

Aide à la Décision

Support de Travaux Dirigés

1. RESOLUTION PAR L'ETRE HUMAIN

Le but de ces exercices n'est pas tant de trouver la solution que de discuter sur la façon d'y arriver, et sur les pièges à éviter.

1.1. Cryp-arithmétique et principe de Newell et Simon

Voici une équation : DONALD + GERALD = ROBERT.

Chaque chiffre de 0 à 9 est assigné à une seule et unique lettre et chaque lettre correspond à un seul et unique chiffre.

Il n'y a pas de zéro non significatif à la gauche des nombres.

La lettre D correspond au chiffre 5.

Trouver la solution à ce problème en testant l'une des approches suivantes :

- Résolution individuelle et introspection.
- Résolution par binôme : l'un résout l'équation en disant ce qu'il pense pendant que le second observe ce que le premier fait.

Autres exemples, si vous souhaitez vous distraire... :

SEND + MORE = MONEY

'S' et 'M' $\neq$ 0 (pas de '0' non significatif)

VINGT + CINQ + CINQ = TRENTE

'V', 'C' et 'T' $\neq$ 0

1.2. « Le Problème difficile »

Soit la question suivante :

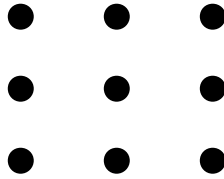
Si le problème que vous avez résolu avant que vous ayez résolu celui-ci était plus difficile que le problème que vous avez résolu après que vous ayez résolu le problème que vous avez résolu avant que vous ayez résolu celui-ci, est-ce que le problème que vous avez résolu avant que vous ayez résolu celui-ci était plus difficile que celui-ci ?

La réponse est-elle oui ou non ?

1.3. Problèmes d'incomplétude

1.3.1. Former quatre triangles équilatéraux avec six allumettes de même taille.

1.3.2. Passer par chacun des neufs points suivants, en traçant au plus quatre segments de droites, sans lever le crayon.



1.3.3. Histoire d'un roi cruel (cité par E. deBono dans « la Pensée Latérale »)

Un Roi veut épouser une jeune fille qui refuse. Il la met d'abord au cachot, puis trouve cette ruse : il va prendre deux cailloux, un noir et un blanc ; elle devra en choisir un ; si c'est le blanc, elle sera libre ; si c'est le noir, elle devra se marier.

La jeune fille voit que le roi choisit deux noirs. Que peut-elle faire pour être libre ?

2. LOGIQUE DES PROPOSITIONS ET LOGIQUE DES PREDICATS

2.1. Logique des propositions – Chaînages avant et arrière

Soit la Base de Règles (BR) suivante, où les lettres majuscules correspondent à des propositions atomiques :

R1 : A	-->	B	R6 : M et L	-->	A
R2 : A	-->	C	R7 : I et B	-->	D
R3 : C	-->	E	R8 : E et D	-->	F
R4 : M	-->	C	R9 : K et F	-->	H
R5 : I et K	-->	A	R10 : L et E	-->	F

2.1.1. Chaînage avant

2.1.1.1. Soit la base de faits initiale {I, L, M}. Indiquer les cycles successifs d'un moteur d'inférence MI fonctionnant en chaînage-avant jusqu'à saturation, avec pour la résolution de conflits les stratégies de choix suivantes :

- stratégie 1 : choix de la première règle dans l'ordre de la base ;
- stratégie 2 : priorité aux règles dont le nombre de prémisses est maximum ; en cas d'égalité, priorité à celle concernant les faits les plus récents, et en dernier critère, l'ordre de la base.

2.1.1.2. Ecrire (pseudo code) l'algorithme du chaînage avant.

2.1.2. Chaînage arrière

2.1.2.1. Soit la base de faits initiale {I, L, M} et le but à déduire F.

Simuler le fonctionnement du moteur d'inférence en chaînage-arrière en construisant l'arbre (ou le graphe) ET/OU correspondant.

