DST Informatique Pour Tous
MP PSI
Durée : 2 heures

19 octobre

La clarté de la présentation et la qualité de la rédaction font partie de l'évaluation. On veillera à utiliser des noms explicites pour les variables et les fonctions introduites. Le candidat pourra clarifier les programmes par l'ajout de commentaires judicieux si nécéssaire.

L'idée originale provient de l'observation de l'exploitation des ressources alimentaires chez les fourmis. En effet, celles-ci, bien qu'ayant individuellement des capacités cognitives limitées, sont capables collectivement de trouver le chemin le plus court entre une source de nourriture et leur nid. Des biologistes ont ainsi observé, dans une série d'expériences menées à partir de 1989, qu'une colonie de fourmis ayant le choix entre deux chemins d'inégale longueur menant à une source de nourriture avait tendance à utiliser le chemin le plus court.

Un modèle expliquant ce comportement est le suivant :

1. Une fourmi (appelée « éclaireuse ») parcourt plus ou moins au hasard l'environnement autour de la colonie ;

2. Si celle-ci découvre une source de nourriture, elle rentre plus ou moins directement au nid, en laissant sur son chemin une piste de phéromones ;

3. Ces phéromones étant attractives, les fourmis passant à proximité vont avoir tendance à suivre, de façon plus ou moins directe, cette piste ;

4. En revenant au nid, ces mêmes fourmis vont renforcer la piste ;

5. Si deux pistes sont possibles pour atteindre la même source de nourriture, celle étant la plus courte sera, dans le même temps, parcourue par plus de fourmis que la longue piste ;

6. La piste courte sera donc de plus en plus renforcée, et donc de plus en plus attractive ;

7. la longue piste, elle, finira par disparaître, les phéromones étant volatiles ;

8. À terme, l'ensemble des fourmis a donc déterminé et « choisi » la piste la plus courte.

L'algorithme général est relativement simple, et repose sur un ensemble de fourmis, chacune parcourant un trajet parmi ceux possibles. A chaque étape, la fourmi choisit de passer d'une ville à une autre en fonction de quelques règles :

Règle 1 Elle ne peut visiter qu'une fois chaque ville ;

Règle 2 Plus une ville est loin, moins elle a de chance d'être choisie (c'est la « visibilité »)

Règle 3 Plus l'intensité de la piste de phéromone disposée sur l'arête entre deux villes est grande, plus le trajet aura de chance d'être choisi

Règle 4 Une fois son trajet terminé, la fourmi dépose, sur l'ensemble des arêtes parcourues, plus de phéromones si le trajet est court

Règle 5 Les pistes de phéromones s'évaporent à chaque itération.

On notera J_i^k la liste des noeuds possible pour une fourmis à l'instant k, positionnée au noeud i, c'est à dire, les noeuds adjacents à i non encore visités par la forumis en question.

La règle de déplacement, appelée « règle aléatoire de transition proportionnelle », est écrite mathématiquement sous la forme suivante la probabilité que la fourmi k choisissent le noeud j sachant qu'elle est en i est $p_{i,j}^k$ vérifiant :

$$\begin{cases} p_{i,j}^k = \frac{\tau_{i,j}(t)^\alpha \times n_{i,j}^\beta}{\sum_{l \in J_i^k} \tau_{i,l}(t)^\alpha \times n_{i,l}^\beta} & j \in J_i^k \\ p_{i,j}^k = 0 & j \notin J_i^k \end{cases}$$

où J_i^k est la liste des noeuds possibles pour une fourmi k lorsqu'elle se trouve sur un noeud i, $n_{i,j}$, la visibilité, qui est égale à l'inverse de la distance de deux noeuds i et j ($n_{i,j} = \frac{1}{d_{i,j}}$) et $\tau_{i,j}(t)$ l'intensité de la piste à une itération donnée t. Les deux principaux paramètres contrôlant l'algorithme sont α et β, qui contrôlent l'importance relative de l'intensité et de la visibilité d'une arête. Une fois la tournée des villes effectuée, une fourmi k dépose une quantité $\Delta\tau_{i,j}^k$ de phéromone sur chaque arête de son parcours :

