TP 1

Problème du voyageur de commerce

Nous allons étudier le parcours de graphes non orientés et à la résolution du problème du voyageur de commerce, tant de manière exacte par **force brute** dans des cas simples que de manière approchée à l'aide d'**algorithmes génétiques** dans des cas plus complexes.
Un graphe G est un **graphe hamiltonien** s'il possède au moins un cycle passant par tous les sommets de G exactement une fois ; un tel cycle est appelé **cycle hamiltonien**.

Un représentant de commerce part de sa ville d'origine et doit passer visiter ses clients dans des villes différentes, une fois et une seule. Il a bien sûr intérêt à minimiser la longueur du cycle qu'il va faire et il souhaite rentrer dans sa ville d'origine. Ce problème est appelé «problème du voyageur de commerce» (*salesman problem*) et est apparu dans les années 1930. Avec le vocabulaire introduit plus haut, il consiste à trouver, dans un graphe hamiltonien (pondéré), un cycle hamiltonien de longueur minimale.

Résolution par force brute

Une approche naïve (dite par force brute) de résolution du problème du voyageur de commerce consiste à tester toutes les boucles hamiltoniennes d'origine donnée, puis à sélectionner celle qui a la plus petite longueur.

A titre d'illustration, si on estime qu'il faut une milliseconde pour tester un trajet, cela nous donne les durées d'exécution suivantes pour le nombre de villes indiqué

Nb de villes	Nb de possibilités	Temps de calcul
5	12	12 millisecondes
10	181 440	0,18 seconde
15	43 milliards	12 heures
20	60 E+15	1928 ans
25	310 E+21	9,8 milliards d'années (!)

1) Écrire une fonction récursive **genere_permutations(L)** prenant en argument une liste L et renvoyant la liste de toutes les permutations de ses éléments, chaque permutation étant elle-même codée dans une liste. Par exemple, genere_permutations([0,1,2])renverra (dans cet ordre ou un autre) :
[0,1,2], [0,2,1], [1,0,2], [1,2,0], [2,0,1], [2,1,0]]

2) Écrire une fonction **genere_boucle(n , depart)** prenant en argument un entier n et un entier depart et renvoyant une liste de toutes les boucles hamiltoniennes possibles d'origine (et d'arrivée) départ sur un graphe complet d'ordre n dont les sommets sont numérotés par des nombres de 0 à $n-1$. On se servira de la fonction précédente.

3) Écrire un fonction **distances_boucles(LB, T)** prenant en argument une liste de boucles hamiltoniennes LB codées par des listes d'entiers et la matrice des poids T du graphe complet où les boucles sont situées, et qui renvoie la liste des distances associées à chacune des boucles dans le même ordre que celui des boucles.

4) Écrire une fonction **meilleure_boucle(LB,T)** de mêmes arguments que **distances_boucles** renvoyant un tuple contenant l'indice de la boucle la plus courte et son poids total.

5) On dispose d'une liste de coordonnées de points repérés dans un repère orthonormé du plan affine Euclidien, codée sous forme de tuples de longueur 2.
Écrire une fonction **coord_vers_marice(LC)** qui prend en argument une telle liste et renvoie une matrice des distances Euclidiennes entre les couples de points.

6) On dispose des coordonnées des points suivants :
$A(0,0)$, $B(1,1)$, $C(2,4)$, $D(1,-3)$, $E(0,-5)$, $F(0,4)$, $G(-1,-5)$, $H(-2,3)$ et $I(-3,0)$.
Résoudre le problème du voyageur de commerce sur le graphe ayant ces points comme sommets, la distance étant la distance Euclidienne. Le voyageur part du point A.

Résolution par algorithme génétique

Le problème du voyageur de commerce ne peut pas être résolu par force brute : pour 25 villes disposées aléatoirement, il faut attendre plus de 9 milliards d'années pour avoir la solution minimale avec cette méthode.
On cherche donc des solutions approchées, accessibles en un temps raisonnable.

Nous allons chercher une réponse approchée par un algorithme génétique : pour commencer nous allons créer une matrice de poids associée à un graphe.

Création de la matrice des poids

1) Écrire une fonction sans argument qui renvoie une liste de 25 tuples (ou listes) de la forme (x, y) générée aléatoirement entre les valeurs (0, 20) correspondant aux coordonnées des villes (points/sommets …).

2) Créer la matrice des poids (des distances) associée à ce problème.

Algorithme génétique

On part d'une génération initiale d'individus aléatoires, **un individu étant ici une boucle hamiltonienne sur le graphe** et correspond donc à une solution approchée (pas forcément performante et encore moins optimale) du problème du voyageur de commerce.

