INF8215 - Intelligence Artificielle : Méthodes et Algorithmes

Compte Rendu TP 2

Amine Bellahsen, Sanae Lotfi, Théo Moins
1965554, 1968682 , 1971821

November 12, 2018

Solution - Exercice 1 : Détective

Dans ce problème, on considère que chacun des habitants possèdent exactement une nationalité, une couleur de maison, une boisson préférée, un métier, et un animal de compagnie. Afin d'avoir un référentiel indépendant de ces caractéristiques, nous avons choisi de déterminer la position de la maison pour toutes les caractéristiques. Par exemple, l'anglais est dans la 3e maison, la maison rouge est la 1ère maison, etc.

On considère ainsi le problème de satisfaction de contraintes suivant :

- Variables : chacune des caractéristiques (Espagnol, Acrobate, Zèbre...) est une variable pouvant aller de 1 à 5, ce qui correspond à la position de la maison correspondante.

  On regroupe les caractéristiques en 5 groupes :

  1. Les nationalités : N_i, $\forall i \in 1..5$, avec $N_1 = Anglais$, $N_2 = Ukrainien$, etc.
  2. Les couleurs de maison : C_i, $\forall i \in 1..5$
  3. La boisson préféré : B_i, $\forall i \in 1..5$
  4. Le métier : M_i, $\forall i \in 1..5$
  5. L'animal de compagnie : A_i, $\forall i \in 1..5$

- Domaine : pour les 25 variables, le domaine est $\{1, 2, 3, 4, 5\}$.

- Contraintes :

  $all\_different(Nationalites)$ (Les 5 nationalités sont présents exactement une fois)

  $all\_different(Couleurs)$ (Les 5 couleurs sont présentes exactement une fois)

  $all\_different(Boissons)$ (Les 5 boissons sont présents exactement une fois)

  $all\_different(Metiers)$ (Les 5 métiers sont présents exactement une fois)

  $all\_different(Animaux)$ (Les 5 animaux sont présents exactement une fois)

  Puis on ajoute en contraintes les 14 affirmations, par exemple:

  "L'anglais habite à la maison rouge." devient $constraint\ Anglais = Rouge$.
  "Le médecin habite dans une maison voisine de celle où demeure le propriétaire du renard." devient $constraint\ abs(Medecin - Renard) = 1$;

Pour bien gérer les exemples qui requierent d'indiquer une certaine position (gauche, droite), nous avons choisi de considérer que nous partant de gauche à droite. par exemple:
La contrainte "La maison verte est immédiatement à droite de la maison blanche." devient *constraint* $Vert = Blanc + 1$;
C'est avec cette modélisation que nous avons ensuite résolu le problème sur MiniZn.
Le résultat obtenu pour les deux questions est:

```
Position des maisons: [Anglais, Ukrainien, Espagnol, Japonais, Norvegien]
[3, 2, 4, 5, 1]
Le buveur d'eau habite a la 1e maison en partant de la gauche.
Le propriétaire du zèbre habite a la 5e maison en partant de la gauche.
----------
Finished in 268msec
```

Ainsi on peut dire que:

- C'est le norvégien, habitant à la 1ère maison, qui boit l'eau.
- C'est le japonais, habitant à la 5ème maison, qui est propriétaire du zèbre.

Solution - Exercice 2 : Round-Robin

On considère le problème de satisfaction de contraintes suivant :

- Variables : $M_{i,j}$, $1 \leq i \leq N$, $1 \leq j \leq N-1$, où $M_{i,j}$ est la variable qui correspond à l'adversaire du joueur i au j^e match.
- Domaine : $\forall (i,j)$, $M_{i,j} \in 1..N$
- Contraintes :

  Chaque équipe i doit jouer les autres équipes exactement une fois :

  $\forall i, all\_different(\{M_{i,j}, j = 1..N-1\})$

  Chaque équipe ne peut pas faire deux matchs pendant la même journée :

  $\forall j, all\_different(\{M_{i,j}, i = 1..N\})$

  Chaque équipe ne peut pas jouer contre lui-même :

$\forall(i,j), M_{i,j} \neq i$

Chaque équipe ne peut pas faire 4 matchs consécutifs à domicile ou à l'extérieur :

$\forall(i,j), location[i, M_{i,j}] + location[i, M_{i,j+1}] + location[i, M_{i,j+2}] + location[i, M_{i,j+3}] < 4$

$\forall(i,j), location[i, M_{i,j}] + location[i, M_{i,j+1}] + location[i, M_{i,j+2}] + location[i, M_{i,j+3}] > 0$

L'adversaire de l'adversaire de l'équipe i doit être l'équipe i :

$\forall(i,j), M_{M_{i,j},j} = i$

L'objectif est donc de trouver la matrice M respectant toutes ces contraintes.

