Rapport Labo 2

Présenté à : Mohammed Najib Haouas

INF8215 : Intelligence artificielle, méthodes et algorithmes

Présenté par : Fabrice Charbonneau – 1798074

Antoine Daigneault-Demers - 1879075

Polytechnique Montréal

2018-11-11

Exercice 1 - Detective

Afin de répondre aux deux questions posées, nous avons donné une valeur allant de 0 à 4 pour chacune des catégories, soit les nationalités, la couleur des maisons, les animaux, les boissons et les travails. Chacune des catégories a été associée à un tableau (Array) afin de plus facilement indiquer des contraintes pour chacune d'entre elles.

Nous avons également mis en place des prédicats pour l'emplacement des maisons, car les énoncés n'indiquent pas toujours directement leur position.

Nous avons ensuite indiqué toutes les contraintes, en plus du fait que les éléments de chacune des catégories sont uniques.

Finalement, la commande «solve satisfy» a été appelée pour trouver une solution qui respecte les contraintes et nous avons affichés les résultats sous la forme d'un tableau. Chaque position (i, j) désigne dans quelle maison (0 à 4) se trouve la personne, objet ou travail.

Avec nos résultats, nous pouvons voir que la personne qui boit de l'eau est le Norvégien. Cette information vient du fait que l'eau (à l'index 4 du tableau) se trouve dans la maison 0, tout comme le Norvégien (index 3 du tableau). Le propriétaire du zèbre est quant à lui le Japonais. En effet, le zèbre, qui se trouve à l'index 3 dans le tableau, est dans la maison 4 tout comme le Japonais (index 4 du tableau).

Exercice 2 - Round Robin

Le but de cet exercice est de fournir un horaire valide pour un tournoi de 14 équipes. Celles-ci ne doivent jouer qu'une seule fois contre chacune des autres équipes. L'horaire est considéré valide uniquement si les équipes ne jouent jamais 4 matchs consécutifs à domicile ou 4 matchs consécutifs à l'étranger.

Pour résoudre se problème, nous avons créé une matrice de taille 14 par 13. 14, car il faut les matchs des 14 équipes, et 13, car chaque équipe joue 13 matchs, soit contre toutes les équipes sauf elle-même.

Nous avons ensuite imposé différentes contraintes pour être certains que la solution trouvée donnait un horaire valide. Premièrement, chaque équipe doit jouer contre chacune des autres et ce une seule fois. La première contrainte s'assure que toutes les équipes contre qui l'équipe i joue sont différentes. La deuxième contrainte s'assure qu'une équipe n'a pas 2 matchs au même moment. La troisième contrainte sert à s'assurer que lorsqu'une équipe A joue contre une équipe B, l'équipe B joue contre l'équipe A. La quatrième sert à vérifier qu'une équipe n'a pas de match contre elle-même. La cinquième contrainte sert à s'assurer qu'une équipe ne joue pas 4 matchs consécutifs à domicile ou à l'étranger.

La symétrie que nous avons identifiée est que si on inverse complètement l'ordre des matchs de chaque équipe (son 1er match devient son dernier, son 2e l'avant-dernier, etc), le résultat

restera le même. Afin d'éliminer les solutions qui sont symétriques à une autre déjà trouvée, nous avons forcé la solution à être dans un ordre en particulier en ajoutant la contrainte suivante: matchs[3,1] > matchs[3,13]. Cela empêche le programme de considérer une solution qui n'est que l'ordre inverse d'une solution déjà considérée. On compare alors les équipes contre qui joue l'équipe 3 aux matchs 1 et 13 (le premier et le dernier). Le choix de l'équipe 3 a été fait en comparant les temps de résolutions avec la même contrainte pour chacune des équipes. Le programme minizinc avait le meilleur temps de résolution avec celle-ci. Cela est probablement dû à la façon dont minizinc gère les variables et optimise la résolution de problèmes à l'interne. L'ajout de cette contrainte a permis de passer d'environ 54 secondes à moins de 500ms, ce qui donne un facteur d'accélération d'environ 108. Il est à noter que ces temps peuvent varier d'une machine à l'autre.

