TP 5 - Flots et taxis de la Marne

Stéphane Grandcolas, François Brucker, Valentin Emiya - Aix-Marseille Université

L2 MPCI - Décembre 2024

Objectif. Dans ce TP, vous allez

— découvrir et manipuler la bibliothèque `networkx` dédiée aux graphes en Python,

— modéliser et résoudre le problème des taxis de la Marne via les flots.

Mise en place de l'environnement de travail. Téléchargez les données et le code sur Ametice, et installez les bibliothèques suivantes via `pip` : networkx, plotly (pour des affichages alternatifs à matplotlib), pandas (pour manipuler des données). Vous travaillerez ici en complétant les fonctions du fichier `taxis.py` et en les appelant dans un fichier `main.py`.

Exercice 1 (Chargement des données brutes)**.** Étudiez la fonction `load_data` fournie et le dictionnaire renvoyé. Affichez les informations relatives à la clé `'Marseille'`. Affichez ensuite le nombre total de villes et de routes.

Un objet `g` de type `networkx.Graph` représente un graphe non-orienté et s'utilise comme un dictionnaire : si `u` est une clé désignant un sommet, `g[u]` est un dictionnaire dont les clés `v` sont les sommets voisins de `u`, et `g[u][v]` est un dictionnaire pouvant stocker des informations sur l'arête `(u,v)`.

Exercice 2 (Construction du graphe routier : sommets et arêtes)**.** Complétez la fonction `build_road_graph` qui prend en entrée les données chargées précédemment et renvoie un graphe non-orienté avec une ville par sommet et une arête entre deux villes voisines. Pour cela, construisez un graphe vide en instanciant la classe `networkx.Graph`, ajoutez les sommets en les désignant par le nom des villes (méthode `add_node`) et les arêtes (méthode `add_edge`). Vérifiez que le graphe comporte le bon nombre de sommets et d'arêtes via les attributs `nodes` et `edges`. À ce stade, il n'est pas demandé d'ajouter les informations sur les capacités, les distances, etc.

Exercice 3 (Ajout des coordonnées géographiques)**.** Complétez la fonction `build_road_graph` en ajoutant la longitude et la latitude de chaque ville. Il suffit pour cela d'ajouter des arguments supplémentaires lors de l'appel à `add_node`. Ces données sont stockées dans un dictionnaire accessible via `g.nodes[u]` où `u` est un sommet du graphe `g`. Affichez les coordonnées de Marseille à partir du graphe obtenu. Affichez ce graphe avec la fonction `draw_road_graph_plt` fournie. Pour briller en société, vous pouvez afficher votre graphe avec la fonction `draw_road_graph_plotly` qui utilise `plotly`.

Exercice 4 (Ajout des durées de parcours entre villes)**.** De la même manière que l'on peut ajouter des informations sur les sommets via `add_node`, on peut ajouter des informations sur les arêtes via `add_edge`. On souhaite ainsi compléter la fonction `build_road_graph` en ajoutant les durées de parcours entre les villes. Dans notre modélisation, les durées de parcours sont des entiers entre `1` et la variable `max_durée` et sont proportionnelles à la distance euclidienne calculée – de façon approximative – à partir des coordonnés (longitude, latitude). Dans la fonction `build_road_graph`,

— construisez dans un premier temps toutes ces distances euclidiennes,

— calculez les distances minimale et maximale,

— convertissez ces distances en durées entières de telle façon que la durée minimale soit `1` et la durée maximale `max_durée`,

— ajoutez ces distances aux arêtes du graphe.

Exercice 5 (Ajout des capacités de parking et des capacités des routes)**.** Ajoutez un entier aléatoire entre 1 et `max_parking` à chaque sommet via une clé `capacité_parking` pour modéliser la capacité de parking de chaque ville et un entier aléatoire entre 1 et `max_capacité_route` à chaque arête via une clé `capacité` pour modéliser la capacité de chaque route.

Exercice 6 (Construction du graphe de flot)**.** Complétez la fonction `build_flow_graph` qui prend en argument le graphe routier précédemment construit un nouveau graphe pour résoudre le problème de flot en utilisant la modélisation proposée [1]. Le graphe créé devant être orienté, on utilisera la classe `networkx.DiGraph`. Dans ce nouveau graphe, la clé de chaque sommet sera de la forme `(ville, temps)` [2] où `ville` est la chaîne de caractères désignant la ville et `temps` est un entier entre `0` et `temps_final` représentant un instant de passage dans cette ville. Les coordonnées géographiques seront attachées aux sommets (mais pas les capacités de parking). Une arête est créée entre un sommet `(u, t)` et un sommet `(v, t+d)` si la durée de parcours entre `u` et `v` est de `d` dans le graphe routier, et la capacité de la route est attachée à l'arête créée. Une arête est créée entre un sommet `(u, t)` et un sommet `(u, t+1)` en y associant comme capacité la capacité de parking de `u` dans le graphe routier. Les `n_villes_départ` villes de départ sont tirées au hasard. Ajoutez le sommet source `'SOURCE'` et reliez-le aux villes de départ avec une capacité `n_taxis_départ`. Ajoutez enfin un sommet fictif puits `'PUITS'` dont le seul arc est `((destination, temps_final), 'PUITS')` de capacité égale à la totalité des taxis ; cela permet de connaître facilement le nombre de taxis arrivés à destination.

Dans un premier temps, testez cette fonction sur le sous-graphe induit par le sommet `'Brest'` et ses voisins (voir méthode `subgraph` de `Graph` et `DiGraph`), avec Brest comme destination.

Exercice 7 (Résolution du problème)**.** Résolvez le problème de flot en utilisant la fonction `nx.maximum_flow`. Quel pourcentage de taxis a atteint la marne ? Faites plusieurs essais en fonction des villes de départ et du temps total alloué.

Exercice 8 (Nombre de taxis dans une ville à un instant donné)**.** À l'instant `t`, combien de taxis sont dans la ville `v` ? Complétez la fonction `nb_taxis_dans_ville_a_t` qui effectue ce calcul.

Exercice 9 (Nombre de taxis sur une route à un instant donné)**.** À l'instant `t`, combien de taxis sont sur une route `(u,v)` ? Complétez la fonction `nb_taxis_entre_villes_a_t(u, v, t, k)` qui calcule le nombre de taxis entre deux villes `u` et `v` à un instant `t` et à la distance `k` de `u`.

Exercice 10 (Représentation graphique)**.** En vous inspirant de `draw_road_graph_plotly`, complétez la fonction `(g, flow_dict, t)` qui trace le graphe routier `g` en ajoutant les informations du flow `flow_dict` à l'instant `t` (nombre de taxis dans chaque ville et sur chaque route à cet instant). Appelez ensuite cette fonction pour chaque instant possible pour générer une suite de figures illustrant le déplacement des taxis.

1. `https://francoisbrucker.github.io/cours_informatique/cours/graphes/projet-bataille-de-la-marne/`

2. Ça tombe bien, les clés des dictionnaires peuvent être des tuples !