Le problème du voyageur de commerce

TSP : Travelling Salesman Problem

1 Introduction

Un problème d'optimisation combinatoire repose sur la donnée d'un ensemble $\mathcal{E}$ et d'une fonction $f : \mathcal{E} \mapsto \mathbb{R}$. Il s'agit de déterminer l'élément e de $\mathcal{E}$ qui optimise $f(e)$. Un algorithme, dit de *force brute*, consiste à énumérer tous les éléments e de $\mathcal{E}$ et à calculer $f(e)$ pour chacun d'eux. Mais cet algorithme devient vite inutilisable en pratique, à cause de la taille de l'ensemble $\mathcal{E}$. On a alors recours à des algorithmes appelés *heuristiques*. Une heuristique est un algorithme qui permet de déterminer rapidement une solution acceptable : la vitesse d'exécution est privilégiée à la qualité de la solution.

2 Le problème

Un voyageur de commerce doit visiter n villes. Il s'agit de déterminer la tournée la plus courte qui passe une et une seule fois par chacune des villes et se termine par la ville d'origine. La situation peut être représenté par un graphe pondéré non orienté $G = (S, A, \omega)$ où chaque sommet de S représente une ville et où chaque arête de A représente une liaison entre 2 villes. Le poids $\omega(x, y)$ d'une arête xy est égal à la distance (en km) entre les villes représentées par les sommets x et y. Une tournée correspond alors à un cycle Hamiltonien et le problème consiste à déterminer un cycle Hamiltonien de poids minimum.

Pour simplifier le problème, on considère que les distances entre villes sont des distances à vol d'oiseau et que G est un graphe complet d'ordre n et de taille $n(n-1)/2$. Dans ce cas, toute permutation de l'ensemble des sommets représente une tournée possible : il y a $n!$ permutations. Cependant, une tournée donnée est représentée par $2n$ permutations. En effet, une tournée étant un cycle, tout sommet de ce cycle peut être un point de départ. De plus, comme le graphe est non orienté, un cycle peut être parcouru dans un sens comme dans l'autre. Au final, il existe exactement $(n-1)!/2$ tournées distinctes. Considérons par exemple un graphe de 4 sommets $S = \{a, b, c, d\}$: les tournées *abcda*, *bcdab*, *cdabc* et *dabcd* sont de même poids (il reste $(4!)/4 = 3!$ permutations possibles) ; les 2 tournées *abcda* et *adcba* sont aussi de même poids (il reste alors $3!/2 = 3$ tournées).

Le problème du voyageur de commerce est donc un problème d'optimisation combinatoire : l'ensemble $\mathcal{E}$ est un ensemble de $(n-1)!/2$ tournée et la fonction objectif que l'on souhaite ici minimiser correspond au poids (en km) d'une tournée.

3 Heuristiques recherchant une tournée acceptable

L'objectif est d'obtenir rapidement une solution qui soit la plus proche possible de la solution optimale.

3.1 Heuristique aléatoire

Cette heuristique consiste à générer aléatoirement un grand nombre de permutations en espérant en trouver une de bonne qualité. En pratique, on se fixe une limite de temps (à vous de la définir) et on ne retient que la meilleure solution obtenue.

3.2 Heuristique du plus proche voisin

On démarre avec une tournée T ne contenant qu'une seule ville. Puis, à chaque itération, on ajoute à T la ville la plus proche de la dernière ville visitée de T. Le processus se termine quand toutes les villes ont été ajoutées à la tournée.

Il s'agit d'une heuristique dite *gloutonne* (*greedy heuristic*) : à chaque itération une nouvelle ville est ajoutée à la tournée sans jamais remettre en cause les ajouts précédents. A vous de voir comment exploiter au mieux cette heuristique.

3.3 Votre proposition

Faire des recherches et proposer au moins une autre heuristique.

4 Heuristiques d'amélioration

Ces heuristiques ont pour objectif d'améliorer une solution existante. En pratique, on recherche une solution en utilisant l'une des heuristiques présentées dans le paragraphe §3 et on essaie d'obtenir une meilleure solution à l'aide d'une heuristique d'amélioration.

4.1 Heuristique d'échange de 2 sommets

Étant donnée une tournée T, on regarde si l'échange de 2 villes dans T produit une tournée de meilleure qualité que T. Par exemple, si on échange b et d dans la tournée $T = abcde$ on obtient une nouvelle tournée $T' = adcbe$. En pratique, on effectue des améliorations successives jusqu'à ce qu'il ne soit plus possible d'améliorer la solution courante. On peut aussi arrêter le processus quand une limite de temps fixée au préalable est dépassée.

4.2 Heuristique de (dé)croisement de deux arêtes

Soit une tournée $T = \cdots ab \cdots cd \cdots$. On (dé)croise les deux arêtes ab et cd pour obtenir la nouvelle tournée $T' = \cdots ad \cdots bc \cdots$. Il suffit ensuite de comparer T et T' et de conserver la meilleure des deux. En pratique, on procède comme pour l'heuristique précédente.

5 Algorithme de force brute

Ce type d'algorithme énumère toutes les tournées possibles, calcule le poids de chacune d'elles et fournit la solution optimale. On testera cet algorithme sur des graphes d'ordre 10 à 20 au plus (à vous de voir). Il s'agit ici d'analyser l'évolution du temps d'exécution en fonction du nombre de villes.

Indication : utiliser le parcours en profondeur d'abord (dfs) et améliorer son usage (par exemple en évitant de poursuivre les visites à partir d'un sommet si cela ne permet pas d'améliorer la solution courante).

6 Les instances

Sur Arche, vous trouverez différentes instances à résoudre. La plus simple est l'instance `communes_10.txt` qui contient la liste des arêtes et leur poids pour un ensemble de 10 villes. La plus grande instance est `communes_777.txt` pour un ensemble de 777 villes. Notez que les sommets sont identifiés par les entiers allant de 0 à $n-1$.

Toutes les instances respectent un même format : la première ligne contient le nombre de sommets n et le nombre d'arêtes $m = n(n-1)/2$. Puis suivent m lignes, chaque ligne décrivant une arête sous la forme : $x \quad y \quad \omega(x,y)$ où $x < y$ et où $\omega(x,y)$ est la distance en kilomètres séparant les villes x et y (distance à vol d'oiseau).

7 Travail demandé

Dans un premier temps, il s'agit de programmer les 5 heuristiques et l'algorithme de force brute en langage `C/C++` ou en Python. Vous devrez ensuite utiliser vos programmes pour résoudre au mieux les différentes instances proposées. Une analyse et un bilan des résultats devra être ensuite présenté.

8 Question bonus

Visualiser les tournées sur une carte de France. Le fichier `communes.xlsx` contient diverses information concernant les communes. On y trouvera les coordonnées GPS des 777 communes mais aussi les *coordonnées cartésiennes* (obtenues par projection sur un plan). Le graphique (nuage de points) se trouvant dans ce fichier est obtenu à l'aide de ces coordonnées cartésiennes.