# Programme de résolution de problème par les techniques de Propagation et Satisfaction de Contraintes

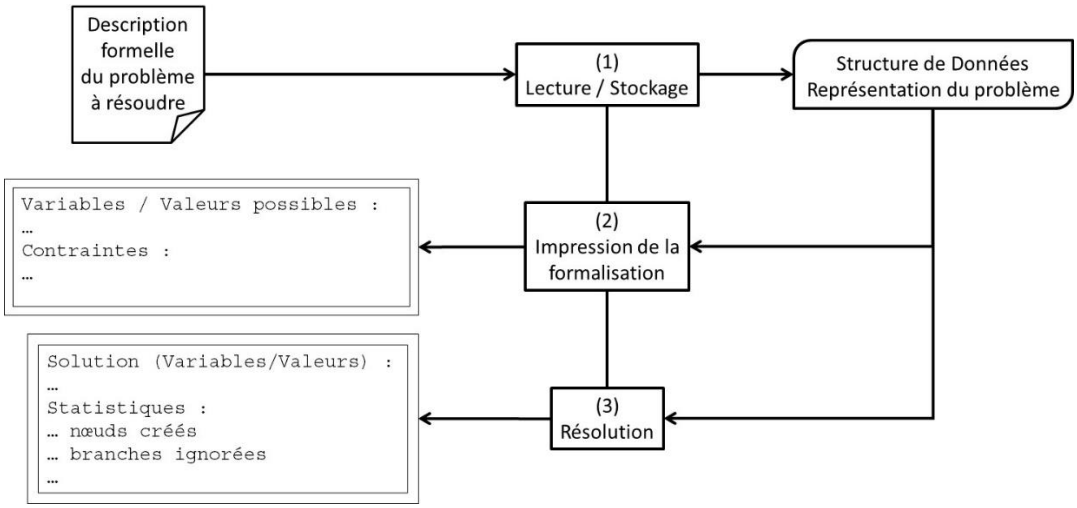
## *Objectif*

Vous devez réaliser un programme générique qui met en œuvre les techniques vues en cours (vérification de cohérence, réduction des domaines de valeurs), afin de résoudre n'importe quel problème décrit au moyen de variables, domaines de valeurs et contraintes.

Vous pourrez ainsi résoudre les différents problèmes vus en cours ou en TD, mais également tout autre problème modélisé de la même façon.

## *Structure générale du programme à réaliser*

Votre programme peut être illustré de la façon suivante :




(1) : lecture de la description d'un problème dans un fichier, et stockage en mémoire.
Un exemple de structure de fichier vous est fourni. Vous n'êtes pas obligés de respecter cette structure, et vous pouvez faire quelque chose de différent.
A vous de définir les structures de données que vous souhaitez utiliser.

(2) : impression du contenu de la structure de données, sous une forme suffisamment explicite et lisible.
On doit pouvoir y trouver clairement l'ensemble des variables et leurs domaines de valeurs, ainsi que les contraintes.

(3) : résolution du problème.
Cette fonction sera développée en plusieurs étapes (voir plus loin dans ce document).
En résultat, elle affiche les valeurs associées aux variables dans la solution trouvée.
Elle affiche également quelques données statistiques sur son exécution.

## Description formelle - Spécification d'un problème à résoudre

Le solveur prend en entrée la description du problème à résoudre. Ce fichier doit contenir :

- la liste des variables,
- la définition des domaines de valeurs associés,
- les contraintes.

*Votre programme doit être générique. C'est-à-dire qu'il doit permettre de prendre en compte n'importe quel problème décrit sous forme de variables + domaines de valeurs + contraintes.*

Voici un *exemple* de ce que vous pouvez utiliser comme fichier d'entrée. Il considère que les variables sont toutes à valeur de type « entier ».

```
5
1 4 1 3 5 7
2 4 2 4 6 8
3 5 1 2 3 4 5
4 2 0 1
5 3 0 1 2
1 1 4
2 5 2
3 1 2
4 10 1 2 3 -1
-1
```

*Explications :*

Les informations sont données dans l'ordre suivant :

- identification des variables,
- définition des domaines de valeurs,
- définition des contraintes.

