L'analyse de graphes avec R : un aperçu avec igraph

Sébastien Plutniak

L'analyse de graphes avec R : un aperçu avec *igraph*

Sébastien Plutniak — École française de Rome
sebastien.plutniak@ehess.fr

Atelier Cytoscape–R
G. Brysbaert & S. Plutniak
École thématique *Analyse de réseaux et complexité*

25 Septembre 2018, Cargèse

Contenu du tutoriel

1 R, au secours ! Avant toute chose…

Des précisions concernant le langage R en général ont été introduites ponctuellement dans ce document. La documentation relative à une fonction est consultable grace à la fonction `?()`. Elle doit devenir un réflexe lors de la découverte de toute nouvelle fonction.

```
?typeof
```

Une autre fonction très utile est `example()`, qui exécute les exemples inclus dans la documentation d'une fonction.

```
example(typeof)
```

2 Chargements et construction du graphe

Commençons par charger le package *igraph* et les données.

```
library(igraph)
```

Pour cet exercice nous reprendrons le cas, classique, du réseau publié dans l'article de W. Zachazy en 1977 [1]. Il représente les relations d'amitié entre les 34 membres d'un club universitaire de karaté, répartis en deux groupes (« factions »).

Les informations sont représentées dans deux tableaux : l'un contenant des informations sur les acteurs, l'autre contenant la description des liens. Chargeons ces deux tableaux.

```
# importer les nœuds
karate_vertices <- read.csv2("karate_vertices.csv")
karate_vertices           # inspecter le tableau des sommets
# importer les arêtes
karate_edges <- read.csv2("karate_edges.csv")
karate_edges              # inspecter le tableau des arêtes
```

Nous allons maintenant construire une représentation ce réseau de relations sous la forme d'un graphe : les acteurs seront les sommets et les relations d'amitié seront les arêtes du graphe. La fonction `graph_from_data_frame()` est une des (nombreuses) fonctions qui permettent de créer un graphe dans *igraph* [2].

```
?graph_from_data_frame
```

Elle prend ici deux arguments, le data frame des arêtes et celui des sommets.

Concernant les arêtes, si le *data frame* contient plus que deux colonnes, alors ces dernières sont interprétées comme les attributs des arêtes. Il en va de même à partir de la deuxième colonne dans le data frame des sommets. On précise enfin que le graphe que l'on souhaite obtenir est non dirigé.

```
karate.net <- graph_from_data_frame(karate_edges,
                                    vertices = karate_vertices,
                                    directed = FALSE)
# on supprime de la mémoire vive les tableaux des liens et des sommets
rm(karate_edges, karate_vertices)
```

3 Inspection : obtenir des informations générales

Question : quel est le type d'objet d'un graphe dans le contexte de l'utilisation d'*igraph* ?

```
mode(karate.net)   # son type
class(karate.net)  # sa classe
```

1. Wayne W. Zachary [1977], « An Information Flow Model for Conflict and Fission in Small Groups », *Journal of Anthropological Research*, 33, 4, p. 452-473, DOI : 10.1086/jar.33.4.3629752.
2. Gábor Csárdi et Tamás Nepusz [2006], « The igraph Software Package for Complex Network Research », *InterJournal*, 1695, 5, p. 1-9, http://igraph.org.

Pour obtenir des informations générales relatives à un objet de type `graph` :

```
karate.net
# informations relatives aux sommets
V(karate.net)
# informations relatives aux arêtes
E(karate.net)
```

Et celles relatives aux attributs :

```
# Extraire les valeurs d'un attribut des arêtes
edge_attr(karate.net, "weight")
# une autre manière de le faire:
E(karate.net)$weight
```

Ici, comme ailleurs en R, il est souvent possible d'employer plusieurs syntaxes pour conduire à un résultat identique.

Questions : que retourne la fonction `V()` par défaut ? Comment obtenir un vecteur avec le numéro de faction de chaque karateka ?

