Visualisation de réseaux avec R

Katherine Ognyanova

2016 - traduit en février 2017

POLNET 2016 Workshop, St. Louis, MO

Katherine Ognyanova, Rutgers University Web: www.kateto.net, Twitter: ognyanova

Traduit par Laurent Beauguitte arshs.hypotheses.org

Contents

Jeux de données et script R sont téléchargeables ici.

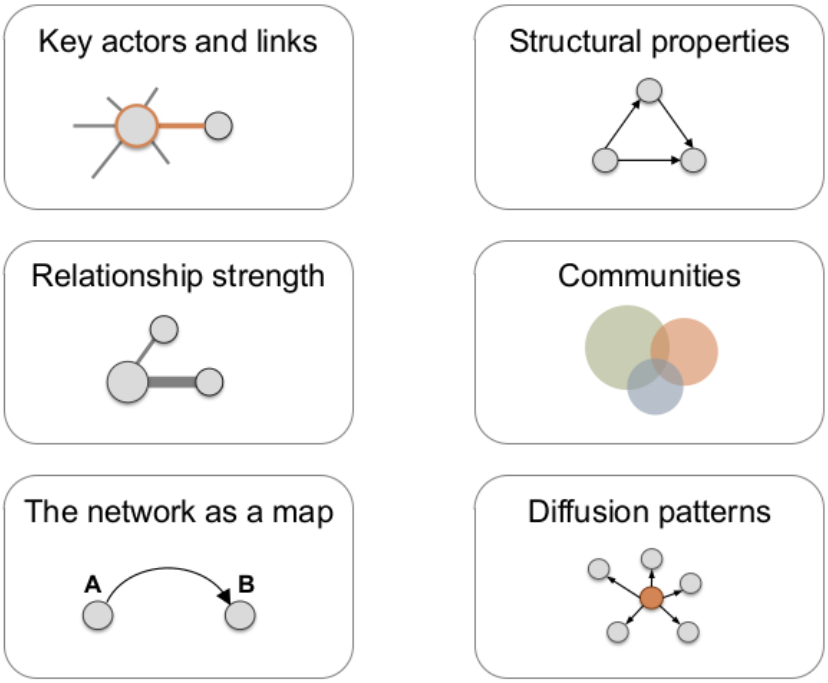
Si vous avez des commentaires, des questions, si vous souhaitez signaler des erreurs, envoyer un e-mail en anglais à rnetviz@ognyanova.net. Si vous avez une remarque sur la traduction française, le destinataire est laurent.beauguitte@cnrs.fr. Enfin, vérifier les dernières versions de ce tutoriel à l'adresse kateto.net.

1 Introduction : visualiser les réseaux

La première question à se poser quand on souhaite visualiser un réseau est l'objectif recherché. Quelles sont les propriétés structurales que l'on souhaite mettre en évidence ?

La visualisation sous forme de graphe n'est pas la seule option disponible - d'autres modes de représentation, voire des graphiques des principales caractéristiques, peuvent être plus appropriés dans certains cas.

Figure 1: Objectifs de la visualisation de réseaux



Dans le graphe, comme dans d'autres modes de visualisation, les variables visuelles permettent de délivrer un message clair. Les principales sont la couleur, la taille, la forme et la position.

Les algorithmes de spatialisation actuels sont à la fois rapides et soucieux d'esthétique. Ils cherchent notamment à éviter les superpositions et les croisements de liens et à assurer des longueurs de liens similaires dans l'ensemble du graphe.

