Examen mi-session

Intelligence Artificielle II (IFT-17587)

Jeudi 1er mars 2001

De 8h30 à 11h15 en salle 3775 PLT

Les documents permis sont le livre, les acétates du cours et une double page de notes

---

1) **(18 pts) Donner la PAGE et les caractéristiques de l'environnement d'un agent en charge de la mise en place des ouvrages dans une bibliothèque. On pourrait supposer qu'un tel agent prend les livres d'une place donnée et les range à la bonne place selon les côtes des livres.**

Percepts : Pixel d'intensité variable (caméra), cote du livre (lecteur de code barre).
Actions :

- Prendre un livre, s'il y a un nouveau livre à ranger.
- Lire la cote.
- Se déplacer (avancer, tourner, arrêter) pour aller ranger un livre ou revenir au chariot des retours et pour éviter les obstacles.
- Déposer un livre lorsqu'il est arrivée à la position où le livre doit être rangé.

But : Le but de l'agent est de mettre les livres, qui sont dans le chariot des retours, à la bonne position dans les rayons de la bibliothèque.

Environnement : L'environnement de l'agent est une bibliothèque. L'environnement est *inaccessible*, parce que l'agent ne peut pas avoir accès à toutes les informations sur l'environnement. L'environnement est *non-déterminé*, parce que les actions de l'agent n'ont pas un effet garanti et il y a de l'incertain. Lorsque l'agent avance, il peut frapper un obstacle qu'il n'avait pas vu. L'environnement est *non-épisodique*, car les actions de l'agent dans le passé peut influencer ses actions. Par exemple, s'il doit replacer un deuxième livre, il va pouvoir tenir compte des obstacles qu'il avait rencontrer lors du rangement du livre précédent. Cela va influencer le chemin qu'il va emprunter. L'environnement est *dynamique*, parce qu'il y a une multitude d'événements qui peuvent se produire pendant que l'agent délibère. Un nouvel obstacle à éviter, un nouveau livre

à ranger, etc. L'environnement est *continu*, car il y a plusieurs perceptions possibles et plusieurs actions possibles.

2) **a) (8 pts) Donner un exemple de chacune des quatre architectures d'agent et expliquer votre choix.**

Tout exemple qui correspond bien aux architectures est accepté. Les quatre architectures sont : agent simple réflexe, agent réflexe avec état interne, agent avec des buts et agent avec utilité.

b) (4 pts) Si on veut un agent qui combinerait « but » et « utilité » qu'elle serait d'après vous l'architecture d'un tel agent ? Donnez un exemple de ce type d'agent et justifiez le.

L'architecture, c'est un mélange des deux architectures but et utilité aux pages 44 et 45 du livre. Elle comprend toutes les composantes de l'architecture du but, auquel ion ajoute une couche sous le but pour tester l'utilité. De cette manière, un agent peut choisir son but, et choisir la meilleure façon de l'atteindre avec l'utilité. Ou si l'agent a plusieurs buts, il peut choisir un but avec la fonction d'utilité. Pour les exemples, tout exemple logique est bon.

3) **Supposons que vous voulez implémenter un programme qui joue au tic-tac-toe, L'utilisateur humain joue toujours O et la machine toujours X. Chaque case sur le board est appelée aij où i est la ligne et j la colonne.**
    a) **(6 pts) Définir les états, le but (test) et les opérateurs pour ce problème, en termes de aij. Combien d'états y a t-il ?**

État : Configuration du jeu = *<a11,a12,…,a33>*
But : Trois X en ligne
Opérateur : Mettre un X ou un O sur le jeu
Nombre d'états : On s'est rendu compte que c'est un problème difficile de combinatoire, par conséquent, tout raisonnement logique sera accepté. Si vous avez trouvé la bonne réponse, vous allez avoir deux points bonis.

La fonction de combinatoire qui est utilisée est {n m} = n! / (m! x (n – m)!) (le nombre de façon de choisir m boules noires identiques parmi un mélange de n boule blanches et noires). Il est à noter que le nombre d'état possible est différent du nombre de nœuds dans l'espace de recherche, parce qu'il y a beaucoup d'états identiques dans l'espace de recherche. La solution est :

1 O => {9 1} = 9 états

1 O, 1 X => {9 1}x{8 1} = 72

2 O, 1 X => {9 2}x{7 1} = 252

2 O, 2 X => {9 2}x{7 2} = 756

3 O, 2 X => {9 3}x{6 2} = 1260

Jusqu'à maintenant, on ne pouvait pas générer des configurations illégales, mais à partir de maintenant, c'est possible d'avoir trois X et trois O dans deux lignes ou colonnes et nous devons en tenir compte.

