Satisfaction Des Contraintes

L'intelligence

artificielle

By

Abdelkabir

Aarab

Écoles Nationales Supérieures d'Arts et Métiers
University Mohammed V de Rabat

Satisfaction Des Contraintes

Écoles Nationales Supérieures d'Arts et Métiers

University Mohammed V de Rabat

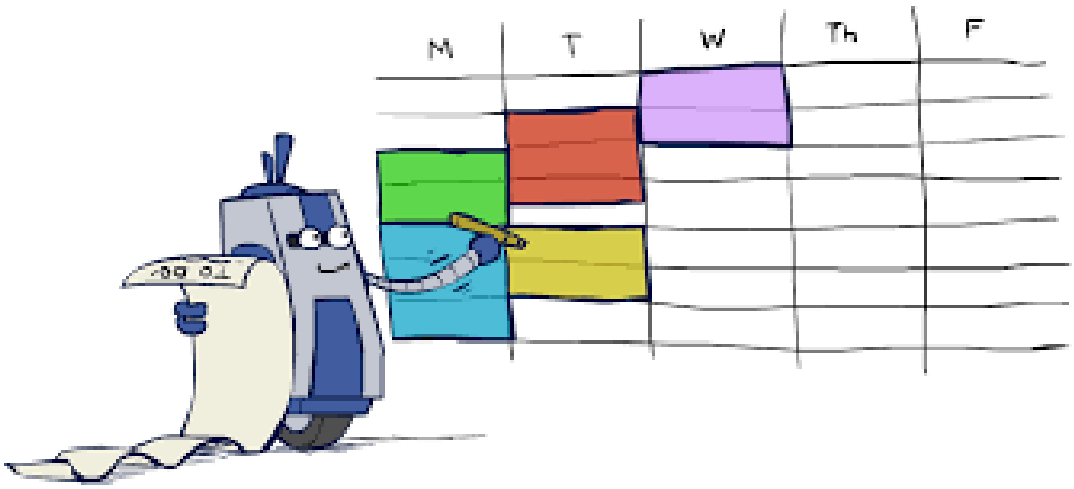
L'intelligence

artificielle

Problème de satisfaction

de

contraintes (CSP)



By

Abdelkabir

Aarab

Encadré par le Professeur :

Nour eddine Joudar

Filiére :

INDIA

Écoles Nationales Supérieures d'Arts et Métiers

University Mohammed V de Rabat

Résumé

Le formalisme des problèmes de satisfaction de contraintes (CSP) permet de repré- senter sous une forme simple et agréable un grand nombre de problèmes. Pour résoudre un CSP, il suffit de modéliser le problème, de spécifier les contraintes et leur résolution étant prise en charge automatiquement par une méthode de résolution (solveur). Tout au long de cette article, nous avons apporté certaines contributions dans le cadre des CSP.

De nombreux problèmes peuvent être exprimés en terme de contraintes comme les problèmes d'emploi de temps, de gestion d'agenda, de gestion de trafic ainsi que certains problèmes de planification et d'optimisation (comme le problème de routage de réseaux de telecommunication). cette article contient trois parties principales : la première partie permet d'une part, de fournir les éléments de base pour formaliser un problème de satisfaction de contraintes, et d'autre part de montrer comment un tel problème peut être ramené à une recherche d'une solution dans un espace d'états. La deuxième partie propose trois algorithmes de recherche de solutions qui mettent à jours les domaines de définitions de variables. La troisième partie permet de comprendre la notion de la consistance dans un système particulier contenant des contraintes unaires ou binaires et montre leur utilisation lors de la recherche d'une solution

Mots clefs : Problème de satisfaction de contraintes (CSP); Maintenance de la consistance d'arc; Heuristiques d'ordonnancement.

Contents

List of Figures

Chapter 1

Problèmes de Satisfaction de Contraintes (CSP)

1.1 Concepts et notations

Definition 1.1.1 (Problème de Satisfaction de Contraintes). Un problème de satisfaction de contraintes (PSC) est modélisé par un triplet P = (X,D,C), défini par:

- un ensemble X = $\{X_1, X_2, \cdots, X_n\}$ de n variables.
- un ensemble D = $\{D_1, D_2, \cdots, D_n\}$ de n domaines finis pour les variables de $\mathcal{X}$.
le domaine D_i est associé à la variable X_i.
- un ensemble fini de m contraintes C = $\{C_1, C_2, \cdots, C_m\}$.
Chaque contrainte C_i, est définie par un couple (v_i, r_i) :

- v_i, est un ensemble de variables $\{X_{i1}, X_{i2}, \cdots, X_{iq}\} \subset \mathcal{X}$
sur lesquelles porte la contrainte C_i. On appelle arité de C_i

la longueur de la séquence v_i, Donc le cardinal de v_i.

- r_i, est une relation, définie par un sous-ensemble du produit cartésien $D_{i1} \times D_{i2} \times \ldots D_{iq}$ des domaines associés aux variables de v_i.
une relation entre les variables de v_i, restreignant les valeurs que peuvent prendre simultanément ces variables.

Definition 1.1.2 (Être pertinente pour).

$\forall k \in [1, m]$, Chaque variable X_i est pertinente pour C_k(noté par $X_i \triangleright C_k$) , si C_k porte sur X_i.

Definition 1.1.3 (Aritéde d'une contrainte).

Étant donné une contrainte C_k et ses variable pertinentes, on définit l'arité de C_k a_k comme le nombre de variables pertinentes pour C_k.

Definition 1.1.4 (CSP binaire).

Un CSP binaire est un CSP P = (X,D,C) dont toutes les contraintes $C_k \in C$ ont une arité égale à 2, c'està dire, chaque contrainte a exactement 2 variables pertinentes.

Un CSP binaire peut être représenté par un graphe de contraintes dans lequel chaque nœud represente une variable, et chaque arc correspond à une contrainte entre 2 variables. Rossi et al. [RPD89] ont montré qu'il est possible de convertir un CSP avec des contraintes n-aires en un CSP binaire équivalent. C'est pour cela que la plupart, des

recherches sur les méthodes pour la résolution de CSP avec domaines finis traitent, pour commencer, les CSP binaires, [Fre82]. Dans cette thèse, nous abordons la résolution de CSP binaires en utilisant plus particulièrement leur représentation matricielle.