1) Les variables sont identifiées par un nombre, et sont simplement numérotées de façon continue, de 1 à n.
Une première ligne du fichier peut les identifier toutes en en donnant simplement le nombre 'n'.

Dans l'exemple, la première ligne

```
5
```

indique qu'il y a 5 variables :

$X = \{ x_1 , x_2 , x_3 , x_4 , x_5 \}$.

2) Nous supposerons que les variables ont toutes des valeurs numériques.
On peut, par exemple, avoir dans le fichier une ligne par variable, avec toutes les valeurs possibles.

Dans l'exemple, la ligne

```
1 4 1 3 5 7
```

indique que la variable n° 1 a 4 valeurs qui sont 1, 3, 5 et 7 : $D_1 = \{ 1 , 3 , 5 , 7 \}$
Si on considère l'exemple donné ci-dessus, les lignes 2 à 6

```
1 4 1 3 5 7
2 4 2 4 6 8
3 5 1 2 3 4 5
4 2 0 1
5 3 0 1 2
```

définissent les domaines de valeurs :

$D_1 = \{ 1 , 3 , 5 , 7 \}$
$D_2 = \{ 2 , 4 , 6 , 8 \}$
$D_3 = \{ 1 , 2 , 3 , 4 , 5 \}$
$D_4 = \{ 0 , 1 \}$
$D_5 = \{ 0 , 1 , 2 \}$

3) Chaque contrainte est définie sur une ligne. Chaque ligne contient :

- le type de la contrainte,
- les variables prises en compte dans la variable,
- des valeurs numériques éventuelles.

Les types de contraintes peuvent être représentés par des codes numériques, par exemple :

| Type | Code |
|---|---|

| | | |
|---|---|---|
| $x_i = x_j$ | 1 | variables égales |
| $x_i \neq x_j$ | 2 | variables différentes |
| $x_i \leq x_j$ | 3 | variable inférieure ou égale à une autre |
| $\Sigma x_i = v$ | 4 | somme de variables égale à une valeur déterminée |
| … | … | |

En faisant cela, les lignes 7 à 10 de l'exemple :

```
1 1 4
2 5 2
3 1 2
4 10 1 2 3 -1
-1
```

définissent successivement les contraintes suivantes :

$x1 = x3$
$x5 \neq x2$
$x1 \leq x2$
$x1 + x2 + x3 = 10$

A vous de définir les types de contraintes que votre système sera capable de traiter, et de définir les codes associés. Bien entendu, plus vous serez capable de représenter et traiter correctement des types de contraintes différents, plus votre programme sera performant et pourra résoudre des problèmes différents.

## Représentation en mémoire

Votre programme doit tout d'abord représenter le problème en mémoire. Il lit les données dans le fichier d'entrée (voir précédemment) et représente son contenu dans des structures de données de votre choix.

Vous êtes libre dans le choix de ces structures de données.

La représentation des variables et de leurs domaines de valeurs ne pose a priori pas de problème majeur. Il peut par exemple s'agit de tableaux, listes, vecteurs, …
Attention : vous devez être capable de représenter n'importe quel domaine de valeurs numériques. Des structures de données statiques sont donc fortement déconseillées.

La représentation des contraintes demande sans doute un peu plus de réflexion. Représenter des contraintes binaires n'est pas la même chose que représenter des contraintes n-aires.

Pour des contraintes binaires, par exemple '=', '≠', '≤', '≥', on peut utiliser des matrices à deux dimensions à valeurs booléennes (matrice d'adjacence d'un graphe de contraintes). Il y aura une matrice par type de contrainte.
Pour les contraintes n-aires, c'est un peu plus difficile. Si on veut par exemple représenter une contrainte de la forme « somme de variables = valeur », il faut représenter à la fois la liste des variables concernées ainsi que la valeur à obtenir. On peut aussi penser à une somme pondérée de variables, pour laquelle il faut aussi représenter les poids associés aux variables.

