Département Génie Informatique et Génie Logiciel

INF8215 – Intelligence artificielle : méthodes et algorithmes

Laboratoire n°2

Prolog, programmation logique, programmation par contraintes

Soumis par

Djemai Mohamed 1911666

Bettache Lyes Heythem 1923715

MELLAH Brahim Redouane1912555

En date du
11 / 11 / 2018

Exercice 1 : Détective

L'objectif de cet exercice est de déterminer un résultat a partir d'une base de connaissance bien définies. En utilisant Minizinc.

On suppose qu'on a 5 nationalités différentes, 5 métiers différents, 5 boissons différentes, 5 animaux différents, et 5 couleurs de maisons différentes. En prenant cette considération, pour permettre à minizinc de résoudre le problème et donne à chaque personne 5 caractéristiques différentes.

On définit les variables et le domaine de chaque variable. Les variables prennent des valeurs entre 1 et 5, de sorte que toutes les variables du modèle sont des variables de décision.

On définit les faits donnés comme contraintes et en ajoutant la supposition précédente comme contrainte. On est capable de résoudre le problème de satisfaction de contrainte.

Le tableau suivant illustre l'assignation des valeurs aux variables par minizinc, on voit que l'eau prend la valeur 1, donc le norvegien qui boit de l'eau parce que lui aussi prend la valeur 1. Pour le zebre, il prend la valeur 5, et donc le japonais qui le possède.

1	2	3	4	5
Norvegien	Ukrainien	Anglais	Espagnol	Japonais
Diplomate	Medecin	Sculpteur	Violoniste	Acrobate
Eau	The	lait	Jus	Cafe
Renard	Cheval	escargot	Chien	**Zebre**
jaune	bleu	rouge	blanc	vert

Exercice 2 : Round-Robin

1- Explication du programme

L'objectif est de trouver un calendrier adéquat pour un tournoi en respectant les critères suivants :

- Chaque équipe joue contre toutes les autres une seule fois exactement.
- On respecte les lieux des matches (soit à domicile soit à l'extérieur) qui sont définis dans un fichier N14a.dzn donné.
- Aucune équipe ne peut jouer 4 matchs successifs (ou plus) à domicile ni 4 matchs successifs (ou plus) à l'extérieur.

On commence par définir un tableau ‘Adversaire’ de taille [N, N-1], tel que chaque ligne représente une équipe, et chaque colonne représente un tour -‘Adversaire[i, j]’ représente l’adversaire de l’équipe $i^{ème}$ dans le $j^{ème}$ tour-. L’objectif est de remplir ce tableau avec tous les adversaires de toutes les équipes.

Les contraintes à définir dans Minizinc sont :

a) Une équipe ne peut pas jouer contre elle-même.
b) Si une équipe A joue contre une équipe B, alors l’équipe B joue contre l’équipe A.
c) Pour chaque équipe, tous les adversaires sont différents.

On définit aussi un autre tableau ‘Lieu’ de même taille que le tableau ‘Adversaire’, tel que chaque ligne représente une équipe, et chaque colonne représente un tour – ‘Lieu[i, j]’ représente le lieu où va être joué le matche de l’équipe $i^{ème}$ dans le $j^{ème}$ tour, soit à domicile, soit à l’extérieur-, on doit remplir ce tableau en respectant le critère de quatre matches successives maximale à domicile ou à l’extérieur.

Les contraintes à définir dans Minizinc sont :

a) Si l’équipe A joue contre une équipe B dans un tour K, alors le lieu de matche est défini dans le tableau PV comme : PV[A,B].
b) Dans chaque ligne du tableau ‘Lieu’, la somme des 4 cases successives doit être différentes de zéro, et différent de 4, comme ça on est sûr qu’il n’y aura pas de 4 matchs successifs (ou plus) à domicile ni 4 matchs successifs (ou plus) à l’extérieur.

2- Explication d’une symétrie dans le problème.

Les solutions symétriques à coûts égaux sont une source de surcharge pour la recherche d'optimisation. Dans notre problème, si on a une solution A, on peut facilement obtenir une autre solution B par inversion des colonnes des tableaux ‘Adversaire et Lieu’ (au lieux de commencer le tournoi par le tour_1 jusqu’au tour_N-1, on commence par le tour_N-1 jusqu’au tour_1).

Pour braiser cette symétrie, on ajoute la contrainte redondante entre le tour_1 et le tour_N-1 d’une seule équipe :

Adveraire[1,1] < Adversaire[1,N-1]

Le temps de résolution est supposé être petit, car en braisant la symétrie, on réduire le nombre de recherches nécessaires pour résoudre le problème.

Cependant dans ce problème on a pratiquement le même temps de résolution (environs `1s`, `200ms`), peut-être parce que la taille de problème est petite (par rapport à un processeur i5-8g).

3- Contrainte globale

Pour respecter l'exigence du nombre de matchs à domicile/à l'extérieur, on peut appliquer une contrainte globale (alldifferent), pour dire que toutes 4 cases successives dans toutes les lignes doivent être différent de 4 et de 0 simultanément.

alldifferent([(venue[i,k]+venue[i,k+1]+venue[i,k+2]+venue[i,k+3]),0,4]);

L'usage d'une contrainte globale est meilleur car :

- Elle simplifie grandement la modélisation du problème.
- L'algorithme de résolution peut résoudre le problème d'une façon plus efficace.

Exercice 3 : Cours à prendre

On désire développer un programme *Prolog* qui donne, pour un cours donné, tous les cours à suivre avant d'être éligible de le prendre.