$$\begin{cases} \Delta\tau_{i,j}^k(t) = \frac{Q}{L^k(t)} & (i,j) \in T^k(t) \\ \Delta\tau_{i,j}^k(t) = 0 & (i,j) \notin T^k(t) \end{cases}$$

où $T_k(t)$ est la tournée faite par la fourmi k à l'itération t, $L_k(t)$ la longueur du trajet et Q un paramètre de réglage. La longueur $L_k(t)$ du chemin $(i_0 = 0, i_1, \ldots, i_p = n-1)$ emprunté par la fourmis k en passant par les noeuds i_l, est calculée comme la somme des pondérations (des poids des arrêtes) :

$$L_k(t) = \sum_{l=1}^{p} d_{i_l, i_{l+1}} \quad p \text{ nombre de noeuds du chemin}$$

A la fin de chaque itération de l'algorithme, les phéromones déposées aux itérations précédentes par les fourmis s'évaporent de $\rho\tau_{i,j}(t)$, ρ étant un paramètre de réglage. Et à la fin de l'itération, on a la somme des phéromones qui ne se sont pas évaporées et de celles qui viennent d'être déposées :

$$\tau_{i,j}(t+1) = (1-\rho)\tau_{i,j}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t).$$

où m est le nombre de fourmis utilisées pour l'itération t et ρ un paramètre de réglage.

Ainsi, à chaque itération la matrice Tau des $\tau_{i,j}$ est modifier par le passage des m fourmis (et n'est d'ailleurs plus forcément symétrique). L'objectif est alors d'obtenir le chemin le plus court qui ici est censé être celui qui est le plus emprunté donc qui a le plus de phéromone.

On supposera avoir importé les bibliothèques `numpy` et `numpy.random` via :

```
import numpy as np
import numpy.random as rd
```

Partie A : Mise en place

On se donne un graphe G pondéré, non orienté et connexe (tous les noeuds son reliés et les arrêtes peuvent être parcourues dans les deux sens). Chaque noeud est numéroté de 0 à $n-1$. On associe à chaque arrête reliant le noeud i au noeud j, une pondération $d_{i,j}$ (la distance entre le noeud i et j).

Les données du graphe pondéré, sont stockés sous forme de liste d'adjacence, dans un dictionnaire `G`, dont les n clés sont les numéros des noeuds, et les valeurs associées sont les listes des couples $[...(j_i, d_{i,j_i}), \ldots]$ où les j_i sont les noeuds adjacent au noeud i (les noeuds voisins de i) et d_{i,j_i} la distance entre les noeuds i et j_i. On parle alors de graphe pondéré, de pondération $d_{i,j}$.

1. On se donne le scripte suivant :

```
G={0:[(1,3),(2,3)],1:[(2,5),(0,3)],2:[(0,3)]}
```

   Représenter le graphe associé.

2. Proposer une fonction `Voisin(G,i)` prenant comme paramètres le dictionnaire `G`, le noeud `i` et renvoyant la liste des noeuds adjacents à i.

3. Programmer une fonction `Distance(G)` prenant comme paramètre le graphe `G` et renvoyant la matrice `D` de type `array`, de taille $n \times n$ vérifiant :

$$D_{i,j} = d_{i,j} \text{ si } i \text{ et } j \text{ sont adjacents, et } D_{i,j} = -1 \text{ sinon}$$

   . Afin que par exemple, appliqué au graphe G précédant, on ait :

```
  Distance(G)
Out[4]:
array([[-1.,  3.,  3.],
       [ 3., -1.,  5.],
       [ 3., -1., -1.]])
```

   On complètera le squelette suivant :

```
 def Distance(G):
       n=len(G)
       D=-np.ones((n,n))
 ..........................
```

4. En appelant les fonctions précédentes, proposer une fonction `initialisation(G)`, renvoyant la liste `J` des noeuds immédiatement "visible" par la fourmi éclaireuse initialement au noeud 0, la matrice `D` et la matrice `Tau` composée des $\tau_{i,j} = 1$, de type `array` et de taille $n \times n$. Ainsi sur le graphe précédant on aura :

```
initialisation(G)
Out[6]:
([1, 2],
 array([[-1.,  3.,  3.],
        [ 3., -1.,  5.],
        [ 3., -1., -1.]]),
 array([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]]))
```

5. Programmer la fonction `Visible(i,G,V)` prenant comme paramètres le noeud `i`, la liste `V` des noeuds déjà visités par la fourmis, le dictionnaire du graphe G et renvoyant la liste des noeuds que la fourmi peut visiter, c'est à dire, les noeuds adjacents non encore visités.

