Projet sur les Graphes - SageMath/Python

Ines Abdeljaoued Tej [1]

Travail attendu : Un rapport avec un maximum de 4 pages converti au format PDF avec une introduction (quel est le problème à résoudre et un aperçu des donnés), la présence d'au moins un graphique, la comparaison de deux modèles sur le même jeu de données et une conclusion.

Les questions à se poser : Le modèle est-il exploitable avec ses performances ? Ses performances vont-elles se dégrader dans le temps ? Le modèle peut-il passer l'échelle et être entraîné sur des jeux de donnés 10, 100 voire 1000 fois plus grand ?

Note finale : Le projet doit être réalisé par un seul élève. On attribue une note finale sur 5 qui est comptée à raison de 2 pts pour la présentation, 2 pts pour les grahiques, 1 pt pour l'implémentation informatique et les simulations numériques utilisées.

Travail à faire : Commencez par installer SageMath sur votre ordinateur (Voir Mme Abdeljaoued pour le format de fichier à installer) ou alors ouvrez un compte sur le cloud cloud.sagemath.org afin de travailler en ligne (solution temporaire à déconseiller). Téléchargez les modules de cartographie nécessaires à la finalisation du projet (voir liens ci-dessous).
Il y a deux volets dans ce projet : d'abord implémenter l'algorithme de la section 1 et rendre un graphe sur une carte géographique. Ensuite, appliquez les connaissances acquises pour obtenir, le plus court chemin (à vol d'oiseaux) parcourant une dizaine de villes choisies au hasard sur la carte de l'Afrique (voir section 2).

1 Tour de la Tunisie

On souhaite traverser quelques villes de Tunisie le plus rapidement possible [2]. On cherchera pour cela, à implémenter quelques outils qui permettent de construire un chemin optimal.

1. La première étape consiste à représenter la liste des villes et de leurs coordonnées via des listes Python. Ces informations sont disponibles sur le net, dans le lien [1] et [3].
   Il faut ensuite créer une fonction qui récupère ces informations depuis le fichier .shx ou autre, et d'obtenir une matrice avec le nom de chaque ville en première colonne, la longitude et la latitude en seconde et troisième colonnes. On obtient une liste de listes, composées chacune d'une chaîne de caractères et de deux réels.
2. Ecrire une fonction distance qui calcule la distance euclidienne entre deux villes.
3. Ecrire une fonction longueur_tour qui retourne la longueur d'un circuit. Un circuit étant un chemin qui parcourt les villes dans l'ordre où elles apparaissent dans la liste de la question 1 et qui commence et se termine à la même ville.

1. `inestej@gmail.com, http://bit.ly/1JEWZiK`
2. Projet inspiré d'un TD proposé par Xavier Dupré à ses élèves de l'ENSAE

4. Ecrire un algorithme qui détecte si un chemin est absurde.
5. La première idée pour construire un chemin plus court est de partir du chemin initial. On échange deux villes aléatoirement puis on calcule la distance du nouveau chemin. Si elle est plus courte, on conserve la modification. Si elle est plus longue, on annule cette modification. On continue tant qu'il n'est plus possible d'améliorer la distance.
6. En exécutant les fonctions permutation et graph, vérfifiez qu'on obtient bien un chemin, même s'il n'est pas optimal.
7. Le résultat n'est pas parfait. Parfois, les chemins se croisent, pour cela, on va essayer de retourner une partie du chemin. Il s'agit de construire une fonction retourne qui retourne un chemin entre deux ville si et j. Avant l'exécution, on a

$$v_1 \to v_2 \to \ldots \to v_{i-1} \to v_i \to v_{i+1} \to \ldots \to v_{j-1} \to v_j \to \ldots$$

après exécution, le chemin devient :

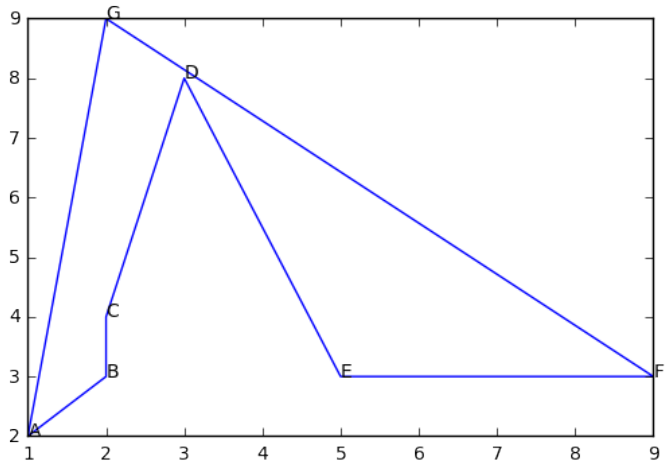
$$v_1 \to v_2 \to \ldots \to v_{i-1} \to v_j \to v_{j-1} \to \ldots \to v_i \to v_{j+1} \to \ldots$$

