Sommaire

1. Problème à satisfaction de contraintes (CSP)

Un problème à satisfaction de contraintes a trois composantes : X, D et C :
X est un ensemble de variables, $\{x_1,\ldots, x_n\}$.
D est un ensemble de domaines, $\{d_1,\ldots,d_n\}$, un pour chaque variable.
C est un ensemble de contraintes qui spécifient les combinaisons admissibles de valeurs.

Chaque domaine d_i est constitué d'un ensemble de valeurs admissibles $\{v_1,\ldots, v_k\}$ pour la variable x_i. Chaque contrainte c_i est constitué d'une paire de <portée, rel>, où portée est un n-uplet de variables qui participent à la contraintes et rel est une relation qui définit les valeurs que ces variables peuvent prendre. Une relation peut être représentée comme une liste explicite de tous les n-uplets de valeurs qui satisfont la contrainte, ou comme une relation abstraite sur laquelle s'appliquent deux opérations:

- Tester si un n-uplet est un membre de la relation.
- Enumérer les membres de la relation.

Par exemple, si $x_1$et x_2 ont tous les deux le domaine {A, B}, la contrainte qui dicte que les variables doivent avoir des valeurs différentes peut s'écrire <(x_1, x_2), [(A, B), (B, A)]> ou <(x_1, x_2), $x_1 \# x_2$>.
L'objectif consiste simplement à trouver un ensemble de valeurs à affecter aux variables, de sorte que toutes les contraintes soient satisfaites.
Nous nous intéressons dans le cadre de ce projet aux problèmes de satisfaction de contraintes binaires en domaines finis.

Un CSP binaire est un CSP P = (V, D, C) dont toutes les contraintes c_i appartient à C ont une arité à 2, c'est à dire, chaque contrainte a exactement 2 variables pertinentes.

Exemple d'un CSP binaire : le coloriage de graphes
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 nœuds voisins n'aient pas la même couleur.

2. Algorithmes de résolution

Les algorithmes de résolution ont pour but de vérifier que toutes les variables peuvent prendre des valeurs qui appartiennent au domaine des variables, tout en respectant les différentes contraintes.

Cet algorithme devra donc générer au moins une solution qui vérifie le système. Il devra pour cela affecter des valeurs aux variables tout en vérifiant que ces valeurs ne s'opposent pas aux contraintes.

Il existe plusieurs types d'algorithmes. Certains fonctionnent de manière systématique en testant toutes les solutions possibles jusqu'à ce qu'ils trouvent une bonne solution. D'autres, plus intuitifs, construisent la réponse à partir d'une solution non satisfaisante quelconque en la modifiant selon les contraintes.

a. Backtrack

Cet algorithme trouve systématiquement une solution s'il en existe une. Pour cela, il affecte au fur et à mesure une valeur de D à chaque variable correspondante dans V. Il vérifie évidemment que la valeur est correcte par rapport aux contraintes.

A chaque affectation, les domaines des variables restantes diminuent. Il arrive qu'à l'affectation d'une variable x_{n-1}, l'algorithme ne puisse trouver de valeur pour x_n car D(x_n) est vide.

A ce stade, l'algorithme revient en arrière, à la dernière variable affectée x_{n-1}. Il modifie la valeur de x_{n-1} en espérant que le domaine de la variable x_n suivante ne sera plus nul. Si après avoir essayé toutes les valeurs du domaine D(x_{n-1}), il n'a pas toujours de solutions pour x_n, il recule et modifie la variable précédente x_{n-2}. Si il n'existe pas de solution, il reculera jusqu'à x_0, sinon il trouve obligatoirement la solution.

Le défaut de cet algorithme est qu'il peut tester toutes les valeurs possibles avant de trouver une solution. Pour le problème de la coloration de graphe, si on lance la procédure avec 100 sommets et 3 couleurs possibles, il existe 3100 solutions (3*3*3*...*3*3) soit 5,15 * 1047 solutions. Si jamais la première bonne solution se trouve dans les dernières possibilités ou si jamais il n'y a pas de solutions, le temps de calcul sera très long.