Figure 2: Quelques types de visualisation

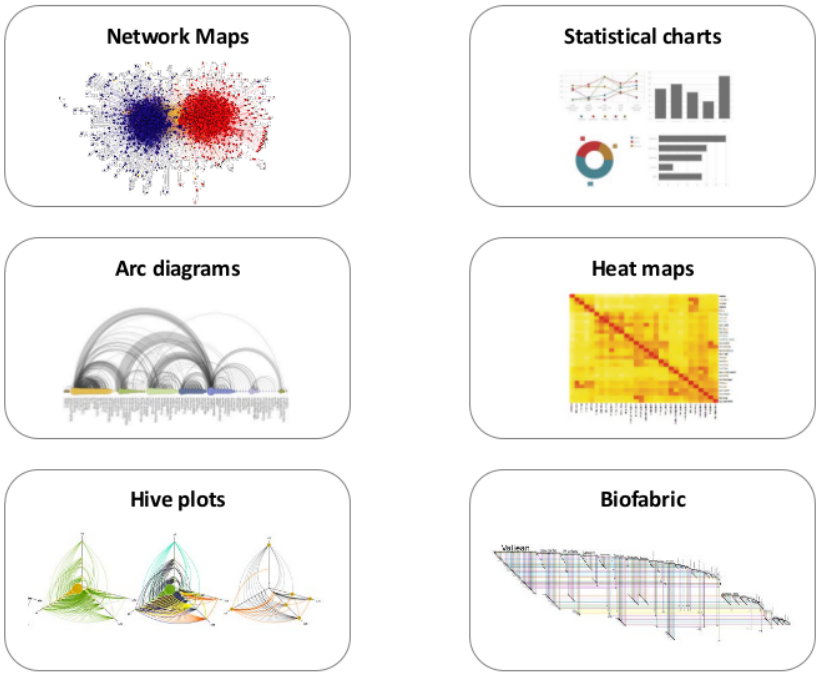


Figure 3: Variables visuelles

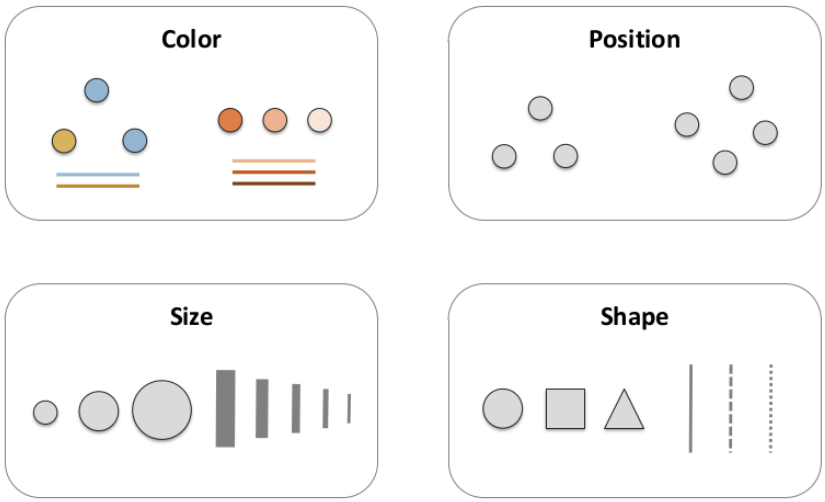
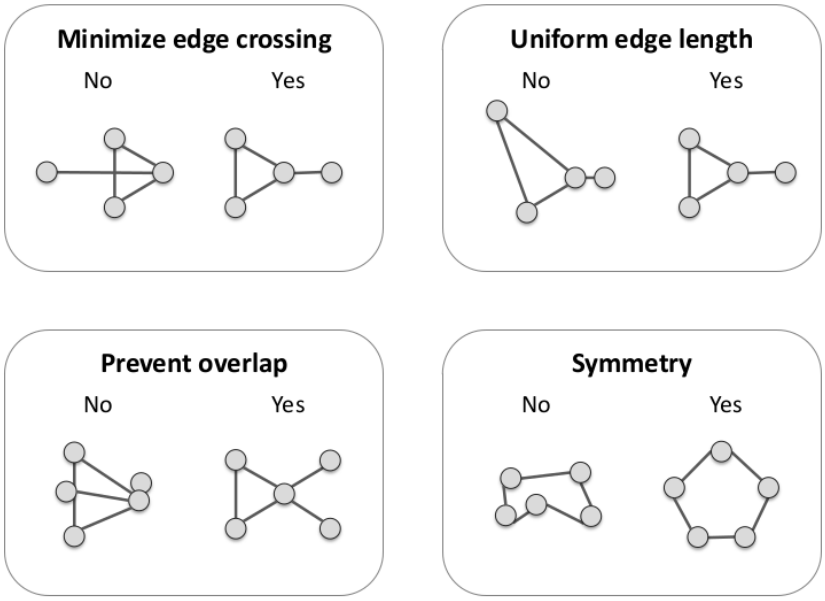


Figure 4: Spatialisations



Ce tutoriel utilise plusieurs packages clés qu'il vous faudra installer. D'autres seront mentionnés mais ne sont pas indispensables.