Bonus:
Pour une équipe donnée, le nombre de matchs à domicile ou à l'extérieur ne doit pas être plus de 3 consécutifs. On peut alors penser à un cas limite où solution contient une équipe qui a comme séquence 0010001000100, où le nombre minimal de fois où il joue à domicile ou à l'extérieur est de 3. Donc, il serait possible d'ajouter une contrainte globale comme at_most qui forcerait toutes les équipes à contenir au moins 3 fois où elle joue à domicile et 3 fois où elle joue à l'extérieur. Son usage pourrait diminuer le temps de résolution, puisque tous les cas où une équipe joue moins de 3 fois à domicile où à l'extérieur seraient ignorés, et ce, sans avoir à vérifier la contrainte qui vérifie de façon directe si une équipe joue plus de 3 fois à l'extérieur ou à domicile consécutives. Par cela, on n'entend pas que cette dernière ne sera plus utile, simplement qu'une certaine partie des solutions possibles sera éliminée plus rapidement avec la nouvelle contrainte globale.

Exercice 3 – Course requirements

Avec le fichier exercice3.pl, il est possible de trouver les cours co-requis à un cours précis en appelant la méthode courseCoRequires(cours, X). Par exemple, les cours corequis à inf1600 peuvent être obtenus avec la commande courseCoRequires(inf1600, X).

Il est aussi possible de trouver les cours qui lui sont directement prérequis avec la fonction coursePreRequires(cours, X). Par exemple, les cours directement prérequis à inf1600 peuvent être trouvés avec la commande coursePreRequires(inf1600, X).

Finalement, il est possible de trouver tous les cours qui doivent avoir été complétés avant ou en même temps que le cours avec la méthode courseRequirements(cours, L). Par exemple, tous les cours qui doivent avoir été complétés avant ou en même temps que inf1600 peuvent être trouvés avec la commande courseRequirements(inf1600, L).

Exercice 4 - Akinator

Afin de pouvoir définir n'importe quelle personne ou objet d'un ensemble de personnes ou d'objets donné, nous avons considéré l'algorithme suivant:

-Avoir une base de connaissances pouvant définir n'importe quelle entité de façon unique avec des booleans
-Avoir une banque d'entités encore possibles (B)
-Tant que B contient plus d'une entité:
    -Recherche dans B l'attribut qui possède le plus grand ratio de différence (l'attribut qui une fois déterminé élimine en moyenne la plus grande quantité d'entités, que ce soit vrai ou faux)
    -Pose la question en lien avec l'attribut trouvé
    -Élimine les entités de B qui ne sont plus possibles
-La réponse est la seule entité qui reste dans B

Afin de faciliter l'implémentation et la compréhension du code en Prolog, nous avons fait le choix d'appliquer cet algorithme à la main afin de créer un arbre statique des 22 personnes ou 22 objets, traduit ensuite dans le code. Un exemple d'arbre pour les objets est présenté dans l'annexe. Le principe est qu'à chaque question, la banque d'entités encore possible est divisée en deux parties de façon la plus égale possible, pour se rapprocher d'une complexité O(log(n)). Les feuilles de l'arbre correspondent à une réponse, c'est-à-dire lorsqu'il ne reste qu'une entité possible. Afin de réutiliser des questions, certaines divisions dans les arbres sont des questions utilisées dans d'autres divisions.

L'implémentation en Prolog est basée sur l'exemple donné dans l'énoncé. Chaque question est donc construite de la même façon, et l'arbre est construit avec des clauses. Finalement, toutes les informations sur chaque entité sont présentes afin de respecter la contrainte de pouvoir questionner la base de connaissances sur n'importe quel objet ou personne.

Annexe

Arbre Akinator objets