Le Problème des Missionnaires et des Cannibales

IHW

31/01/2023

1 Le problème des missionnaires et des cannibales

Le problème des missionnaires et des cannibales est un problème classique en Intelligence Artificielle qui consiste à déterminer la séquence optimale d'actions pour transporter une certaine quantité de missionnaires et de cannibales d'une rive à l'autre d'un fleuve à bord d'une barque de capacité limitée. Les contraintes du problème incluent la nécessité de ne pas laisser un nombre supérieur de cannibales à un endroit donné sans la présence d'au moins un missionnaire pour les surveiller.

1.1 Représentation

La situation est représentée par deux rives, la rive gauche et la rive droite, ainsi que le fleuve qui les sépare. La barque peut être située sur l'une ou l'autre rive ou sur le fleuve.
On peut représenter la situation par un tableau avec quatre colonnes correspondant aux différents emplacements, et deux lignes correspondant aux missionnaires et aux cannibales :

	Rive gauche	Fleuve	Rive droite
Missionnaires	M_L		M_R
Cannibales	C_L		C_R

Le nombre de missionnaires et de cannibales est représenté par des variables M_L, C_L, M_R et C_R. Dans la suite du problème leur nombre sera definit par la variable n.
On peut également ajouter une colonne pour représenter la position de la barque, qui peut être sur la rive gauche, sur la rive droite ou sur le fleuve :

	Rive gauche	Fleuve	Rive droite	Bateau
Missionnaires	M_L		M_R	1
Cannibales	C_L		C_R	1

La position du bateau est représentée par une variable p représentant ca capacité, et peut prendre les valeurs "0" pour la rive gauche, "1" pour la rive droite ou "" pour le fleuve. Si le bateau est sur la rive gauche ou la rive droite.

1.2 Exemple

Pour résoudre ce problème, nous pouvons utiliser des tableaux pour représenter l'état actuel des missionnaires et des cannibales, ainsi que la position du bateau. Voici les tableaux pour chaque étape de la résolution :

1.2.1 Étape 0

M_G	**C_G**	**Bateau**	**M_D**	**C_D**	**Nb_déplacements**	**Déplacements**
3	3	1	0	0	0	-

L'étape 0, représente l'état initial du problème avec 3 missionnaires, 3 cannibales ainsi que la barque positionné sur la rive gauche.

1.2.2 Étape 1

M_G	C_G	Bateau	M_D	C_D	Nb_déplacements	Déplacements
3	1	0	0	2	1	2 CG → 2 CD

Dans l'étape 1, le bateau est sur la rive droite avec deux cannibales à bord.

1.2.3 Étape 2

M_G	C_G	Bateau	M_D	C_D	Nb_déplacements	Déplacements
3	2	1	0	1	2	1 CD → 1 CG

Dans l'étape 2, le bateau est sur la rive gauche avec un cannibales à bord.

1.2.4 Étape 3

M_G	C_G	Bateau	M_D	C_D	Nb_déplacements	Déplacements
3	0	0	0	3	3	2 CG → 2 CD

Dans l'étape 3, le bateau est sur la rive droite avec deux cannibales à bord.

1.2.5 Étape 4

M_G	C_G	Bateau	M_D	C_D	Nb_déplacements	Déplacements
3	1	1	0	2	4	1 CD → 1 CG

Dans l'étape 4, le bateau est de nouveau sur la rive gauche avec un cannibale à bord.

1.2.6 Étape 5

M_G	C_G	Bateau	M_D	C_D	Nb_déplacements	Déplacements
1	1	0	2	2	5	1 CG 1 MG→1 CD 1 MD

Dans l'étape 5, le bateau est de retour sur la rive droite avec un missionnaire et un cannibale à bord.

1.2.7 Étape 6

M_G	C_G	Bateau	M_D	C_D	Nb_déplacements	Déplacements
2	2	1	1	1	6	1 CD 1 MD→1 CG 1 MG

Dans l'étape 6, le bateau est de retour sur la rive gauche avec un missionnaire et un cannibale à bord.

1.2.8 Étape 7

M_G	C_G	Bateau	M_D	C_D	Nb_déplacements	Déplacements
0	2	0	3	1	7	2 MG → 2 MD

Dans l'étape 7, le bateau est de nouveau sur la rive droite avec 2 missionnaires à bord.

1.2.9 Étape 8

M_G	C_G	Bateau	M_D	C_D	Nb_déplacements	Déplacements
0	3	1	3	0	8	1 CD → 1 CG

Dans l'étape 8, le bateau est de retour sur la rive gauche avec un cannibale à bord.