3 O, 3 X => {9 3}x{6 3} = 1680, le nombre d'état dans lequel l'humain a déjà gagné = 8 x {6 3} = 160

4 O, 3 X => {9 4}x{5 3} = 1260, le nombre d'états où X gagne = 8 x {6 4} = 120

4 O, 4 X => {9 4}x{5 4} = 630, le nombre d'état dans lequel O gagne et non X = 2 x 8 x {6 4} = 240

5 O, 4 X => {9 5}x{4 4} = 126, le nombre d'états où X gagne = 8 x {6 1} = 48

Donc, le nombre d'états possibles est 6046 – 160 – 120 – 240 – 48 = 5478

b) (4 pts) Quelle algorithme de recherche pensez vous utiliser pour implémenter un tel programme ? Pourquoi ?

Il y a plusieurs réponses ici et nous acceptons toutes les réponses pourvu qu'elles soient justifiées.

c) (3pts) Décrire brièvement une fonction qui évalue la valeur d'un état, tel que décrit en a), pour le joueur-machine.

Toute fonction raisonnable sera accepté. L'idée c'est que la fonction doit pénalisé les coups perdants et donner une « récompense » pour les coups gagnants. Par exemple, le nombre de X dans une même ligne moins le nombre de O dans une même ligne est une fonction possible.

4) **Dans l'espace de recherche ci-dessous, le nombre placé à côté de chaque lien dénote le coût associé au lien entre nœuds, et h est le coût estimé du nœud jusqu'au but final G.**

   a) **(4 pts) Donner alors l'ordre dans lequel les nœuds sont visités lorsqu'on utilise le A*. Donner alors f(n) pour chacun des nœuds n.**

f(B)=3, f(C) =14, f(D)=11, f(E)=6, f(H)=14, f(F) = 11, f(I)=22, f(G)= 12
A, B, E, D, H, C, F, G.

   b) (4 pts) Un algorithme « vorace » du type meilleur-d'abord peut-il faire mieux ? Combien de nœuds, un tel algo traverse-t-il avant d'atteindre le but G.

Les nœuds visités sont : A, B, E, D, H, C, G. Donc oui, dans cet exemple, l'algorithme meilleur-d'abord est plus efficace, il visite un nœud de moins.

   c) (5 pts) Autant la largeur-d'abord que le A* consomment de la mémoire (C'est exponentiel !). Comment pouvez vous modifier A* de façon à utiliser moins d'espace mémoire ? Quelle est alors la complexité en espace du nouvel algo ?

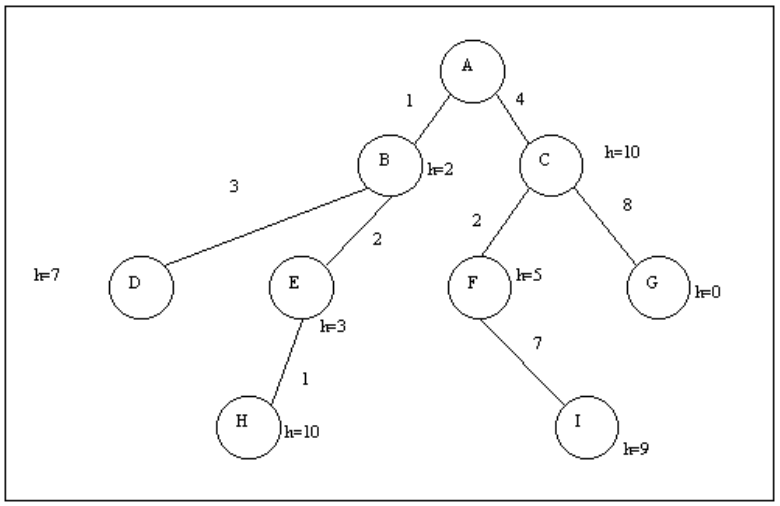
On peut utiliser l'algorithme IDA* avec le coût comme limite de profondeur. Pour la complexité en espace, voir le livre à la page 106. Dans le pire des cas, IDA* demande bf^*/d où d est le plus petit coût d'un opérateur, b le facteur de branchement et f^* le coût de la solution optimale. Dans notre cas, le facteur de branchement est 2, le coût de la solution optimale est 12 et le coût du plus petit opérateur est 1, donc l'algorithme demande 24 nœuds. Dans la plupart des cas, on peut estime le nombre de nœuds visité par bd où d est la profondeur de la solution, ce qui donne 2 x 2 = 4 nœuds.

   d) (4 pts) Si H est aussi un but, lequel parmi largeur-d'abord, profondeur-d'abord et le profondeur-itératif peut trouver la solution optimale ?

Seulement la recherche en profondeur d'abord va trouver la solution optimale H.

e) **(5 pts) La fonction heuristique h pose un problème dans la mesure où elle surestime trop le coût de C à G. Quelle propriété de A* n'est plus remplie si h a ce problème ?.**

A* ne sera plus optimal.