Definition 1.1.5 (Matrice de Contraintes)**.**

Une Matrice de Contraintes R est un tableau rectangulaire $m \times n$ tel que

$$\mathrm{R}_{\alpha j} = R\left[\alpha, j\right] = \begin{cases} 1 \text{ Si la variable } X_j \rhd C_\alpha \\ 0 \text{ Sinon} \end{cases}$$

S'il s'agit d'un CSP binaire, dans R il y a juste deux entrées non-nulles pour une contrainte C_α comme cela est montré sur la figure 1.1. Dans l'exemple, les variables $X_!et X_2$ sont les deux variables pertinentes pour C_α. Cela est représenté dans la matrice de contraintes avec le numéro 1 à l'intersection entre la colonne de chaque variable et la ligne correspondant à C_α. Dans le graphe de contraintes, cela est indiqué par l'arête α qui lie la variable X_2 avec la variable X_n

Definition 1.1.6 (Instanciation)**.** Étant donné un CSP P=(X,D,C) on appelle **instanciation I (ou affectation)** une application qui associé à chaque variable X_i une valeur $\mathrm{I}(\mathrm{X}_i) \in D_i$

Definition 1.1.7 (Instanciation Partielle)**.**
Étant donné un CSP P=(V,D,C) on appelle **instanciation partielle** I_p de $V_p = \left\{X_{p1}, X_{p2}, \cdots, X_{pj}\right\} \subset V$ une application qui associe à chaque variable $X_{pi} \in V_P$ une valeur $I_p(X_{pi}) \in D_{pi}$(le domaine de X_{pi})

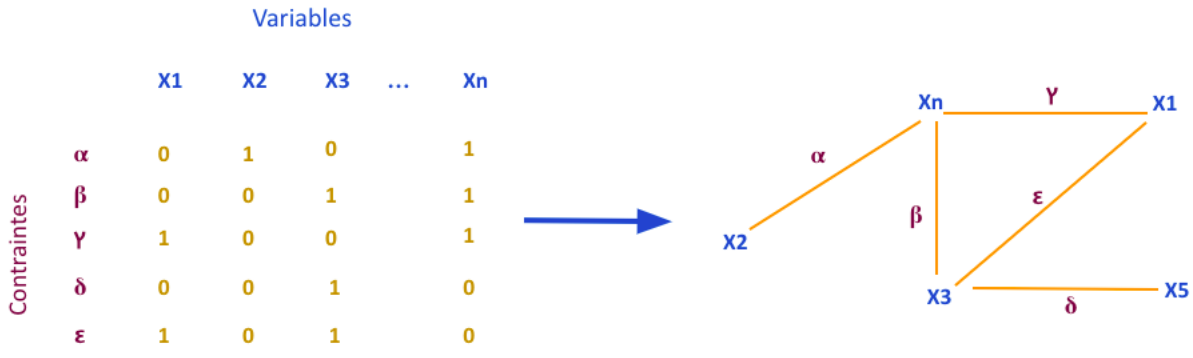


Figure 1.1: Matrice de Contraintes et son Graphe de Contraintes

Remarque: Dans le cas où $V_P = V$ alors I_p est une instanciation complète.

Definition 1.1.8 (Satisfaction de contrainte)**.**
Étant donné un CSP P=(V,D,C). une instanciation partielle
I_p **satisfait** la ontrainte $C_i = (v_i, r_i)$ de C(noté $I_p \models C_i$) ssi $v_i \in V_p$ et (I_p de v_i) $\in r_i$.
À l'opposé, on dira qu'une instanciation I_p viole C_i ssi $v_i \in V_p$ et $I_p(v_i) \notin r_i$.

Definition 1.1.9 (Instanciation consistante)**.** Étant donné un CSP P=(V,D,C),
une instanciation partielle I_p
des variables est dit **consistante ssi:**

$$\forall C_i = (v_i, r_i) \in C, \text{ telle que } v_i \in V_p, I_p \models C_i.$$

Definition 1.1.10 (Solution(S) d'un CSP)**.**
Une solution S de P=(V,D,C) est une instanciation consistante de toutes les variables,
($V_p = V$). On dit alors que l'instanciation S satisfait P (noté $S \models P$).
L'ensemle des solutions de P sera noté S_p.

Exemple 1

Soit le CSP composé des variables X=$\{ X_1, X_2, X_3\}$, des domaines

$\mathrm{D}_1 = \{1, 2\}, D_2 = \{2, 3, 4\}$ et $D_3 = \{4, 7\}$ et des contraintes
$\mathrm{C}_1 = \Big((x_1, x_2, x_3), \{(1, 3, 4), (2, 2, 4)\}\Big)$ et $C_2 =$
$\Big((x_1, x_2, x_3), \{(1, 2, 4), (1, 2, 7), (1, 3, 4), (1, 3, 7), (1, 4, 7), (2, 3, 4), (2, 3, 7), (2, 4, 7)\}\Big)$.
La contrainte C_2 correspond à une contrainte de différence deux à deux entre ses variables tandis que C_3 impose l'égalité entre x_3 et la somme de ses deux autres variables. $\mathrm{A}_p = \Big((x_1), (1)\Big)$ est une affectation partielle du CSP. $A_t = \Big((x_1, x_2, x_3), (1, 3, 4)\Big)$ est une affectation totale solution, du CSP car le triplet $(1, 3, 4)$ satisfait les deux contraintes C_1 et C_2.A_t est la seule solution du CSP, par exemple $\Big(\ (\mathrm{x}_1, x_2, x_3), (2, 2, 4)\Big)$ n'est pas solution car le triplet $(2, 2, 4)$ ne satisfait pas C_2.

1.2 Sudoku

Un autre exemple classique est la résolution d'un sudoku. Dans ce cas, les variables sont les 81 cases de la grille, le domaine de chaque variable sont les nombres de 1 à 9 et les contraintes ne sont pas la répétition des chiffres des lignes, des colonnes, des sous-réseaux. Dans ce cas, il est également donné une affectation initiale (les numéros déjà présents dans la grille)

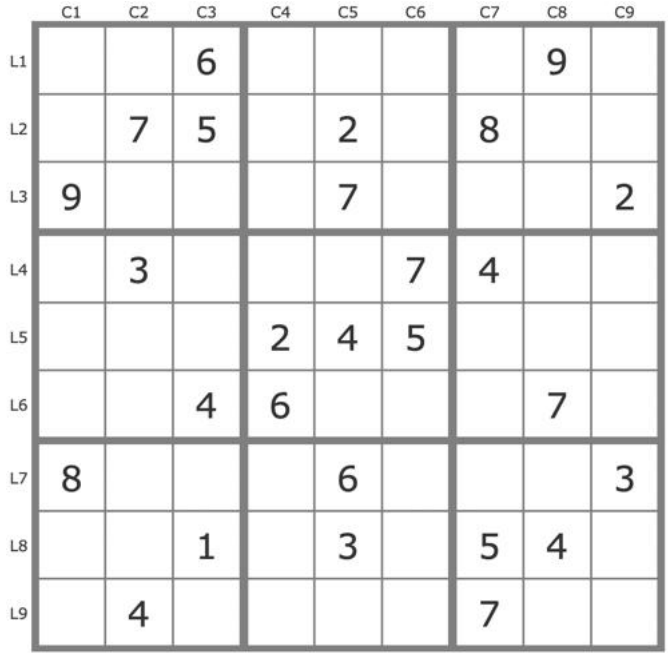


Figure 1.2: probleme de sudoku

Chapter 2

Exemples de CSP

Les exemples suivants sont d'un intérêt particulier, car ils sont des prototypes pour une grande classe de problèmes pratiques.

