Système expert : application à la résolution des tours de Hanoï.

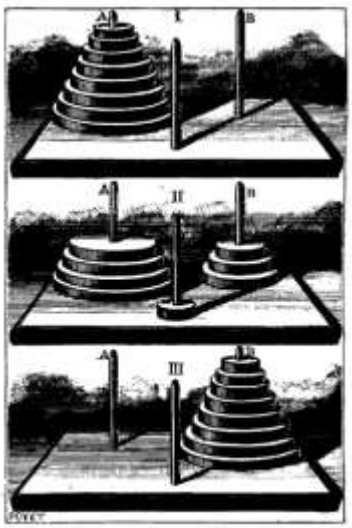
1. Contexte

Nous voulons mettre en place un système à base de règles (système expert) qui par inférence en chainage doit résoudre le problème des tours de Hanoï. L'énoncé du problème des tours de Hanoï est le suivant (source Wikipedia) :

Les tours de Hanoï (originellement, la tour d'Hanoïa) sont un jeu de réflexion imaginé par le mathématicien français Édouard Lucas, et consistant à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire », et ceci en un minimum de coups, tout en respectant les règles suivantes :

- On ne peut déplacer plus d'un disque à la fois ;
- On ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.

On suppose que cette dernière règle est également respectée dans la configuration de départ. Ci-dessous, vous avez une illustration « historique » du jeu (source Wikipedia).



La tour d'Hanoï.

Dans le cadre de ce projet, on se placera dans un cadre simple, à savoir que nous n'avons que 3 tours (que nous appellerons pics) et nous avons 3 disques à déplacer, chacun de diamètre de 1,2 et 3 (en cm par exemple). Nous référencerons les disques avec leur taille.

Suggestion : dans l'environnement Python afin de symboliser la situation de jeu à un instant donné, vous pouvez créer une classe

```
class Jeu_Hanoi:
    def __init__(self):
        self.pic = (np.zeros([3,3],dtype=np.int))
        self.nombre_palet = (np.zeros(3,dtype=np.int))
        pass
```

qui contient :

- pic : un tableau Numpy de taille *3x3*, le premier indice (entre 0 à 2) correspondant au numéro du pic (de la gauche vers la droite), le second indice (entre 0 et 2), le disque possiblement présent. Par exemple la case pic*[b,0]* contient le premier disque de l'empilement sur le pic numéro *b*.
- nombre_palet : un tableau Numpy de taille *3*, indiquant par exemple nombre_palet*[b]* le nombre de disque présent à l'instant étudié sur le pic numéro *b*.

A l'initialisation la situation sera la suivante (tous les disques sont sur le premier pic rangés en taille décroissante en partant du bas donc 3 -> 2-> 1) :

```
jeu = Jeu_Hanoi()
jeu.nombre_palet[0] = 3
jeu.pic[0,0] = 3
jeu.pic[0,1] = 2
jeu.pic[0,2] = 1
```

Le reste est à 0 du fait l'initialisation.

Et la situation finale désirée (tous les disques sont sur le dernier pic rangés en taille décroissante en partant du bas donc 3 -> 2-> 1) :

```
Jeu_final = Jeu_Hanoi()
Jeu_final.nombre_palet[2] = 3
Jeu_final.pic[2,0] = 3
Jeu_final.pic[2,1] = 2
Jeu_final.pic[2,2] = 1
```

2. Base experte

Ce problème peut se résoudre algorithmiquement d'une manière optimale. Cependant nous allons mettre en place un système expert qui chercher une résolution à partir de règles simples.

Après échange avec un expert, nous avons identifié les prémisses suivantes pour les règles qui vont constituer les éléments constitutifs de la base experte :

- On ne peut déplacer un disque d'un pic d'indice *b* sur un pic d'indice *d* que si le pic d'indice *b* contient au moins 1 disque ;

  Suggestion : cette condition peut être validée à partir de la valeur jeu.nombre_palet[b]>0.

- On ne peut déplacer le disque supérieur du pic d'indice *b* (donc la case jeu.pic[b,jeu.nombre_palet[b]-1]) sur le pic d'indice *d* que si le pic d'indice *d* est vide ou son disque en position supérieure est de taille plus grande (donc la case jeu.pic[d,jeu_nombre_palet[d]-1]).

  Suggestion : cette condition peut être validée à partir de la comparaison de la valeur de jeu.pic[b,jeu.nombre_palet[b]-1]) et la valeur de jeu.pic[d,jeu.nombre_palet[d]-1].

- Pour éviter de boucler sans fin durant la résolution du problème, on ne peut pas effectuer un déplacement si, après déplacement, on se retrouve dans une situation déjà étudiée.

