6 Réalisation d'un solver de PPC

L'objectif de cette partie est d'implanter un solveur de CSP en Python. Les classes vous sont proposées sur Celene. Vous compléterez ces classes pour réaliser le solveur.

6.1 Contraintes et CSP

La classe Constraint est proposée pour représenter les contraintes. Cette classe est virtuelle car dépendant de la contrainte, les méthodes suivantes nécessitent d'être redéfinies :

— la méthode `unsat` qui teste si les domaines sont incompatibles avec la contrainte ;
— la méthode `checkSupport` qui vérifie si pour une variable donnée, une valeur donnée a un support dans la contrainte.

La contrainte Different. La classe Different définit la contrainte $\neq$ comme suit : une instance de cette classe ayant deux attributs v_1 et v_2 correspond à la contrainte $v_1 \neq v_2$.

Implanter les fonctions `unsat` et `checkSupport` pour cette classe.

La contrainte Superieur. Ecrire la classe Superior qui définit la contrainte binaire $>$.

La contrainte Table. Ecrire la classe Table qui définit une contrainte binaire donnée en extension. Les éléments de la table sont des couples qui définissent la contrainte.

De nouvelles contraintes. Comme extension, vous pouvez proposer de nouvelles contraintes binaires.

La classe CSP. Une instance de la classe CSP représente un CSP. Etudier la classe CSP proposée.

6.2 Solveur de CSP

La classe Solver réalise un solveur de CSP. Elle est définie partiellement, l'objectif est de la compléter avec les fonctions nécessaires pour les différents algorithmes de recherche (backtrack, backtrack avec forward checking et backtrack avec propagation de contraintes).

Chacune de ces fonctions prend comme argument le CSP à résoudre. Lorsque le solveur décide de créer des sous-cas (il fait des branchements), un nouveau CSP qui est une copie du CSP initial, mais avec des domaines plus restreints, sera créé pour chaque sous-cas. Cela simplifie le retour en arrière (backtrack).

6.2.1 Implantation des algorithmes de recherche

Définir les fonctions pour chacun des algorithmes de recherche suivant.

1. Backtrack.
2. Backtrack avec forward checking : pour cela il sera nécessaire d'implanter une fonction qui réalise le forward checking.
3. Backtrack avec propagation de contraintes : pour la propagation de contraintes, vous implanterez deux algorithmes AC-1 et AC-3.

Dans ces algorithmes de recherche, pour créer des branchements, on choisit la première variable non affectée et ordonner les valeurs dans le domaine de cette variable dans l'ordre fixé lors de la création du domaine.

6.2.2 Implantation des heuristiques

Dans chaque algorithme de recherche, lorsque le solveur fait un branchement, il choisit une variable x parmi les variables qui ne sont pas encore affectées et crée des branches suivant un ordre sur les valeurs dans le domaine restant de x. Le choix de la variable x peut être guidé par une heuristique de choix de variable. L'ordre sur les valeurs peut aussi être guidé par une heuristique de valeurs.

Implanter les heuristiques suivantes pour le choix de variables :

1. L'heuristique statique qui ordonne les variables dans l'ordre décroissant de leur degré.
2. L'heuristique dynamique qui choisit la variable ayant le plus petit domaine restant.

Implanter les heuristiques suivantes pour l'ordre de valeurs :

1. L'heuristique dynamique qui ordonne les valeurs dans l'ordre décroissant de leur nombre de supports, pour toutes les contraintes concernant la variable.
2. L'heuristique qui ordonne les valeurs dans l'ordre inverse de l'ordre précédent.