2.1 Le coloriage de graphe

Le problème de coloriage de graphe avec 3 couleurs consiste à colorier un graphe non-orienté comprenant n nœuds. Chaque nœud doit être colorié avec une des trois couleurs disponibles de telle sorte que deux noeuds voisins n'aient pas la même, couleur. Ce problème est utilisé pour rnodéliser certains types de problèmes d'ordonnancement et de gestion de ressources. La figure 1.2 montre le problème de coloriage d'une carte avec 3 couleurs (noir, blanc, gris). Les contraintes sont toutes du même type: ne pas colorier avec la même couleur les pays voisins.
Ce problème peut être modélisé comme un CSP en représentant chaque région comme une variable. Le domaine de chaque variable est défini par 3 couleurs possibles. Ainsi, nous obtenons le CSP suivant.
$-X = \{x_1, x_2, x_3, ..., x_7\}$
$-D = d_1 = d_2 = d_3 = ... = d_7 = \{(noir, blanc, gris\}$

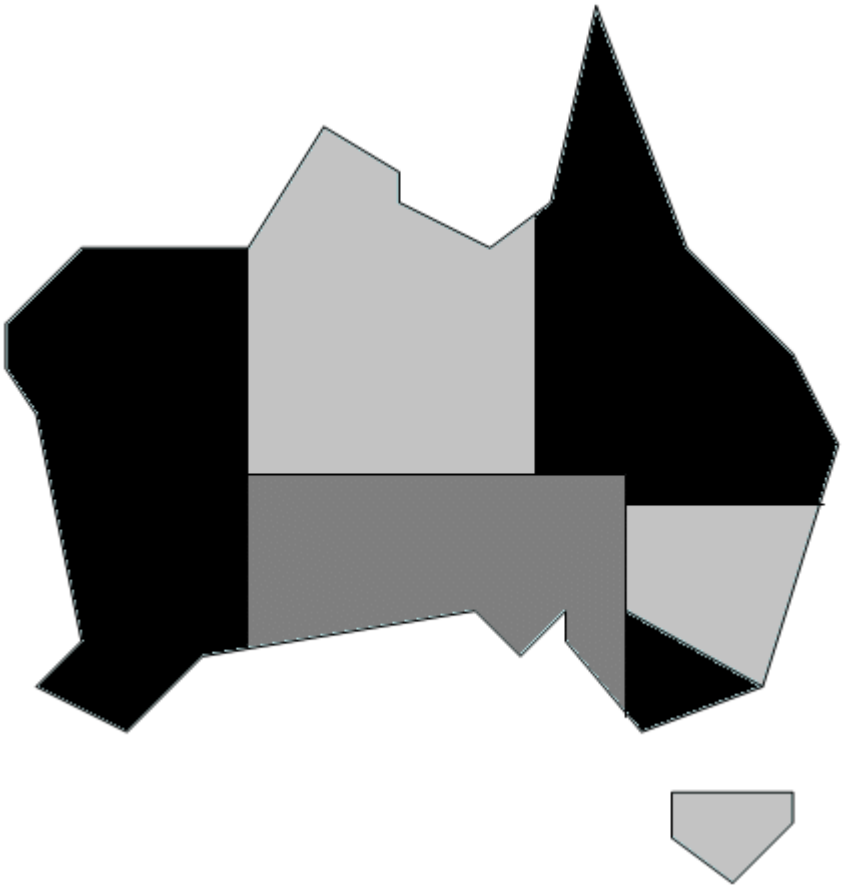


Figure 2.1: Exemple 1: Coloriage avec 3 couleurs

$-C = \{x_i \neq x_j / x_i \text{ et } x_j \text{ sontvoisins } \}$

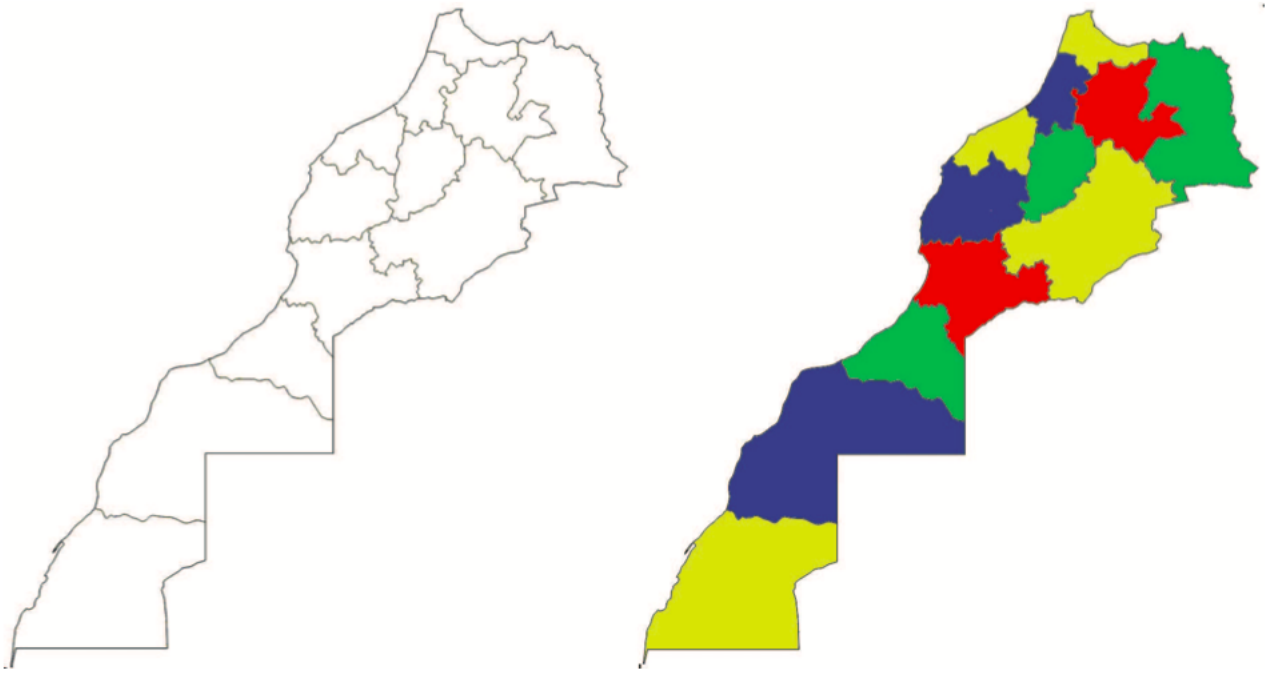


Figure 2.2: Exemple 2: Coloriage de la carte géographique du Maroc

2.2 Les N-reines

Les N-reines

Placer sur un échiquier de N x N cases. N reines qui ne s'attaquent pas (suivant les règles classiques des échecs),il ne faut donc pas plus d'une reine par ligne, par colonne et par diagonale. Il s'agit d'un problème équivalent à la rechercher d'un ensemble intérieurement stable dans un graphe de N x N nœuds (un nœud pour chaque case), chaque arête correspondant à une diagonale, une verticale ou une horizontale. Il peut s'envisager pour d'autres types de pièces.