Les principaux packages qui seront utilisés sont igraph (développé par Gabor Csardi et Tamas Nepusz), sna & network (développé par Carter Butts et l'équipe Statnet), visNetwork (développé par Benoit Thieurmel) et ndtv (développé par Skye Bender-deMoll)[1].

```
install.packages('igraph')
install.packages('network')
install.packages('sna')
install.packages('ndtv')
install.packages('visNetwork')
```

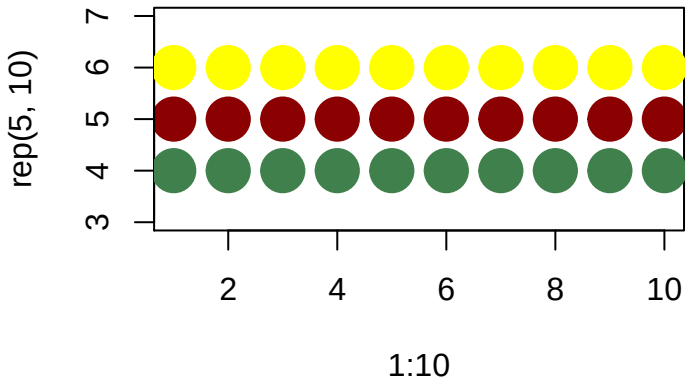
2 Couleurs et fontes avec R

2.1 Les couleurs des graphiques R

Les couleurs sont agréables mais surtout, elles aident les personnes à différencier les types d'objets ou les valeurs d'un attribut. Dans de nombreuses fonctions de R, vous pouvez utiliser des noms de couleur, des valeurs hexadécimales ou RGB.

Dans le script basique ci-dessous, `x` et `y` sont les coordonnées des points, `pch` détermine la forme du point, `cex` sa taille et `col` sa couleur. Pour connaître les paramètres disponibles avec R base, taper `?par`.

```
plot(x=1:10, y=rep(5,10), pch=19, cex=3, col="dark red")
points(x=1:10, y=rep(6,10), pch=19, cex=3, col="557799")
points(x=1:10, y=rep(4,10), pch=19, cex=3, col=rgb(.25,.5,.3))
```

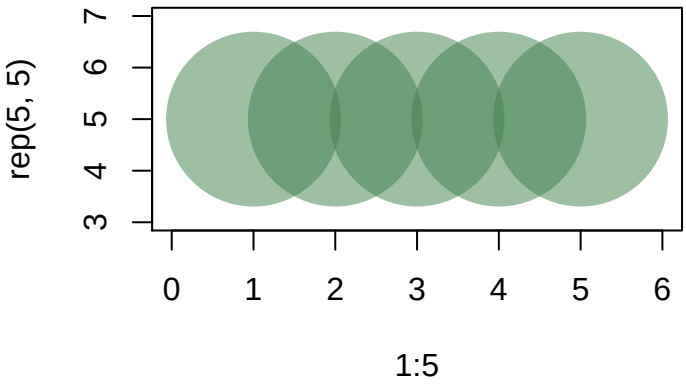


Par défaut, les valeurs RGB varient de O à 1. Ces valeurs peuvent être modifiées pour varier entre 0 et 255 en précisant `rgb(10, 100, 100, maxColorValue=255)`.

Nous pouvons régler l'opacité/la transparence d'un élément à l'aide du paramètre `alpha` (valeur entre 0 et 1).

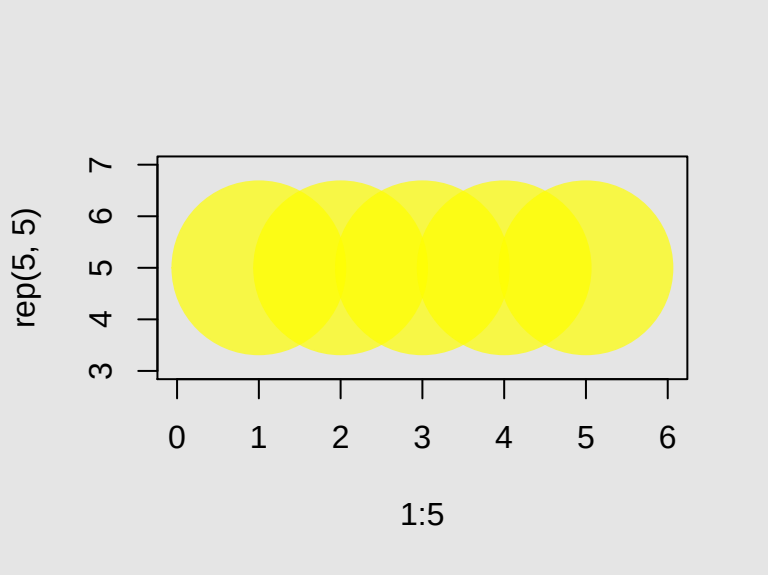
```
plot(x=1:5, y=rep(5,5), pch=19, cex=12, col=rgb(.25,.5,.3, alpha=.5), xlim=c(0,6))
```