2.1.2.2. Même question avec le but H.

2.1.2.3. Ecrire l'algorithme (pseudo-code) correspondant au chaînage arrière.

2.1.2. Généralisation

Dans les questions précédentes, les règles étaient de la forme :

SI <ensemble de faits reliés par l'opérateur « ET » ALORS <fait décidé comme 'vrai'>

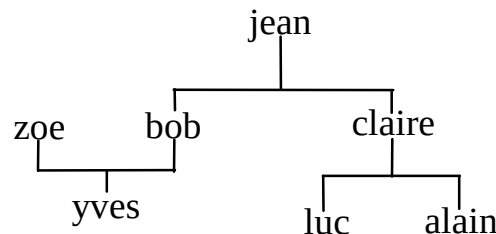
Imaginez maintenant des règles plus générales de la forme :

SI <expression booléenne> ALORS <expression booléenne>

Discutez des modifications à apporter pour les algorithmes décrits précédemment.

2.2. Logique des prédicats – Arbre généalogique

Soit la famille suivante, représentée au moyen d'un (extrait d') arbre généalogique :



2.2.1. Formaliser grâce aux prédicats pere et mere l'arbre généalogique précédent.

pere(?x, ?y) signifie « x a pour père y »

mere(?x, ?y) signifie « x a pour mère y »

Vous formaliserez la définition de ces prédicats en utilisant le langage ci-dessus.

2.2.2. Ecrire dans le langage ci-dessus, les règles qui représentent les relations parent, grand-pere, frere&sœur (au sens large: ayant au moins un parent commun) en vous aidant des prédicats pere et mere.

2.2.3. Négation par absence

Ecrire une règle pour compléter la base de faits avec la relation enfant_unique.

2.3. Inscription au congrès CIA

2.3.1. Formaliser les informations suivantes, relatives à l'inscription au congrès CIA ("Chercheurs en IA"), qui comporte un colloque scientifique et des cours :

- Pour l'inscription au colloque, une réduction de 50% est accordée à tout étudiant âgé de moins de 28 ans à la date du congrès et à tout auteur qui présente une communication. Il n'y a pas de tarif spécial étudiant ou auteur pour les cours.
- Les laboratoires de recherche d'université et du CNRS, qui inscrivent des participants en nombre supérieur à 3, bénéficient d'une réduction de 30% sur les droits d'inscription au colloque et aux cours.
- Cette réduction est également applicable à toute entreprise membre de l'ACIA (Association des chercheurs en IA) sous les mêmes conditions mais seulement sur les droits d'inscription au colloque.
- Les réductions ne sont applicables que sur les inscriptions reçues un mois au moins avant la date de début du congrès, qui est le 11-01-03.

Vous ferez cette formalisation tout d'abord en utilisant la logique des propositions, puis vous transposerez dans la logique des prédicats.

2.4. Que faites-vous de vos soirées ?

Soit la base de règles définie comme suit :

