M2 – SDR (TP1-2-3)

Calcul des propriétés structurelles des réseaux

Clémence Magnien, Lionel Tabourier, Fabien Tarissan

Cette première séance de TD/TP est destinée à vous familiariser avec les notions statistiques de bases concernant les grands réseaux d'interaction et à vous forger une intuition concernant l'analyse et le calcul de ces propriétés à travers deux cas concrets.
Dans la suite de la feuille (et les TPs suivants), nous représenterons naturellement les réseaux par des graphes, que nous noterons $G = (V, E)$, où V est l'ensemble des nœuds (aussi appelés *sommets*) du graphe numérotés de 0 à $|V| - 1$ et E l'ensemble des liens (aussi appelés *arêtes*).

1 Pour commencer

Exercice 1 — *Préparation* Créer quelques exemples de graphes *à la main* qui vous permettront de vérifier les résultats de vos programmes. Vous les stockerez dans des fichiers pour lesquels chaque ligne est de la forme
x y
indiquant l'existance d'un lien entre le nœud x et le nœud y.

2 Manipulations et propriétés de base

2.1 Charger un graphe en mémoire

Dans la suite (et les TP suivants), *programme* désignera un programme ou une combinaison de programmes dans le ou les langage de votre choix.

Exercice 2 — *Taille d'un graphe* Réaliser un programme calculant le nombre de nœuds d'un graphe (sans le stocker en mémoire) et l'écrivant sur la sortie standard, qu'on redirigera vers un fichier `graphe.n`. Vous pourrez vous aider pour cela de l'aide mémoire (cf. section 5).

Exercice 3 — *Degrés des nœuds* Réaliser un programme calculant le degré (*i.e.* son nombre de liens) de chaque nœud d'un graphe (sans le stocker en mémoire), et qui l'écrit sur la sortie standard, qu'on redirigera vers un fichier `graphe.deg`.

Exercice 4 — *Chargement d'un graphe* À l'aide des résultats des deux exercices précédents, réaliser un programme lisant un graphe et le stockant en mémoire sous la forme d'un tableau de tableaux, de façon optimale en espace. Si possible, les tableaux seront contigus en mémoire.

2.2 Quelques propriétés statistiques de base

Exercice 5 — *Nombre de nœuds isolés* Réaliser un programme calculant le nombre de sommets de degré 0 d'un graphe, ainsi que la densité, le degré moyen, le degré minimum et le degré maximum du graphe. Y a-t-il besoin de stocker le graphe en mémoire pour cela ?

On rappelle que la *distribution des degrés* d'un graphe G est la fonction qui à un entier k associe le nombre de nœuds du graphes ayant un degré égal à k.

Exercice 6 — *Distribution des degrés* Écrire un programme qui calcule la distribution des degrés d'un graphe et l'écrit dans un fichier dans lequel chaque ligne est au format
d n
où n le nombre de nœuds du graphe ayant un degré d (pous pourrez utiliser les outils proposés en section 5.1)

2.3 Application à des cas réels

Récupérer le réseau d'interaction protéine-protéine de la drosophile [1] (que nous nommerons par la suite PPI-DROSO) ainsi qu'un réseaux de connexions entre routeurs de l'Internet [2] (que nous nommerons INET) tous deux disponibles dans le répertoire `/Enseignants-Public/tarissan/sdr/2014/`. Attention, ces graphes peuvent contenir des boucles.

Exercice 7 — *Suppression des boucles* Écrire un programme qui supprime les boucles présentes dans un graphe.

Exercice 8 — *Application* On applique maintenant tout ce qu'on a implémenté jusqu'à présent sur le cas des réseaux PPI-DROSO et INET.

1. À partir des programmes précédents, calculer les caractéristiques des réseaux PPI-DROSO et INET ainsi que leur distribution des degrés.
2. Afficher ces distributions à l'aide de GNUPLOT (cf. section 5.2)
3. Commenter les résultats.

Exercice 9 — *Distribution cumulative* Écrire un programme qui calcule la distribution cumulative des degrés pour un graphe donné. Utiliser ce programme sur les réseaux précédents et afficher le résultat à l'aide de GNUPLOT. Est-ce cohérent avec l'affichage précédent ?

3 Densité locale *v.s.* densité globale

La densité locale est une notion qui tente de capturer le rapport entre le nombre de triangles (trois sommets liés entre eux) et le nombre de triplets connexes (trois sommets et au moins deux arêtes). Il existe en particulier deux définitions proches liées à cette notion :

Définition 1 (Clustering Coefficient) *Soit $G = (V, E)$ un graphe et $v \in V$ un sommet. Soit $N(v)$ l'ensemble des voisins de v et $E(N(v)) = \{(u_1, u_2) \in E \mid u_1, u_2 \in N(v)\}$ l'ensemble des liens entre ces voisins. Le* coefficient de clustering $\mathrm{cc}(v)$ de v *est défini par*

$$\mathrm{cc}(v) = \frac{2|E(N(v))|}{d^\circ(v)(d^\circ(v) - 1)}.$$

Le coefficient de clustering du graphe G *est alors la moyenne des coefficients de clustering pour chacun des sommets de G de degré supérieur ou égal ou à 2.*

Définition 2 (Transitive Ratio) *Soit G un graphe. Le* transitive ratio $\mathrm{tr}(G)$ de G *est défini par*

$$\mathrm{tr}(G) = \frac{3N_\Delta}{N_\vee},$$

où N_Δ est le nombre de triangles dans le graphe, et $N_\vee$ le nombre de triplets connexes.

