Programmation par contraintes

Le but de ce projet est d'écrire un solveur pour résoudre des problèmes de programmation par contraintes. On va traiter des problèmes avec un seul type de contraintes binaires : deux variables ne peuvent pas avoir la même valeur. On va donc pouvoir résoudre des problèmes tels que le sudoku ou les problèmes de coloration.

Pour résoudre un CSP, on doit combiner deux idées : utiliser une méthode de recherche et raisonner sur les contraintes. L'algorithme `ForwardChecking` effectue une sorte de recherche en profondeur d'abord – à chaque étape, on tente d'affecter une variable – et on raisonne sur les contraintes impliquant seulement la variable que l'on vient d'affecter.

Avec `ForwardChecking`, on vérifie donc l'arc cohérence [1] mais on limite la vérification à une petite partie des variables. On peut aussi vérifier l'arc cohérence pour toutes les variables : cela va coûter plus cher, mais peut restreindre plus vite le domaine des valeurs possibles pour les autres variables. On a présenté un algorithme simple appelé `ac3` pour rendre un CSP arc cohérent (ou découvrir que le problème n'a pas de solutions).

Le premier but du projet est d'implémenter `ForwardChecking` avec l'utilisation des trois heuristiques vues en cours pour choisir la variable suivante et l'ordre des valeurs. On cherchera à estimer le gain de ces heuristiques sur un jeu de données : ici, on vous fournit 50 jeux de sudoku.

Le second but est d'implémenter une variante de `ForwardChecking` qui, au lieu de raisonner simplement sur les contraintes impliquant la variable qui vient d'être affectée utilise `ac3` pour rendre le CSP arc-cohérent. On utilisera le même jeu de données pour savoir si cela a un impact sur les performances.

On fournit un code java pour modéliser une variable (classe `Variable<T>`, une contrainte binaire (classe `BinaryConstraint<T>`) et un problème de satisfaction par contraintes (CSP, classe `BinaryCSP<T>`). Le paramètre `T` représente le type des variables. On fournit aussi des classes pour les applications sur le sudoku (on aura besoin des classes `Sudoku` et `Digit`) et pour les problèmes de coloration (classe `GraphColouring`).

Travail à effectuer

Complétez la classe `BinaryCSP<T>` avec les méthodes `forwardCheck()` et `forwardCheckAC3()`. A priori, vous n'avez pas besoin de modifier les autres classes (à part les méthodes `main`).

Le fichier `sudokus.txt` contient 50 jeux, la classe `Sudoku` contient des méthodes pour lire ce fichier, créer une instance, et lancer l'exécution des méthodes `ForwardChecking` et `ForwardCheckingAC3`. On fournit aussi deux exemples de problème de coloration : `australia.txt` correspond à l'exemple vu en cours, et `gc.txt` contient un problème beaucoup plus difficile (instance connue sous le nom de `myciel7`). La classe `GraphColouring` contient des méthodes pour lire ces fichiers et appeler le solveur.

Evaluez le changement de performance dû à l'utilisation des heuristiques pour choisir les variables et les valeurs.

1. une variable est arc coherente si toutes les valeurs associees a la variable satisfont toutes les contraintes binaires ; le CSP est arc coherent si toutes les variables sont arc-cohérentes.

Evaluez les différences de performance entre le `ForwardChecking` classique et sa variante qui utilise `ac3`. Pouvait-on s'attendre à ces résultats ?

Soumission

1. Le projet est à rendre le **6 janvier 2017** sur la plateforme `mycourse`
2. le projet est à faire en binôme. Il y aura une soutenance commune pour les deux projets, conservez les groupes sauf en cas d'impossibilité (envoyez un email rapidement)
3. Vous devez soumettre l'ensemble des fichiers sources dans une archive `zip`.
4. Vous devez remettre un document au format pdf contenant vos conclusions et vos résultats. N'oubliez pas d'indiquez le nom des auteurs du travail.