5) **(10 pts) Supposons qu'on veut céduler un problème de salles de cours. Dans ce problème, on a c salles et k cours, chaque cours débute à Sk et finit à Ek, avec Sk et Ek des entiers naturels. Sachant qu'aucun cours ne doit partager avec un autre la même salle, que les salles sont seulement disponibles entre 1 :00 et 5 :00, formulez ce problème comme un CSP (constraint solving problem). Suggérez alors une heuristique pour résoudre ce problème et expliquer pourquoi pensez-vous que votre heuristique peut résoudre un tel problème** .

Le problème consiste à choisir une classe pour chaque cours.

Variables : C1, …, Ck (les différentes classes pour chacun des cours)

Domaine : {1, …, c}

Contraintes : Pour tout les i,j, si (Ei >= Ej > Si OU Si <= Sj < Ei) alors (Ci != Cj)

Si < Ei, Si > 1 (pour 1 :00), Ei < 5

Vous pouvez utiliser n'importe quelle heuristique vu en classe, comme par exemple l'heuristique min-conflicts.

6) Pour chacune des assignations suivantes, dites si KB |== a est valide

a) **(3 pts) KB = (A Ú B ® C) Ù A et a = C**

A	B	C (α)	A∨B	A∨B→C	(A∨B→C)∧A
T	T	T	T	T	T
T	T	F	T	F	F
T	F	T	T	T	T
T	F	F	T	F	F
F	T	T	T	T	F
F	T	F	T	F	F
F	F	T	F	T	F
F	F	F	F	T	F

Quand KB est vrai, a (C dans ce cas) est toujours vrai, donc KB |== a est valide.

b) (3 pts) KB = (A ® B) et a = B

A	B	A→B
T	T	T
T	F	F
F	T	T
F	F	T

KB |== a n'est pas valide, parce que a (B dans ce cas) n'est pas toujours vrai quand KB est vrai (voir ligne 4).

c) (3 pts) KB = (A ® B) Ù (B ® C) et a = (A ® C)

A	B	C	A→B	B→C	(A→B) ∧ (B→C)	A→C
T	T	T	T	T	T	T
T	T	F	T	F	F	F
T	F	T	F	T	F	T
T	F	F	F	T	F	F
F	T	T	T	T	T	T
F	T	F	T	F	F	T
F	F	T	T	T	T	T
F	F	F	T	T	T	T

Quand KB est vrai, a (A → C dans ce cas) est toujours vrai, donc KB |== a est valide.

7) Donner l'axiome successeur comme il a été introduit dans le cours pour les actions suivantes :

a) (3 pts) Cassé(x,Resultat(a,s)) où x est une variable dénotant un verre

a = tomber(x) ∨ [cassé(x,s) ∧ a ≠ réparer(x)]

b) (3 pts) Avoir(lait,Resultat(a,s))

a = acheté(lait) ∨ [avoir(lait,s) ∧ a ≠ renverser(lait)]

8) **Dans l'algorithme de rétropropagation qui est utilisée pour les réseaux de neurones, l'erreur pour une unité de sortie est donnée par la formule suivante :** $d_k \leftarrow o_k(1-o_k)(t_k-o_k)$**. Pour une unité cachée, l'erreur est donnée par la formule suivante :** $d_h \leftarrow o_h(1-o_h)\sum_{k\in sorties} w_{kh}d_k$ **.**

   a) **(3 pts) Expliquer pourquoi on n'utilise pas la même formule pour les unités cachées.**

On n'utilise pas la même formule, parce que pour les unités de sortie, on a accès à la valeur désirée de l'unité directement, ce qui n'est pas le cas pour les unités cachées.

   b) **(2 pts) Expliquer l'idée derrière l'utilisation de la somme** $\sum_{k\in sorties} w_{kh}d_k$ **pour calculer l'erreur faite par l'unité cachée.**

Pour calculer l'erreur faite par l'unité cachée, on regarde sa participation dans l'erreur des unités de sortie. Donc, pour chaque unité de sortie à laquelle l'unité cachée est connectée, on pondère l'erreur de l'unité de sortie par le poids du lien. Plus le poids est fort, plus l'unité cachée a participé à l'erreur de l'unité de sortie. En d'autres mots, l'erreur de l'unité cachée, c'est la somme de ses participations aux erreurs des unités de sortie auxquelles elle est connectée.

9) **(5 pts) Dans les algorithmes génétiques, expliquer les effets du taux du mutation sur la population. Que se passe-t-il s'il est trop grand ou trop faible ?**

Dans les algorithmes génétiques, si le taux de mutation est trop élevée, les individu de la population vont toujours être modifiés, donc, il n'y aura pas de convergence vers un meilleur individu. La population va être trop hétérogène.

Si le taux de mutation est trop faible, alors la population va devenir très homogène. Tous les individus vont être identiques ou très proches. La qualité de la population ne pourra plus augmenter. On perd la chance qu'une mutation nous donne un individus plus fort.