Au début nous avons ajouté notre base de connaissances des faits (cours(X)), et aussi les faits qui liées les cours entre eux (prerequis(X, Y), corequis(X,Y)). Le programme parcourt les faits de façon récursive, et stocke les cours à prendre dans une liste. On utilise aussi la fonction setof(), qui permet à prolog de retourner toutes les solutions trouvées et pas juste la premiere solution trouvée, aussi il permet de supprimer les cours doubles dans la liste. Notre programme commence par appelé la règle coursAPrendreComplet(C, L), cette règle appelle la règle coursRequis(A, B) dans le cas ou on demande la liste des cours a prendre pour un cours corequis avec un autre cours de façon récursive. Dans le cas où on demande la liste des cours à prendre pour un cours corequis on commence par déterminer le cours avec qui il est corequis avec la règle coursCRequis() et détermine sa liste, Parce que la liste de cours à prendre pour un cours corequis est la même.

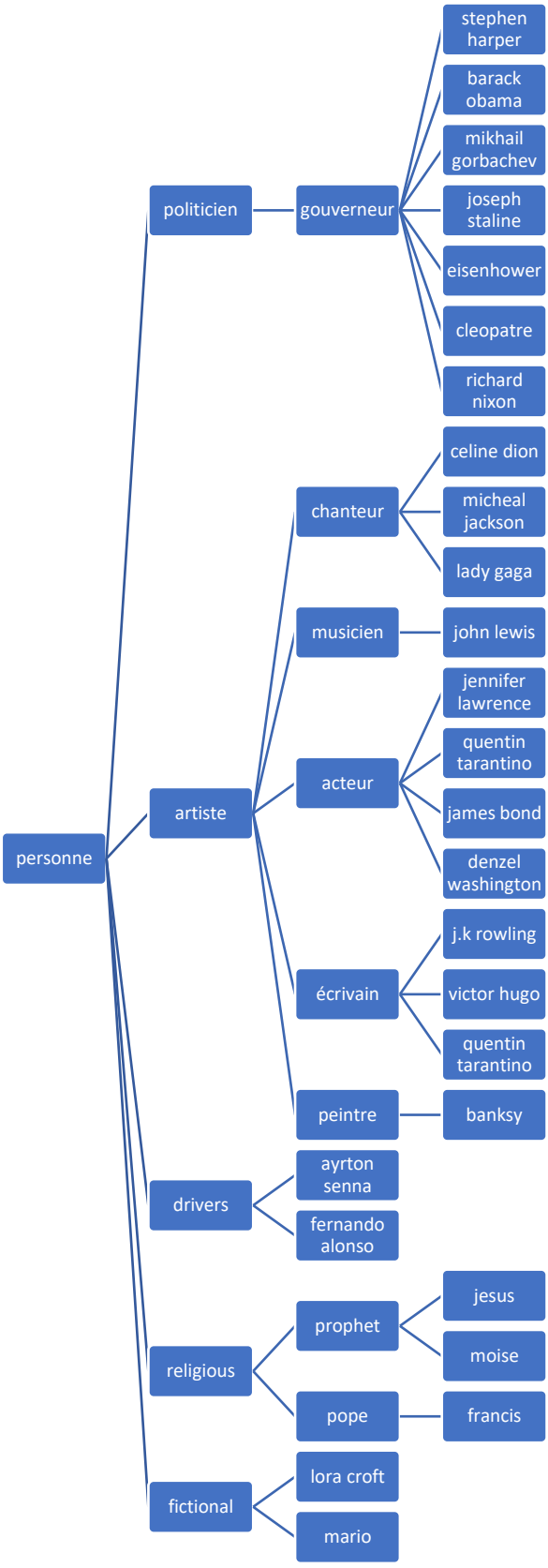
Exercice 4 : Akinator

La recherche se fait sous forme d'un arbre de recherche, comme montré dans les figures suivantes pour le programme des personnes, et le programme des objets. A chaque nœud on pose une question à l'utilisateur pour donner l'information complémentaire pour pouvoir continuer la recherche dans l'arbre jusqu'à trouver la solution. L'algorithme fait la recherche en profondeur.

Les listes de personnes et d'objets ne représentent que les feuilles de l'arbre de recherche, donc on est obligé de construire la base de connaissances pour guider la recherche en mettant des règles représentent les nœuds intermédiaires. Pour le faire on classifie les listes sur plusieurs catégories pour arriver à la solution en posant le minimum de questions. Les figures précédentes montrent bien les classes de chaque liste. Par exemple pour, on peut mettre les chanteurs et les acteurs sous la classe artiste. Aussi, on peut mettre la cuisinière et l'aspirateur sous la classe des appareils électroménager. L'algorithme est capable de trouver l'appartenance d'une personne a une certaine classe en interrogeant directement la base de connaissances. Par exemple en mettant artiste(micheal_jackson), le programme retourne 'true', directement sans poser des questions.

On définit les feuilles de l'arbres comme des clauses simples, et des règles pour classifier les personnes et les objets, pour avoir une base de connaissances consistante. Prolog exécute le programme du haut vers le bas, et applique l'algorithme de retour en arrière. En écrivant personne(X), prolog cherche dans tout le programme en essayant d'unifier X, il retourne la première solution trouvée. En chaque nœud, on pose une question l'utilisateur pour passer au prochain nœud dans l'arbre. Par exemple en posant la question ' X gouverne ?', et la réponse est 'yes', alors on passe au nœud politicien(X), si la réponse est 'non', on pose la question qui est 'X est un artiste ?' et ainsi de suite jusqu'à quand arrive à unifier X. et retourner le résultat trouvé.

Arbre de recherche des personnes



Arbre de recherche des objets