TP : Heuristiques

Dans ce TP on va travailler sur la résolution approchée de 2 problèmes classiques de la théorie de l'informatique avec la notion d'heuristique. Dans le cas du voyageur de commerce, on essaiera même d'aller plus loin en présentant une méthode métaheuristique : la descente locale.

1 Problème du sac à dos : exemple

On s'intéresse au problème du sac à dos avec les objets suivants :

Objet	1	2	3	4
Poids (kg)	3	2	1	4
Valeur (£)	4	3	1	9

en respectant la contrainte de poids maximal $W = 8$ kg.

1. 1)
   Justifier que la solution optimale est une solution où 3 objets sont mis dans le sac (donc que les solutions avec 1, 2 ou 4 objets ne sont pas optimales).
1. 2)
   En étudiant toutes les configurations avec 3 objets déterminer la solution optimale, en précisant quels sont les objets pris, le poids total et la valeur totale.
1. 3)
   On va dorénavant étudier le cas des algorithmes gloutons suivants les trois heuristiques suivantes :
   — le "meilleur" objet est le plus cher ;
   — le "meilleur" objet est le plus léger ;
   — le "meilleur" objet est celui qui a le meilleur rapport valeur/poids
   En appliquant l'algorithme glouton avec les 3 heuristiques, déterminer les solutions approchées qu'ils donnent. Commenter.

2 Problème du voyageur de commerce

Le problème du voyageur de commerce est un autre problème classique de théorie de l'informatique. Dans ce problème, on dispose de n villes et on cherche à déterminer la "boucle" qui passe par toutes les villes (et revient à la ville de départ) qui soit la plus courte.

Formellement, on donne une matrice carrée de taille n dont le coefficient M_{ij} donne la distance entre la ville i et la ville j et on veut obtenir une liste d'entiers entre 0 et $n-1$ codant la tournée la plus courte.

Par exemple, avec les valeurs suivantes :

	Clermont-Ferrand	Bordeaux	Toulouse	Lyon	Marseille
Clermont-Ferrand	0	376	377	167	475
Bordeaux	376	0	244	556	646
Toulouse	377	244	0	538	404
Lyon	167	556	538	0	314
Marseille	475	646	404	314	0

on peut imaginer une tournée Clermont → Toulouse → Bordeaux → Marseille → Lyon → Clermont de distance totale $377+244+646+314+167 = 1748$ km ou une tournée Toulouse → Lyon → Bordeaux → Marseille → Clermont → Toulouse de distance totale 2592 km donc beaucoup moins efficace.

Si l'on veut résoudre le problème de manière optimale, il faut déterminer toutes les combinaisons possibles, au nombre de $n!$, donc là encore une complexité exponentielle : dans le cas d'un grand nombre de villes, il vaut mieux avoir rapidement une solution pas trop éloignée de la solution optimale plutôt que chercher longtemps la solution optimale.

Un heuristique permettant de trouver une solution et suivant un algorithme glouton est la suivante : depuis la ville i on cherche à aller dans la ville j la plus proche parmi celle qui n'ont pas été encore visitées.

2. 1)
   Etant donnée une liste de n liste de n éléments codant la matrice des distances entre les n villes, écrire

une fonction `glouton` qui renvoie la liste des villes à parcourir pour la boucle suivant l'algorithme glouton. On pourra remarquer que la ville de départ n'a pas d'influence puisque le trajet est une boucle donc on prendra 0.

On souhaite maintenant essayer d'améliorer cet algorithme en utilisant une méta-heuristique. L'idée va donc consister en les opérations suivantes :

— d'abord, utiliser l'algorithme glouton afin d'obtenir une solution heuristique ;
— modifier légèrement la tournée en inversant deux villes dans celle-ci, et voir si la solution obtenue est meilleure ;
— si elle est meilleure, inversez les deux villes et recommencer les inversions avec la nouvelle tournée. Une fois que les inversions ne donnent plus de tournée meilleure, s'arrêter.

2. 2)
Ecrire un programme `longueur_tournee` qui renvoie la longueur d'une tournée obtenue à partir d'une matrice de distance M (liste de listes) et une tournée T (une liste d'entiers).

2. 3)
Ecrire une fonction `ameliorable` qui prend comme arguments une matrice M, une tournée T et deux indices i et j ($0 < i < j < n$) et qui renvoie un booléen disant si la tournée obtenue en inversant les villes en position i et j est plus courte.

2. 4)
Ecrire une fonction `meta` qui prend comme arguments une matrice M et une tournée T et applique la métaheuristique afin d'améliorer la solution obtenue.