1.2.10 Étape 9

M_G	C_G	Bateau	M_D	C_D	Nb_déplacements	Déplacements
0	1	0	3	2	9	2 CG → 2 CD

Dans l'étape 9, le bateau est de nouveau sur la rive droite avec 2 cannibales à bord.

1.2.11 Étape 10

M_G	C_G	Bateau	M_D	C_D	Nb_déplacements	Déplacements
1	1	1	2	2	10	1 MD → 1 MG

Dans l'étape 10, le bateau est de retour sur la rive gauche avec un missionnaire à bord.

1.2.12 Étape 11

M_G	C_G	Bateau	M_D	C_D	Nb_déplacements	Déplacements
0	0	0	3	3	11	1 MG 1 CG→ 1 MD 1 CD

Dans cette dernière étape, le bateau est de retour sur la rive gauche avec un missionnaire et un cannibale à bord.

1.3 Solution complète

Voici la solution complète en termes de déplacements du bateau :

Etape 1 : 2 CG → 2 CD
Etape 2 : 1 CD → 1 CG
Etape 3 : 2 CG → 2 CD
Etape 4 : 1 CD → 1 CG
Etape 5 : 1 CG 1 MG → 1 CD 1 MD
Etape 6 : 1 CD 1 MD → 1 CG 1 MG
Etape 7 : 2 MG → 2 MD
Etape 8 : 1 CD → 1 CG
Etape 9 : 2 CG → 2 CD
Etape 10 : 1 MD → 1 MG
Etape 11 : 1 MG 1 CG → 1 MD 1 CD

Cette solution est la plus optimal en prend en compte un total de 11 étapes pour résoudre le problème des missionnaires et des cannibales.

2 Etude Expérimentale

Il existe plusieurs méthodes pour résoudre le problème des missionnaires et des cannibales, notamment la recherche en profondeur d'abord (DFS), la recherche en largeur d'abord (BFS), l'algorithme A* et l'algorithme IDA*.
Le problème des missionnaires et des cannibales peut être modélisé sous la forme d'un arbre de recherche, où chaque nœud représente un état possible de la situation. Les arcs de l'arbre représentent les différents mouvements possibles des missionnaires et des cannibales. L'algorithme de tree_search consiste à explorer systématiquement toutes les solutions potentielles en utilisant une stratégie de recherche en profondeur DFS ou en largeur BFS.
L'algorithme commence en explorant le nœud racine, qui représente l'état initial. Lorsqu'un nœud est exploré, l'algorithme génère tous les états fils possibles en utilisant les différents mouvements possibles des missionnaires et des cannibales. L'algorithme s'arrête lorsqu'un nœud final est trouvé, qui représente une solution au problème. Ce nœud final peut être défini comme un nœud où tous les missionnaires sont de l'autre côté de la rivière, sans qu'aucun cannibal ne soit en minorité.
En utilisant l'algorithme tree_search, il est possible de trouver la solution optimale au problème des missionnaires et des cannibales, c'est-à-dire la solution qui implique le moins de déplacements de la barque.

2.1 Implémentation de l'Algorithme

Nous avons implémenté l'algorithme de recherche en arbre en utilisant le langage Python et en utilisant les fonctions suivantes:

- **suppr** : cette fonction calcule la différence entre deux vecteurs.

- **add** : cette fonction calcule la somme de deux vecteurs.
- **v_add** : cette fonction retourne toutes les actions possibles pour ajouter des personnes à bord du bateau.
- **v_suppr** : cette fonction retourne toutes les actions possibles pour retirer des personnes à bord du bateau.
- **tree_search** : cette fonction implémente l'algorithme de recherche en arbre en utilisant les fonctions précédentes.

2.2 Résultats et Analyse

Après avoir exécuté notre algorithme, nous avons obtenu un nombre total d'étapes égal à 11. Ce nombre d'étapes est le minimum nécessaire pour transporter les trois missionnaires et les trois cannibales de la rive gauche à la rive droite de la rivière.
Pour résoudre ce problème, nous utilisons l'algorithme Tree Search. Tree Search est un algorithme de parcours de graphe qui consiste à explorer systématiquement toutes les possibilités en utilisant une structure de données sous la forme d'un arbre. Le noeud de départ de l'arbre représente l'état initial du problème et chaque noeud enfant représente un état possible qui est obtenu à partir de l'état parent en effectuant une action.
Dans notre programme, nous utilisons l'algorithme Tree Search pour explorer toutes les combinaisons possibles d'actions pour transporter les missionnaires et les cannibales d'une rive à l'autre. Nous utilisons une liste memo pour enregistrer les états visités afin de ne pas explorer deux fois les mêmes combinaisons. Nous définissons également les fonctions v_add et v_suppr pour retourner les actions possibles qui peuvent être effectuées en fonction de l'état actuel. Les fonctions add et suppr permettent de mettre à jour l'état en fonction de l'action choisie.
Lorsque nous trouvons un état final qui correspond à la condition où tous les missionnaires et les cannibales se trouvent de l'autre côté du fleuve, nous stockons le nombre d'étapes nécessaires pour atteindre cet état dans la variable solution.