Pour résoudre ce problème par CSP, il faut choisir une bonne modélisation, c'est-à-dire définir l'ensemble des variables X ainsi les domaines de chaque variables D(X), ensuite identifier les contraintes C entre les variables. Il existe plusieurs façons de modéliser ce problème par CSP, mais dans notre rapport, on va juste étudier une seule modélisation et c'est la plus efficace pour le cas des n reines. Par exemple, pour modéliser le problème de 4 reines sur un échiquier de 4 lignes et 4 colonnes, on considère 4 variables, et chaque variable est associée à une ligne i de telle sorte que cette variable désigne le numéro de la colonne sur laquelle la reine se trouve, c'est-à-dire

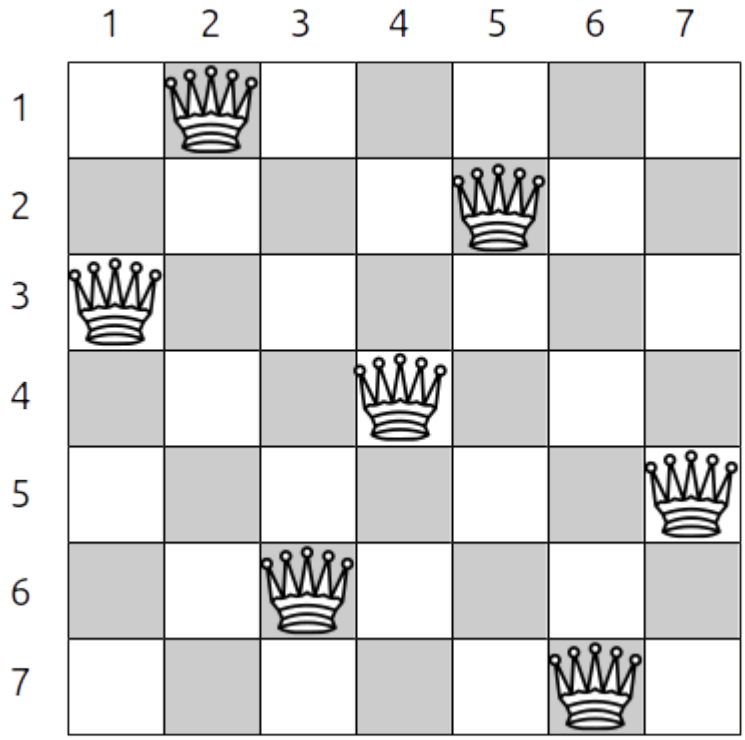


Figure 2.3: Exemple: 7-reines

qu'il y a une seule reine par ligne.

On peut résumer cette modélisation comme suit :

$-X = \{x_1, x_2, x_3, x_4\}$

$-D = d_1 = d_2 = d_3 = d_4 = \{1, 2, 3, 4\}$

$-C = \{c_1, c_2, c_3\}$ avec :

1. $c_1 = \{x_i \neq x_j / \forall (i,j) \in D \times D, i \neq j\}$ (Colonnes différents).
2. $c_2 = \{x_i + i \neq x_j + j / \forall (i,j) \in D \times D, i \neq j\}$ (Diagonales montant différents).
3. $c_3 = \{x_i - i \neq x_j - j / \forall (i,j) \in D \times D, i \neq j\}$ (Diagonales descendant différents).

2.3 Le problème du Zèbre

Le problème du Zèbre

Attribué à Lewis Carroll, pasteur logicien et écrivain anglais auteur de nombreux autres puzzles. On considère cinq maisons, toutes de couleurs différentes (rouge, bleu, jaune, blanc, vert), dans lesquelles logent cinq personnes de profession diffé- rente (peintre, sculpteur, diplomate, docteur et violoniste) de nationalité différente (anglaise, espagnole, japonaise, norvégienne et italienne) ayant chacune une boisson favorite (thé. jus de fruits, café, lait et vin) et des animaux favoris (chien, escargots, renard, cheval et zèbre). On dispose des faits suivants: l'Anglais habite la maison rouge, l'Espagnol possède un chien, le Japonais est peintre, l'Italien boit du thé, le Norvégien habite la, première maison à gauche, le propriétaire de la maison verte boit du café, la maison verte est à droite de la blanche, le sculpteur élève des escargots, le diplomate habite la maison jaune, on boit du lait dans la maison du milieu, le Norvégien habite à coté de la maison bleue, le violoniste boit du jus de fruit, le renard est dans la maison voisine du médecin, le cheval est à côté de la maison du diplomate. Il s'agit de trouver le possesseur du zèbre et le buveur de vin. En fait le problème n'admet qu'une seule solution et il s'agit de la trouver.

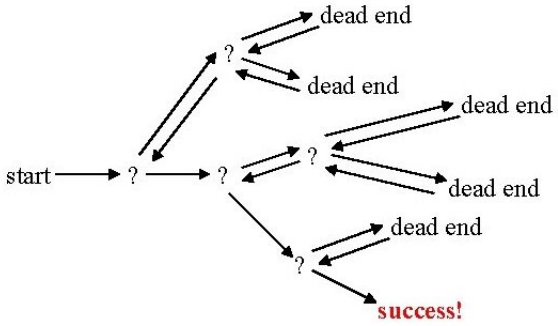
Chapter 3

Méthodes de résolution des CSP

Il existe différentes approches pour résoudre les problèmes de satisfaction de contraintes. Dans la figure 1.4, nous montrons un résumé non-exhaustif des différentes approches existantes pour aborder un CSP. Celles qui sont prises en compte dans cette article sont: Algorithmes Génétiques, Décomposition, Réparation par Escalade. Représentation par Réseaux de Contraintes. Nous pouvons aussi classifier en deux classes les méthodes de résolution: Méthodes Complètes et Incomplètes. Les méthodes complètes font une recherche arborescente en instanciant les variables une par une et en effectuant un retour en arrière en cas d'échec. Les méthodes incomplètes font une réparation d'une configuration en parcourant de manière non systématique (aléatoire) l'espace de recherche.