- | | |
|---|--|
| <p>1 REGLE <i>décision_dodo</i>
SI fatigue SUP envie_de_sortir
ALORS décision ← dodo</p> <p>2 REGLE <i>décision_sortir1</i>
SI fatigue INF envie_de_sortir
ALORS décision ← sortir</p> <p>3 REGLE <i>décision_sortir2</i>
SI fatigue EGAL envie_de_sortir
ET lendemain EGAL congé
ALORS décision ← sortir</p> <p>4 REGLE <i>influence</i>
SI fatigue EGAL envie_de_sortir
ET lendemain DIFF congé
ET nb_de_personnes SUP_EGAL 2
ET CONNU (désir_entourage)
ALORS décision ← désir_entourage</p> <p>5 REGLE <i>télé</i>
SI fatigue EGAL envie_de_sortir
ET lendemain DIFF congé
ET INCONNU (désir_entourage)
ET programme_télé EGAL bon
ALORS décision ← télé</p> <p>6 REGLE <i>ciné1</i>
SI décision EGAL sortir
ET choix_prioritaire PARMi
{cinéma danse théâtre opéra}
ET prix_par_personne INF_EGAL 40
ALORS sortie ← cinéma
manger ← maison</p> | <p>7 REGLE <i>ciné2</i>
SI décision EGAL sortir
ET choix_prioritaire EGAL cinéma
ET prix_par_personne SUP_EGAL 150
ET faim EGAL vrai
ALORS sortie ← cinéma
manger ← restaurant</p> <p>8 REGLE <i>fast1</i>
SI manger EGAL restaurant
ET ambiance PARMi {rapide branchée}
ALORS type_restaurant ← fast-food</p> <p>9 REGLE <i>fast2</i>
SI manger EGAL restaurant
ET prix_par_personne INF 50
ALORS type_restaurant ← fast-food</p> <p>10 REGLE <i>indécis</i>
SI décision EGAL sortir
ET INCONNU(choix_prioritaire)
ET nb_de_personnes EGAL 2
ALORS sortie ← choix_de_l_autre</p> <p>11 REGLE <i>gentille</i>
SI décision EGAL sortir
ET ambiance PARMi
{galante intéressée indécise}
ET nb_de_personnes EGAL 2
ALORS sortie ← choix_de_l_autre</p> <p>12 REGLE <i>sans_idée</i>
SI décision EGAL sortir
ET INCONNU (sortie)
ALORS manger ← restaurant</p> |
|---|--|

Les attributs utilisés dans ces règles sont de deux types :

	attributs	demandable	affichable
1	fatigue	X	
2	envie_de_sortir	X	
3	décision		
4	désir_entourage		
5	nb_de_personnes	X	
6	prix_par_personnes	X	
7	ambiance	X	
8	lendemain	X	
9	programme_télé	X	
10	faim	X	
11	sortie	X	X

12	manger	X	X
13	choix_prioritaire		
14	type_restaurant		X
15	choix_de_l_autre	X	

2.4.1. Quelles sont les inférences possibles dans le cas suivant :

fatigue	2
envie_de_sortie	3
choix_prioritaire	théâtre
prix_par_personne	40

2.4.2. Déterminer la valeur de « sortie » dans le cas suivant :

fatigue	2
envie_de_sortie	2
nb_personnes	2
choix_de_l_autre	théâtre
lendemain	congé

2.4.3. Quelles sont les inférences possibles dans le cas suivant :

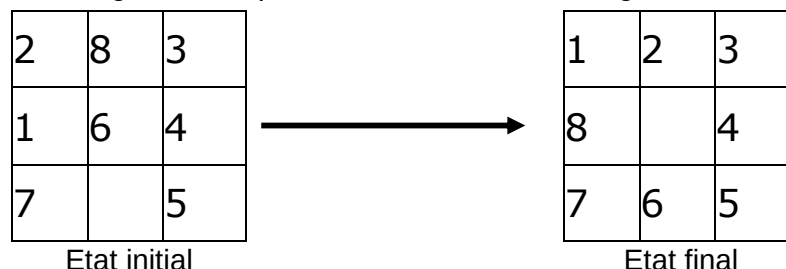
fatigue	1
ambiance	branchée
choix_prioritaire	cinéma

3. ALGORITHMES DE RECHERCHE

Application au jeu du taquin :

Le taquin est apparu aux Etats-Unis vers la fin des années 1870. La légende veut qu'il ait été inventé par un sourd muet qui voulait ranger ses cubes dans la boîte sans les en sortir. Il est plus probable que ce jeu soit dû au célèbre problémiste Américain Sam Loyd. Il s'est rapidement répandu aux Etats-Unis puis en Europe où il a connu un succès comparable au Rubik's cube en son temps...

Il est très simple : par des déplacements successifs, il faut ramener les pièces du Taquin à leur position de départ, indiquée au chargement du problème, comme dans la figure ci-dessous :



3.1. Résolution par construction du graphe

Essayez de tracer le graphe représentant l'ensemble des solutions de progression du jeu. Qu'en concluez-vous ?

3.2. Algorithme de recherche au moyen d'un graphe.

Imaginez des algorithmes de recherche utilisant un graphe.