En traduisant ce problème CSP sur MiniZn, le temps d'exécution est relativement long (environ 2 minutes).

Cependant, on observe une symétrie dans notre problème : si la matrice $(M_{i,j})$ est solution, la matrice obtenue en faisant $(M_{i,N-1-j})$ le sera aussi (cela revient à considérer le calendrier "à l'envers", c'est à dire que la dernière journée se fera en premier, l'avant dernière en deuxième, etc). On peut facilement vérifier que chacune des contraintes sera respectées.

Pour briser la symétrie, nous pouvons ajouter la contrainte redondante suivante :

$$M_{1,1} < M_{1,13}$$

Ainsi, en ajoutant cette contrainte, $(M_{i,N-1-j})$ ne sera plus solution si $(M_{i,j})$ l'est.

On brise alors la symétrie et l'on impose un ordre ; cela permet de ne pas explorer l'espace de solutions symétriques, et d'aboutir plus rapidement à une solution (on ne cherche pas à reprouver des états dont le symétrique était déjà prouvé faux).

Le temps d'exécution de l'algorithme passe d'environ 2 minutes à une centaine de millisecondes.

Un exemple de la sortie de notre programme est donné par la figure 1.

	$Rd1$	$Rd2$	$Rd3$	$Rd4$	$Rd5$	$Rd6$	$Rd7$	$Rd8$	$Rd9$	$Rd10$	$Rd11$	$Rd12$	$Rd13$
$Team1$	2	7	12	8	10	6	4	5	13	9	11	14	3
$Team2$	1	5	4	6	3	7	9	12	8	11	10	13	14
$Team3$	13	11	10	4	2	9	8	6	5	7	14	12	1
$Team4$	12	6	2	3	5	14	1	7	10	8	9	11	13
$Team5$	11	2	7	9	4	13	14	1	3	6	8	10	12
$Team6$	10	4	8	2	13	1	7	3	14	5	12	9	11
$Team7$	9	1	5	14	12	2	6	4	11	3	13	8	10
$Team8$	14	12	6	1	11	10	3	13	2	4	5	7	9
$Team9$	7	10	13	5	14	3	2	11	12	1	4	6	8
$Team10$	6	9	3	12	1	8	11	14	4	13	2	5	7
$Team11$	5	3	14	13	8	12	10	9	7	2	1	4	6
$Team12$	4	8	1	10	7	11	13	2	9	14	6	3	5
$Team13$	3	14	9	11	6	5	12	8	1	10	7	2	4
$Team14$	8	13	11	7	9	4	5	10	6	12	3	1	2

Figure 1: Exemple d'un calendrier qui respecte les contraintes du problème, où chaque ligne i contient les adversaires du joueur i et chaque colonne j représente le tour j.(Rd = Round)

BONUS

Pour faire respecter le nombre de matchs à domicile/à l'extérieur, on peut construire une contrainte globale régulière ("Regular" dans Prolog), c'est à dire représenter la contrainte par un automate fini déterministe. Nous avons défini l'automate fini déterministe correspondant à ce problème dans la figure 2. Le tableau correspondant à l'automate est donné par la figure 3.

La contrainte globale "Regular" va nous permettre de remplacer les 2 contraintes mentionnées précédemment, assurant que chaque équipe ne peut pas faire 4 matchs consécutifs à domicile ou à l'extérieur.La réduction du nombres des contraintes est un 1er avantage pour l'utilisation des contraintes globales. Utiliser cette contrainte globale permettra aussi de faciliter la modélisation (surtout si on décide par exemple d'avoir une contrainte plus difficile à modéliser concernant le nombre de match maximal à jouer à domicile ou à l'extérieur). Par ailleurs, la résolution sera accélérée puisqu'on vérifie en même temps si on tient compte de plusieurs contraintes simples. Le plus grand avantage réside dans le fait que les algo-

rithmes de propagation des contraintes globales sont efficaces et rapides. La contrainte "Regular" en particulier se base sur la définition d'un automate fini, et on sait que les automates finis déterministes ont plusieurs aspects avantageux : simplicité de leur définition, facilité de manipulation, aisance de la programmation informatique et élégance des propriétés mathématiques entre-autres. (ref: wikipedia)

En conclusion, en utilisant la contrainte globale "Regular", on s'attend à avoir un temps de résolution plus rapide et une modélisation plus simples et flexibles de notre problème.