[1] Ces deux derniers packages concernent la visualisation dynamique, la traduction est en cours.



Avec les valeurs hexadécimales, la transparence se règle avec la fonction `adjustcolor` du package `grDevices`. Nous allons également mettre un fond gris à l'aide de la fonction `par()`. La taille des marges peut se régler à l'aide des paramètres `par(mar=c(bottom, left, top, right))`. Il est enfin possible de superposer des graphiques en spécifiant `par(new = FALSE)`.

```
par(bg="gray90")
col.tr <- grDevices::adjustcolor("557799", alpha=0.7)
plot(x=1:5, y=rep(5,5), pch=19, cex=12, col=col.tr, xlim=c(0,6))
```

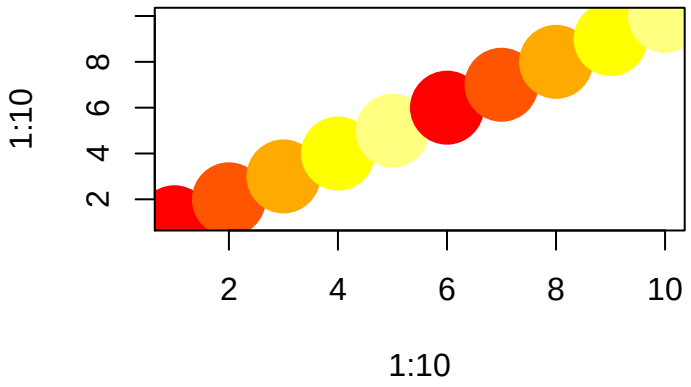


Si vous souhaitez utiliser les noms de couleur, voici comment tous les lister.

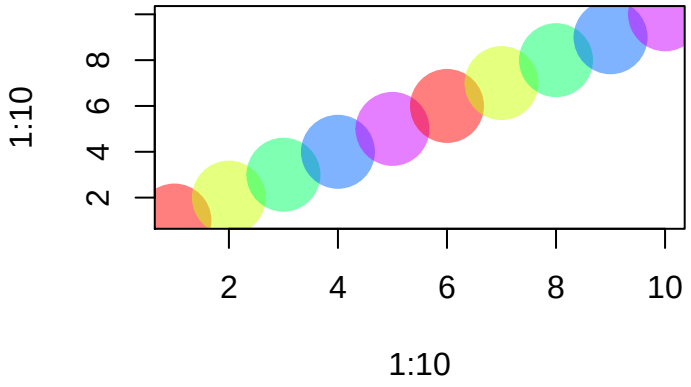
```
colors()                          # Liste de tous les noms de couleurs
grep("blue", colors(), value=TRUE) # Couleurs contenant "blue" dans leur nom
```

Nous avons souvent besoin de gammes de couleurs ou de varier la valeur d'une couleur donnée. R propose des palettes prédéfinies.

```
par(bg="white")
pal1 <- heat.colors(5, alpha=1)   #  5 couleurs chaudes, opaque
pal2 <- rainbow(5, alpha=.5)      #  5 couleurs chaudes, transparent
plot(x=1:10, y=1:10, pch=19, cex=5, col=pal1)
```