6. On se donne une liste `C`$= [i_0, \ldots, i_p]$ de noeuds correspondants à un chemin ((i_j, i_{j+1}) est donc une arête du graphe), programmer une fonction `longueur(D,C)` renvoyant la longueur du chemin `C`.

7. Proposer une fonction `maximum(Tau,i)` prenant comme paramètre la matrice `Tau` le noeud `i` et qui renvoi le noeud j tel que $\tau_{i,j}$ soit le maximum des $\tau_{i,k}$ avec $k \in [|0, n-1|]$.

Partie B : Initialisation

On suppose ici $m = 1$ et $k = 0$.

On se donne une première fourmi, l'éclaireuse. Lr es paramètres α, β, Q et ρ sont fixés. Initialement la matrice `Tau`, dont les coefficiants sont les $\tau_{i,j}$ (l'intensité de la piste) est composée de 1. On se donne un graphe G d'ordre n (constitué de n noeuds) et la matrice $D \in \mathcal{M}_n(\mathbb{R})$ des pondérations, dont les coefficients $d_{i,j}$ prennent pour valeur la distance entre les noeuds i et j, s'ils sont reliés et la valeur -1 sinon. Les matrices D, Tau et la liste J sont initialisés de la façon suivante :

```
J,D,Tau=initialisation(G)
```

1. Proposer une fonction `Loi(D,Tau,u,i,J)` prenant comme paramètres la matrice `D` le uplet `u`$=(\alpha, \beta, Q, \rho)$, l'entier `i`, la liste `J` des noeuds "visitble" par la fourmi partant de i (noté J_i), et la matrice `Tau` renvoyant la liste `P` des couples $(j, p_{i,j})$ des noeuds j de J_i associé à la valeur $p_{i,j}$:

$$p_{i,j} = \frac{\tau_{i,j}^{\alpha} \times n_{i,j}^{\beta}}{\sum_{l \in J_i} \tau_{i,l}(t)^{\alpha} \times n_{i,l}^{\beta}} \quad j \in J_i$$

On complètera :

```
def Loi(D,Tau,u,i,J):
    a,b,q,r=u
    P=[]
    S=0
    for .........
        S+=.............
    for ..........
       ........................

    return P
```

2. On suppose disposer de la liste `P` (contenant les couples $(j, p_{i,j})$), proposer la fonction `Freq(P)`, qui renvoie la liste des fréquences cumulée vérifiant :

$$F[k] = \sum_{j=0}^{k} p_{i,u_j} \text{ les } u_j \text{ sont les noeuds de} J_i$$

3. On suppose disposer de la liste `P` (contenant les couples $(j, p_{i,j})$), la liste `J` des voisins "visible" par la fourmi à partir du noeud i (noté J_i). La fourmi choisit un noeud j parmi les noeuds de la liste `J` avec la probabilité $p_{i,j}$.

   On se propose d'utiliser la méthode dites d'inversion :

   La commande `rd.random()` renvoi un nombre au hasard uniformément. La probabilité qu'il soit compris entre a et b est proportionnelle à la taille de l'intervalle $[a, b]$, soit $\mid b - a \mid$. Si on dispose de la liste des fréquences définies précédemment, dans le cas qui nous intéresse, on cherche alors j tel que le nombre `x=rd.random()` soit dans l'intervalle $]F[j-1], F[j]]$ (avec comme convention $F[-1] = 0$) comme la liste des fréquences cumulées est croissante en cherche finalement le plus petit j tel que $F[j]$ soit plus grand que x.