3.1 Recherche arborescente par retour-en-arrière (Le Bachtracking)

L'ensemble des affectations totales d'un CSP détermine l'espace de recherche qu'ont à explorer les méthodes de résolutions procédant par calcul des solutions. Cet espace de recherche peut se structure sous la forme d'un arbre qui associe à chaque variable une profondeur donnée et qui fait correspondre à chaque nœud une affectation



possible pour la variable du niveau considéré en associant à chaque nœud interne autant de fils qu'il existe d'affectations possibles pour la variable suivante. Toute branche menant de la racine à un nœud caractérise donc une affectation partielle du CSP, et dans le cas d'une feuille, une affectation totale. Du fait de l'ordonnancement des variables, de l'exhaustivité et de la non-redondance des choix de valeur pour chaque variable, nous garantissons alors la bijection entre feuilles de l'arbre et affectations totales du CSP. Dans ce cadre, la recherche arborescente par retour-arrière (backtracking) est une méthode de parcours en profondeur d'abord qui consiste à reconsidérer l'affectation de variable la plus récente en cas d'inconsistance. Nous distinguons deux méthodes - générer- et-tester et tester-et-générer - selon que le test de consistance s'effectue uniquement sur les feuilles, ou bien sur chaque nœud interne.

Le Backtracking est une stratégie de recherche qui a été largement utilisée dans les solveurs de contraintes. Dans le cadre CSP, le fonctionnement de base est de choisir une seule variable à la fois, et d'envisager pour elle une valeur de son domaine, en s'assurant que la nouvelle valeur choisie est compatible avec l'affectation partielle courante. Si l'affectation actuellement choisie viole certaines contraintes une autre valeur qui est disponible est prise. Si toutes les variables sont affectées, alors le problème est résolu. Si, à n'importe quelle étape, aucune valeur ne peut être affectée à une variable sans violer une contrainte, l'affectation de la dernière variable (juste avant l'échec) est révisée, et une autre valeur, lorsqu'elle est disponible, est assignée à cette variable. Cela procède de suite jusqu'à ce qu'une solution soit trouvée ou que tous les tests des combinaisons de valeurs aient échoué.

Algorithm 1 Algorithme Bachtracking chronologique

```
1: A ← ()
2: procedure BACKTRACK(A)                          ▷ The g.c.d. of a and b
3:     if isFull(A) then
4:         return A
5:     else
6:         select x_i ∈ X\vars_(A)
7:         for each v_i ∈ D(X_i) do
8:             x_i ← v_i
9:             if isLocallyConsistent(A ∪(x_i = v_i)) then
10:                Backtrack(A ∪(x_i = v_i))
11:
```

Backtracking avec L'anticipation est une simple amélioration de l'algorithme précédent. En fait, à chaque choix d'une affectation d'une variable, les domaines des variables non

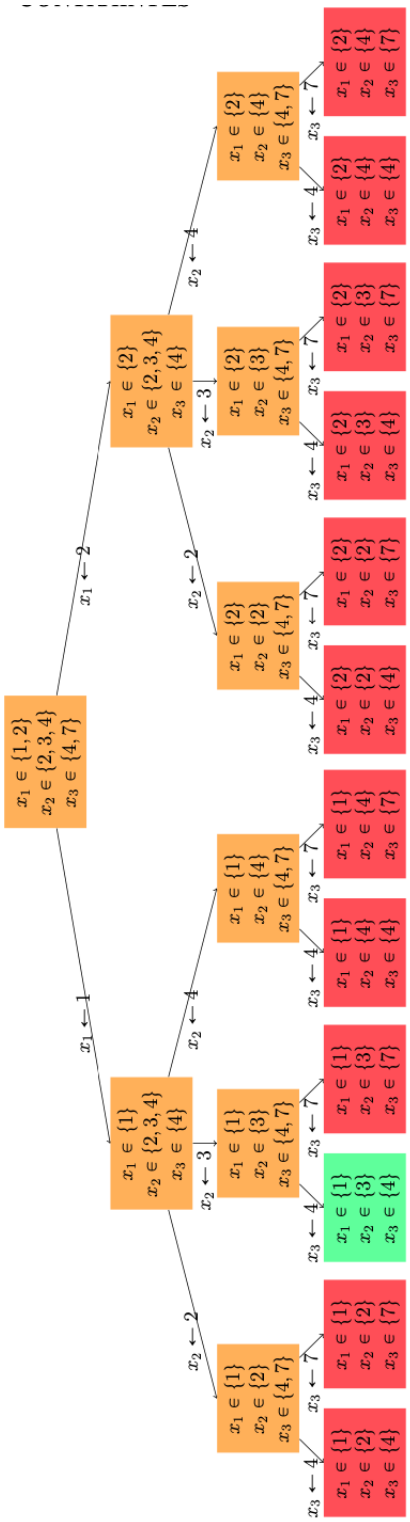


Figure 3.1: Arbre développé par la méthode générer-et-tester pour le CSP de l'exemple 1

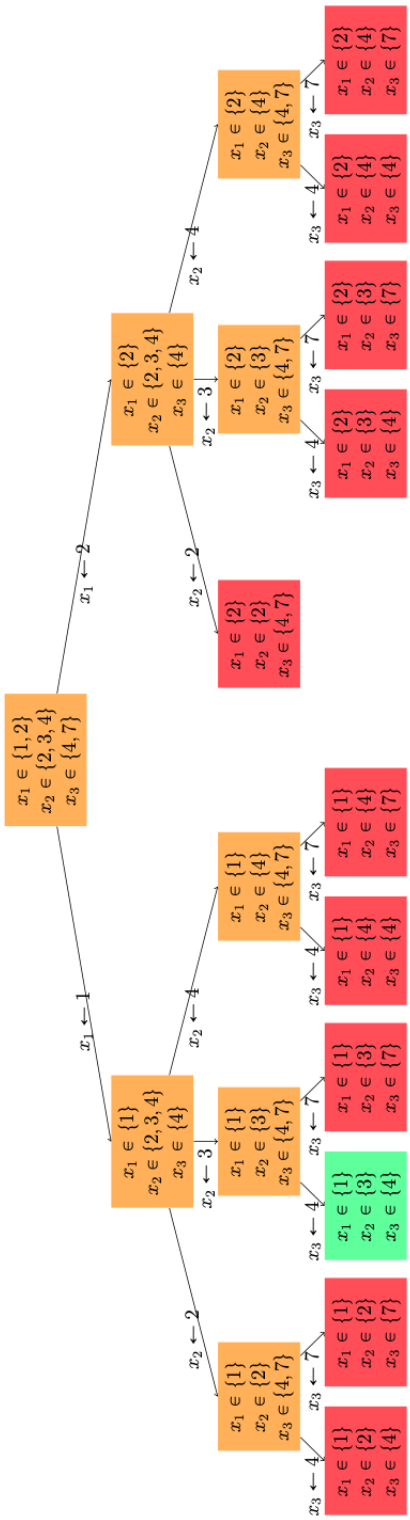


Figure 3.2: Arbre développé par la méthode tester-et-générer pour le CSP de l'exemple 1

