X8II060

UFR Sciences et Techniques

M1 ALMA
2016-2017

Rapport projet Programmation par contrainte ?

Auteurs :
Montalvo Araya, Mario
Bégaudeau

Février 2017

Introduction

Le travail demander consiste à développer un solveur CPS de la forme X,D,C avec X l'ensemble des variables, D l'ensemble des domaines et C l'ensemble des contraintes.

Le but est de basé le solveur sur un algorithme de type branch-and-prune comme indiquer sur le sujet.

Pour ce rapport préliminaire le travaille demander était de résoudre le "problème des dames" vu en cours grâce à un algorithme de backtracking.

Enfin je signale que notre projet est codé en java et la classe à avec le main est BackTrackingQueen.java .

Le backtracking

La méthode de backtracking consiste essayer de réduire les domaines des différentes variables un par un en utilisant les contraintes du problème.

On vérifie à chaque "pas" de l'algorithme la validité des différentes valeurs que peut prendre une variable donnée grâce aux contrainte et si on trouve une valeur valide on avance.

Pour mieux expliquer avec un exemple sur un échiquier de 4*4 case :

La queen de la ligne 1 essaye de prendre la valeur 1 (la valeur indiquant sur quel case de la ligne elle ce trouve, comme donc le CM2). Les contraintes du problème ne l'interdise pas on continue.

La queen de la ligne 2 essaye de prendre la valeur 1. Pas possible à cause des contraintes donc on essaye la valeur suivante. Même problème avec la valeur 2. Et enfin pour la valeur 3 les contraintes l'autorise donc on continue et ainsi de suite.

Si on arrive à un ensemble de domaines réduit à une valeur pour chaque variable sans problème avec les contraintes alors on a une solution.

Avec cette méthode il n'y a pas de réduction des domaines à l'avance on ce contente de vérifié les contraintes à chaque "pas".

Les domaines

La représentation

Nous avons choisi de représenter les domaines comme un ensemble ordonner d'entier pour le problème des dames.

Comme nous utilisons le java nous avons utilisé des **TreeSet<Integer>**. En effet cette structure de donnée est naturellement ordonné.

Les opérations

Nous avons implémenté différentes opérations sur les domaines nous allons les listées en indiquant leur complexités.

- **La copie** : On utilise pour cela un constructeur par recopie en java. Nous ne somme pas sûr du coût de cette fonction mais il s'agit d'un classique constructeur par recopie : public Domain (TreeSet<Integer> valeurs) this.valeurs=valeurs ;
- **accès aux valeurs** : Comme nous utilisons des TreeSet nous ne pouvons pas accéder directement aux valeur. Nous somme obliger de parcourir le TreeSet jusqu'à la valeur choisi comme une liste chaînée

donc nous somme en **O(n)**. Cependant comme avec le backtracking nous voulons parcourir tout le domaine à chaque fois cela ne pose pas problème.
— **suppressions aux bornes** : Elle ce fait en temps constant avec les fonctions déjà implémenter des TreeSet **O(1)**
— **suppression valeur quelconque** : Elle ce fait aussi grâce aux fonctions des TreeSet en **O(log n)**

Les noeuds

La représentation

Pour l'instant nous représentons un noeud comme une ArrayList de domaine. Cela permet un accès rapide au différents domaine.

Les opérations

Nous avons implémenté différentes opérations sur les noeud nous allons les lister en indiquant leur complexités.

Les opérations

Nous avons implémenté différentes opérations sur les noeuds nous allons les lister en indiquant leur complexités.
— **La copie** : On utilise pour cela un constructeur par recopie en java comme pour les domaines. Nous ne somme pas sur du coût de cette fonction mais il sagit d'un classique constructeur par recopie.
— **accès aux domaines** : Comme nous utilisons des ArrayList l'accès à un des domaines ce fait en temps constant **O(1)**.
— **modification** : Pareil comme nous utilisons des ArrayList nous faisons cela en temps constant **O(1)**
— **Affectation d'un domaine** : Pour rajouter un domaine nous le rajoutons à la fin de l'ArrayList des domaines en faisant un simple add() **O(1)**

l'algorithme

Nous allons pas re-expliquer la méthode de backtracking utilisé mais juste discuté de la complexité de l'algorithme en lui même.

Cette méthode a une très mauvaise complexité. En effet la complexité est en **O(n !)**.

En effet il faut voir comme il est décrit dans le cours nous parcourons l'arbre de possibilité en entier jusqu'à toutes les feuilles. Sachant que l'on considère une "impasse" et une solution comme une feuille de l'arbre.

Conclusion

En implémentant la méthode de backtracking nous avons très vite pu voir ses limites. En effet avec la complexité en $\mathbf{O(n!)}$ l'algorithme met déjà pas mal de temps pour un échiquier de 15 case.

Il est donc nécessaire que nous réussissions à appliquer les méthodes de contraction vu en cours pour arriver à un résolutions des problèmes en un temps raisonnable.