Travaux pratiques d'IA

Série 1: Formalisation

1 Les trois missionnaire

1.1 Description

Voir le document

1.2 Formalisation du problème recherche

1. Donnez une représentation des états
On a deux rives, un groupe de trois missionnaires et un groupe de trois cannibales. On peut représenter par rive: le nombre de missionnaire et de cannibale par un couple (m,c)
Comme il y a deux rives (gauche 0, droite 1 que j'ai choisi pour ce cas) on a (m0, c0, m1, c1)

2. Quels sont les opérateurs possibles?
* On a des opérateurs `allez` pour aller de la rive gauche vers la rive droite.
* On a des opérateur `retour` pour aller de la rive droite vers la rive gauche
* Comme le bateau ne peut contenir que deux personnes, on a un ensemble de passagers contenant des couples ou des singletons: {m; c; (m,m); (c,c); (m,c)}.
* Les mouvements se font dans les deux sens

3. Définissez les conditions pour lesquels les opérateurs sont applicables Un opérateur est applicable si: 1. Le nombre de personnes déplacées n'est pas plus grand que le nombre de personnes disponibles. (par exemple déplacer 2 cannibals alors qu'il y en a que 1) 2. Il n'y a pas plus de cannibals que de missionnaires dans chaque côtés de la rivière.

4. Implémenter un algorithme de recherche pour résoudre le problème en utilisant un arbre de recherche correspondant à la description que vous avez choisi.

```
//Algorithme de recherche
init <- (3,3,0,0)
Transition(init)

function Transition(old)
for transition in transitions
    new <- transition(old)
    if test(new) == true:
        if new == (0,0,3,3)
            print(new)
        else
            recherche(new)
```

Remarque: J'ai donné dans le pseudo code une définition récursive. Cependant dans mon implémentation, je ferrai une représentation non récursive

1.3 Complexité

1. L'espace de recherche correspond à tout les états du système. On a alors les répartition possible entre les 3 missionnaires et les 3 cannibals multiplié par le fait que la barque peut se trouver d'un côté ou de l'autre de la rivière:
    1. 4*4*2= 32 possibilités
2. Le nombre d'états n'amenant pas à la mort d'un missionnaire sont assez limités (s'ils sont les trois, les cannibals peuvent être répartis comme bon nous semble alors que s'il y en a un isolé, il faut qu'il soit avec au plus un cannibal):
    1. 4+1+1+4= 10 possibilités
3. Le nombre d'état accessible depuis l'état initial dépendent des transitions:
    1. Comme il y a 5 transitions possibles. On a 5 états accessible.
    2. On a cependant que deux transitions qui ne sont pas funestes. On a donc 2 états accessibles.

2 La tour de Hanoi

2.1 Description

Voir le document.

Questions

1. Formalisez le problème
    - Un état du système peut être représenté par 3 piles contenant les disques. Les disque sont alors représenté comme des entiers (plus c'est grand plus le chiffre est élevé) On a ainsi une liste de 3 piles.
    - Dans le cas de 3 disque (avec les disques sur la pile de gauche) le système serait [(1,2,3), (0,0,0), (0,0,0)] et l'état final serai [(0,0,0),(0,0,0),(1,2,3)]
    - Les transitions peuvent être représentés par des couples (départ, arrivé) qui décrivent le mouvement des disque. Les éléments départ et arrivé peuvent chacun prendre trois valeurs: "gauche", "milieu","droite" représentant les piles par leur position. Ainsi ("gauche", "milieu") représente le mouvement du disque au sommet de la pile gauche vers la pile du milieu. Il y a donc 6 transitions possibles.
2. En comptant tout les état possibles du système, on a:
    1. Sans les règles (3+6+1)*6= 60 possibilités
    2. Avec les règles 3+(6*2)+1= 16 possibilités
3. Voir le code

3 Algorithme de recherche général

On classe les sommets en 3 catégories: * La catégorie A des sommets déjà visités (ceux qui apparaissent dans l'arbre de parcours) * La catégorie B des sommets adjacents à ceux de la catégorie A mais pas encore visités (sommets qui peuvent être atteints) * La catégorie C des sommets invisibles qui n'ont pas encore été rencontrés du tout (qui ne peuvent pas être atteints depuis un sommet déjà visité)

```
Liste <- vide; liste.push(s_I)
repeat
    s_courant<-liste.pop()
    if(s_courant == s_G)
        break
    list.push(Gamma(s_courant))
until liste.len() == 0
    if (S_courant == S_G)
        backtrack solution
    else
        pas de solution
```