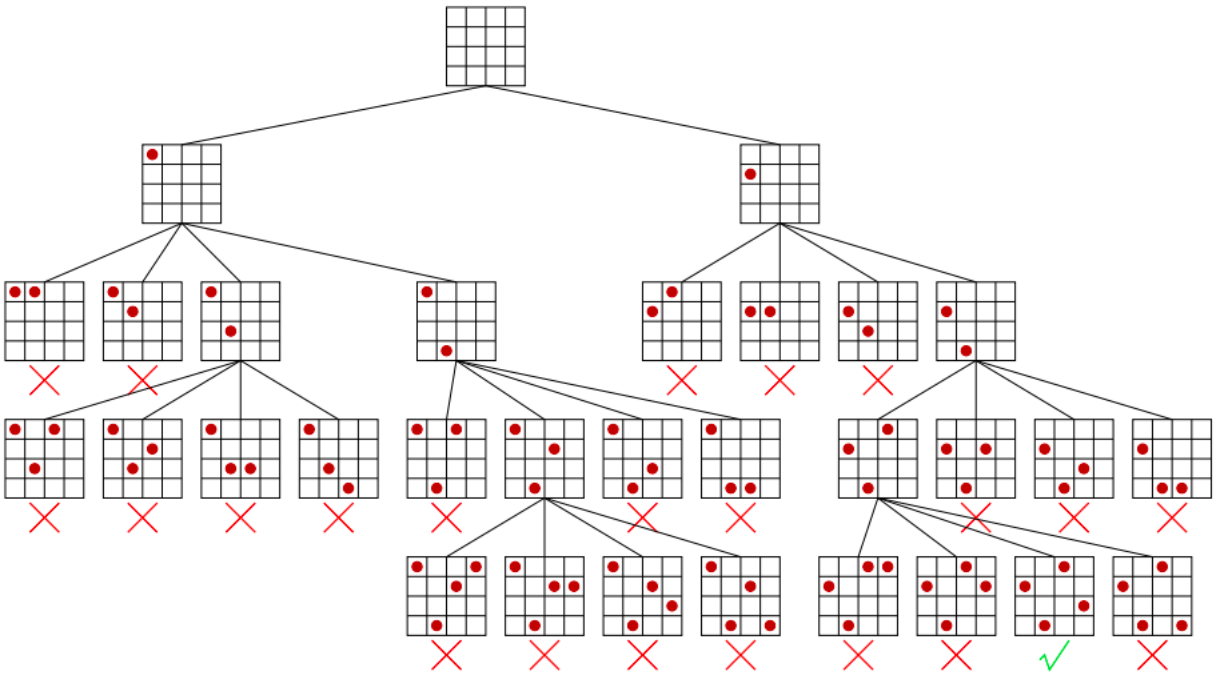


Figure 3.3: Algorithme backtrack : 4 reines.

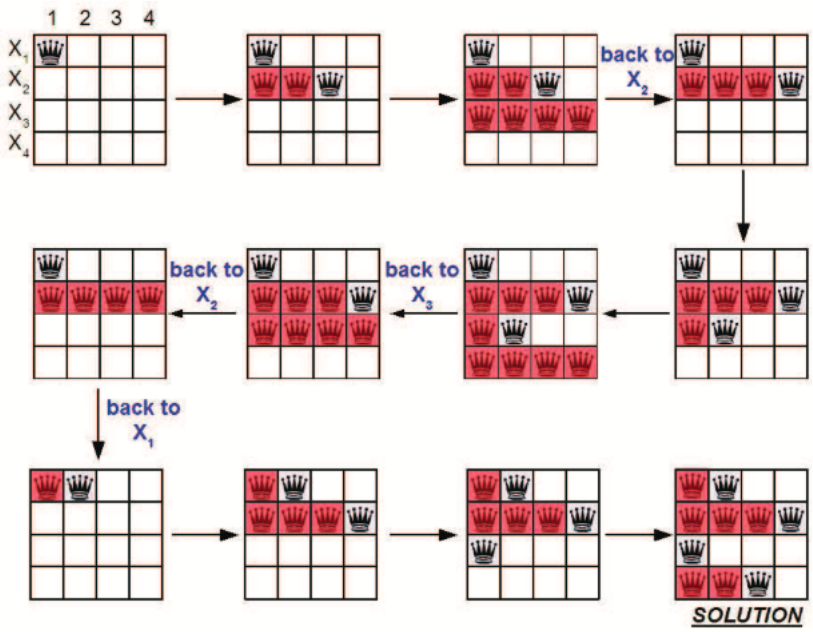


Figure 3.4: Exécution du Backtracking sur l'exemple des 4 reines avec L'anticipation

affectés encore, sont réduits aux valeurs qui sont compatibles avec l'affection en cours et ceci en utilisant le système des contraintes.

3.2 Forward Checking

L'algorithme Forward-Checking (FC) est la procédure la plus simple pour la vérification de chaque nouvelle instanciation contre les variables futures non instanciées. Le but de la vérification en avant est de propager les informations des variables affectées aux variables non encore instanciées. L'algorithme FC a le même principe que l'algorithme Backtracking, en ajoutant simplement une étape de filtrage de domaines des variables non instanciées, à chaque fois qu'une valeur est affectée à une variable. Le pseudo-code de procédure FC est présenté dans l'algorithme 2. FC est une procédure récursive qui tente de prévoir les effets de choisir une affectation sur les variables non encore attribuées. Chaque fois qu'une variable est affectée, FC vérifie les effets de cette instantiation sur les domaines des variables futures (ligne 8). Donc, toutes les valeurs des domaines de variables futures qui sont incompatibles avec la valeur attribuée (vi) de la variable courante (xi) sont enlevés (lignes 20-21). Les variables futures concernées par ce processus de filtrage ne sont que ceux qui partagent une contrainte avec xi (ligne 19). Par ailleurs, chaque domaine d'une variable futur est filtré afin de garder uniquement les valeurs cohérentes avec les variables déjà ins- tanciées. Par conséquent, le FC n'a pas besoin de vérifier la cohérence des nouvelles affectations contre celles déjà instanciées comme dans l'algorithme BT. Le Forward checking est alors la meilleure façon de prévenir les affectations qui garantissent l'échec plus tard.

