Centralités
Projet

K. Bertet – J.L. Guillaume – A. Huchet

Travail demandé :

- Ce TEA est à faire en binôme
- TEA à rendre sur moodle pour le 7 novembre 2021 au plus tard. N'attendez pas le 6 novembre pour commencer le projet.
- Le rendu comportera un unique fichier notebook contenant tous les développements et toutes les explications nécessaires. Chaque centralité sera notée indépendamment. La note portera sur : la qualité et la justesse de l'implémentation, les explications et notamment celles sur la complexité.

Les mesures de centralités sont des fonctions associant à chaque sommet d'un graphe une valeur, censée représenter l'importance du sommet dans le graphe.

Il existe différentes mesures, qui permettent de mettre en avant différentes caractéristiques.

Les mesures de centralités peuvent être utilisées pour identifier les personnes les plus influentes dans un réseau social, les points de passages les plus importants dans un réseau routier, ou les personnes qui transmettent le plus un virus dans le cas d'une épidémie.

Échauffement

Jeu d'essai

Commencez par créer un jeu d'essai. Nous utiliserons Python, avec le module Networkx (voir TP1). Par exemple pour créer un graphe aléatoire avec 100 sommets et 500 arêtes :

```
import networkx
G = networkx.gnm_random_graph(100,500)
```

Visualisation

Créez une liste de nombres réels de même longueur que le nombre de sommets dans votre graphe. Essayez par exemple avec des réels compris dans l'intervalle [0,1] :

```
color_map = [x/len(G.nodes()) for x in range(len(G.nodes()))]
```

À l'aide de matplotlib.pyplot, affichez le graphe que vous venez de générer. Nous utiliserons la fonction draw() de Networkx, en passant la liste générée précédemment en paramètre.

```
import matplotlib.pyplot
networkx.draw(G, node_color=color_map)
matplotlib.pyplot.show()
```

Par défaut, Matplotlib utilise une palette de couleur allant du violet au jaune, nommée *viridis*.

Les sommets sont coloriés en fonction de la liste que vous avez générée. Plus la valeur color_map[x] est élevée (relativement à la valeur maximale de cette liste), plus le sommet x aura une couleur proche du jaune. À l'inverse, plus color_map[x] est faible, plus le sommet x aura une couleur proche du violet.

D'autres palettes de couleur sont disponibles, libre à vous d'en choisir une autre. Référez-vous à la documentation de la fonction draw() pour apprendre à changer de palette de couleur.

Centralité Degré

La plus ancienne et la plus simple des centralités est la centralité degré, décrite par Alexander Bavelas en 1948. Il s'agit simplement du degré du sommet concerné.

$$C_d(v) = deg(v)$$

Écrivez une fonction python pour calculer la centralité degré de tous les sommets du graphe généré plus tôt, puis affichez le graphe en coloriant les sommets en fonction de cette centralité.

Centralité Closeness

D'autres centralités sont basées sur les plus courts chemins. C'est le cas de la centralité Closeness, décrite par Alexander Bavelas en 1950, qui est l'inverse de la moyenne des longueurs de tous les plus courts chemins depuis le sommet concerné. Plus le sommet v est proche des autres sommets, plus il est central.

$$C_c(v) = \frac{n-1}{\sum_u d(v,u)}$$

Écrivez une fonction python pour calculer la centralité Closeness de tous les sommets du graphe généré plus tôt, puis affichez le graphe en coloriant les sommets en fonction de cette centralité. Attention, vous ne devez pas utiliser les présentes dans Networkx mais recoder l'algorithme intégralement, y compris les calculs de plus courts chemins.

Centralité Betweenness

La centralité Betweenness, décrite par Linton Freeman en 1977 est aussi basée sur les plus courts chemins. Il s'agit de calculer un plus court chemin entre toutes les paires de sommets du graphe. La centralité Betweenness est le nombre de plus courts chemins passant par le sommet en question.

$$C_b(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Écrivez une fonction python pour calculer la centralité Betweenness de tous les sommets du graphe généré plus tôt, puis affichez le graphe en coloriant les sommets en fonction de cette centralité. Attention, vous devez recoder l'algorithme.

Centralité Expected Force

La centralité Expected Force, décrite par Glenn Lawyer en 2015, permet de modéliser le « spreading power » d'un sommet. Elle est définie comme suit :

$$C_i = -\sum_{j \in J} \hat{d}_j \log(\hat{d}_j)$$

Où l'ensemble J est l'ensemble des listes de trois sommets, construites comme suit :

- Les trois sommets le long d'un chemin de longueur deux partant du sommet i
- Ou, le sommet i et deux de ses voisins

L'ensemble J contient des listes (ordonnées) de sommets, c'est-à-dire que deux listes de sommets appartenant à J peuvent avoir les mêmes sommets si les listes sont ordonnées différemment.

$\hat{d}_j$ est le degré normalisé du groupe de sommets , $j \in J$ c'est à dire le nombre d'arêtes ayant une seule extrémité dans j, normalisé. Pour normaliser cette valeur, il suffit de la diviser par la somme de tous les d_j :

$$\hat{d}_j = \frac{d_j}{\sum_{j \in J} d_j}$$

Écrivez une fonction python pour calculer la centralité Expected Force de tous les sommets du graphe généré plus tôt, puis affichez le graphe en coloriant les sommets en fonction de cette centralité.

Calcul de complexité de ces quatre centralités

Pour chacune de ces centralités, évaluez la complexité de vos algorithmes et détaillez le calcul. Mesurez également le temps d'exécution comme vu en TP.

Pour aller plus loin

Modifiez votre implémentation des centralités Closeness et Betweenness pour utiliser les fonctions de plus courts chemins présentes dans Networkx. Comparez les temps de calcul de votre implémentation et de l'implémentation de Networkx.