INF4230 – Intelligence artificielle
Examen de mi-session – Hiver 2013

Éric Beaudry
Département d'informatique
Université du Québec à Montréal

Lundi 18 janvier 2013 – 18h00 à 21h00 (3 heures) – Locaux SH-2140

Instructions

- Aucune documentation permise.
- Les appareils électroniques, incluant les calculatrices, sont strictement interdits.
- Répondez directement sur le questionnaire à l'intérieur des endroits appropriés.
- Aucune question ne sera répondue durant l'examen. Si vous croyez qu'une erreur ou qu'une ambiguïté s'est glissée dans le questionnaire, indiquez clairement la supposition que vous avez retenue pour répondre à la question.
- Ne détachez pas les feuilles du questionnaire.
- Le côté verso peut être utilisé comme brouillon. Des feuilles additionnelles peuvent être demandées au surveillant.

Identification

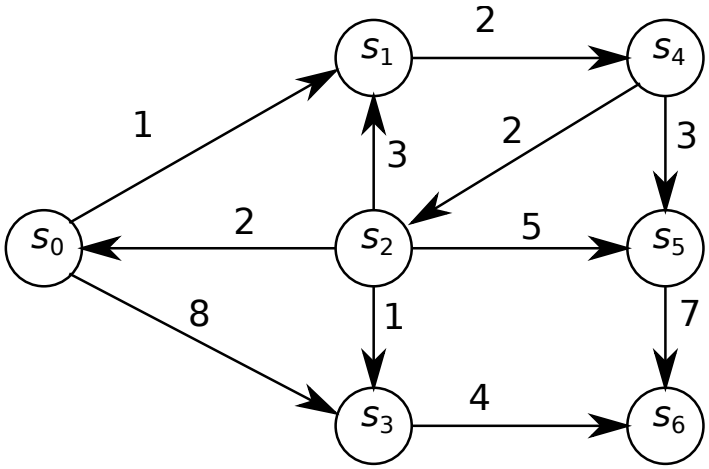
Nom : Solutionnaire

Résultat

Q1 (/5)	Q2 (/5)	Q3 (/5)	Q4 (/5)	Q5 (/5)	TOTAL (/25)

1 Algorithme A* [5 points]

Soit l'espace d'états, les transitions, les fonctions heuristiques h_1 et h_2, et le but G suivants.



État	$h_1()$	$h_2()$
s_0	9	10
s_1	8	9
s_2	5	6
s_3	0	3
s_4	7	10
s_5	7	6
s_6	0	0

$G = \{s_6\}$

(a) Faites la trace de l'algorithme A* en utilisant l'état initial s_0, l'heuristique h_1 et le but G. [3 points]

Étape	État choisi	Liste *open* $(s, f, g, parent)$ triée par f	Liste *closed* $(s, g, parent)$
#0	–	$(s_0, 9, 0, -)$	
#1	s_0	$(s_3, 8, 8, s_0), (s_1, 9, 1, s_0)$	$(s_0, 0, -)$
#2	s_3	$(s_1, 9, 1, s_0), (s_6, 12, 12, s_3)$	$(s_0, 0, -), (s_3, 8, s_0)$
#3	s_1	$(s_4, 10, 3, s_1), (s_6, 12, 12, s_3)$	$(s_0, 0, -), (s_1, 1, s_0), (s_3, 8, s_0)$
#4	s_4	$(s_2, 10, 5, s_4), (s_6, 12, 12, s_3), (s_5, 13, 6, s_4)$	$(s_0, 0, -), (s_1, 1, s_0), (s_3, 8, s_0),$ $(s_4, 3, s_1)$
#5	s_2	$(s_3, 6, 6, s_2), (s_6, 12, 12, s_3), (s_5, 13, 6, s_4)$	$(s_0, 0, -), (s_1, 1, s_0), (s_2, 5, s_4),$ $(s_4, 3, s_1)$
#6	s_3	$(s_6, 10, 10, s_3), (s_5, 13, 6, s_4)$	$(s_0, 0, -), (s_1, 1, s_0), (s_2, 5, s_4),$ $(s_3, 6, s_2), (s_4, 3, s_1)$
#7	s_6	$(s_5, 13, 6, s_4)$ Solution trouvée : $\langle s_0, s_1, s_4, s_2, s_3, s_6 \rangle$ Coût : 10	$(s_0, 0, -), (s_1, 1, s_0), (s_2, 5, s_4),$ $(s_3, 6, s_2), (s_4, 3, s_1), (s_6, 10, s_3)$

(b) L'heuristique h_1 est-elle admissible ? Justifiez. [1 point]

Oui. Elle ne surestime jamais le coût restants.
$h^*(s_0) = 10$, $h^*(s_1) = 9$, $h^*(s_2) = 5$, $h^*(s_3) = 4$, $h^*(s_4) = 7$, $h^*(s_5) = 7$, $h^*(s_6) = 0$.