   Programmer une fonction `choix(P,J)` renvoyant le noeud choisit et prenant comme paramètre la liste J des noeuds "visible" et la liste de probabilité `P`, en complétant le squelette suivant :

```
def choix(P,J):
    F=Frep(P)
    r=rd.random()
    j=0
    ..............
        ............
    return ..........
```

4. Compléter le programme de la fonction `Chemin(G,u)` prenant comme paramètre le dictionnaire `G` du graphe, le uplet $u = (\alpha, \beta, Q, \rho)$ et renvoyant la liste `V` des noeuds visités quand la fourmis éclaireuse atteint le noeud $n-1$ ou une liste vide si elle ne peut plus avancer sans reprendre un noeud déjà emprunté.

```
def Chemin(G,u):
    J,D,Tau=initialisation(G)
    n=len(G)
    V=[0]
    P=Loi(D,Tau,u,0,J)
    j=choix(P,J)
    V.append(j)
    while j<n-1:
        J=.................
        if len(J)==0:
            return ......
        ..............
       ..............
       ..............
    return V
```

5. Programmer la fonction `Pheromone(D,Tau,Q,r,V)` prenant comme paramètre les matrices `D` et `Tau`, les flotants `Q`, `r`=ρ et la liste `V`, du chemin parcouru par la fourmi, qui modifie la matrice `Tau`, c-a-d les valeur $\tau_{i,j}$, si V est non vide de la façon suivante :

$$\tau_{i,j}(1) = (1-\rho)\tau_{i,j}(0) + \Delta\tau_{i,j}(0)$$

Où si on pose L la longueur du chemin V on a :

$$\begin{cases} \Delta\tau_{i,j}(0) = \frac{Q}{L} & \text{i et j des noeuds de } V \\ \Delta\tau_{i,j}(0) = 0 \text{ sinon} \end{cases}$$

Partie C : Construction du programme de recherche aléatoire

On se donne maintenant m fourmis, les paramètres α, β, Q et ρ sont fixés. On répète l'expérience pendant $Time$ itérations. On se donne un graphe G d'ordre n (constitué de n noeuds) et la matrice $D \in \mathcal{M}_n(\mathbb{R})$ des pondérations, dont les coefficients $d_{i,j}$ prennent pour valeur la distance entre les noeuds i et j, s'ils sont reliés et la valeur -1 sinon. Ces données sont stockées dans le tableau D. Si on se place à l'itération t, on note $Tau \in \mathcal{M}_n(\mathbb{R})$ la matrice des coefficients τ_{ij} à l'instant t, stokées dans le tableau Tau.
Les fourmis partent toutes du noeud 0 pour rejoindre le noeud $n-1$. Dès que ce dernier noeud est atteint elle s'arrête et le chemin correspond à la liste V des noeuds parcourues d'en l'ordre pour relier le noeud 0 au noeud $n-1$.

1. Proposer une adaptation du programme `Chemin`, qu'on nommera `Chemin_m(D,Tau,G,u,m)` prenant comme nouveau paramètre `m` et renvoyant la liste des listes des chemins parcourus par chaque fourmi :

```
def Chemin_m(D,Tau,G,u,m):

        C=[]
        n=len(G)
        ..................
            V=[0]
            J=Voisin(0,G)
            P=Loi(D,Tau,u,0,J)
            j=choix(P,J)
            V.append(j)
            test=True
            while j<n-1 and test:
                ..................
                ................
               ..................
                  ...........
                ............
                  ..............
            C.append(V)
        return C
```

2. En adaptant le programme `Pheromone()` précédant, proposer une nouvelle fonction `Pheromone_m(D,Tau,Q,r,C)` prenant comme paramètre la liste `C` des m listes des chemins parcourus par les m fourmis à la place la liste V du chemin parcouru par une unique fourmi et modifiant la matrice `Tau` :

3. On dispose du programme `Fourmis()` suivant :

```
def Fourmis(G,m,Times,u):
      J,D,Tau=initialisation(G)
      a,b,Q,r=u
      for t in range(Times):
            C=Chemin_m(D,Tau,G,u,m)
            Pheromone_m(D,Tau,Q,r,C)
      return Tau
```

On suppose que le chemin le plus court est bien celui qui est le plus emprunté donc qui a le plus de phéromone. Proposer une fonction `Optimum(G,u,m,Times)`, renvoyant le chemin optimum (on pourra appeler la fonction `maximum()` de la partie A).