Exemple :
On va exécuter l'algorithme du backtrack avec les contraintes suivantes :

$$\begin{cases} x_1 < x_2 \\ x_1 < x_3 \\ x_1 < x_4 \\ x_1 < x_5 \\ x_3 < x_6 \\ x_4 > x_6 \\ x_5 \in \{1,2\} \\ x_i \in \{1,2,3\}, i \neq 5 \end{cases}$$

Le graphe suivant montre le nombre de tests qu'on aura effectué :

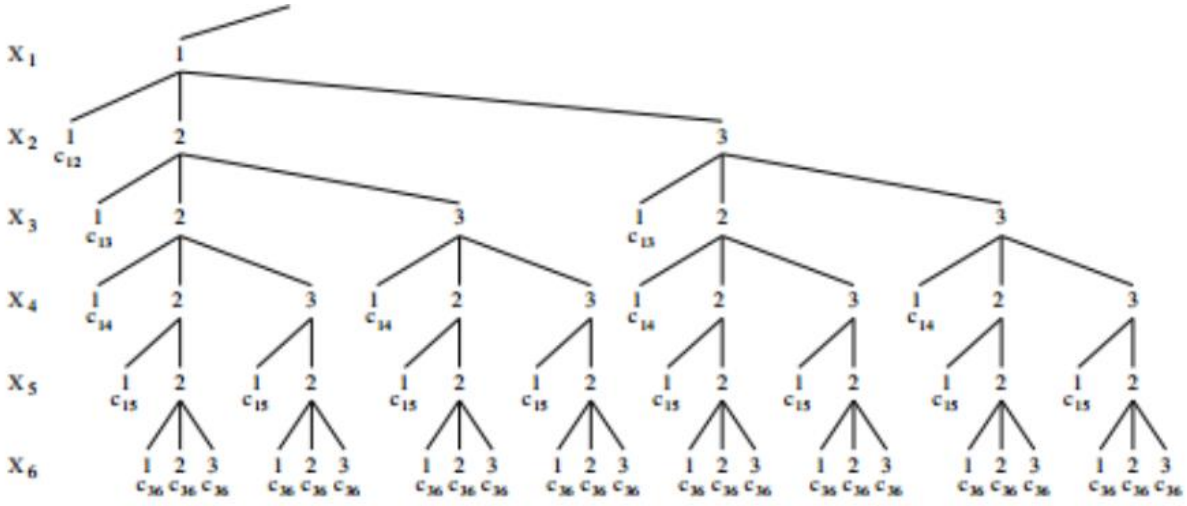

Figure 1 : Graphe résumant les tests du backtrack

b. Foward Checking

Cet algorithme est assez proche du backtrack, il affecte des valeurs aux variables au fur et à mesure. La différence se trouve dans la gestion des impasses, quand l'algorithme ne trouve plus de solutions.

Le foward checking, avant de choisir une valeur pour une variable x_n, vérifie que cette affectation correspond aux contraintes et vérifie que les autres variables x_i (i>n) pourront être affectées. Si l'affectation de x_i ne peut être faite, l'algorithme choisi une autre valeur pour x_n.

Si jamais aucune solution, pour x_n, ne permet l'affectation des autres variables, la procédure fera un retour en arrière sur x_{n-1} pour changer sa valeur. Ce point reste identique au backtrack. Si le CSP n'a pas de solution, on recule jusqu'à la variable x_0.

Exemple :
On va exécuter l'algorithme du foward checking avec les contraintes suivantes :

$$\begin{cases} x_1 < x_2 \\ x_1 < x_3 \\ x_1 < x_4 \\ x_2 < x_5 \\ x_3 > x_5 \\ x_3 \in \{1,2\} \\ x_i \in \{1,2,3\}, i \neq 3 \end{cases}$$

Le graphe suivant montre le nombre de tests qu'on aura effectué :