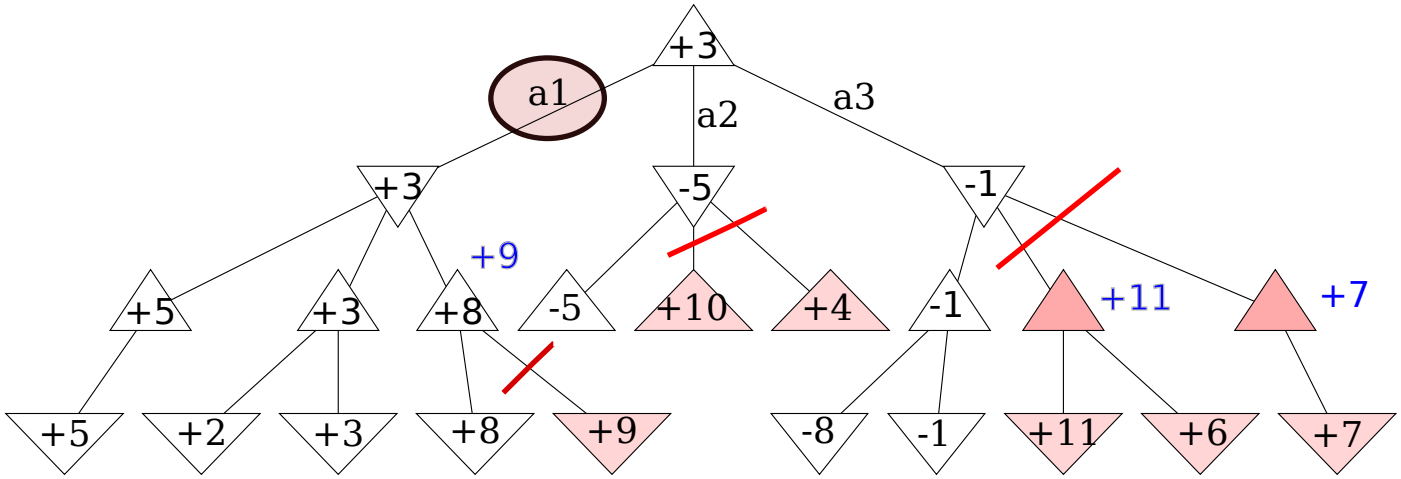
(c) Toujours en supposant le but G, l'heuristique h_2 est-elle admissible ? Justifiez. [1 point]

Non, puisque la fonction heuristique h_2 surestime le coût restant dans les états $\{s_2, s_4\}$.

2 Jeux – algorithme minimax avec élagage alpha-beta [5 points]

(a) Donnez la trace de l'algorithme minimax avec élagage alpha-beta sur l'arbre de jeu ci-dessous. [3 points]

1. Indiquez dans le triangle de chaque noeud visité sa valeur minimax.
2. Rayez au moyen d'une barre les arcs que l'algorithme a élagués.
3. Encerclez l'action retournée par l'algorithme (la décision).



Les noeuds sur fond rouge n'ont pas été pas visités. Les valeurs en bleu sont les valeurs qu'aurait retourné l'algorithme minimax sans élagage alpha-beta.

(b) L'algorithme a-t-il fait un élagage «parfait» sur l'arbre de recherche en (a) ? Si oui, justifiez votre réponse. Sinon, indiquez un exemple de changement de valeur d'une feuille qui réduirait le nombre de noeuds visités. [1 point]

Non. Si on remplace la valeur +2 de la 2e feuille (en partant de la gauche) par une valeur plus grande que +5, alors la feuille +3 sera élaguée.

(c) Vrai ou faux : au jeu de tic-tac-toe, les humains peuvent encore battre les machines qui implémentent l'algorithme minimax, car le nombre de noeuds à explorer est encore beaucoup trop grand pour les machines modernes de 2013. Justifiez. [1 point]

Faux. L'arbre de recherche du tic-tac-toe est relativement petit (moins de $9! = 362880$ feuilles). Avec une machine moderne, il est plutôt facile de parcourir tout l'arbre de recherche en moins d'une seconde. Ainsi, le temps de réflexion de la machine est pratiquement imperceptible par un humain. Puisque la machine peut explorer tout l'arbre de jeu, alors elle jouera parfaitement. Au jeu de tic-tac-toe, il est possible de ne jamais perdre en jouant parfaitement. En conclusion, les humains peuvent au mieux annuler contre une machine au tic-tac-toe. Mais il est impossible de battre la machine au tic-tac-toe.