Suggestion : pour ce dernier point, vous pouvez générer, pour chaque situation de jeu, un nombre unique en utilisant des puissances de 2. Par exemple à l'initialisation, le nombre peut se calculer suivant :

jeu.pic =[[3 2 1] [0 0 0] [0 0 0]] -> nombre = 2^2+2^1+2^0

A la situation finale désirée :

Jeu.pic =[[0 0 0] [0 0 0] [3 2 1]] -> nombre = (2^2+2^1+2^0)*2^6

Ainsi à chaque étape de la résolution, la situation créée devient un nouveau fait que vous pouvez ajouter dans une liste Python par exemple appelée situation_etudiee qui enregistrera toutes les situations de jeu interdites pour le futur. Vous pouvez proposer votre propre stratégie.

Enfin pour finaliser la base experte, l'expert nous indique qu'il faut appliquer les règles suivantes :

- Règle 1 : si (le pic 0 est non vide) et (le déplacement du disque supérieur du pic 0 sur le pic 1 est autorisé) et (la situation résultante de ce déplacement n'a pas été observée) alors réaliser (le déplacement du disque supérieur du pic 0 sur le pic 1).
- Règle 2 : si (le pic 1 est non vide) et (le déplacement du disque supérieur du pic 1 sur le pic 2 est autorisé) et (la situation résultante de ce déplacement n'a pas été observée) alors réaliser (le déplacement du disque supérieur du pic 1 sur le pic 2).
- Règle 3 : si (le pic 2 est non vide) et (le déplacement du disque supérieur du pic 2 sur le pic 1 est autorisé) et (la situation résultante de ce déplacement n'a pas été observée) alors réaliser (le déplacement du disque supérieur du pic 2 sur le pic 1).
- Règle 4 : si (le pic 1 est non vide) et (le déplacement du disque supérieur du pic 1 sur le pic 0 est autorisé) et (la situation résultante de ce déplacement n'a pas été observée) alors réaliser (le déplacement du disque supérieur du pic 1 sur le pic 0).
- Métarègle : En cas de plusieurs règles éligibles, on choisit la règle de plus petit indice (si règle1 et règle3 sont applicable, on exécute la règle1).

3. Réalisation

Nous allons réaliser le programme SE permettant de résoudre le problème de la tour d'Hanoï à partir de la base experte définie par l'expert dans la section 2.

- Réaliser une fonction nombre_situation(jeu) qui convertit une situation de jeu en un nombre unique et retourne ce nombre.
- Réaliser une fonction pic_vide(indice_pic1,jeu) qui vérifie que le pic d'indice indice_pic1 n'est pas sans palet, elle retourne un Boolean.
- Réaliser une fonction regle_jeu(indice_pic1,indice_pic2,jeu) qui vérifie si l'on peut déplacer le palet supérieur du pic d'indice indice_pic1 vers le pic d'indice indice_pic2, elle retourne un Boolean.
- Réaliser une fonction effectue_deplacement(indice_pic1,indice_pic2,jeu) qui effectue le déplacement du palet supérieur du pic d'indice indice_pic1 vers le pic d'indice indice_pic2. La fonction déplace le palet et incrémente et décrémente le nombre de palets sur chacun des pics concernés, elle modifie donc la variable jeu.
- Réaliser une fonction situation_non_vue(indice_pic1,indice_pic2,jeu,situation_etudiee) qui, dans le cas d'un déplacement du palet supérieur du pic d'indice indice_pic1 vers le pic d'indice indice_pic2, vérifie que l'on ne retourne pas dans une situation déjà étudiée. La fonction retourne un Boolean.

A partir de ces différentes fonctions, vous allez développer le moteur d'inférence en chainage avant qui à chaque itération, à partir de la situation initiale :

- Sélectionne la ou les règles éligibles suivant (dans le cas général) :
  si (pic_vide(indice_pic1,jeu)) et (regle_jeu(indice_pic1,indice_pic2,jeu)) et (situation_non_vue(indice_pic1,indice_pic2,jeu,situation_etudiee)) alors règle éligible
- Applique la règle sélectionnée : effectue le déplacement, met à jour la liste situation_etudiee et incrémente une variable comptant le nombre de cout joué.
- Après chaque coup effectué, afficher le déplacement et la nouvelle situation de jeu.

Cette inférence se fait jusqu'à obtenir la situation finale désirée.

- Tester votre programme et analyser le nombre de coup nécessaire pour résoudre le problème.
- Dans le cas étudié, quel est le nombre de coups optimal pour résoudre le problème (à comparer avec votre résolution).
- Peut-on ajouter d'autres règles pour optimiser la résolution ?
- Selon le temps restant étendre votre programme à d'autres cas (par exemple, 4 ou 5 palets, 4 pyramides).