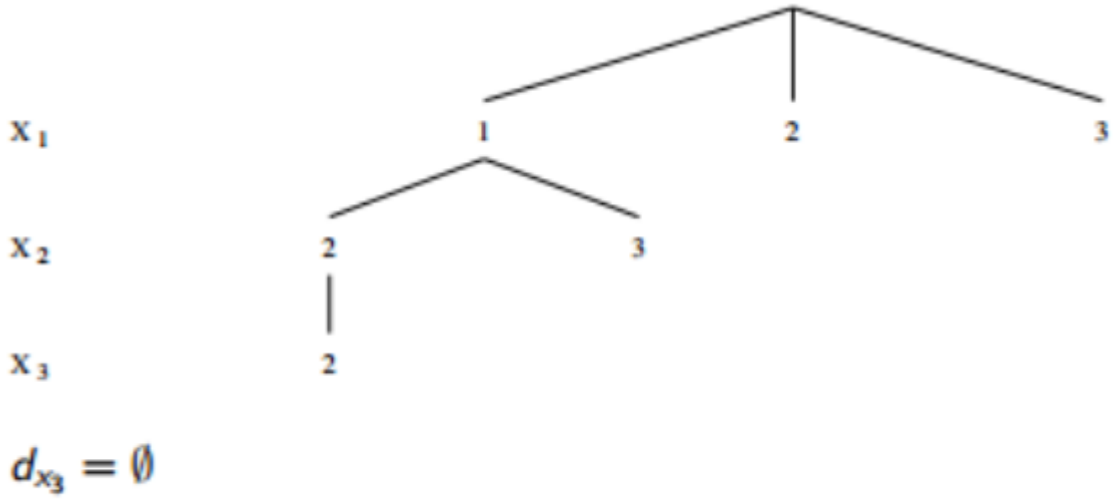

Figure 2 : Graphe résumant les tests du foward checking

c. Filtrage par arc consistance

Le filtrage a pour but de supprimer de chaque domaine toutes les variables dans un support (une valeur a_i d'un domaine dom(i) à un support a_i dans le domaine dom(j) si le couple (a_i, a_j) est autorisé par la contrainte liant les variables i et j).

i. Real-full look-ahead

L'algorithme a pour objectif de résoudre des instances de CSP et réalise une recherche en profondeur d'abord avec les retours en arrières. A chaque étape de la recherche, une assignation de variable est effectuée suivie par un processus de filtrage qui correspond à établir la consistance d'arc.

ii. MAC

L’algorithme consiste à instancier une variable par une valeur de son domaine est considérée comme la réduction de ce domaine à la valeur choisie ; cette réduction est donc propagée pour obtenir la consistance d'arc. Ce processus est répété lors de chaque instanciation.

Cette technique permet d'élaguer considérablement l'espace de recherche, et semble (en l'état actuel des connaissances) présenter le meilleur compromis entre surcoût occasionné par le filtrage et réduction de l'espace de recherche.

3. Description des structures crées

Pour pouvoir représenter les graphes d’arc consistance, il a fallu créer deux structures :

La première, nommée var_domaine représente les valeurs qui doivent correspondre entre deux variables selon les contraintes. Elle est représentée par un tableau à deux dimensions de pointeurs d’entier. Si la case 0, 1 de l’instance de la structure est à 1, cela veut dire que la variable x_1 avec la valeur 0 peut être associée avec la variable x_2 de valeur 1.

La seconde structure, appelée tab_variable regroupe les liens entre les variables. Elle est représentée par un tableau à deux dimensions de pointeurs sur des var_domaine. Ces pointeurs sont tous initialisés à Null à départ, et prennent l’adresse d’une instance de la structure var_domaine s’il y a un arc entre deux variables.