3 Problèmes à satisfaction de contraintes (CSP) [5 points]

Lors des récentes élections générales en Orangeland, le Parti d'intelligence artificielle (PIA) a fait élire 26 députés, nommés $a, b, c, \cdots, z$, et a obtenu la majorité au parlement. Le chef du PIA vous mandate pour former son conseil des ministres. Il y a six (6) postes de ministre à combler : **F**inances, **É**ducation, **D**éveloppement durable, **A**griculture, **S**ciences et technologies, et **J**ustice. Les quatre (4) contraintes suivantes doivent être rigoureusement respectées :

C1 : un député doit être compétent dans son ministère ;

C2 : un député ne peut obtenir plus d'un ministère ;

C3 : le conseil des ministres doit être paritaire (3 hommes et 3 femmes) ;

C4 : un ministre ne doit avoir d'antécédents criminels.

Vous avez décidé de modéliser ce problème à l'aide d'un CSP. Ce CSP a six variables $\{F, E, D, A, S, J\}$ ayant pour domaine les valeurs $\{a, b, c, \cdots, z\}$.

(a) Avant même de lancer un algorithme de recherche (ex. : *backtracking search*), quelles contraintes (C1 à C4) permettent de réduire le domaine de chaque variable ? Expliquez intuitivement votre démarche. [1 point]

La contrainte C1 peut être utilisée pour filtrer le domaine des variables en gardant uniquement les députés compétents pour chaque ministère.
La contrainte C4 peut aussi être utilisée pour filtrer le domaine des variables en éliminant les députés ayant des antécédents criminels.

Pour les sous-questions suivantes supposez les domaines de valeurs suivants :

$Dom(F)$	$\{a, b, c, d, g\}$	$Dom(A)$	$\{a, z\}$
$Dom(E)$	$\{a, b, e, f, g\}$	$Dom(S)$	$\{a, e, b, z\}$
$Dom(D)$	$\{e, z\}$	$Dom(J)$	$\{a\}$

(b) Un algorithme de résolution de CSP a été présenté en classe. Ce dernier est basé sur une recherche de type *backtracking search*. L'algorithme consiste à assigner une variable à la fois, et ce, jusqu'à ce que toutes les variables soient assignées. Le choix de la prochaine variable à assigner peut avoir un impact considérable sur les performances de l'algorithme.

Nommez une heuristique vue en classe et présentée dans le livre que vous pourriez utiliser pour choisir la première variable (ministère) à assigner. Si vous ne vous rappelez pas du nom exact, expliquez-la intuitivement. Enfin, **dites quelle est la première variable à assigner** selon cette heuristique. [1 point]

L'heuristique *Minimum Remaining Values* (MRV) consiste à choisir la variable ayant **le moins de valeurs possibles**. Dans le cas présent, MRV choisi la variable J.
Attention : MRV ne choisit pas la **variable la plus contrainte**. La **variable la plus contrainte** est la variable ayant le plus grand nombre de contraintes binaires avec d'autres variables. Cette deuxième heuristique peut être utilisée dans le cas où plusieurs minimisent MRV.

(c) La contrainte C2 se traduit par un ensemble de 15 contraintes binaires $C2 = \{F \neq E, F \neq D, F \neq A, \cdots S \neq J\}$? Supposons que nous venons d'assigner $F = a$. Suite à cette assignation, indiquez les nouveaux domaines des variables $\{E, D, A, S, J\}$ en appliquant le *forward-checking*. Tenez uniquement compte des contraintes binaires C2. [1 point]

$F =$	a	$Dom(A)$	$\{z\}$
$Dom(E)$	$\{b, e, f, g\}$	$Dom(S)$	$\{e, b, z\}$
$Dom(D)$	$\{e, z\}$	$Dom(J)$	$\{\}$

(d) À titre de rappel, voici l'algorithme AC-3 (*arc-consistency* 3) présenté en classe et dans le livre.

```
function AC-3(csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (Xi, Xj) ← Remove-First(queue)
    if Remove-Inconsistent-Values(Xi, Xj) then
      for each Xk in Neighbors[Xi] do
        add (Xk, Xi) to queue

---------------------------------------------------------------------

function Remove-Inconsistent-Values(Xi, Xj) returns true if succeeds
  removed ← false
  for each x in Domain[Xi] do
    if no value y in Domain[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
      then delete x from Domain[Xi]; removed ← true
  return removed
```

Et revoici les domaines actuels des variables :

$Dom(F)$	$\{a,b,c,d,g\}$	$Dom(A)$	$\{a,z\}$
$Dom(E)$	$\{a,b,e,f,g\}$	$Dom(S)$	$\{a,e,b,z\}$
$Dom(D)$	$\{e,z\}$	$Dom(J)$	$\{a\}$

Donnez le résultat final (nouveaux domaines des variables) de l'algorithme AC-3 en considérant uniquement les contraintes binaires $C2$. Les étapes intermédiaires ne sont pas demandées. Notez qu'aucune assignation n'a encore été faite. [1 point]

$Dom(F)$	$\{c,d,g\}$	$Dom(A)$	$\{z\}$
$Dom(E)$	$\{f,g\}$	$Dom(S)$	$\{b\}$
$Dom(D)$	$\{e\}$	$Dom(J)$	$\{a\}$

(e) Expliquez intuitivement comment adapter l'algorithme *backtracking search* pour considérer la contrainte $C3$.

