Le voyageur de commerce

PSI 2022/2023

Un représentant de commerce doit visiter un certain nombre de villes pour vendre ses marchandises. Le problème pour le voyageur est de trouver le chemin le plus court permettant de passer une et une seule fois par chacune des villes et de revenir au point de départ.

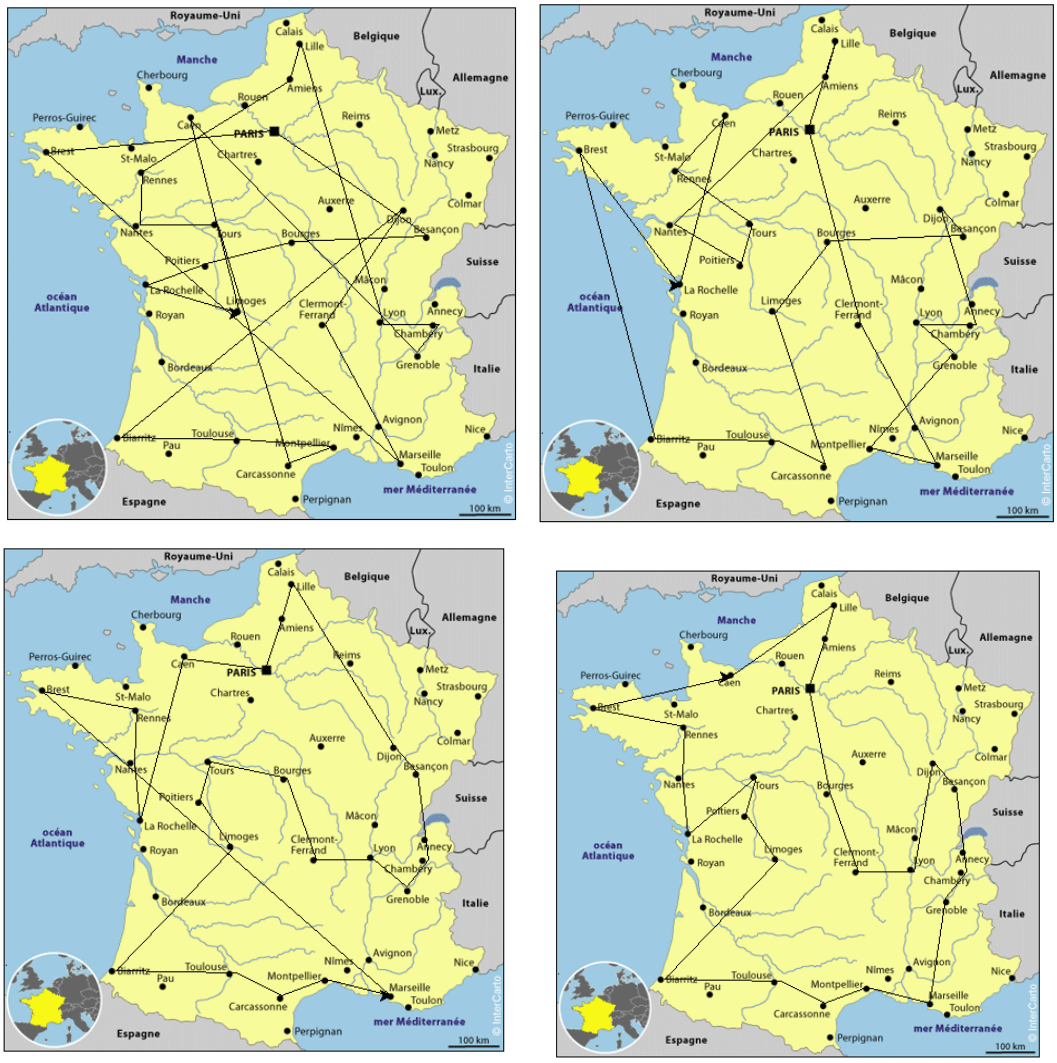

FIGURE 1 – *Parcours d'un voyageur de commerce traversant 23 villes françaises. En haut à gauche : trajet initial aléatoire. Hen haut à droite : trajet après 100 optimisations. En bas à gauche : après 300 optimisations. En bas à droite : après 1000 optimisations.*

Ce problème en apparence très simple est en fait incroyablement difficile : il n'existe pas à l'heure actuelle d'algorithme permettant de trouver la solution exacte en un temps raisonnable. Il est néanmoins

possible de trouver le chemin optimal avec une *métaheuristique*, un algorithme d'optimisation basé sur des probabilités. C'est cette méthode que nous allons utiliser aujourd'hui.

Mise en place

Dans la suite, on s'intéresse aux 23 villes françaises suivantes.

n°	Villes	Coordonnées
0	Limoges	(-20,-45)
1	La Rochelle	(-105,-20)
2	Poitiers	(-50,0)
3	Bourges	(30,20)
4	Besançon	(150,25)
5	Paris	(10,120)
6	Brest	(-195,100)
7	Marseille	(125,-180)
8	Clermont	(57,-55)
9	Dijon	(128,50)
10	Chambéry	140,-80
11	Biarritz	(-127,-157)
12	Toulouse	(-17,-159)
13	Montpellier	(65,-165)
14	Carcassone	(25,-180)
15	Tours	(-40,35)
16	Nantes	(-110,35)
17	Rennes	(-108,83)
18	Amiens	(25,165)
19	Lille	(35,200)
20	Lyon	(110,-55)
21	Grenoble	(160,-55)
22	Caen	(-65,130)

Les coordonnées indiquées dans ce tableau correspondent à la position en pixels de chaque ville sur la carte du fichier *France.gif* à télécharger sur le site cahier de prépa. Ces coordonnées sont stockées sous la forme d'une liste de tuples nommée `ville` dans le script Python voyageur.py.

