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
```

```
import numpy as np
import numpy.random as rd
G={0:[(1,3),(2,3)],1:[(2,5),(0,3)],2:[(0,3)]}
# Partie A question 2
def Voisin(G,i):
    V=[]
    for j in G[i]:
        V.append(j[0])
    return V
# Question 3
def Distance(G):
```

```
    n=len(G)
    D=-np.ones((n,n))
    for i in G:
        for u in G[i]:
            D[i,u[0]]=u[1]
    return D
    # Question 4
def initialisation(G):
    n=len(G)
    return Voisin(G,0),Distance(G),np.ones((n,n))

   # Question 5
def Visible(i,G,V):
    J=Voisin(G,i)
    Visit=[]
    for u in J:
        if u not in V:
            Visit.append(u)
    return Visit
   # Question 6
def longueur(D,C):
    L=0
    for j in range(len(C)-1):
        L+=D[C[j],C[j+1]]
    return L
    #Question 7
def maximum(Tau,i):
    M=0
    j=0
    for k in range(len(Tau[i])):
        if Tau[i,k]>M:
            j=k
    return j
#Partie B Question 1

def Loi(D,Tau,u,i,J):
    a,b,q,r=u
    P=[]
    S=0
    for v in J:
        S+=Tau[i,v]**a/D[i,v]**b

    for v in J:
        P.append((v,Tau[i,v]**a/D[i,v]**b)/S)

    return P

  # Question 2
def Freq(P):
      F=[]
      for u in P:
```

```
            F.append(u[1])
        return np.cumcum(F)

  #Question 3
def choix(P,J):
        F=Freq(P)
        r=rd.random()
        j=0
        while F[j]<r:
            j+=1
        return P[j][0]

#Question 4

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=Visible(j,G,V)
            if len(J)==0:
               return []
            P=Loi(D,Tau,u,j,J)
            j=choix(P,J)
            V.append(j)
        return V

 #Question 5
def Pheromone(D,Tau,Q,r,V):
      Tau=(1-r)*Tau
      if len(V)>0:
          L=longueur(D,V)
          for i in range(len(V)-1):
              Tau[V[i],V[i+1]]+=Q/L

 # Partie C Question 1

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

        C=[]
        n=len(G)
        for k in range(m):
            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:
                J=Visible(j,G,V)
                test=False
                if len(J)!=0:
                  test=True
                  P=Loi(D,Tau,u,j,J)
                  j=choix(P,J)
                  V.append(j)
            C.append(V)
        return C
    #Question 2
def Pheromone_m(D,Tau,Q,r,C):
      Tau=(1-r)*Tau
      for V in C:
          L=longueur(D,V)
          for i in range(len(V)-1):
               Tau[V[i],V[i+1]]+=Q/L
 # Question 3
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

def Optimum(G,u,m,Times):
    n=len(G)
    Tau=Fourmis(G,m,Times,u)
    V=[0]
    j=0
    while u!=n:
        j=maximum(Tau,j)
        V.append(j)
    return j
```