DS 2013 Programmation par contraintes

Damien Crémilleux

5 juin 2014

1 Question de cours

1.1 Question 1.1

En Prolog les contraintes sont passives. En effet, il est possible d'exprimer une contrainte (comme X¡Y), mais Prolog ne pourra pas déduire de valeur à partir de cette contrainte (par exemple si l'on a (X or Y, Y=0) Prolog ne pourra pas inférer X=1). La contrainte exprimée servira uniquement à tester les réponses obtenues par la suite (comme X=2,Y=1, X¿Y). Ainsi, on ne peut pas inférer de valeur à l'aide de contrainte passive, d'où la nécessité d'utiliser des solveurs avec la propagation des contraintes.

1.2 Question 1.2

Dans un solveur de contraintes, la propagation et le labeling s'alternent. La propagation consiste à appliquer les contraintes et le labeling consiste à énumérer les valeurs pour une variables. Ainsi le solveur va propager les contraintes pour éliminer les valeurs incohérentes, puis on donne une valeur à une variable, et on boucle ainsi pour générer les solutions du problème.

1.3 Question 1.3

Les algorithmes de propagation dits d'*hyper arc consistency* sont sont NP-complets et donc très couteux. Les algorithmes de bornes sont plus efficaces car ils se rapprochent de la solution par une propagation rapide (on n'enlève pas toutes les valeurs incohérentes, mais cela permet de gérer des intervalles). La suppression des valeurs incohérentes est ensuite réalisée à l'aide du labeling.

1.4 Question 1.4

L'approche *generate and test* consiste à énumérer toutes les solutions possibles, et à vérifier que les contraintes sont vérifiées. Cette méthode peut être très lourde (l'arbre de recherche est immense) lorqu'il y a beacoup de variables. En outre, cette méthode est lente lorsque l'on cherche toutes les solutions ou lorsque l'on cherche à prouver qu'il n'existe pas de solution. L'utilisation de cette méthodes est donc valable pour les problèmes de petite taille. L'approche *constrain and generate* consiste à propager les contraintes, puis à effectuer le labeling sur les valeurs restantes. Cette approche permet de résoudre des problèmes plus gros et d'effectuer plusieurs tactiques pour trouver les solutions satisfaisantes. Elle recquiert un solveur de contrainte.

1.5 Question 1.5

La génération d'un emplois du temps est un problème de contrainte. En effet, il n'existe pas d'algorithme ou de méthodes pour résoudre ce problème. En outre, différentes stratégies et heuristiques (comme placer d'abord les cours avec le plus d'élèves, etc) peuvent être mises en place pour aboutir à un résultat. Enfin, les contraintes sont omniprésentes dans ce problème (les élèves ne peuvent pas être à deux endroits en même temps, un seul prof par matière, etc).

2 Problème

2.1 Question 2.1

```
% DS 2013 - Damien Cremilleux

:-lib(ic).

solve(Tab) :-
        tabPersonne(Tab),
        poserContraintes(Tab),
        labeling(Tab),
        printTab(Tab).

tabPersonne(Tab) :-
        dim(Tab,[4,4]),
        Tab :: 0..1.

poserContraintes(Tab) :-
        contrainte1(Tab),
        contrainte2(Tab),
        contrainte3(Tab),
        contrainte4(Tab),
        contrainte5(Tab),
        contrainte6(Tab),
        contrainte7(Tab).

contrainte1(Tab) :-
        Case is Tab[4,3],
        Case #= 1.

contrainte2(Tab) :-
        Case is Tab[2,4],
        Case #= 0.

contrainte3(Tab) :-
        Case is Tab[3,1],
        Case #= 0.

contrainte4(Tab) :-
```

```
        (multifor([I,J],[1,1],[4,4]),
         param(Tab)
        do
            Case1 is Tab[I,J],
            Case2 is Tab[J,I],
            (Case1 #= 0) => (Case2 #= 0)
        ).

contrainte5(Tab) :-
        (for(I, 1, 4),
         param(Tab)
        do
            Case1 is Tab[I,2],
            Case2 is Tab[1,I],
            (Case1 #= 1) => (Case2 #= 1)
        ).

contrainte6(Tab) :-
        (for(I, 1, 4),
         param(Tab)
        do
            Case1 is Tab[2,I],
            Case2 is Tab[4,I],
            (Case1 #= 1) => (Case2 #= 1)
        ).

contrainte7(Tab) :-
        (for(I, 1, 4),
         param(Tab)
        do
            Case1 is Tab[I,1],
            Case2 is Tab[I,2],
            Case3 is Tab[I,3],
            Case4 is Tab[I,4],
            Case1 or Case2 or Case3 or Case4
        ).

printTab(Tab) :-
        dim(Tab,[4,4]),
        (for(I, 1, 4), param(Tab) do
            (for(J, 1, 4), param(Tab, I) do
                X is Tab[I,J],
                (var(X) -> write("_"); printf("_%2d",[X]))
            ),
            nl
        ),
        nl.
```

2.2 Question 2.2

2.3 Question 2.3

Lors du TP de Prolog, la solution obtenue était plus longue du point de vue temps d'exécution. En effet la solution en Prolog repose sur l'approche *generate and test* car Prolog n'a que des contraintes passives. En outre, il est plus facile d'exprimer les contraintes et donc de programmer ce problèmes en CLP. Enfin, le code obtenu est plus lisble.

2.4 Question 2.4

À cause de la proposition 4, on constate une symétrie dans le tableau : case[i,j] = case[j,i]. On peut donc poser une contrainte supplémentaire afin d'éliminer cette symétrie et d'accélérer la recherche de solution.

2.5 Question 2.5

2.6 Question 2.6

Ici, il n'y a pas de raison de raffiner le labeling.