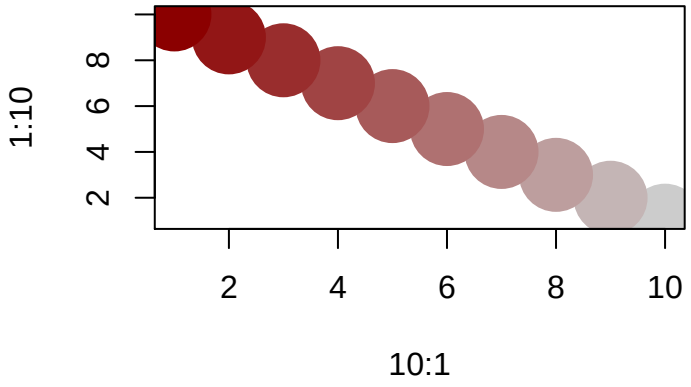



```
plot(x=1:10, y=1:10, pch=19, cex=5, col=pal2)
```



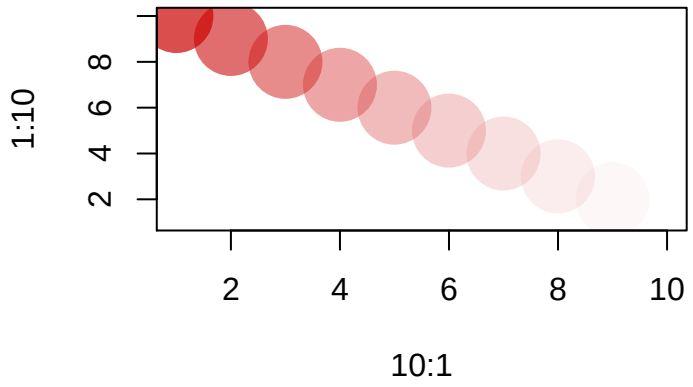
Il est possible de créer ses propres gammes avec `colorRampPalette` et de choisir le nombre de nuances à afficher.

```
palf <- colorRampPalette(c("gray80", "dark red"))
plot(x=10:1, y=1:10, pch=19, cex=5, col=palf(10))
```



Pour utiliser la transparence avec `colorRampPalette`, utilisez le paramètre `alpha=TRUE`.

```
palf <- colorRampPalette(c(rgb(1,1,1,.2),rgb(.8,0,0,.7)), alpha=TRUE)
plot(x=10:1, y=1:10, pch=19, cex=5, col=palf(10))
```



Trouver de bonnes combinaisons de couleurs est délicat et les possibilités de construction de gammes de couleurs dans R restent limitées. Heureusement, des packages spécialisés sont disponibles.

```
# Si RColorBrewer n'est pas installé, exécutez la ligne suivante
# install.packages('RColorBrewer')
library('RColorBrewer')
```

Ce package contient une fonction principale, `brewer.pal`. Pour l'utiliser, il suffit de sélectionner la gamme voulue et le nombre de nuances souhaité. Voici quelques exemples.

```
display.brewer.pal(8, "Set3")
```

Set3 (qualitative)

```
display.brewer.pal(8, "Spectral")
```

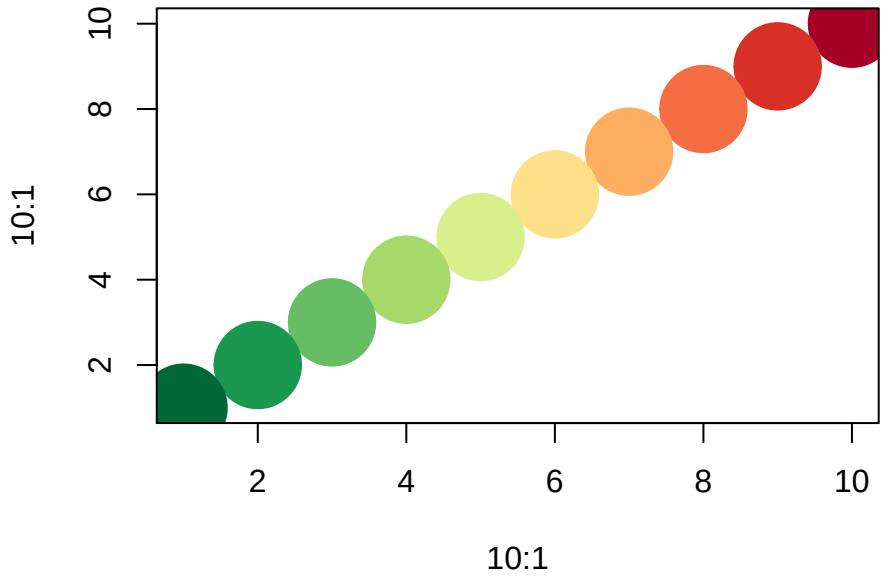
Spectral (divergent)

```
display.brewer.pal(8, "Blues")
```

