M1 - DAC
IAMSI
Intelligence Artificielle et
Manipulation Symbolique de l'Information

Année 2016-2017

Examen réparti n° 2

UPMC SORBONNE UNIVERSITÉS

Rappels : **Aucun document n'est autorisé.** Le barème n'est donné qu'à titre indicatif. Une annexe contenant des rappels de cours est donnée en fin de sujet. **Ce sujet comporte 4 pages.**

Exercice 1 Programmation en ASP – 5 points

1. Construire, en ASP, un programme $a(M, N, Q)$ qui calcule la valeur Q de la fonction d'Ackermann en fonction des entrées M et N.
   Remarque : rappelons que la fonction d'Ackermann se définit pour tout entier m et n comme suit : $\begin{cases} Ackermann(0,n) = n+1 \\ Ackermann(m+1,0) = Ackermann(m,1) \\ Ackermann(m+1,n+1) = Ackermann(m, Ackermann(m+1,n))) \end{cases}$
2. écrire en ASP un prédicat $gamme/1$ qui donne exactement $n+1$ ensembles réponses qui instancient chacun $gamme/1$ avec une valeur différente prise entre 0 et n.
   Remarque : par exemple pour $n = 4$ on aura cinq ensembles réponses : $S_0 = \{gamme(0)\}, S_1 = \{gamme(1)\}, S_2 = \{gamme(2)\}, S_3 = \{gamme(3)\} et S_4 = \{gamme(4)\}$.
3. Construire, toujours en ASP, un prédicat $valAckermann(m, n, q)$ qui renvoie toutes les valeurs de la fonction $Ackermann(m, n)$ pour toutes les valeurs de m et n telles que $0 \leq m \leq 3$ et $0 \leq n \leq 10$.
   Remarque : on pourra faire appel à deux prédicats $gamme\_m/1$ et $gamme\_n/1$ qui donneront $4*11 = 44$ ensembles réponses instanciant chacun $gamme\_m(M)$ et $gamme\_n(N)$ avec un couple différent de valeurs de M et de N telles que $0 \leq M \leq 3$ et $0 \leq N \leq 10$.

Exercice 2 Interprétation de trois programmes en ASP – 3 points

Donner tous les ensembles réponses des trois programmes suivants en détaillant et en justifiant tous les calculs.

$$\Pi_1 \begin{cases} a. \\ \neg b \leftarrow a, \ \textbf{not} \ \neg c. \\ \neg c \leftarrow \ \textbf{not} \ \neg b. \end{cases} \qquad \Pi_2 \begin{cases} 2\{p(a),\ p(b),\ p(c)\}3. \\ p(a) \leftarrow p(X). \end{cases} \qquad \Pi_3 \begin{cases} 3\{p,s,q,r\}3. \\ 1\{s,r\} \leftarrow p. \end{cases}$$

Exercice 3 "STRIPS" – 5 points

1. En quoi consiste la représentation STRIPS d'une action ?
2. Quelles sont les structures de données requises pour mettre en œuvre la résolution d'un problème de planification en STRIPS ?
3. Imaginez qu'il y ait une grève de métro et que vous vouliez rentrer chez vous depuis Jussieu soit en bus, soit à pied, soit en vélo. Donner trois règles qui correspondent à ces trois moyens de transports, sachant que :
   — Pour prendre le bus, il faut se trouver sur la station de bus et avoir un ticket
   — Pour prendre le vélo, il faut soit avoir un vélo et l'avoir garé devant Jussieu, soit posséder un pass Vélib, aller à la station de Vélib et s'assurer qu'il y ait des vélos disponibles.

Exercice 4 Ensembles fréquents – 2 points

Dans le tableau 1, les E_i correspondent à 8 exemples d'apprentissage, décrits par 4 attributs : A, B, C et *classe*.

Ici, nous faisons appel à l'algorithme *Apriori* pour engendrer tous les ensembles fréquents de l'ensemble d'apprentissage E donné dans le tableau 1 avec un support minimal $minsup = 0.3$.

1. Rappeler les étapes de l'algorithme *Apriori*.
2. Engendrer tous les ensembles fréquents à l'aide de l'algorithme *Apriori*.

Exemple	A	B	C	classe
E_1	a_1	b_1	c_3	$+$
E_2	a_1	b_3	c_1	$+$
E_3	a_2	b_2	c_1	$-$
E_4	a_3	b_1	c_3	$+$
E_5	a_3	b_2	c_3	$-$
E_6	a_3	b_1	c_2	$-$
E_7	a_3	b_3	c_2	$-$
E_8	a_1	b_3	c_3	$+$

TABLE 1 – *Ensemble d'apprentissage E*

Exercice 5 Extraction de règles d'association – 2 points

1. Indiquer comment on extrait les règles d'association à partir d'un ensemble fréquent.
2. Engendrer trois règles d'association de confiance 100% à partir des ensembles fréquents trouvés à la question précédente.

Exercice 6 Construction d'un arbre de décision – 3 points

1. Rappeler comment on détermine le gain d'un attribut à partir de l'entropie pondérée par le nombre d'éléments et expliquer à quoi correspond cette formule.
2. Construire l'arbre de décision qui discrimine tous les exemples de l'ensemble d'apprentissage A (voir tableau 1) pour les deux classes '+' et '−' en faisant appel à la méthode de construction employée par l'algorithme ID3.

Remarque : le tableau 2, fourni en annexe, permet de traiter le problème sans calculette.