4 Visualisation : obtenir une représentation graphique

4.1 Impression écran

Pour pouvoir décliner plus facilement les visualisations, on commence par copier notre graphe dans un nouvel objet.

```
x <- karate.net
plot(x)
plot.igraph(x)
```

Rappel : `plot()` est la fonction générique en R pour commander une sortie graphique. Elle s'adapte à la classe de l'objet auquel elle est appliquée : pour les objets de classe « `igraph` » elle exécute la méthode `plot.igraph()`.

Lorsque l'on souhaite visualiser un graphe sous une forme réticulaire, un choix doit être fait entre les différentes méthodes existantes pour optimiser le placement des sommets les uns par rapport aux autres et limiter le croisement des arêtes. Sans paramétrage explicite, la fonction `plot.igraph()` sélectionne automatiquement une méthode en fonction des propriétés du graphe.

```
# détails sur le choix automatique de l'algorithme de placement
? layout_nicely
```

Pour prendre connaissance des algorithmes implémentés tapez `layout_` puis la touche d'autocomplétion <TAB>.

```
# détails sur le choix automatique de l'algorithme de placement
plot(x, layout = layout_in_circle(x))

x$layout <- layout_with_fr(x) #  Fruchterman-Reingold layout algorithm
plot(x)
```

```
x$layout <- layout_with_dh(x) #  Davidson-Harel layout algorithm
plot(x)
```

Il est possible de contraindre la position initiale des sommets. Si l'on souhaite par exemple grouper visuellement les sommets en fonction de leurs factions, on définira des positions initiales différentes en fonction de la faction.

```
# data.frame avec des coordonnées arbitraires :
initial.coords <- cbind(
      as.numeric(as.character(factor(V(x)$faction,
                                     levels=c(1,2), labels=c(-1, 1)))),
      as.numeric(as.character(factor(V(x)$faction,
                                     levels=c(1,2), labels=c(-1, 1))))
      )

# on transmet les coordonnées initiales à l'algorithme de placement:
plot(x,
     coords = initial.coords,
     vertex.color = V(x)$faction)

# idem avec choix de l'algorithme de placement (Fruchterman-Reingold):
plot(x,
     vertex.color = V(x)$faction,
     layout = layout_with_fr(x, coords = initial.coords) )
```

La fonction `plot.igraph()` accepte de nombreux paramètres, dont nous verrons ci-après les plus utiles pour l'impression de graphes.

Rappel : en R, les couleurs acceptent différentes définitions :

— une chaîne de caractères explicite, ex. : `darkslategray4` ou textttbrown

— une chaîne de caractère produite par la fonction `rgb()` dont la syntaxe est la suivante :
`rgb(`<valeur de rouge>, <valeur de vert>, <valeur de bleu>, <transparence>`)`.
par exemple :

```
rgb(.1, .8, .0, .95 )
```

Lorsque l'on souhaite affecter des couleurs différentes aux sommets d'un graphe il est alors utile de stocker les différentes valeurs dans un vecteur que l'on entre dans la fonction `plot()` ou dans un vecteur que l'on introduit dans le graphe en tant qu'attribut des sommets.

Voici un exemple (non exhaustif) des paramètres graphiques associés à la visualisation de graphes :

```
plot(x,
  # Paramètres des sommets
     vertex.size = 10 * degree(x) / max(degree(x)) + 3,
     vertex.color = as.character(
                         factor(betweenness(x),
                                labels = rev(heat.colors(
                                  length(unique(betweenness(x)))
                                  )))
```

```
                        ),
     vertex.label = V(x)$name,
     vertex.label.cex = .6,
     vertex.label.color = "black",
     vertex.shape = "square", # "circle"
     vertex.label.color="black",
  # Paramètres des arêtes
     edge.arrow.size = 0.5,
     edge.width = E(x)$weight,
     edge.color = rgb(.1, .1, .1, .8),
  # Paramètres généraux
     layout = x$layout,
     main = "Titre du graphe",
     sub = "sous titre")
```

La taille des sommets est proportionelle à leur degré ; la couleur des sommets suit un gradient coloré en fonction de la valeur de centralité d'intermédiarité ;