Caractéristique principale : votre algorithme ne doit pas disposer au préalable du graphe complet, mais doit le construire au fur et à mesure de son déroulement, selon ses besoins.

Deux fonctions peuvent vous aider à construire des algorithmes permettant de trouver la « meilleure » solution, c'est-à-dire celle qui nécessite le moins d'opération possible :

- Fonction d'évaluation du coût entre la position de départ et celle étudiée par l'algorithme à un moment donné : il suffit dans le jeu du taquin de considérer que chaque étape a un coût de '1'.
- Fonction d'estimation du coût restant pour passer de la position étudiée par l'algorithme à un moment donné à la position finale du jeu. Vous devez supposer que cette fonction existe et rend une valeur entière, sans vous préoccuper pour l'instant de son algorithme.

3.3. A*

Soient les deux fonctions heuristiques suivantes :

- $h1(n)$ = nombre de tuiles mal placées.
- $h2(n)$ = distance Manhattan totale (nombre de cases pour se rendre à la position désirée pour chaque tuile).

3.3.1. Développer l'arbre de recherche en utilisant l'algorithme A* selon l'heuristique $h1$.

3.3.2. Même question avec l'heuristique $h2$.

3.4. Comparaison des algorithmes

Comparez l'efficacité des algorithmes qui ont été vus au cours de l'exercice 3 en fonction des critères suivants :

- complétude : garantie de trouver une solution si elle existe ;
- optimalité : garantie de trouver la « meilleure » solution s'il y en a une ;
- complexité en temps : nombre d'états qui doivent être traités avant de trouver la solution ;
- complexité en espace : nombre d'états qui doivent être conservés en mémoire pour trouver la solution.

Pour cela, vous utiliserez les valeurs suivantes :

b = facteur de branchement de l'arbre de recherche (nombre maximum de successeurs associés à un état)

d = profondeur dans l'arbre de recherche de la solution trouvée par l'algorithme

m = profondeur maximale de l'arbre de recherche (peut être égal à l'infini)

4. ALGORITHMES DE JEUX

Application : tic-tac-toe

On s'intéresse à la modélisation du jeu de Tic-Tac-Toe à deux joueurs. Dans ce jeu on dispose au départ d'une grille vide 3 x 3. Supposons que la machine marque des croix (X) et l'humain des cercles (O). Les deux joueurs inscrivent à tour de rôle une croix et un cercle. Le gagnant est le premier qui forme un alignement de trois marques identiques.

4.1. Evaluation sur un niveau

La machine doit jouer. Elle évalue les différentes possibilités de jeu à sa disposition.

Ecrivez (pseudo code) l'algorithme d'une fonction appelée MAX1 qui permet, depuis une position de jeu connue, d'évaluer tous les coups jouables et de choisir la meilleure solution.

Vous utiliserez pour cela une fonction *int evaluate ()* qui rend la valeur d'une position. Plus la valeur rendue sera élevée, plus une position de jeu sera avantageuse pour la machine.

Pour l'instant, n'écrivez pas l'algorithme de cette fonction : supposez tout simplement qu'elle existe.

4.2. Evaluation à 2 niveaux

La machine évalue ses possibilités de jeu et les réponses possibles de son adversaire.

Ecrivez (pseudo code) l'algorithme d'une fonction MAX2 qui permet, depuis une position de jeu connue, d'évaluer toutes les combinaisons jouables et de choisir la meilleure solution.

Vous utiliserez la même fonction *int evaluate ()* que pour la question précédente.

4.3. Algorithme MinMax

Généralisez l'algorithme précédent de façon à utiliser le mécanisme MinMax selon deux conditions d'arrêt de l'analyse par la machine :

4.3.1. : jusqu'à l'atteinte d'une solution finale d'un jeu ;

4.3.2. : jusqu'à une profondeur donnée en paramètre.

Pour le 4.3.1, vous utiliserez une fonction *int est_final ()* qui indique si la machine gagne, l'adversaire gagne, ou si la partie est nulle.

Pour 4.3.2, vous utiliserez la fonction *int evaluate ()* qui rend la valeur d'une position.