On écrit ensuite une fonction permettant de créer la génération suivante à partir des principes suivants :

- ☞ plus un individu est performant (i.e plus la route qu'il emprunte est courte), plus il a de chances de se reproduire. Un tirage au sort entre individus reproducteurs est conduit selon le principe de la roulette, principe qui sera détaillé plus loin ;
- ☞ à chaque nouvelle génération, les individus peuvent voir leur code génétique muter avec une probabilité p, qui est un paramètre (inconnu...). Une **mutation** consiste à sélectionner deux indices dans le code génétique d'un individu et à inverser le code génétique de cet individu entre ces deux indices. Ici, cela signifie inverser le sens du trajet entre deux villes dans la boucle hamiltonienne définissant l'individu (ce qui préserve évidemment le caractère hamiltonien du parcours) ;
  Par exemple si on a l'individu [5, 3, 2, 1, 4, 0, 8, 7, 6] et que l'on procède à une mutation entre les termes d'indice 1 et 5, on arrive à l'individu [5, **0, 4, 1, 2, 3**, 8, 7, 6] ;
- ☞ deux parents créent deux enfants par un système de ***croisement en un point*** : le premier donne le début (dont l'indice de fin est déterminé aléatoirement) de son parcours à un fils et le second parent donne le début de son parcours à l'autre fils. Les parcours des fils sont alors complétés par les parcours du parent dont le « code génétique »(ici, la boucle hamiltonienne qui le définit) n'a pas été utilisé, en ajoutant à la fin du code génétique du fils les sommets qui n'y apparaissent pas encore, pris dans l'ordre d'apparition dans le code du second parent.
  Par exemple si on a tiré l'indice 4 et que les deux parents sont les individus
  $p1$ = [5, 3, 2, 1, 4, 7, 8, 0, 6] et $p2$ = [3, 1, 0, 5, 8, 6, 4, 2, 7] alors les deux fils seront les individus
  $f1$ = [**5, 3, 2, 1**, 0, 8, 6, 4, 7] et $f2$ = [**3, 1, 0, 5**, 2, 4, 7, 8, 6]

3) La fonction **random.shuffle(L)** mélange aléatoirement la liste L (et retourne None).
En utilisant la fonction **random.shuffle**, écrire une fonction **cree_initiale(N)** qui renvoie la génération initiale, codée comme liste de listes. Cette génération comportera N individus, qui seront chacun représentés par une liste (des entiers de 0 à 24) contenant les indices des 25 villes dans un ordre aléatoire.

4) Écrire une fonction **meilleure_individu(population, M)** qui prend en argument une liste de listes d'individus représentés par leur parcours (que l'on n'oubliera pas de fermer) ainsi que la matrice des poids, et qui renvoie un tuple contenant la distance totale parcourue par le meilleur individu de population et son parcours.

5) Écrire une fonction **mutation(individu, probmut)** qui prend en argument un individu décrit par son cycle hamiltonien et codé sous forme d'une liste et qui renvoie l'individu une fois son code génétique muté, comme décrit en début de cette partie. Les indices de mutations sont aléatoires, générés par **random.sample()**.

6) Écrire une fonction **crossover(p1, p2)** prenant en argument deux individus parents p1 et p2 et renvoyant leurs deux fils, obtenus par le procédé de croisement décrit en début de cette partie.

7) La roulette est une des façons de sélectionner les individus reproducteurs. Un individu donné a une probabilité proportionnelle à $f(d)$ de se reproduire où f est une fonction donnée décroissante et d la distance de parcours de l'individu.
Plus précisément, on dispose au départ d'une génération, codée sous forme de liste de listes. On calcule la liste D des distances totales parcourues par chaque individu, puis la liste F des $f(d)$. À partir de cette liste F, on crée une liste R, représentant la fonction de répartition de la variable aléatoire X, où $P(X = x_k)$ est
par définition la probabilité que l'individu d'indice k se reproduise. On a ainsi : $R_k = \frac{\sum_{j=0}^{k} F_j}{\sum_{j=0}^{N-1} F_j}$
On tire au hasard un flottant x entre 0 et 1. L'entier k tel que $R_k \leq x < R_{k+1}$ donne l'indice de l'individu choisi pour se reproduire.
Écrire une fonction **genere_roulette(population, T)** qui prend en argument une génération et la matrice des poids T et qui renvoie la liste r décrite ici. On pourra prendre :
$f(d) = 1/(d - 0.99m)\text{^}3$, m étant la distance parcourue parcourue par l'individu le plus performant de la génération considérée.

8) Écrire une fonction **indiceroulette(R)**, qui prend en argument la liste précédemment construite et qui renvoie l'indice d'un individu tiré au sort pour se reproduire.

9) Créer une fonction **generation_suivante** en utilisant les éléments précédents (génération par *cross-over* et mutation aléatoire).

10) La tester sur plusieurs générations en prenant soin de ne pas lancer de calculs trop longs (en travaillant par exemple sur une partie de la liste de ville et sur peu de générations).
Exemples de sorties sur 25 villes, des meilleurs individus des générations 1, 2, 4, 9, 40 et 1