## Résolution du problème

Le problème étant représenté en mémoire, reste à le résoudre.

Nous avons vu en cours que la recherche consiste en un parcours d'arbre en profondeur d'abord. Une mise en œuvre récursive de l'algorithme de résolution est certainement la plus immédiate.

Remarque : La mise en œuvre du parcours par une procédure récursive entraine qu'il n'est pas nécessaire de représenter l'arbre en mémoire. Lorsqu'on parle de « parcours de l'arbre », il s'agit en fait « d'enchaînement des appels récursifs » de la procédure de résolution.
N'oubliez pas que la représentation d'un nœud de l'arbre prend de la place en mémoire, et que l'arbre peut vite devenir volumineux s'il n'y a pas d'élagage par vérification continue des contraintes ou réduction des domaines de valeurs. Dans le passé, certaines équipes se sont vues piégées par l'espace mémoire nécessaire pour représenter des arbres volumineux…

Reportez-vous au cours pour une description des algorithmes demandés aux étapes 1 à 3.

*Etape 1.*

Vous pouvez dans un premier temps mettre en œuvre la méthode naïve (et lourde à l'exécution), mais simple à construire, consistant à « parcourir » tout l'arbre (parcours en profondeur d'abord), et vérifier sur les feuilles que toutes les contraintes sont respectées.

Une procédure « méthode_triviale », si elle est mise en œuvre de façon récursive, prend en paramètre une « affectation », telle qu'identifiée en cours, à savoir la liste des affectations déjà faites sur la branche parcourue, et les domaines de valeurs de l'ensemble des variables.
Elle rend comme résultat, en cas de succès, une affectation complète.

Un pseudo-code est proposé en annexe.

L'ordre de prise en compte des variables, et l'ordre de prise en compte des valeurs pour une variable, n'a pas d'importance pour cette étape.

*Etape 2.*

*Il s'agit ici d'une amélioration du code produit lors de l'étape 1 : le code nécessaire à la réduction des domaines de valeurs est ajoutée à la procédure initiale.*

Une fois cet algorithme « méthode triviale » mis en place, vous pouvez passer à la mise en place de la réduction des domaines de valeurs. Chaque affectation d'une variable 'x' réduit les domaines de valeurs des autres variables 'y' « sous contrainte » avec 'x'. Ces domaines de valeurs réduits sont propagés dans les branches lors du parcours en profondeur d'abord. Un domaine de valeur vide provoque l'élagage de la branche en cours (pas de solution).
Nous avons vu en cours que cette méthode réduit les branches à venir, car elle limite les valeurs à tester pour les variables pas encore assignées. Elle permet le cas échéant de stopper l'analyse d'une branche s'il n'y a plus de valeur possible pour une variable.

Un pseudo-code est proposé en annexe.

*Etape 3.*

Vous pouvez ensuite permettre à votre système de choisir entre plusieurs stratégies de « construction » de l'arbre au moyen de différentes stratégies.
Les étapes 1 et 2 vont naturellement provoquer l'assignation de valeurs aux variables dans l'ordre de leur numérotation.
Lors de l'étape 3, vous pouvez mettre en œuvre d'autres stratégies évoquées en cours, par exemple choisir les variables avec les plus grands / plus petits domaines de valeurs possibles.

Pour les équipes qui réalisent cette étape 3, il sera intéressant de résoudre les mêmes problèmes avec des stratégies différentes, et de comparer les données statistiques fournies par le programme.

*Etape 4.*

Dernière amélioration : mettre en œuvre la « cohérence d'arête ». Lorsque tout le reste marche uniquement.

Puissance de votre programme

N'oubliez pas que la puissance d'un programme de PSC se juge en particulier sur les types de contraintes qui peuvent être prises en compte.
Les contraintes de type « différent de » permettent de modéliser de nombreux problèmes, mais ce n'est pas suffisant. Si vous prenez l'exemple du carré magique (voir support de TD), vous devez traiter les contraintes de type « somme de variables = valeur ».
Dès l'étape 2 de la réalisation, il vous faut également réfléchir aux contraintes que vous voulez pouvoir traiter avec votre programme.