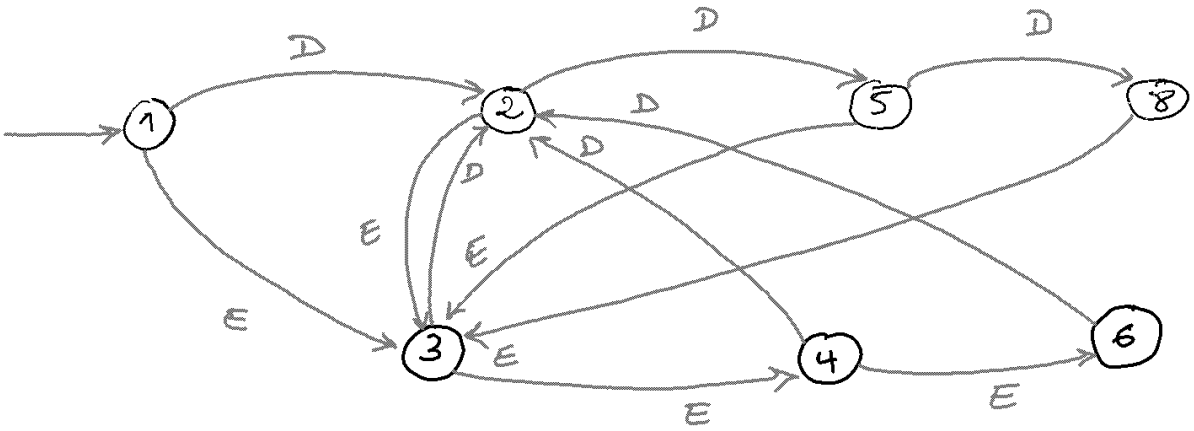

Figure 2: Automate fini déterministe correspondant au problème. (D: match à Domicile, E: match à l'extérieur)

	Dom	Ext
1	2	3
2	5	3
3	2	4
4	2	6
5	7	3
6	2	0
7	0	3

Figure 3: Tableau correspondant à l'automate, l'état 0 étant un état interdit

Solution - Exercice 3 : Cours à prendre

Dans cet exercice, nous possédons une liste de cours, leurs corequis directs, et leurs prérequis directs également. Nous désirons écrire un programme qui donne, pour un cours donné, tous les cours à suivre avant d'être éligible de le prendre.

Ainsi, la première étape serait donc d'indiquer au programme Prolog les cours dont nous disposons, leurs corequis et leurs prérequis. Nous avons pour cela crée les faits $cours(X)$, $corequis(X, Y)$ où X est un co-requis directement de Y, et ensuite $prerequis(X, Y)$ où X est un pré-requis directement de Y.

Désormais, nous avons besoin de règles nous permettant de récupérer tout les cours prérequis et corequis à n'importe quel cours. Pour cela nous définissons deux règles: $isprerequis(X, Y)$ puis $iscorequis(X, Y)$.

$isprerequis(X, Y)$ est vérifié si au moins l'une des trois règles suivantes est vérifiée :

- X est prerequis ou corequis directement de Y.

Exemple: log1000 est un corequis direct de inf1900, et prérequis direct de log2410.

- Il existe un cours Z dont X est un prérequis direct, et que $isprerequis(Z, Y)$ est bien vraie sachant la règle précédente.

Exemple: inf1005c est prérequis direct de log1000, et log1000 est corequis direct de inf1900.

- Il existe un cours Z dont X est corequis direct, et que $isprerequis(Z, Y)$ est bien vraie sachant les deux règles précédentes.

Exemple: log2810 est corequis de inf2010, if inf2010 est prérequis direct de inf2705.

$iscorequis(X, Y)$ est vérifié si au moins l'une des trois règles suivantes est vérifiée :

- Y est corequis direct de X.
Exemple: log2990 est corequis direct de inf2705 donc iscorequis(inf2705,log2990) est vraie.
- X et Y sont corequis direct de Z.
Exemple: mth1007 et log2990 sont corequis direct de inf2705.
- $isprerequis(X, Z)$ et $iscorequis(Z, Y)$ sont vérifiées.
Ces deux dernières règles servent à montrer qu'être corequis est symétrique, et si X et Y sont corequis direct de Z alors ils sont corequis entre eux.
Exemple: $isprerequis(inf1010, inf2010)$ et $iscorequis(inf2010, log2810)$ sont vraies.