Annexe

Annexe ASP : Axiomes de choix et contraintes de cardinalité

Axiomes de choix

Un axiome de choix permet de choisir parmi un ensemble de littéraux positifs un certain nombre d'entre eux qui seront considérés comme des faits. Ainsi, toute règle contenant en tête $\{p, q, r\}$ peut se remplacer au choix par un ensemble X de règles où $X \subseteq \{p, q, r\}$. Cela crée autant de version alternatives du programme contenant cette règle, et tout *answer set* de l'une des versions de ce programme est un *answer set* du programme avec l'axiome de choix.

Contraintes de cardinalité

Il est aussi possible d'exprimer des contraintes de cardinalité. Ainsi dans le **corps** d'une règle $\{p, q, r\} \geq k$ est satisfait par X si et seulement si au moins k des atomes de l'ensemble $\{p, q, r\}$ sont présents dans X (soit si et seulement si $|X \cap \{p, q, r\}| \geq k$.

Axiomes de choix contraints

Un littéral de la forme $k_1\{a_1, \ldots, a_n\}k_2$ en **tête** de règle, est en fait le cumul d'un axiome de choix et de deux contraintes de cardinalité prises comme contraintes d'intégrité. Ainsi, "k_1 $comp_1^{-1}$ $\{a_1, \ldots, a_n\}$ $comp_2$ k_2." est sémantiquement strictement équivalente aux deux règles suivantes : "$\{a_1, \ldots, a_n\}$." et "$\leftarrow$ *not* k_1 $comp_1^{-1}$ $\{a_1, \ldots, a_n\}$ $comp_2$ k_2.". Si la règle possède un corps, celui-ci est conservé. Ainsi "$1\{p, q\}1 \leftarrow r, not\ s$." est équivalent à "$\{p, q\} \leftarrow r, not\ s$." et "$\leftarrow not\ 1\{p, q\}1, r, not\ s$.".

Algorithme APRIORI

APRIORI($B, minsup$)
Entrées : B ensemble de transactions
$minsup$ seuil de support
$C_1 \longleftarrow$ {singletons}
$k \longleftarrow 1$
tant que $C_k \neq \emptyset$ faire
    pour chaque $c \in C_k$ faire
        compte(c) = 0
        pour chaque $t \in B$ faire
            si $c \subset t$ alors compte(c) $\longleftarrow$ compte(c) +1
    $F_k = \{c \in C_k \mid \frac{\text{compte}(c)}{|B|} \geq minsup\}$
    $k \longleftarrow k + 1$
    $C_k \longleftarrow$ APRIORI-GEN(F_{k-1})
retourner $\cup_k F_k$

APRIORI-GEN(F)
Entrée : F ensemble d'itemsets fréquents de longueur k
$C \longleftarrow \{c = f_1 \cup f_2 \in F \times F$ tels que $|c| = k + 1\}$
pour chaque $c \in C$ faire
    pour chaque $i \in c$ faire
        si $c \backslash \{i\} \notin F$ alors
            $C \longleftarrow C \backslash \{c\}$
retourner C

Mesure d'information

— Entropie de Shannon de la classe C :

$$H_S(C) = -\sum_k p(c_k)\log(p(c_k))$$

— Forme conditionnelle de l'entropie de Shannon :

$$H_S(C|A) = -\sum_i p(v_i)\sum_k p(c_k|v_i)\log(p(c_k|v_i))$$

— Gain d'information de l'attribut A pour la classe C :

$$I_S(A,C) = H_S(C) - H_S(C|A)$$

Entropie pondérée (cas 2 classes)

On considère que la classe C peut prendre 2 valeurs $c_1 =' +'$ et $c_2 =' -'$.

Le tableau ci-dessous donne l'entropie pondérée $H_S(C)$ dans ce cas, pour un ensemble qui contient p éléments de la classe '+' et n éléments de la classe '−'.

$H_S(C)$ s'écrit alors : $-(n+p)\left(\frac{p}{n+p}\log\left(\frac{p}{n+p}\right)+\frac{n}{n+p}\log\left(\frac{n}{n+p}\right)\right)$

p/n	**1**	**2**	**3**	**4**	**5**
5	3.90	6.04	7.63	8.92	10.00
4	3.61	5.51	6.90	8.00	
3	3.25	4.85	6.00		
2	2.75	4.00			
1	2.00				

TABLE 2 – *Entropie pondérée*

Algorithme de construction d'arbre de décision (variante ID3 / CART)

— Entrée : $\mathcal{B}$, la base d'apprentissage
— Sortie : un arbre de décision

1. Pile $\mathcal{P} = \{\mathcal{B}\}$
2. Tant que $\mathcal{P}$ n'est pas vide :
   — soit $\mathcal{E}$ la tête de $\mathcal{P}$
   — si le critère d'arrêt est atteint pour $\mathcal{E}$ alors créer une feuille qui contient la classe majoritaire présente dans $\mathcal{E}$
   — sinon, avec $\mathcal{E}$:
   (a) calculer $H(C|A_j)$ pour tous les attributs A_j
   (b) choisir l'attribut A_j qui maximise $I_S(A_j, C)$
   (c) créer un nœud dans l'arbre de décision avec A_j
   (d) à l'aide de A_j, partitionner $\mathcal{E}$: créer autant de sous-ensembles d'exemples que de valeurs différentes de A_j
   (e) empiler les sous-ensembles obtenus dans $\mathcal{P}$