**Annexe – Résolution par la « méthode triviale »**

```
fonction méthode_triviale
entrée :     < X_A , D , X , C >
                 ensemble des variables assignées X_A
                 ensemble des domaines de valeurs D = { D_i }
                                  // D_i est un singleton si x_i ∈ X_A
                 ensemble des variables X
                 définition des contraintes C
résultat :   { < x_i , v_i > }
                 ensemble de couples variable / valeur
             en cas de succès :
                 toutes les variables sont référencées et ont une valeur
             en cas d'échec :
                 l'ensemble est vide
si X_A = X
    alors    // c'est une feuille de l'arbre
             // toutes les variables x_i sont assignées
             // chaque domaine de valeurs D_i contient une seule valeur
         si vérification_des_contraintes ( D , C ) = vrai
             alors    retourner « succès »
                          // chaque variable prend la valeur unique contenue
                          // dans son domaine de valeurs
             sinon    retourner « échec »
         fsi
    sinon    // créer les descendants du nœud courant et continuer le
             // parcours en profondeur
         soit une variable x_k choisie dans X\X_A
         pour toute valeur v_k,j du domaine de valeurs D_k de x_k faire
             soit Z un ensemble d'assignations variables / valeurs
             Z ← méthode_triviale ( X_A+{x_k} , D ← ( D_k={v_k,j} ) , X , C )
                      // Appel récursif avec les changements suivants :
                      // - ajout de x_k dans les variables assignées
                      // - domaine de valeurs de x_k réduit à 'v_k,j'
             si Z ≠ ∅
                 alors    retourner Z
                          // arrêt dès qu'une solution est trouvée
             fin_si
         fin_pour
         retourner ∅      // échec si aucune valeur de x_k n'est acceptable
fin_si
```

**Annexe – Résolution avec réduction des domaines de valeur**

```
fonction méthode_réduction_domaines
entrée :     < X_A , D , X , C >
                 ensemble des variables assignées X_A
                 ensemble des domaines de valeurs D = { D_i }
                                 // D_i est un singleton si x_i ∈ X_A
                 ensemble des variables X
                 définition des contraintes C
résultat :   { < x , v > }
                 ensemble de couples variable / valeur
             en cas de succès :
                 toutes les variables sont référencées et ont une valeur
             en cas d'échec :
                 l'ensemble est vide
si X_A = X
    alors    // c'est une feuille de l'arbre
             // toutes les variables x_i sont assignées
             // chaque domaine de valeurs D_i contient une seule valeur
         retourner succès      // chaque variable prend la valeur unique
                               // contenue dans son domaine de valeurs
                 // si on atteint une feuille
                 // et que toutes les réductions de domaines de valeurs
                 // ont été effectuées
                 // alors le nœud correspond à une solution au problème
    sinon    // créer les descendants du nœud courant et continuer le
             // parcours en profondeur
         soit une variable x_k choisie dans X\X_A
         X'_A ← X_A + { x_k }      // ajout de x_k dans les variables assignées
         pour toute valeur v_k,j du domaine de valeurs D_k de x_k faire
             D' ← ( D ← ( D_k={v_k,j} )
                      // domaine de valeurs de x_k réduit à 'v_k,j'
             D' ← réduction_des_domaines ( X'_A , D' , X , C )
                      // réduction des domaines de valeurs pour toutes
                      // les variables non assignées
             si aucun_domaine_vide ( D' )
                      // abandon de l'assignation x_k = v_k,j
                      // si un domaine de valeurs devient vide
                 alors
                      soit Z un ensemble d'assignations variables / valeurs
                      Z ← méthode_réduction_domaines ( X'_A , D' , X , C )
                      si Z ≠ ∅
                          alors    retourner Z
                                   // arrêt dès qu'une solution est trouvée
                      fin_si
             fin_si
         fin_pour
         retourner ∅      // échec si aucune valeur de x_k n'est acceptable
fin_si
```