FIPA 1A : Programmation avancée - Projet

M. Léger

19 Avril 2019

Le but de ce projet est de résoudre le problème du voyageur du commerce sur des graphes complets avec distance euclidienne.

I. Problème du voyageur de commerce

Le problème du voyageur de commerce consiste en la recherche d'un cycle Hamiltonien (cycle passant une unique fois par chaque nœud) de poids minimal dans un graphe donné.

Pour simplifier le problème, nous allons considérer pour ce projet uniquement des graphes complets, où chaque nœud du graphe est donné par un point de coordonnées x et y réelles. Le poids d'une arrête entre deux nœuds du graphe est simplement la distance euclidienne entre les deux points du plan que représentent ces nœuds.

Autrement dit, si on considère que les points sont des villes (ou des points d'intérêt quelconques) sur une carte, le but est de trouver un chemin le plus court possible, qui passe une seule et unique fois par chaque ville, sauf pour la ville de départ, qui est également la ville d'arrivée (on forme bien un cycle). Par convention, on considérera qu'on part toujours du premier point du fichier.

II. Format de données

Les données d'entrée sont au format *csv*. La première ligne indique le nombre n de villes et les n lignes suivantes décrivent chacune l'emplacement d'une ville sous la forme x,y où x est l'abcisse et y l'ordonnée de la position de la ville, sous forme de nombres flottants.

Exemple 1 *Prenons par exemple le fichier suivant:*

```
4
0.0,1.0
1.0,1.0
1.0,0.0
0.0,0.0
```

Nous nommerons ces points 1, 2, 3 et 4 dans l'ordre d'apparition (voir figure 1). La distance (ou le poids d'une arête) entre deux points P *et* Q *est simplement la distance euclidienne :*

$$dist(P,Q) = \sqrt{(x_P - x_Q)^2 + (y_P - y_Q)^2}$$

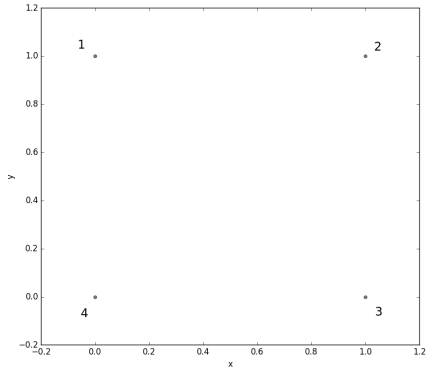

Figure 1: Représentation (géo)graphique des quatre villes de l'exemple 1.

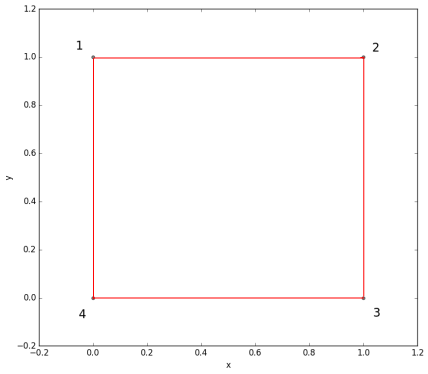

Figure 2: Solution optimale pour l'exemple 1

Ainsi, on a $dist(1,2) = dist(2,3) = dist(3,4) = dist(4,1) = 1$
et $dist(1,3) = dist(2,4) = \sqrt{2}$.

La distance totale d'un cycle est la somme des distances des arêtes du cycle. Dans cet exemple, il n'existe que trois cycles Hamiltoniens différents (comme on est dans un graphe non-orienté, on ne se soucie pas du sens de parcours d'un cycle) : le parcours optimal est 1-2-3-4-1 *(figure 2) et sa distance totale (ou son poids) est de* 4. *Les deux autres cycles Hamiltoniens sont* 1-2-4-3-1 *(figure 3) et* 1-3-2-4-1 *(figure 4). Ils ne sont pas optimaux car leur distance totale est de* $2 + 2 * \sqrt{2}$, *qui est plus grande que* 4.

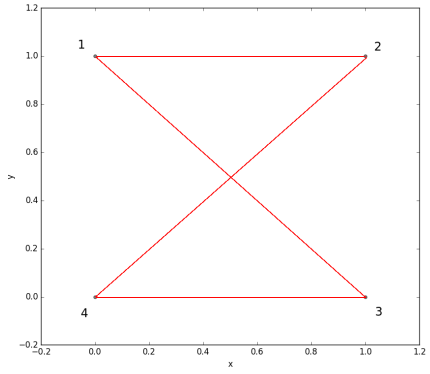

Figure 3: Solution non optimale pour l'exemple 1.

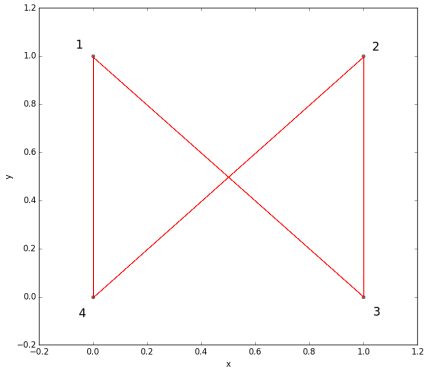

Figure 4: Solution non optimale pour l'exemple 1

III. Complexité, solutions approchées

On a vu dans l'exemple précédent que l'on pouvait trouver la solution optimale en énumérant toutes les possibilités et en choisissant le cycle le plus court. Mais ce n'est pas toujours possible, à cause de la complexité du problème. En effet, pour un problème à n villes, il existe de l'ordre de $O(n!)$ cycles Hamiltoniens possible (plus précisément, il en existe $\frac{(n-1)!}{2}$). En supposant que le calcul d'un cycle et de sa

distance totale ne prend qu'une microseconde, les estimations de temps nécessaires au calcul de toutes les solutions possibles en version naïve pour n villes est donné par la table 1.