Algorithm 2 Algorithme Forward Checking

```
1: procedure FC(A)                                          ▷ The g.c.d. of a and b
2:     if isFull(A) then
3:         return A
4:     else
5:         select x_i ∈ X\vars_(A)
6:         for each v_i ∈ D(X_i) do
7:             x_i ← v_i
8:             if Filter(A , (x_i = v_i)) then
9:                 FC(A ∪(x_i = v_i))
10:            else
11:                for each (x_j ∉ vars(A) such that ∃c_{i,j} ∈ C) do
12:                    restore D_j
13:                end for
14:            end if
15:        end for
16:    end if
17: end procedure
18: function FILTER(A , (x_i = v_i))
19:    for each (x_j ∉ vars(A) such that ∃c_{i,j} ∈ C) do
20:        for each (v_j ∈ D_j such that (v_i, v_j) ∉ c_{i,j}) do
21:            remove v_j fromD_j
22:        end for
23:        if D_j = ∅ then
24:            return False
25:        end if
26:    end for
27:    return True
28: end function
```

=0

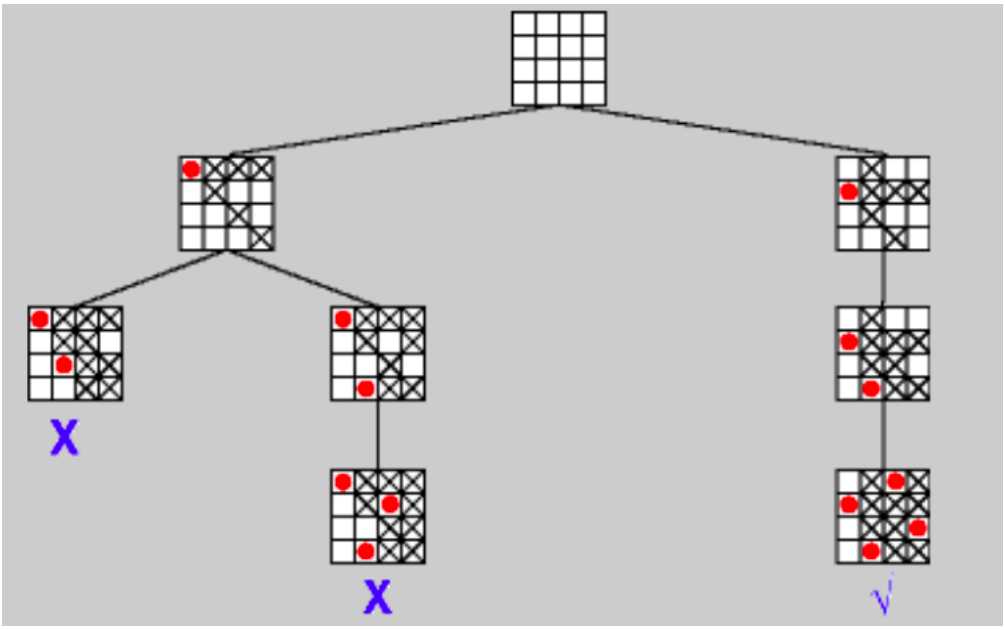


Figure 3.5: Exécution du Fonward-checking sur l'exemple des 4 reines avec L'anticipation

3.3 Filtrage des domaines (la consistance d'arc)

Le concept de la Filtrage des domaines est l'un des concepts le plus important dans les CSP pour améliorer la performance de l'algorithme Backtrack.

Filtrage des domaines:

Les méthodes de filtrage sont fondamentalement des méthodes de point fixe qui réduisent itérativement les domaines de variable jusqu'à obtenir une propriété voulue - communément appelée degré de consistance - ou la preuve de l'inconsistance du CSP.

Les méthodes de filtrage peuvent être utilisées soit avant de mener une recherche de solutions soit en cours de recherche (nous parlerons de maintien de consistance). Dans le second cas, le principe consiste à déclencher la méthode de filtrage à chaque affectation de variable en intégrant l'élimination des valeurs non retenues pour la variable et en en propageant les effets sur les domaines des variables restantes. Lorsque le filtrage détecte l'inconsistance du CSP courant, il est alors inutile d'explorer le sous-arbre associé au nœud courant. Le nœud est dit en échec et la recherche se poursuit par retour-arrière.

L'algorithme qui enlève les valeurs des domaines des variables d'un CSP jusqu'à ce qu'il soit consistant d'arc (on dit que l'algorithme filtre les domaines des variables) s'appelle

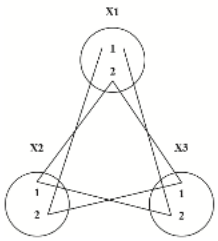


Figure 3.6: Exemple d'un problème arc consistant mais sans solution

AC (pour Arc Consistency). Il existe différentes versions de cet algorithme: AC1, AC2, AC3, ..., chaque version étant plus efficace (en général) que la précédente.

Definition 3.3.1 (Variable voisine)**.**

Deux variables sont voisines l'une de l'autre si et seulement si elles sont liées par une contrainte.

Definition 3.3.2 (La consistance d'arc)**.**

Une valeur vérifie la consistance d'arc aussi longtemps qu'elle a au moins un support (une valeur compatible) dans le domaine de chaque variable voisine. On dit alors qu'elle est arc-consistante.

Exemple 1:

$-X = \{x_1, x_2, x_3\}$

$-D = d_1 = d_2 = d_3 = \{1, 2\}$

$-C = \{x_1 \neq x_2, x_1 \neq x_3 et x_2 \neq x_3\}$

voire vigure 3.6

Exemple 2 :

Soit le CSP composé des variables $x_1 \in \{2\}$ et $x_2 \in \{2, 3\}$ et de la contrainte $x_1 \neq x_2$. La réduction de domaine par arc-consistance laisse inchangé le domaine de x_1, mais réduit le domaine de x_2 à 3, car la contrainte ne sera jamais satisfaite si x_2 prend la valeur 2.

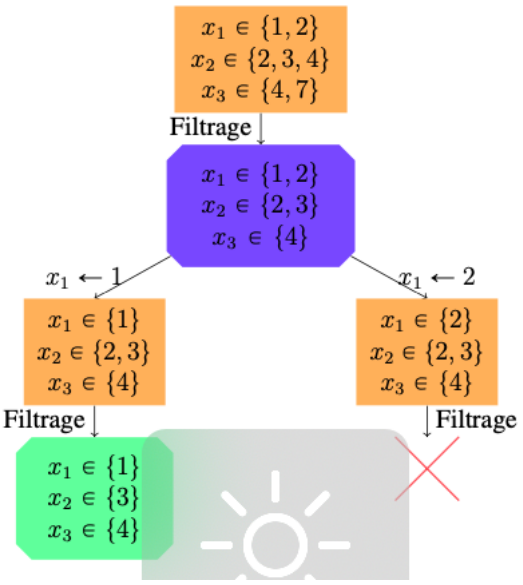


Figure 3.7: Arbre de recherche développé par filtrage sur le CSP de l'exemple 3

Exemple 3

Soit le CSP composé des variables X=$\{ X_1, X_2, X_3\}$, des domaines $D_1 = \{1,2\}, D_2 = \{2,3,4\}$ et $D_3 = \{4,7\}$ et des contraintes $C_1 = \Big((x_1,x_2,x_3), \{(1,3,4),(2,2,4)\}\Big)$ et $C_2 = \Big((x_1,x_2,x_3), \{(1,2,4),(1,2,7),(1,3,4),(1,3,7),(1,4,7),(2,3,4),(2,3,7),(2,4,7)\}\Big)$. La contrainte C_2 correspond à une contrainte de différence deux à deux entre ses variables tandis que C_3 impose l'égalité entre x_3 et la somme de ses deux autres variables. $A_p = \Big((x_1),(1)\Big)$ est une affectation partielle du CSP. $A_t = \Big((x_1,x_2,x_3),(1,3,4)\Big)$ est une affectation totale solution, du CSP car le triplet $(1,3,4)$ satisfait les deux contraintes C_1 et C_2.A_t est la seule solution du CSP, par exemple $\Big(\ (x_1,x_2,x_3),(2,2,4)\Big)$ n'est pas solution car le triplet $(2,2,4)$ ne satisfait pas C_2.