2. 5)
Définir une fonction `comparaison` qui pour une matrice de distances de ville M renvoie la longueur de la tournée obtenue par l'algorithme glouton et celle de la métaheuristique. L'appliquer aux deux matrices de distances pour la liste de 5 villes et celle de 19 villes données ci-dessous.

```
M5=[[0,376,377,167,475],[376,0,244,556,646],
[377,244,0,538,404],[167,556,538,0,314],[475,646,404,314,0]]
noms_5villes={0:'Clermont', 1:'Bordeaux', 2:'Toulouse', 3:'Lyon', 4:'Marseille'}

M9 = [[0,182,830,918,822,841,974,393,719,705,884,520,863,738,895,627,781,1138,285],
[182,0,648,619,675,659,792,216,842,555,840,338,819,546,713,445,599,907,250],
[830,648,0,793,1051,536,719,610,866,1219,1379,310,1358,566,730,245,503,1002,855],
[918,619,793,0,279,510,466,402,197,523,597,586,876,300,276,562,425,304,679],
[822,675,1051,279,0,774,771,478,109,280,318,741,297,585,555,747,698,506,558],
[841,659,536,510,774,0,284,520,667,974,1092,398,1071,201,318,329,84,650,844],
[974,792,719,466,771,284,0,602,664,1005,1089,584,1068,215,192,506,216,498,926],
[393,216,610,402,478,520,602,0,366,609,797,300,776,369,514,372,444,690,324],
[719,842,866,197,109,667,664,366,0,326,425,598,404,462,473,640,591,434,506],
[705,555,1219,523,280,974,1005,609,326,0,206,909,185,772,799,958,898,769,407],
[884,840,1379,597,318,1092,1089,797,425,206,0,1069,21,915,873,1065,1015,824,586],
[520,338,310,586,741,398,584,300,598,909,1069,0,1048,379,549,106,357,795,540],
[863,819,1358,876,297,1071,1068,776,404,185,21,1048,0,894,852,1044,995,803,565],
[738,546,566,300,585,201,215,369,462,772,915,379,894,0,147,331,117,442,678],
[895,713,730,276,555,318,192,514,473,799,873,549,852,147,0,498,233,330,824],
[627,445,245,562,747,329,506,372,640,958,1065,106,1044,331,498,0,290,771,652],
[781,599,503,425,698,84,216,44,591,898,1015,357,995,117,233,290,0,566,768],
[1138,907,1002,304,506,650,498,690,434,769,824,795,803,442,330,771,566,0,940],
[285,250,855,679,558,844,926,324,506,407,586,540,565,678,824,652,768,940,0]]
noms_19villes={0:'Biarritz', 1:'Bordeaux', 2:'Brest', 3:'Dijon', 4:'Grenoble',
5:'Le Havre', 6:'Lille', 7:'Limoges', 8:'Lyon', 9:'Marseille', 10:'Monaco',
11:'Nantes', 12:'Nice', 13:'Paris', 14:'Reims', 15:'Rennes', 16:'Rouen',
17:'Strasbourg', 18:'Toulouse'}
```

2. 6)
Expliquer le but du programme suivant, sachant que la fonction `permutations` du module `itertools` fournit la liste de tous les tuples obtenus en faisant des permutations de la liste en argument.

```
import itertools
```

```

def exhaustif(M) :
    n = len(M)
    indices = [i for i in range(1,n)]
    tournees = list(itertools.permutations(indices))
    T_id = [0] + list(tournees[0]) + [0]
    l_id = longueur_tournee(M,T_id)
    for T in tournees :
        T = [0] + list(T) +[0]
        l = longueur_tournee(M,T)
        if l < l_id :
            T_id = T
            l_id = l
    return T_id,l_id
```

2. 7)
   Comparer le temps d'exécution des deux programmes et la qualité des solutions obtenues pour la liste de 11 villes suivantes. Commenter.

```
M11 = [[0, 619, 659, 216, 842, 555, 338, 546, 713, 445, 907],
[619, 0, 510, 402, 197, 523, 586, 300, 276, 562, 304],
[659, 510, 0, 520, 667, 974, 398, 201, 318, 329, 650],
[216, 402, 520, 0, 366, 609, 300, 369, 514, 372, 690],
[842, 197, 667, 366, 0, 326, 598, 462, 473, 640, 434],
[555, 523, 974, 609, 326, 0, 909, 772, 799, 958, 769],
[338, 586, 398, 300, 598, 909, 0, 379, 549, 106, 795],
[546, 300, 201, 369, 462, 772, 379, 0, 147, 331, 442],
[713, 276, 318, 514, 473, 799, 549, 147, 0, 498, 330],
[445, 562, 329, 372, 640, 958, 106, 331, 498, 0, 771],
[907, 304, 650, 690, 434, 769, 795, 442, 330, 771, 0]]
noms_11villes = {0:'Bordeaux', 1:'Dijon',2:'Le Havre',3:'Limoges', 4:'Lyon',
 5:'Marseille', 6:'Nantes',7:'Paris', 8:'Rennes',9:'Strasbourg'}
```