Nombre de villes	Nombre de solutions possibles	Temps de calcul estimé
4	3	$\sim$ 3 microsecondes
6	60	$\sim$ 60 microsecondes
8	2520	$\sim$ 2.5 millisecondes
10	181440	$\sim$ 181.4 millisecondes
12	19958400	$\sim$ 19.9 secondes
14	$\sim 3.1 * 10^9$	$\sim$ 51.9 minutes
16	$\sim 6.5 * 10^{11}$	$\sim$ 7.6 jours
18	$\sim 1.8 * 10^{14}$	$\sim$ 5.6 années
20	$\sim 6.1 * 10^{16}$	$\sim$ 1.9 millénaires
22	$\sim 2.5 * 10^{19}$	$\sim$ 0.8 millions d'années
24	$\sim 1.3 * 10^{22}$	$\sim$ 409.6 millions d'années
26	$\sim 7.8 * 10^{24}$	$>$ âge de l'Univers

Table 1: Estimation du temps de calcul pour toutes les solutions de manière naïve

On se rend bien compte que pour de grandes valeurs de n, le calcul de toutes les solutions est impossible. Dans ce cas, on crée ce qu'on appelle des *heuristiques* : des solutions approchées, ou non-optimales. Le but est alors de trouver un cycle Hamiltonien qui soit le plus court possible (en distance totale), sans forcément savoir si on a réussi à obtenir le meilleur ou s'il en existe un encore plus court.

IV. **Travail demandé**

Les algorithmes utilisés et le langage d'implémentation sont au choix. Les programmes devront au moins prendre comme argument un nom de fichier à traîter et retourner, en sortie standard ou bien dans un fichier:

- le meilleur cycle trouvé sous la forme *solution:1-x-x-x-...-1* (évidemment, pas x mais les points choisis pour le cycle)
- la distance totale du cycle, sous la forme *distance:xxx*
- le temps de calcul mesuré depuis la lecture du fichier jusqu'a l'affichage (ou l'écriture) de la solution, sous la forme *temps:xxx secondes*

Questions:

(1) Proposez un programme permettant de trouver **la solution optimale**. Le programme sera évalué sur le problème donné par le fichier *test10.csv* (ou sur un problème similaire) à partir des trois points suivants : méthode algorithmique, implémentation et temps de calcul. *(Environ un tiers de la note)*

(2) Proposez un programme permettant de trouver **une solution qui soit la meilleure possible**. Le programme sera évalué à partir des quatre points suivants : méthode algorithmique, implémentation, distance totale de la solution trouvée et temps de calcul. Le problème choisi pour l'évaluation dépendra de l'efficacité de la solution proposée. *(Environ deux tiers de la note)*

V. **Indications**

- Pour la première question, tout est déjà expliqué dans le sujet. Pour la seconde question, il va falloir être imaginatif (ou bien chercher sur Internet) et produire une heuristique la plus efficace possible.
- Pour cela, une première idée intéressante est celle des plus proche voisins : lorsque vous êtes sur une ville donnée, vous pouvez calculer les distances par rapport à chacune des autres villes non visitées, et choisir d'aller à la plus proche et ainsi de suite.
- Si vous avez réussi à réaliser cette première heuristique, il est possible d'analyser le résultat pour identifier les améliorations possibles. Une amélioration classique consiste à éviter les croisements. En effet, si un cycle contient un ou plusieurs croisements, il est possible d'en trouver un plus court en changeant l'ordre de parcours pour supprimer ces croisements (voir figures 5 et 6). La difficulté de cette amélioration vient alors de ces questions : comment détecter automatiquement les croisements? Si on supprime un croisement, comment choisir le nouvel ordre de parcours des points concernés?
- Il existe d'autres heuristiques plus ou moins efficaces et plus ou moins compliquées, comme par exemple des algorithmes génétiques. Il n'est pas conseillé de s'y intéresser avant d'avoir déjà implémenté et testé votre première heuristique (la plus simple étant celle des plus proche voisins).

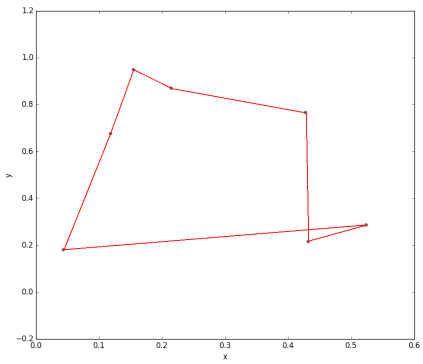

Figure 5: Exemple de cycle Hamiltonien avec un croisement.

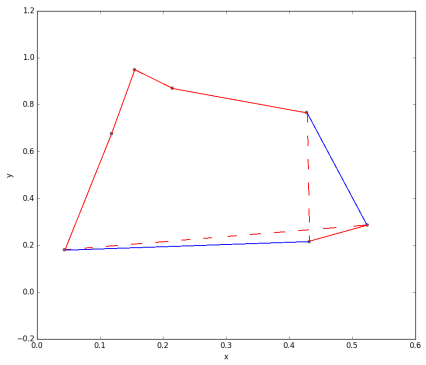

Figure 6: Amélioration du cycle : le croisement est supprimé (en pointillés) et de nouveaux arcs sont rajoutés (en bleu) pour compléter le cycle