1. A l'aide de la fonction `shuffle`(L) du module `random`, créer une liste `chemin_init` contenant une permutation aléatoire de la liste des entiers de 0 à 22. Par exemple :

```
chemin_init = [4, 18, 10, 11, 16, 9, 7, 3, 12, 1, 13, 2, 8, 6, 15, 0, 14, 5,
    17,20,21,22]
```

2. A l'aide du module turtle, remplir la fonction `tracer_chemin(villes,chemin)` qui trace le chemin parcouru par un voyageur parcourant les 20 villes dans l'ordre donné par la liste `chemin` et revenant à son point de départ. On pourra utiliser les fonctions suivantes :

bgpic('France.gif')	utilise le fichier 'France.gif' comme fond d'écran
up(), down()	soulève la tortue, abaisse la tortue
goto(x,y)	déplace la tortue jusqu'au point de coordonnées (x, y)

3. On note M_{ij} la distance entre deux villes d'indice i et j de la liste `villes`. Remplir la fonction `matrice_distances(villes)` qui renvoie la matrice des distances `M` contenant toutes les distances M_{ij} (on remarquera que la matrice est symétrique et que tous ses éléments diagonaux sont nulles).

4. Ecrire une fonction `distance_parcourue(M,chemin)`, où `M` est la matrice des distances précédente, qui renvoie la distance totale `d` parcourue par un voyageur traversant toutes les villes dans l'ordre donné par la liste `chemin` et revenant à son point de départ.

La métaheuristique

Une métaheuristique est un algorithme d'optimisation visant à résoudre des problèmes d'optimisation difficile (souvent issus des domaines de la recherche opérationnelle, de l'ingénierie ou de l'intelligence artificielle) pour lesquels on ne connaît pas de méthode classique plus efficace.

La métaheuristique utilisée ici s'appuie sur l'algorithme de Métropolis (1953) utilisé pour décrire l'évolution d'un système thermodynamique. Elle se divise en deux étapes :

a) on construit un nouveau chemin de parcours en effectuant une permutation aléatoire de deux villes dans le chemin de parcours initial.

b) On calcule la distance parcourue avec la permutation et on la compare à la distance parcourue sans permutation. On applique le critère de Métropolis. Si ce critère est validé, la permutation est acceptée et le voyageur parcourt le nouveau chemin. Si le critère n'est pas validé, le voyageur reste sur le chemin initial.

Permutation

On change l'ordre de parcours des villes en changeant de façon aléatoire l'indice de deux villes dans la liste `chemin`. Pour cela, on choisit deux entiers aléatoires i et j tel que $0 \leq i < j \leq N-1$ et on permute les villes correspondantes :

$$\begin{pmatrix} \texttt{chemin}[0] \\ \dots \\ \texttt{chemin}[i] \\ \texttt{chemin}[i+1] \\ \dots \\ \texttt{chemin}[j-1] \\ \texttt{chemin}[j] \\ \dots \\ \texttt{chemin}[N-1] \end{pmatrix} \longrightarrow \begin{pmatrix} \texttt{chemin}[0] \\ \dots \\ \texttt{chemin}[j] \\ \texttt{chemin}[j-1] \\ \dots \\ \texttt{chemin}[i+1] \\ \texttt{chemin}[i] \\ \dots \\ \texttt{chemin}[N-1] \end{pmatrix}$$

5. Remplir la fonction `permut(chemin)` qui effectue l'opération ci-dessus et renvoie la nouvelle liste `nv_chemin`. On rappelle que la fonction `randint(a,b)` du module `random` renvoie un entier aléatoire compris entre a et b (inclus).

Critère de Métropolis

Il s'énonce comme suit :

- Soit la distance parcourue par le voyageur en suivant le nouveau chemin est inférieure à la distance initialement parcourue. Dans ce cas, la permutation est acceptée.
- Soit la distance $d(C')$ est supérieure à la distance $d(C)$. Dans ce cas, la permutation est acceptée avec une probabilité p tel que :

$$p \leq \mathrm{e}^{\frac{d(C')-d(C)}{T}}$$

où T est un paramètre appelé température effective du système [1]. Plus cette température est grande, plus des configurations "mauvaises" seront acceptées par l'algorithme. A contrario, si la température est très basse, seules des perturbations faisant diminuer la distance parcourue par le voyageur seront acceptées. Il est très important d'accepter de "mauvaises" perturbations car elles permettent d'explorer une plus grande partie de l'espace des solutions et ainsi d'éviter de s'enfermer trop vite autour d'un minimum local.

6. Ecrire une fonction `Metropolis(chemin,M,T)` prenant en argument une liste `chemin`, la matrice des distances `M`, et un flottant `T`, et renvoyant la liste contenant le chemin parcouru par le voyageur de commerce après permutation et application du critère de Métropolis.

Abaissement de la température

En faisant décroître la température petit à petit, le système finit par "geler" dans un état de distance minimale. A chaque itération, on fera décroître la température suivant une suite géométrique :

$$T_{i+1} = (1-x)T_i$$

où $x = 0.001$.

7. Ecrire une fonction `Voyageur(T0,Niter)`, une température initiale T_0 et le nombre d'itérations désiré N_{iter}, et permettant d'obtenir la solution optimale suivant l'algorithme du recuit simulé. Tracer la solution obtenu pour $T_0 = 50$ et $N_{\text{iter}} = 200, 2000, 15000$.

1. On reconnaît le facteur de Boltzmann des physiciens !