Maintenant, nous devons définir la règle coursAPrendreComplet pour énumérer tous les cours à suivre avant d'être éligible de prendre un certain cours. Nous optons à retourner une liste comme résultat, pour cela nous utilisons la méthode $setof$. Nous utilisons la méthode *delete* qui nous permet de supprimer les doublons dans le résultat.

```
?- coursAPrendreComplet(log2990).
[inf1005c,inf1010,inf2010,inf2705,log2810,mth1007]
true.

?- coursAPrendreComplet(log2410).
[inf1005c,inf1010,inf1500,inf1600,inf1900,inf2205,log1000]
true.
```

Ensuite pour récupérer tous les corequis d'un cours, nous avons crée la règle $testcorequis(X, Y)$ qui permet de tester si X est corequis de Y, ou avoir la liste complète des corequis de Y (voir exemples).
Pour tester si un cours X est prerequis d'un autre cours Y, on peut utiliser la règle $testprerequis(X, Y)$. Contrairement à $testcorequis(X, Y)$, cette fonction contient un *cut* et ne permet donc pas d'avoir la liste complète des prerequis, puisque cette fonctionnalité est déjà assurée par $coursAPrendreComplet(X, Y)$
Il faut aussi noter que si un cours est un corequis, alors il est aussi prérequis.
Pour les corequis, si on désire avoir la liste complète ou vérifier, on fait comme suit:

```
?- testcorequis(X,log1000).
X = inf1600 ;
```

```
X = inf1900 ;
X = inf2205 ;
false.

?- testcorequis(mth1007,inf2705).
true ;
false.
```

Si on désire vérifier si un cours est prerequis d'un autre, on fait comme suit:

```
?- testprerequis(inf1010,inf2010).
true.
```

Il faut noter qui si un cours n'a aucun prérequis ni de corequis (Exemple inf1005c), alors la règle *coursAPrendreComplet* **renvoie une liste vide.**

```
?- coursAPrendreComplet(inf1005c).
[]
true.
```

Solution - Exercice 4 : Akinator

Pour la construction des deux questionnaires, nous avons tenu à respecter les deux critères notés dans le sujet de TP, à savoir la flexibilité pour l'ajout de nouvelles entités, tout en optimisant le nombre de questions à poser.

Or, les deux listes à considérer contenant 22 éléments, il est clair que le nombre optimal de questions à poser (puisque l'on peut répondre que par oui ou non) vaut $\lfloor log_2(22) \rfloor + 1$, soit un maximum de 5 questions à poser au pire des cas. En effet, une personne qui joue ce jeu a la même probabilité de penser à chacune des personnes dans la liste. Cela fait que la meilleure construction du questionnaire est telle que chaque question coupe l'espace des possibilités en deux, ce qui va permettre d'avoir un arbre binaire complet.

Néanmoins, il est difficile de trouver un questionnaire qui est optimal tout en ayant des questions généralisables, car cela demande de diviser en deux la liste des possibilités de la manière la plus égale possible, et ce à chaque étape du questionnaire. Or, la plupart des

caractéristiques intuitives qu'ont les éléments de la liste ne sont pas binaires, par exemple le métier (il y a plus que 2 métiers) ou la nationalité (il y a plus que 2 nationalités) pour les personnes. De plus, il est important de s'assurer qu'il est possible de répondre à nos questions pour n'importe quelle personne de la liste, si jamais un nouvel élément doit s'indexer à l'arbre. Par exemple, tous les personnages fictifs n'ont pas forcément de nationalité ou de date de naissance, donc on ne peut pas poser la question sur la nationalité à tout le monde.

Ainsi, les deux questionnaires que nous avons conçus posent au maximum 6 questions (au lieu de 5 s'il avait été optimal) pour deviner la personne ou l'objet, ce qui reste proche de la borne inférieur, tout en ayant l'avantage d'être généralisable facilement pour l'ajout de nouvelles entités, ce qui permet de construire une base de connaissances avec des catégories comme l'ensemble des hommes ou l'ensemble des artistes pour l'exemple des personnes et l'ensemble des équipements électroménagers pour les objets.