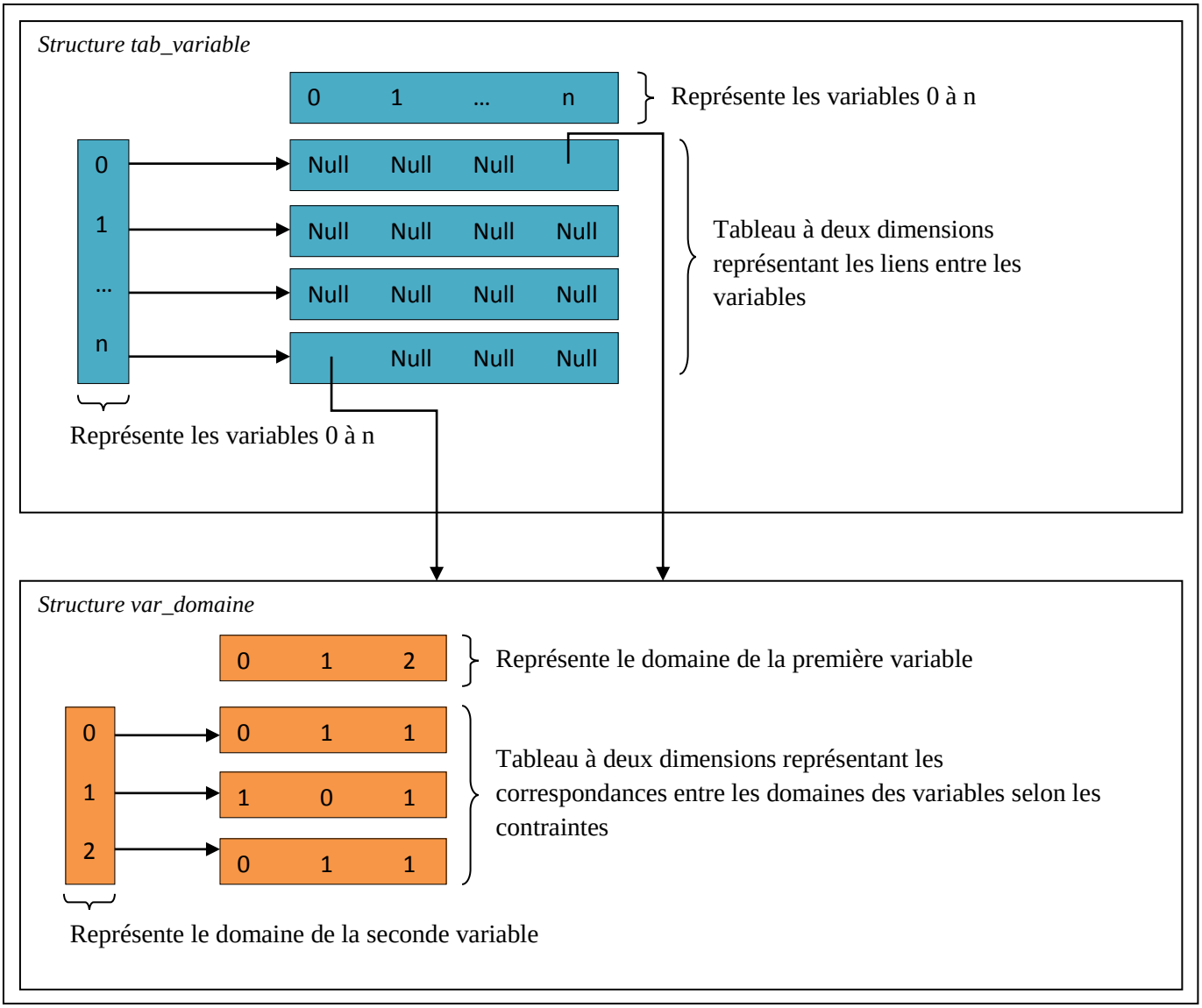

Figure 3 : Relations entre les deux structures

4. Génération d'un graphe aléatoire

a. Procédures pour générer le graphe

La génération d'un graphe aléatoire passe par deux fonctions :

- void genereGraphe(int nbSommet,int nbArrete,int nbDomaine,float sat)
  Cette fonction aura pour but de créer une instance de la structure tab_variable et de la configurer en fonction des paramètres nbSommet et nbArrete, qui sont le nombre de sommets et le nombre d'arrêtes dans le graphe. Les arrêtes seront générées aléatoirement, et les correspondances seront effectuées à l'aide de la fonction genereDomaine décrite ci-dessous.
- var_domaine* genereDomaine(float s,int nbDomaine,int var1,int var2)
  Cette fonction a pour but de créer une instance de la structure var_domaine et de la configurer en fonction des paramètres de la fonction. On aura donc un tableau à deux dimensions d'entiers de taille nbDomaine. Les correspondances des domaines seront générées aléatoirement, en ne dépassant pas le coefficient de satisfiabilité s.

b. Exemple d'un graphe aléatoire

Voici un schéma représentant un des cas qui seront générés si on veut un graphe contenant 4 sommets, 4 arrêtes, un domaine de 3 éléments et un coefficient de satisfiabilité égal à 0.67.

	0	1	2	3
0	NULL	Adresse var_domaine 1	NULL	Adresse var_domaine 2
1	Adresse var_domaine 1	NULL	Adresse var_domaine 3	NULL
2	NULL	Adresse var_domaine 3	NULL	Adresse var_domaine 4
3	Adresse var_domaine 2	NULL	Adresse var_domaine 4	NULL

Tab_variable

	0	1	2
0	1	0	1
1	1	1	1
2	0	0	1

Var_domaine 1

	0	1	2
0	1	1	1
1	1	1	1
2	0	0	0

Var_domaine 2

	0	1	2
0	0	1	0
1	0	1	1
2	1	1	1

Var_domaine 3

	0	1	2
0	1	0	1
1	1	1	0
2	1	1	0

Var_domaine 4

Figure 4 : Exemple d'une génération d'un graphe aléatoire