4.2 Impression dans un fichier

On souhaite rediriger la sortie de la fonction plot() ailleurs que vers l'écran. Le principe général est le suivant :

1. on ouvre une sortie avec une des fonctions suivantes : `png()`, `jpeg()`, `pdf()`, `svg()`...
2. on exécute la fonction `plot()`
3. on ferme la sortie avec la fonction `dev.off()`

```
# exécutez soit
png("karate.png", width=2000, height=2000, units="px", pointsize=40)
# ou
svg("karate.svg", pointsize=20, width=12, height=12)
# ou
pdf("karate.pdf", pointsize=11)

# puis
   plot(karate.net, main= "Karate Club")

# et enfin :
dev.off()
```

5 Édition : modifier un graphe

5.1 Éditer les sommets et arêtes

```
delete_edges()
delete_vertices()

# supprimer les 5 premiers sommets :
delete_vertices(karate.net, 1:5)
```

5.2 Éditer les attributs

On souhaite ajouter un « age » aux 34 karatekas (donc, formellement, un attribut aux sommets du graphe) :

```
V(karate.net)$age <- sample(15:30, 34, replace = T)
```

La fonction `sample()` échantillonne aléatoirement 34 valeurs entre 15 et 30 compris (avec remise). NB : il n'est pas nécessaire de déclarer la création de l'attribut par une commande préalable.

Inspectez l'objet `karate.net`. On souhaite maintenant supprimer l'attribut âge.

```
delete_vertex_attr(karate.net, "age")
```

Inspectez à nouveau l'objet `karate.net`. (Pourquoi l'attribut `age` est-il toujours présent ?)

6 Indiçage et extraction : manipuler les parties d'un graphe

6.1 Indiçage

L'indiçage des sommets se réalise en respectant la syntaxe suivante :
V(<objet graphe>)[<indice numérique ou chaîne de caractères ou booléen>]

```
V(karate.net)[1]        # le premier sommet
V(karate.net)["Mr Hi"] # idem avec le nom de sommet
V(karate.net)[1:10]     # les 10 premiers sommets
V(karate.net)["John A"] # un autre avec le nom de sommet
```

L'indiçage des arêtes s'effectue de la même manière, en donnant des arguments à la fonction `E()`. Cherchons les liens d'amitié ayant un poids égal à 5.

```
E(karate.net)[ E(karate.net)$weight == 5 ]
```

Il est possible de construire des expressions plus complexes pour sélectionner certaines arêtes. La commande suivante sélectionne toutes les arêtes existant entre John A et les autres sommets :

```
E(karate.net)[ V(karate.net)["John A"]  %--%  V(karate.net) ]
```

La syntaxe est :
`A %--% B` les arêtes entre A et B (voir exemple précédent)

Pour les graphes dirigés, la syntaxe d'*igraph* admet en outre les opérateurs :
`A %->% B` les arêtes de A vers B
`A %<-% B` les arêtes de B vers A

On peut donc évaluer une expression sur un vecteur qui renverra les indices numériques pertinents, puis donner ce résultat aux fonctions `E()` ou `V()`. Voici un exemple.

```
degree(karate.net) > 4
```

La ligne précédente évalue l'expression > 4 et renvoie un vecteur de booléens (vrai/faux). La ligne suivante extrait les sommets ayant un degré total > 4 en utilisant l'expression précédente :

```
V(karate.net)[ degree(karate.net) > 4 ]
```

La fonction `which()` renvoie les indices des éléments d'un vecteur booléen correspondant aux valeur `TRUE` :

```
which( degree(karate.net) > 4  )
```

Il est parfois utile de combiner ces fonctions :

```
V(karate.net)[ which(degree(karate.net) > 4  ) ]$faction
```

L'expression précédente renvoie les numéros de faction des karatékas qui ont plus que 4 relations.