Le principe de l'AC3 s'impose sur la recherche de support pour chaque valeur sur chaque contrainte impliquée sur deux variables pour supprimer les valeurs non viables. pour cela, on utilise la liste Q qui contient ces paires de variables pour les étudier à l'aide de la fonction $Revise(x_i, x_j)$ qui détermine l'existence d'au moins un support pour chaque valeur de di sur cij. Si une valeur vi est supprimée parce qu'elle n'a pas de support sur $c_{i,j}$ AC3 réétudier la viabilité de toutes valeurs v_k de d_k par rapport à $c_{k,i}$

ALGORITHM 4: Algorithme AC-3

procedure AC-3()
1. **foreach** $(x_i, x_j \in E_G)$ **do** $Q \leftarrow Q \cup (x_i, x_j); (x_j, x_i);$
2. **while** $(Q \neq \emptyset)$ **do**
3. $(x_i, x_j) \leftarrow Q.pop()$;
4. **if** $(Revise(x_i, x_j))$ **then**
5. **if** $(D_i = \emptyset)$ **then** return false;
6. **else**
7. $Q \leftarrow Q \cup (x_k, x_i) \mid x_k, x_i \in E_G, k \neq i, k \neq j$
8. return true ;

function Revise(x_i, x_j)
9. *change* $\leftarrow$ false ;
10. **foreach** $(v_i \in D_i)$ **do**
11. **if** $(\nexists v_j \in D_j$ *such that* $(v_i, v_j) \in c_{ij})$ **then**
12. remove v_i from D_j ;
13. *change* $\leftarrow$ true ;
14. return *change* ;

Figure 3.8: Algorithme AC-3

en plaçant la paire de variables (x_k, x_i) dans Q avec $k \neq i$ et $k \neq j$.

3.4 Conflict-Directed Backjumping (CBJ)

Pendant que la procédure Backtracking n'effectue qu'un simple retour arrière lorsqu'une affectation d'une variable échoue en mettant en cause la dernière variable instanciée, les méthodes rétrospectives tentent d'identifier les causes de l'échec. Elles tirent parti de cette information pour sélectionner un meilleur point de retour. Dans l'algorithme CBJ "Conflict-directed backjumping", pour chaque variable instanciée x_i, CBJ maintient un ensemble dit ensemble de conflits. Cet ensemble contient toutes les variables responsables de l'échec de l'extension de l'affectation A. Cette extension restera impossible tant que les valeurs de A mises en cause dans cette contrainte sont présentes dans l'affectation courante. Tant que le retour en arrière n'atteint pas l'une des variables de l'ensemble de conflits, une extension complète est impossible car aucune valeur ne peut être affectée à la variable x_i. L'algorithme fera donc un saut (Backjump) vers la dernière variable affectée dans l'ensemble de conflits (figure 1.3).

Le pseudo-code de CBJ est illustré dans le figure ci suivant. Au lieu d'enregistrer seulement la variable la plus profonde, CBJ enregistre pour chaque variable x_i l'ensemble des variables qui étaient en conflit avec une certaine affectation de x_i. Ainsi, CBJ maintient un ensemble EMCS[i] (Earliest Minimal Conflict Set) pour chaque variable x_i où il stocke les variables appartenant à la plus proche violation de contraintes avec une affectation de x_i. Chaque fois qu'une variable x_i est choisie pour être instancié (ligne 4), CBJ initialise EMCS[i] à l'ensemble vide. Ensuite, CBJ initialise le domaine courant de xi à son domaine initial (lignes 5-6). Ensuite, une valeur v_i cohérente à l'état de la recherche actuelle est recherché pour la variable x_i. Si v_i est incompatible avec la solution partielle courante, alors v_i est retirée de domaine actuel d_i (ligne 16), et x_j est ajoutée à EMCS[i] tels que c_{ij} est la première contrainte violés par la nouvelle affectation de x_i (ligne 17). EMCS[i] peut être considéré comme le sous-ensemble des variables passées en conflit avec x_i. Quand un dead-end survient sur le domaine de la variable x_i, CBJ revient à la dernière variable, dite x_j, dans EMCS[i] (lignes 10, 11 et 18). Les informations contenues dans EMCS[i] sont remportées en haut à l'EMCS[j] (ligne 14). Par conséquent, CBJ effectue une forme de « backtracking intelligent" à la source du conflit permettant à la procédure de recherche d'éviter de

ALGORITHM 2: Algorithme Conflict-directed Backjumping

```
procedure CBJ(A)
1.  if (isFull(A)) then
2.      return A ;
3.  else
4.      choose x_i ∈ X \vars(A) ;
5.      EMCS[i] ← ∅ ;
6.      D_i ← D_i^0 ;
7.      foreach (v_i ∈ D_i) do
8.          x_i ← v_i ;
9.          if (isConsistent(A ∪ (x_i, v_i))) then
10.             CS ← CBJ(A ∪ (x_i, v_i)) ;
11.             if (x_i ∉ CS) then
12.                 return CS
13.             else
14.                 EMCS[i] ← EMCS[i] ∪ CS \x_i ;
15.         else
16.             remove v_i from D_i ;
17.             let c_ij be the earliest violated constraint by (x_i = v_i) ;
18.             EMCS[i] ← EMCS[i] ∪ x_j ;
19.     return EMCS[i] ;
```

redécouvrir le même échec dû à la même raison.

Quand un dead-end survient, l'algorithme CBJ fait un saut vers la variable coupable. Pendant le processus de backjumping, CBJ efface toutes les affectations des variables moins prioritaire à la variable coupable ce qui cause une perte d'effort significatif fait pour atteindre ces affectations.

[1] HAMMOUJAN Saida, Raisonnement Par Contraintes: Approches De Satisfaction De Contraintes Dynamiques Et Distribuées, THÈSE DE DOCTORAT, 2016

[2] Maria-Cristina, Riff-Rojas, R esolution de probl'emes de satisfaction de contraintes avec des algorithmes evolutionnistes, THÈSE DE DOCTORAT, 2010

[3] Vincent Vigneron, Programmation par contraintes et découverte de motifs sur données séquentielles, THÈSE DE DOCTORAT, 2017