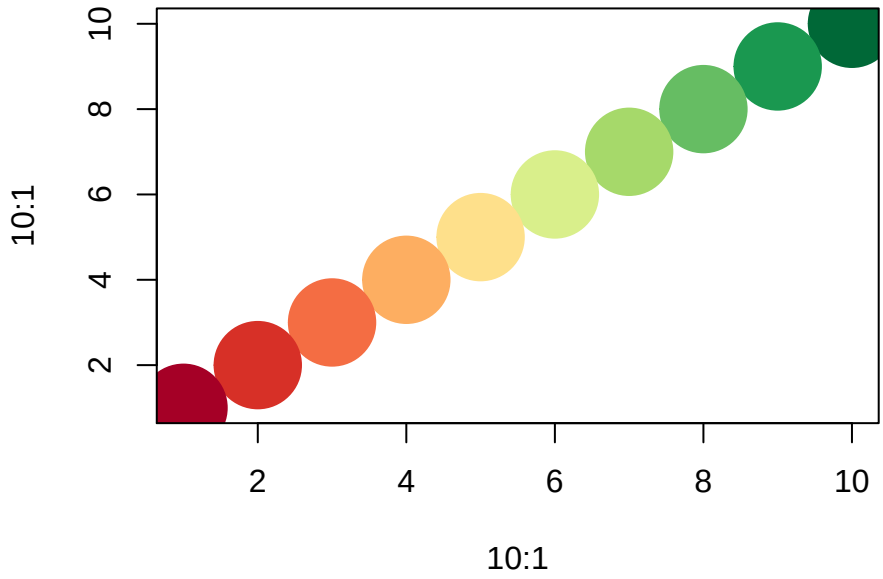
Blues (sequential)

Pour utiliser `RColorBrewer` dans un graphique.

```
pal <- brewer.pal(10, "RdYlGn")
plot(x=10:1, y=10:1, pch=19, cex=6, col=pal)
```




```
plot(x=10:1, y=10:1, pch=19, cex=6, col=rev(pal))
```



Enfin, pour connaître l'ensemble des gammes disponibles, `display.brewer.all()`.

2.2 Polices dans les graphiques

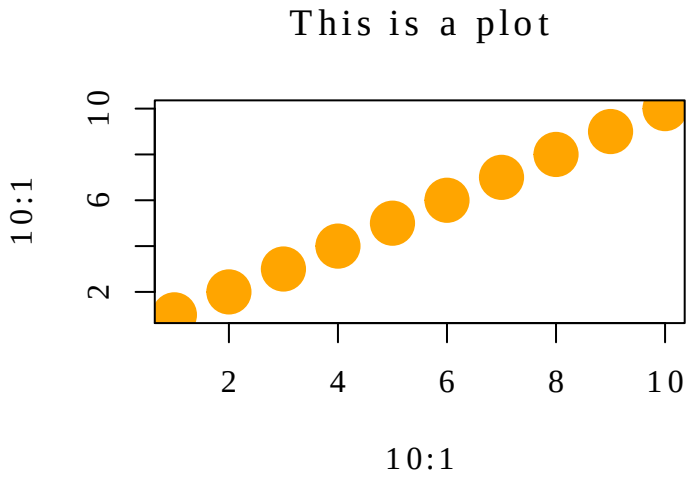
Utiliser des polices différentes dans les graphiques peut être délicat avec R. C'est notamment le cas si vous êtes sous Windows - les personnes sous Mac & Linux peuvent survoler cette partie.