- Hypothèses pour la liste des personnes:
On devine la personne choisie par le joueur en posant, entre-autres, les question suivantes:

1. Est-ce que la personne est vivante? on considère que les personnages fictifs ne sont pas vivants (cela inclut James Bond, Mario et Lara Croft).

2. Est-ce que la personne est un artiste? on considère que les acteurs, les chanteurs, les réalisateurs, les écrivains et Banksey sont des artistes. On ne considère pas que Hideo Kojima est un artiste.

3. Est-ce que la personne est une femme? On considère que cette question est valide pour les personnages fictifs également. Par exemple, Mario est un homme, Lara Croft est une femme. On fait l'hypothèse que Banksey est un homme.

4. Est-ce que la personne a gouverné plus que 5 ans? même si cette question semble être difficile, nous utilisons cette question pour différencier entre Richard Nixon et Dwight D. Eisenhower puisque les deux ont été des présidents des états unis.

5. Est-ce que la personne a gouverné plus que 5 ans? même si cette question semble être difficile, nous utilisons cette question pour différencier entre Richard Nixon et Dwight D. Eisenhower puisque les deux ont été des présidents des états unis.

6. Est-ce qu'il s'agit d'une personnalité religieuse? ici on parle des personnalités religieuses connues. Par exemple Jesus, Moise et le Pape François.

Les autres questions ne contiennent pas d'ambiguïtés.

- Hypothèses pour la liste des objets:
On devine l'objet choisi par le joueur en posant, entre-autres, les question suivantes:

1. L'objet se trouve-t-il dans la cuisine ? On considère que le cactus, la lampe, le papier et le téléphone (entre autres) des objets qui ne se trouvent pas à la cuisine.
2. L'objet doit-il être branché pour fonctionner? On considère que le téléphone, la lampe et l'ordinateur font partie des objets qui doivent être branchés pour fonctionner, et que le piano par exemple ne fait pas partie de cette catégorie.
3. L'objet est-il portable a l'extérieur de la maison? On considère que le papier, la clé, le portefeuille et le sac-à-dos font pas de cette catégorie.
4. L'objet sert-il a ranger des billets? On considère que la réponse est oui pour le portefeuille uniquement.

Les autres questions ne contiennent pas d'ambiguïtés.

- Implémentation PROLOG :
Nous avons implémenté un algorithme pour deviner la personne lorsqu'on demande "personne(X).", en posant des questions auxquelles le joueur répond par oui ou non. Nous pouvons également vérifier si une personne X est: vivant(X), femme(X), homme(X), chanteur(X), chef_etat(X), acteur(X), ecrivain(X), realisateur(X) et pilote(X).

Le code permet aussi de deviner un objet lorsqu'on demande "objet(X).", en posant des questions par la suite. Nous pouvons également vérifier si un objet X est: cuisine(X) (objet se trouve à la cuisine), appareil_electromenager(X), vaisselle(X), meuble(X), nettoyant(X), nettoyant_electrique(X), nettoyant_sol(X)., couvert(X) (couverts de table), instrument_musique(X) ...etc.

Pour illustrer l'utilisation de notre code prolog, considérons les exemples suivants:

- Deviner "Dwight D. Eisenhower" avec notre code prolog:

```
?- personne(X).
_3502 est vivant(e) ? non.
```

```
_3502 gouverne un pays ? |: oui.
_3502 gouverne les USA ? |: oui.
_3502 a-t-il gouverne plus que 5 ans ? |: oui.

X = dwight_d_eisenhower .
```

- Chercher tous les artistes dans la base de connaissances:

```
?- artiste(X).
X = michael_jackson ;
X = lady_gaga ;
X = victor_hugo ;
X = j_k_rowling ;
X = banksy ;
X = quentin_tarantino ;
X = denzel_washington ;
X = jennifer_lawrence.
```

- Deviner l'objet "cactus" avec notre code prolog:

```
?- objet(X).
_4280 se trouve-t-il dans la cuisine ? non.
_4280 sert a nettoyer? |: non.
_4280 doit-il etre branche pour fonctionner? |: non.
_4280 est portable a l exterieur de la maison? |: non.
_4280 est un instrument de musique? |: non.
_4280 est une plante? |: oui.
X = cactus .
```

- Chercher tous les objets qui se trouvent à la cuisine:

```
?- cuisine(X).
X = fourchette ;
```

```
X = assiette ;
X = four ;
X = cuisiniere ;
X = cafetiere ;
X = grille_pain ;
X = table ;
X = casserole ;
X = detergent_a_vaisselle.
```