3 Améliorations et extentions

3.0.1 Implémentation de l'algorithme A*

L'implémentation de l'algorithme A* consiste à utiliser une heuristique pour orienter la recherche et trouver un chemin plus rapide vers la solution. En utilisant une heuristique pour évaluer la fonction de coût, le programme peut explorer les états les plus prometteurs en premier et éviter les chemins inutiles. Cela peut améliorer considérablement la vitesse de convergence vers la solution.

3.0.2 Application à d'autres problèmes

Cette extension consiste à utiliser l'algorithme développé pour résoudre d'autres problèmes tels que les huit reines, le labyrinthe, etc. En utilisant les mêmes méthodes de recherche, l'algorithme peut être facilement adapté à de nouveaux problèmes en définissant simplement les actions et les conditions d'état possibles.

3.0.3 Intégration de l'interface graphique

Cette extension consiste à ajouter une interface graphique pour visualiser l'arbre de recherche et les étapes de la solution. Cela peut aider à mieux comprendre comment la recherche s'est déroulée et à déboguer le programme plus facilement. De plus, l'interface graphique peut rendre l'algorithme plus attrayant pour les utilisateurs, en leur permettant de visualiser la progression de la recherche dans un format plus accessible.

3.0.4 Implémentation de fonction d'analyse

L'ajout d'une fonction "coût" dans le programme permettrait de comparer les temps d'exécution de différentes parties du code et d'optimiser le programme en conséquence. Cette fonction pourrait également aider à identifier les goulots d'étranglement dans le programme.

La fonction "graph" serait utile pour visualiser les nœuds du graphe créé par le programme à chaque étape. Cela permettrait de mieux comprendre le fonctionnement du programme et de détecter les éventuelles erreurs ou incohérences dans le graphe. Il serait également possible d'utiliser cette fonction pour visualiser différentes métriques, telles que la distance ou le coût, associées à chaque nœud du graphe.

4 Implémentation en Python

Dans ce rapport, nous avons choisi d'implémenter la solution du problème des missionnaires et des cannibales en utilisant la méthode de recherche en profondeur d'abord. Le code Python suivant montre la façon dont la solution a été implémentée.

Listing 1: Code Python

```
import copy

n = int(input("Entrez le nombre de missionnaires et de cannibales : "))
p = int(input("Entrez la capacite de la bateau : "))

start = [n, n, 1]
size = n
possible_actions = []

for i in range(p + 1):
    for j in range(p - i + 1):
        if i + j <= p:
            possible_actions.append([i, j, 1])

memo = []
solution = 0

def suppr(vec1, vec2):
    return [vec1[i] - vec2[i] for i in range(3)]

def add(vec1, vec2):
    return [vec1[i] + vec2[i] for i in range(3)]

def v_add(vec):
    actions = []
    for action in possible_actions:
        if action[0] + action[1] >= 1:
            x = add(vec, action)
            y = suppr([size, size, 1], x)
            if x[0] >= 0 and x[1] >= 0 and x[0] <= size and x[1] <= size and (x[0] >= x
                actions.append(action)
    return actions

def v_suppr(vec):
    actions = []
    for action in possible_actions:
        if action[0] + action[1] >= 1:
            x = suppr(vec, action)
            y = suppr([size, size, 1], x)
            if x[0] >= 0 and x[1] >= 0 and x[0] <= size and x[1] <= size and (x[0] >= x
                actions.append(action)
    return actions

def tree_search(vec, moves):
```

```
    global solution

    if vec in memo:
        return
    else:
        memo.append(vec)

    moves = copy.deepcopy(moves)
    moves += 1

    if vec[2] == 0:
        actions = v_add(vec)
    else:
        actions = v_suppr(vec)

    for action in actions:
        new_vec = copy.deepcopy(vec)
        for i in range(3):
            if new_vec[2] == 0:
                new_vec[i] += action[i]
            else:
                new_vec[i] -= action[i]
        if sum(new_vec) == 0:
            print("Etape Finale", moves, "=", [new_vec[0], new_vec[1], new_vec[2], size-
            solution = moves
            return True
        if tree_search(new_vec, moves):
            print("Etape",moves, "=", [new_vec[0], new_vec[1], new_vec[2], size-new_vec
            return True
    return False

tree_search(start, 0)

print("Le nombre d'etape totale est de=", solution)
```