Dans la fonction récursive implémentant l'algorithme *backtracking search*, on ajoute deux compteurs *nbhommes* et *nbfemmes*. Dès qu'un de ces compteurs dépasse 3, on fait un retour en arrière (*backtracking*). Ou mieux : dès qu'un compteur atteint 3, on élimine le sexe saturé dans le domaine des variables restantes. Remarque : alterner l'assignation homme-femme peut ne pas fonctionner, ou au mieux, ne pas être efficace.

4 Logique du premier ordre [5 points]

(a) Traduisez les énoncés suivants en logique du premier ordre. [2 points]

1. Les étudiants aiment la bière ou [1] la lecture.
2. Tous les étudiants qui travaillent forts obtiennent leur diplôme.
3. Gilles est un étudiant.
4. Si une personne aime la lecture, alors elle lit régulièrement.
5. Gilles ne lit pas régulièrement.

1. $\forall x : etudiant(x) \rightarrow (aime(x, biere) \vee aime(x, lecture))$
2. $\forall x : etudiant(x) \wedge travfort(x) \rightarrow obtientdiplome(x)$
3. $etudiant(Gilles)$
4. $aime(x, lecture) \rightarrow litreg(x)$
5. $\neg litget(Gilles)$

(b) Convertissez les formules sous forme clausale. Numérotez les clauses. [1 point]

1. $\neg etudiant(x_1) \vee aime(x_1, biere) \vee aime(x_1, lecture)$
2. $\neg etudiant(x_2) \vee \neg travaillefort(x_2) \vee obtientdiplome(x_2)$
3. $etudiant(Gilles)$
4. $\neg aime(x_3, lecture) \vee litreg(x_3)$
5. $\neg litreg(Gilles)$

1. Ce «ou» doit être interprété comme un «ou inclusif», c'est-à-dire qu'un étudiant doit aimer au moins la bière, la lecture ou les deux.

(c) Prouvez que «Gilles aime la bière». Commencez par écrire la négation de l'énoncé à prouver. Numérotez cette nouvelle clause en reprenant la numérotation en (b). Ensuite, utilisez la technique de résolution. À chaque étape, indiquez par leur numéro les deux clauses utilisées et l'unificateur (une substitution) requis. Rappel : le but est d'obtenir une clause vide (une contradiction). [2 points]

6. $\neg aime(Gilles, bire)$
– –
Avec 4 et 5, et l'unificateur le plus général $\{x_3 = Gilles\}$, on obtient :
7. $\neg aime(Gilles, lecture)$
Avec 1 et 7, et upg $\{x_1 = Gilles\}$, on obtient :
8. $\neg etudiant(Gilles) \vee aime(Gilles, biere)$
Avec 3 et 8, et l'upg $\{\}$, on obtient :
9. $aime(Gilles, biere)$
Avec 6 et 9, et l'upg $\{\}$, on obtient :
10. $false$ (la clause vide).
Nous avons montré une contradiction. Donc, Gilles aime la bière.

5 Questions générales [5 points]

(a) Dans le TP1, un problème définit un (1) seul robot-camion, *n* emplacements (lieux) et *m* colis. En supposant qu'un robot-camion peut contenir un nombre illimité de colis, quelle est la taille de l'espace d'états ?

Le robot-camion peut être à *n* emplacements. Chaque colis est sur le robot-camion ou à *n* emplacements, donc (n+1) possibilités. Donc : $n \cdot (n+1)^m$ états possibles.

(b) L'environnement du TP1 était-il statique ou dynamique ?

Statique.

(c) Vrai ou faux : un agent de type réflexe simple (*simple reflex agents*) décide d'une action à exécuter en se basant uniquement sur les données sensorielles courantes et non sur un historique de données sensorielles.

Vrai.

(d) Vrai ou faux : l'algorithme d'exploration par escalade (*hill-climbing*) trouve toujours une solution optimale.

Faux. Les méthodes de recherche locale sont sensibles aux optimums (minimums ou maximums) locaux. L'escalade sera piégé dans le premier optimum local rencontré.

(e) Vrai ou faux : l'algorithme d'exploration par recuit simulé (*simulated annealing*) ne peut être utilisé en pratique, car il requiert une trop grande quantité de mémoire.

Faux. La méthode du recuit simulé peut demander beaucoup de temps, mais demande très peu de mémoire, car elle ne mémorise que l'état courant.

/********** Fin de l'examen ! **********/