6.2 Extraction de sous-graphes

Plusieurs fonctions sont disponibles pour extraire des parties d'un graphe :

```
induced_subgraph()     # extraire à partir de sommets.
# arguments :  un graphe, un vecteur indiçant les sommets,
# renvoie : un sous-graphe.

make_ego_graph()  # voisinage d'un sommet
# extrait le sous-graphe du voisinage d'ordre n d'un sommet donné.

subgraph.edges()       # extraire à partir d'arêtes
# arguments :  un graphe, un vecteur indiçant les arêtes,
# renvoie : un sous-graphe.
```

Quelques exemples :

```
# sous-graphes des sommets de degrés > 4 :
karate.sub.net <- induced_subgraph(karate.net,
                      V(karate.net)[ degree(karate.net) > 4 ]
                      )
plot(karate.sub.net,
     main="Karate club : sous-graphes des sommets de degré > 4"
     )

# sous-graphes des sommets liés par des arêtes de poids >= 3 et =< 5 :
karate.sub2.net <- subgraph.edges(karate.net,
                                  which(E(karate.net)$weight %in% 3:5 )
                                    )
plot(karate.sub.net, main=" Karate club : sous-graphe des liens > 2")

# voisinage de degré 1 du sommet "John A" :
mrhi.net <- make_ego_graph(karate.sub.net, 1,
```

```
                                V(karate.sub.net)["John A"]
                                )
plot( mrhi.net[[1]] )
```

NB : cet indiçage est nécessaire car la fonction `make_ego_graph()` renvoie une liste de graphes.

7 Métriques : mesurer les propriétés d'un graphe

7.1 Mesures globales

```
gorder()       # nombre de sommets
gsize()       # nombre d'arêtes
length(E())    # une autre manière d'obtenir le nombre d'arêtes
               # (car E() renvoie un vecteur de nombres entiers)

diameter(karate.net, directed = F) # diamètre du graphe
diameter(karate.net, directed = T, unconnected = TRUE, weights = NULL)
```

Comparez les résultats précédents : pourquoi sont-ils égaux ?

```
# moyenne des longueurs des chemins
mean_distance(karate.net)
edge_density(karate.net)                    # densité du graphe
reciprocity()                    # réciprocité (pour graphes dirigés)
transitivity(karate.net, type="global") # transitivité
```

Consultez l'aide de la fonction pour identifier les différents modes de calcul de la transitivité.

7.2 Mesures locales

On souhaite maintenant calculer les degrés totaux des sommets et ajouter un attribut nommé « degree » aux sommets.

```
degree(karate.net)
```

La fonction `degree()` renvoie un vecteur contenant les degrés des sommets. On ajoute ce résultat comme attribut des sommets.

```
V(karate.net)$deg <- degree(karate.net)
```

Consultez la documentation pour les modes de calcul (degrés entrant, sortant, totaux). Pour obtenir le graphe de la distribution cumulées des degrés (utile pour identifier des propriétés générales d'un graphe comme le fait d'être sans échelle) on combine les fonctions `plot()` et `degree_distribution()`.

```r
plot(degree_distribution (karate.net, cumulative = T))
# centralité d'intermédiarité des sommets
betweenness(karate.net)
# centralité d'intermédiarité des arêtes
edge_betweenness(karate.net)
# centralité de proximité
closeness(karate.net)
# à nouveau la transitivité mais ici calculée pour  chaque sommet
transitivity(karate.net, type="localundirected")
# plus courts chemins
distances(karate.net)
```

La fonction `distances()` renvoie une matrice contenant la longueur du plus court chemin entre chaque paire de sommets du graphe.

Si l'on souhaite exporter les résultats des calculs sous forme tabulaire, par exemple les noms des sommets et leur valeur de degré :

```r
# création d'un tableau (data frame) :
degree.tab <- data.frame( name = V(karate.net)$name,
                          degree = degree(karate.net))
# écriture dans un fichier :
write.table(degree.tab, "name-degree.csv", row.names = F)
```

8 Classification des sommets d'un graphe

8.1 Algorithmes de classification

Huit algorithmes sont implémentés dans *igraph*. Ils prennent pour argument principal un graphe et renvoient un objet de type `communities`. Commençons avec l'algorithme `edge betweeness` qui procède par déconstruction progressive du graphe en supprimant les arêtes dans l'ordre décroissant de leur score de centralité (l'algorithme n'admet pas les arêtes valuées).

```r
karate.comm <- cluster_edge_betweenness(karate.net, weights = NULL)
```

Un objet `communities` est une liste qui contient les éléments suivants :

```r
names(karate.comm)
```

L'élément « membership » contient un vecteur donnant la classe affectée à chaque sommet on utilise cette information pour retravailler le graphe comtpe tenu de la classification.