Exercice 10 — *Coefficients de clustering et transitive ratio*

1. Écrire un programme permettant de calculer ces coefficients pour un réseau donné.
2. Appliquer ce programme sur le fichier PPI-DROSO et INET.
3. Comment analyser les valeurs ?
4. Tester l'efficacité de votre programme (le temps mis pour effectuer le calcul).

1. récupéré sur la base de donnée en ligne `http://160.80.34.4/mint/Welcome.do`
2. récupéré à partir du site `http://svnet.u-strasbg.fr/merlin`

4 Parcours en largeur et applications

On rappelle ci-dessous l'algorithme de parcours en largeur pour un graphe donné et un sommet d'origine.

Algorithm 1: Parcours en largeur d'un graphe G à partir d'un sommet s fixé.

```
F ← CréerFileVide()
MettreDansFile(F,s)
Marquer(s)
while F est non vide do
    u ← RetirerPremier(F)
    Afficher u
    for v voisin de u dans G do
        if NonMarqué(v) then
            MettreDansFile(F,v)
            Marquer(v)
        end
    end
end
```

Exercice 11 — *Parcours en largeur* Écrire un programme qui, étant donné un graphe G et l'un de ses sommets s, affiche le parcours en largeur de G à partir du sommet s.

Les exercices suivants exploitent cet algorithme pour calculer un certain nombre de propriétés d'un graphe donné.

4.1 Composantes connexes

Exercice 12 — *Taille des composantes connexes* En modifiant le programme précédent, écrire un programme qui calcule la taille de chaque composante connexe d'un graphe donné. Utiliser ce programme pour afficher la distribution de la taille des composantes connexes sur les réseaux PPI-DROSO et INET. Commenter.

Exercice 13 — *Composante connexe principale* Modifier le programme précédent pour qu'il extraie la composante principale d'un graphe donné. Utiliser ce programme sur les deux réseaux étudiés.

4.2 Centralité

La *betweenness centrality* (ou centralité d'intermédiarité) est une propriété qui tente de capturer l'importance d'un nœud (ou d'un lien) pour la propagation d'un signal/message, autrement dit sa importance en tant qu'intermédiaire dans la structure du réseau. Elle repose essentiellement sur la notion de plus court chemin.
Par convention, nous noterons σ_{st} le nombre de plus courts chemins entre les nœuds s et t d'un graphe G et $\sigma_{st}(v)$ le nombre de plus courts chemins entre les nœuds s et t passant par v (on suppose ici $s \neq t \neq v$). La centralité d'intermédiarité d'un nœud v est alors définie par la formule suivante :

$$BC(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

On pourra remarquer qu'un cas particulier et simple pour le calcul de $BC(v)$ a lieu lorsqu'il existe exactement 1 plus court chemin pour toutes les paires de sommets s, t de G. Dans ce cas $BC(v)$ vaut exactement $\sum_{s \neq t \neq v} \sigma_{st}(v)$, c'est à dire le nombre de plus courts chemins dans G passant par v.

Exercice 14 — *Ensemble des plus courts chemin* Adapter l'algorithme de parcours en largeur pour qu'il donne **tous** les plus courts chemins. Sa complexité est-elle modifiée ?

Exercice 15 — *Nombre de plus courts chemins* Étant donné un ensemble des plus courts chemins partant de s, comment calculer le nombre de plus courts chemins passant par $v \neq s$ dans cet ensemble ? Implémenter une fonction calculant ce nombre pour chaque sommet.

Exercice 16 — *Centralité d'intermédiarité*

1. À partir de l'exercice précédent, donner un algorithme permettant de calculer la centralité de chaque sommet d'un graphe.
2. Quelle est sa complexité ?
3. Implémenter cet algorithme et l'appliquer aux réseaux PPI-DROSO et INET.
4. Afficher la distribution des valeurs obtenues.
5. Afficher la distribution cumulative de ces valeurs.

5 Aide mémoire

5.1 Commande Unix

- `cut -f` i, j fic : extraction des champs i et j à partir d'un fichier fic
- `sort` fic : ordonne le fichier fic
- `uniq -c` fic : compte le nombre d'occurrence de lignes identiques consécutives (à combiner avec `sort` pour obtenir une distribution)
- *cmd* `>` fic : redirection de la sortie de la commande *cmd* dans le fichier fic.
- `time` *cmd* : donne le temps mis par la machine pour exécuter une commande *cmd*.

5.2 Gnuplot

GNUPLOT est un programme permettant d'obtenir une représentation graphique à partir d'un jeu de données structuré. Beaucoup de documentations et tutoriels existent [3]. Je ne donne ici que la commande principale qui va vous être utile pour afficher les résultats. Il suffit d'exécuter la commande `gnuplot` dans un terminal, puis `plot "fic" using i:j`, ce qui conduira à afficher les données contenues dans le fichier fic en utilisant les valeurs de la colonne i pour abscisse et celles de la colonne j pour ordonnée.
Quelques options utiles :

- `set logscale x` : utilise une échelle logarithmique (et non linéaire) pour l'axe des x (*idem* avec `y`).
- `set xrange` $[v_1 : v_2]$: n'affiche les données que pour les valeurs de x comprises entre v_1 et v_2 (*idem* avec `yrange`).
- `set xlabel` *nom* : Donne le nom *nom* à l'axe des x (*idem* avec `ylabel`).

3. `http://www.gnuplot.info/`
`http://www.info.univ-angers.fr/~gh/tuteurs/tutgnuplot.htm`