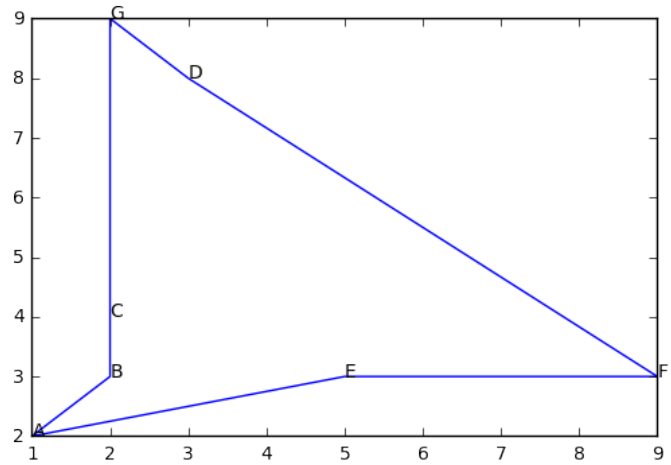
8. De la même manière qu'à la question 5, on choisit au hasard deux villes et on applique la fonction précédente. Si la distance est plus courte, on garde ce changement, sinon, on revient à la configuration précédente. On répète cette opération tant que la distance du chemin total diminue.

```
import matplotlib.pyplot as plt
import numpy
import random
import copy
def get_tour():
    tour = [["A", 1, 2], ["B", 2, 3], ["C", 2, 4], ["D", 3,8], ["E", 5,3]]
    tour += [["F", 9,3], ["G", 2,9]]
    return(tour)

def distance(tour, i, j):
    dx = tour[i][1]-tour[j][1]
    dy = tour[i][2]-tour[j][2]
    return(dx**2+dy**2)**0.5

def longueur_tour(tour):
    d = 0
    for i in xrange(0,len(tour)-1):
        d += distance(tour, i, i+1)
    d += distance(tour, 0, -1)
    return(d)

def graph(tour):
    x = [t[1] for t in tour]
    y = [t[2] for t in tour]
    x += [x[0]]
    y += [y[0]]
```

```
    plt.plot(x,y)
    for ville, x, y in tour:
        plt.text(x,y,ville)
    plt.show()

def permutation(tour):
    best = longueur_tour(tour)
    fix = 0
    while True:
        i = random.randint(0, len(tour)-1)
        j = random.randint(0, len(tour)-1)
        if i==j: continue
        e = tour[i]
        tour[i] = tour[j]
        tour[j] = e
        d = longueur_tour(tour)
        if d>= best:
            fix += 1
            e = tour[i]
            tour[i] = tour[j]
            tour[j] = e
        else:
            best = d
            fix = 0
        if fix > 10000: break

def retourne(tour, i, j):
    while i <= j:
        e = tour[i]
        tour[i] = tour[j]
        tour[j] = e
        i += 1
        j -= 1

def croisement(tour):
    best = longueur_tour(tour)
    fix = 0
    while True:
        i = random.randint(0,len(tour)-2)
        j = random.randint(i+1,len(tour)-1)
        retourne(tour, i, j)
        d = longueur_tour(tour)
        if d >= best:
            fix += 1
            retourne(tour, i,j)
        else:
            fix = 0
```

```
            best = d
        if fix > 10000 : break

def enchaine(tour):
    best = longueur_tour(tour)
    tttt = copy.deepcopy(tour)
    print "debut", best
    nom = 0
    while True:
        croisement(tour)
        d = longueur_tour(tour)
        print "croisement", d, best

        permutation(tour)
        d = longueur_tour(tour)
        print "permutation", d, best

        if d < best:
            best = d
            tttt = copy.deepcopy(tour)
            nom = 0
        elif nom >2:
            break
        else:
            nom += 1
            for k in range(0,3):
                i = random.randint(0, len(tour)-2)
                j = random.randint(i+1, len(tour)-1)
                e = tour[i]
                tour[i] = tour[j]
                tour[j] = e
        return tttt

if __name__ == "__main__":
    tour = get_tour()
    graph(tour)
    tour = enchaine(tour)
    graph(tour)
```





9. Afficher les graphes des chemin ainsi obtenus partant respectivement de Tunis, de Béjà, de Gabes et de Sfax.

```
from geopy.geocoders import Nominatim
geolocator = Nominatim()
villes = ["Tunis", "Bizerte", "Sousse", "Gabes", "Sfax", "Kairouan"]
villes += ["Tozeur", "Nabeul"]
villes += ["Ariana", "Manouba", "Le Kef"]
villes += ["Kebili", "Jendouba", "Beja"]
villes += ["Mateur", "Kasserine", "Gafsa", "Seliana"]
villes += ["Monastir", "Mahdia"]

tour = []
for i in villes:
    location = geolocator.geocode(i)
    tour += [[i, location.latitude, location.longitude]]

print(tour)

if __name__ == "__main__":
    tour = enchaine(tour)
    graph(tour)
```

10. En utilisant la netographie ci-jointe, dessinez le graphe optimal sur la carte.

2 Parcours du continent africain

En utilisant la netographie ci-dessous, et en cherchant d'autres modules de Python, générez des graphes sur des cartes de l'Afrique permettant de modéliser des chemins optimaux sur le continent africain (au moins 10 villes visibles sur la carte).

Références

[1] http ://www.statsilk.com/maps/download-free-shapefile-maps

[2] http ://www.distancesfrom.com

[3] http ://www.diva-gis.org/gdata

[4] http ://www.sigterritoires.fr/index.php/cartes-topographiques-150000-du-nord-de-la-tunisie-de-1943/

[5] https ://data.sfgov.org/Public-Safety/SFPD-Incidents-Previous-Year-2015-/ritf-b9ki

[6] http ://www.portailsig.org/content/python-leaflet-folium-ou-comment-creer-des-cartes-interactives-simplement

[7] https ://www.youtube.com/watch ?v=TTEr_nv43TE