TP 1 : Igraph/R - Introduction

Rushed Kanawati

24 octobre 2016

Ce TP est à réaliser en binôme. Un seul rapport est à rendre à la fin de la séance

Résumé

L'objectif de ce TP est de se familiariser avec l'outil `igraph` : un outil d'analyse et de visualisation de graphes.

igraph : installation

`igraph` est une bibliothèque de manipulation, d'analyse et de visualisation de graphes, disponible sous forme de logiciel libre. Il est disponible au téléchargement sur le site `http://igraph.sourceforge.net/`. La bibliothèque est disponible pour utilisation avec C, python ou R. Nous l'utilisons ici avec le langage `R`. Exécuter un shell `R` (ex. `Rstudio`) et dans ce shell exécuter les commandes suivantes :

1. `install.packages("igraph")` # pour installer le package igraph
2. `library(igraph)` # pour charger la bibliothèque dans le shell R
3. `help(igraph)` # affiche les pages d'aide concernant ce package

Génération et chargement de graphs

Création de graphes

La fonction `graph.empty()` permet de générer un graphe vide :

```
g <- graph.empty(directed=FALSE)
```

Il est possible d'ajouter des nœuds et des liens à un graph en utilisant les fonctions `add.vertices` et `add.edges`.

```
g <- add.vertices(g,4)
```

```
g <- add.edges(g,c(1,2))
```

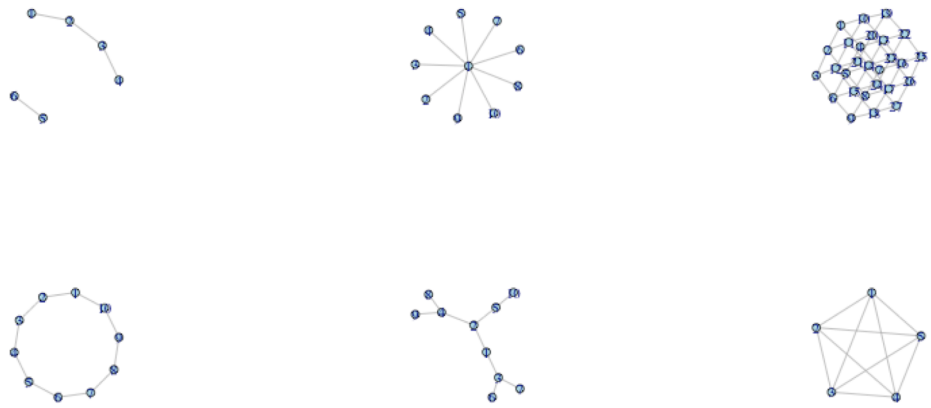

FIGURE 1 – Exemple de génération de graphes

On peut aussi supprimer des nœuds et des liens en utilisant les fonctions `delete.vertices` et `delete.edges`. La fonction `summary(g)` retourne une description texte du graphe. La fonction `plot(g)` permet de visualiser le graphe.

Génération de graphes simples

`igraph` offre un ensemble de fonctions de génération de graphes simples. Des exemples sont les suivants :

```
graph( c(1,2,2,3,3,4,5,6), directed=FALSE )
graph.star(10, mode="undirected")
graph.lattice(c(3,3,3), directed=FALSE)
graph.ring(10, directed=FALSE)
graph.tree(10, 2, mode="undirected")
graph.full(5, loops=FALSE, directed=FALSE)
```

La figure 1 illustre les graphes générés par les commandes précédentes.

Graphes aléatoires

`igraph` offre aussi un ensemble de fonction de génération de graphes aléatoires, notamment le modèle *Erdös-Renyi*, le modèle de graphes petits monde (Watts-Strogratz) et le modèle de l'attachement préférentiel (Barabasi-Albert). Des exemple d'utilisation sont donnés ci-après :

```
er_graph <- erdos.renyi.game(100, 2/100)
ws_graph <- watts.strogatz.game(1, 100, 4, 0.05)
ba_graph <- barabasi.game(100)
```

Lecture de graphs

La fonction `read.graph` permet de charger un graphe sauvegardé dans un fichier. Le format est à spécifier avec l'attribute format. Des exemples sont :

```
karate <- read.graph("http://cneurocvs.rmki.kfki.hu/igraph/karate.net", format="pajek")
setwd( votre répertoire de travail)
football <- read.graph("football.gml", format="gml")
```

La fonction `write.graph` permet de sauvegarder un graphe dans le format désiré.

Mesures dans les graphes

`igraph` offre un ensemble riche de fonctions de mesures dans les graphes. Voici les principales fonctions :

- `vcount(g)` : retourne le nombre de nœuds dans g
- `ecount(g)` : retourne le nombre de liens dans g
- `graph.density(g)` : donne la densité du graphe g
- `diameter(g)` : retourne le diamètre du graphe g
- `degree(g)` : retourne le degrés de chaque nœud dans g
- `degree.distribution(g)` : calcule la distribution de degrés de g
- `transitivity(g)` : calcule le coefficient de clustering du graphe g
- `shortest.paths(g)` : retourne un matrice qui donne les longueurs de plus courts chemins entre chaque couple de nœuds.
- `betweenness(g)` : calcule la centralité d'intermédiarité
- `closeness(g)` : calcule la centralité de proximité.
- `is.connected(g)` : retourne `TRUE` si le graphe est connexe.
- `clusters(g)` : retourne une liste des composantes connexes dans le graphe.
- `neighbors(g,x)` retourne la liste des voisins du nœud x dans g

Visualisation de graphes

L'objet `V(g)` (resp. `E(g)`) représente la liste de nœuds (resp. liens) du graphe `g`. Il est possible d'ajouter des attributs aux nœuds et aux liens d'un graphe. Par exemple on peut ajouter un attribut `color` afin de personnaliser la visualisation d'un graphe.

```
V(g)$color <- sample( c("red", "black"), vcount(g), rep=TRUE)
E(g)$color <- "grey"
plot(g)
```

L'attribut `shape` peut être utilisé pour contrôle la forme d'un nœud (ex. circle, square). L'attribut `layout` peut être ajouté à un objet graphe pour déterminer l'algorithme de visualisation à appliquer. Exemple :

```
g <- graph.ring(10)
g$layout <- layout.circle
plot(g)
```

Exercices

1. Développer un script *R* qui permet de retourner un résumé des principales caractéristiques topologiques d'un graphe.
2. Développer un script qui permet d'afficher la distribution de degrés d'un graphe.
3. Etudier la variation des diamètres de graphes et de la transitivité dans les trois modèles de base de graphes aléatoires (Erdös-Renyi, Watts, Barbasi) en fonction de nombre de nœuds.
4. Télécharger les graphes suivants `dolphins.gml`, `polblogs.gml`, `football.gml`, `karate.gml`, `polbooks.gml` sur le site `http://lipn.fr/~kanawati/ars`. Dans ces graphes les communautés sont désignées par l'attribut `value`. Afficher ces graphes en colorant les communautés par des couleurs différentes.
5. Pour chacun des graphes, extraire le sous-graphe égo-centré sur le nœud le plus central selon les différentes centralités.