprend celui de l'algorithme « simple retour-arrière», en ajoutant simplement une étape de filtrage à chaque fois qu'une valeur est affectée à une variable. Comme on vient de le voir, on peut effectuer différents filtrages plus ou moins forts, permettant d'établir différents niveaux de consistance locale (nœud, arc, chemin, . . .).

Ce principe de filtrage est repris dans la fonction récursive « anticipation (*A, (X, D, C*))» décrite ci-dessous.

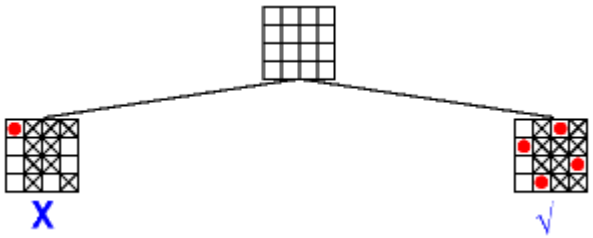
Fonction *Anticipation* (A, (X, D, C)) : **Booléen**
Début
Si (*A* n'est pas consistante) **Alors**
Retourner *faux*
Fin Si
Si toutes les variables de X sont affectées **Alors**
/* *A est une affectation totale consistante « une solution »* */
Retourner *vrai*
Sinon /* *A est une affectation partielle consistante* */
Choisir une variable X_i de X qui n'est pas encore affectée
Pour toute valeur $V_i \in D_i$ **Faire**
/* *filtrage des domaines par rapport à* $A \cup (X_i, V_i)$ */
Pour toute variable X_j de X qui n'est pas encore affectée **Faire**
$D'_j \leftarrow \{V_j \in D_j / A \cup \{(X_i, V_i), (X_j, V_j)\}$ est consistante$\}$
Si D'_j est vide **Alors**
Retourner *faux*
FinSi
Fin Pour
Si *Anticipation* ($A \cup \{(X_i, V_i)\}$, (X, D', C)) = *vrai* **Alors**
Retourner *vrai*
FinSi
Fin Pour
Retourner *faux*
Finsi
Fin

- **Exemple de trace d'exécution de « Anticipation»**

Considérons de nouveau le problème du placement de 4 reines sur un échiquier 4 x 4. L'enchainement des appels successifs à la fonction « Anticipation/nœud» peut être représenté par l'arbre ci-après (les valeurs supprimées par le filtrage sont marquées d'une croix).



Si on applique un filtrage plus fort, qui rétablit à chaque étape la consistance d'arc, l'enchainement des appels successifs à la fonction « Anticipation/arc» correspondante est représenté par l'arbre ci-après (les valeurs supprimées par le filtrage sont marquées d'une croix).



Ainsi, on constate sur le problème des 4 reines que le filtrage des domaines permet de réduire le nombre d'appels récursifs : on passe de **27** appels pour «simple retour-arrière» à **8** appels pour l'algorithme d'anticipation avec filtrage simple établissant une consistance de nœud. En utilisant des filtrages plus forts, comme celui qui établit la consistance d'arc, on peut encore réduire la combinatoire de **8** à **3** appels récursifs. Cependant, il faut noter que plus le filtrage utilisé est fort, plus cela prendra de temps pour l'exécuter.