On souhaite maintenant imprimer le graphe avec une couleur par classe.

```r
V(karate.net)$color <- as.character(
                         factor(karate.comm$membership,
                           labels = rainbow(length(karate.comm)
                                            )))
```

Détail de la ligne précédente (non spécifique à *igraph*) :

— la fonction `factor()` permet de recoder un vecteur
— la fonction `length()` appliquée à un objet `communities` donne le nombre de communautés ;
— la fonction `rainbow()` renvoie un vecteur contenant un nombre de couleurs tirés d'une palette « arc-en-ciel » égal au nombre donné en argument.
— la fonction `as.character()` convertit la sortie de la fonction `factor()` en chaîne de caractères.

On imprime le graphe sur la sortie standard :

```
plot(karate.net, main="Karate club",
     sub="classification :  edge betweeness")

plot(karate.net,
  main = "Karate Club : partitionnement",
  sub = paste("algorithm:",  karate.comm$algorithm,
            " |  modularity: ", as.character(
                                round( modularity(karate.comm), 2)),
            " | n community: ", as.character(
                                 length(karate.comm) )
             )
)
```

Comme illustré ci-dessus, il est possible d'ajouter de multiples informations à la sortie graphique. La fonction `paste()` est très utile pour convertir des objets en chaînes de caractères et les assembler.

Voici les autres algorithmes implémentés :

```
cluster_fast_greedy() # greedy optimization of modularity
cluster_walktrap()    # short random walks
cluster_spinglass()   # based on statistical mechanics
cluster_leading_eigen() # based on the leading eigenvector of
                        # the community matrix
cluster_label_prop() # based on propagating labels
cluster_infomap()    # minimizes the length of a random walker
cluster_optimal()    # maximizing the modularity measure

# avec "spinglass" on peut limiter le nombre de classes :
V(karate.net)$color <- membership(cluster_spinglass(karate.net,
                                                    spins = 2))
plot(karate.net)
```

Exercice : comparez visuellement les résultats de deux ou trois algorithmes de classification. Conseil : définissez au préalable la position des sommets pour pouvoir comparer plus aisément les résultats.

NB : il est possible de préciser sa propre palette de couleur en donnant à l'argument « labels » de la fonction `factor()` un vecteur contenant les valeurs de ces couleurs. Il est également possible d'utiliser une autre palette standard, comme `heat.colors()`

8.2 Discriminations simples des sommets

```r
# décomposition des k-core
# (sous graphes maximaux dans lesquels chaque sommet est de degré k)
coreness(karate.net)

# composantes connexes d'un graphe
components(karate.net, mode = c("strong"))

ego_size(karate.net, 1,
         nodes = V(karate.net)$name == "Mr Hi", mode = c("all"))
```

```r
articulation_points()
```

Cette fonction renvoie les indices des points d'articulation du graphe, c.-à-d. des sommets dont la suppression fait augmenter le nombre de composantes connexes du graphe.

Voici un exemple :

```r
plot(karate.net)
# on identifie le(s) points d'articulation
articulation.points(karate.net)
```

La série de commandes suivante supprime le(s) point(s) d'articulation puis visualise le graphe.

```r
karate.no.art.net <- delete_vertices(karate.net,
                  V(karate.net)[ articulation.points(karate.net)  ]
                  )
plot(karate.no.art.net)

# examinons à nouveau les composantes connexes du graphe :
components(karate.no.art.net)

# nombre de composantes connexes :
components(karate.no.art.net)$csize
```