Pour importer les polices du système d'exploitation dans R, nous pouvons utiliser le package `extrafont`.

```
# install.packages('extrafont')
library('extrafont')
# Import des fontes du système - cela peut être long
font_import()
fonts()                    # Liste des polices disponibles
loadfonts(device = "pdf") # Indiquer device = "pdf" pour produire un pdf
```

Maintenant que les polices sont disponibles, il est possible de les insérer dans les graphiques.

```
plot(x=10:1, y=10:1, pch=19, cex=3,
     main="This is a plot", col="orange",
     family="Arial Black" )
```



Quand vous sauvegardez vos graphiques en pdf, vous pouvez y incorporer les polices disponibles.

```
# Il faut d'abord que R puisse trouver ghostscript sur votre ordinateur :
# Sys.setenv(R_GSCMD = "C:/Program Files/gs/gs9.10/bin/gswin64c.exe")

# pdf() envoit les graphiques produits avant dev.off() dans un fichier pdf :
pdf(file="ArialBlack.pdf")
plot(x=10:1, y=10:1, pch=19, cex=6,
     main="This is a plot", col="orange",
     family="Arial Black" )
dev.off()

embed_fonts("ArialBlack.pdf", outfile="ArialBlack_embed.pdf")
```

3 Format des données, volume et préparation

Dans ce tutoriel, nous utiliserons tout d'abord deux petits jeux de données relatifs aux médias. Le premier concerne un réseau d'hyperliens et de mentions entre des sources médiatiques. Le second est un réseau de liens entre des médias et des consommateurs.

Si ces jeux de données sont de petite taille, la plupart des principes liés aux graphiques produits s'appliquent également aux réseaux plus grands. C'est la raison pour laquelle nous utiliserons rarement certaines variables visuelles comme la forme des sommets : elle serait impossible à distinguer sur un graphe de grande taille. Pour représenter de très gros réseaux, il peut même être nécessaire de ne pas représenter les liens et de se concentrer sur l'identification et la visualisation des communautés de sommets.

La taille des réseaux que vous pouvez visualiser avec R est limitée par la RAM de votre ordinateur. Il faut rappeler que, bien souvent, la visualisation de grands réseaux est moins utile que produire des graphiques donnant les principales caractéristiques du graphe.

3.1 Jeu de données 1: format liste de liens (*edgelist*)

Le premier jeu de données comprend deux fichiers "Media-Example-NODES.csv" et "Media-Example-EDGES.csv" (téléchargeables ici).

```
nodes <- read.csv("Data/Dataset1-Media-Example-NODES.csv", header=T, as.is=T)
links <- read.csv("Data/Dataset1-Media-Example-EDGES.csv", header=T, as.is=T)
```

Examiner les données.

```
head(nodes, 4)
```

```
##    id               media media.type type.label audience.size
## 1 s01            NY Times          1  Newspaper            20
## 2 s02     Washington Post          1  Newspaper            25
## 3 s03 Wall Street Journal          1  Newspaper            30
## 4 s04           USA Today          1  Newspaper            32
```

```
head(links, 4)
```

```
##   from  to weight      type
## 1  s01 s02     10 hyperlink
## 2  s01 s02     12 hyperlink
## 3  s01 s03     22 hyperlink
## 4  s01 s04     21 hyperlink
```

```
nrow(nodes); length(unique(nodes$id))
```

```
## [1] 17
```

```
## [1] 17
```

```
nrow(links); nrow(unique(links[,c("from", "to")]))
```

```
## [1] 52
```

```
## [1] 49
```

Il y a plus de liens (52) que de combinaisons uniques origine-destination (49). Cela signifie qu'il y a dans les données des liens multiples entre certaines paires de sommets. Nous allons agréger ces liens multiples en sommant leur intensité à l'aide de la fonction `aggregate`.

```
links <- aggregate(links[,3], links[,-3], sum)
links <- links[order(links$from, links$to),]
colnames(links)[4] <- "weight"
rownames(links) <- NULL
```

3.2 Jeu de données 2 : format matrice

```
nodes2 <- read.csv("Data/Dataset2-Media-User-Example-NODES.csv", header=TRUE, as.is=TRUE)
links2 <- read.csv("Data/Dataset2-Media-User-Example-EDGES.csv", header=TRUE, row.names=1)
```

Examiner les données.

```
head(nodes2, 4)
```

```
##    id media media.type media.name audience.size
## 1 s01   NYT          1  Newspaper            20
## 2 s02  WaPo          1  Newspaper            25
## 3 s03   WSJ          1  Newspaper            30
## 4 s04  USAT          1  Newspaper            32
```

```
head(links2, 4)
```

```
##     U01 U02 U03 U04 U05 U06 U07 U08 U09 U10 U11 U12 U13 U14 U15 U16 U17
## s01   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## s02   0   0   0   1   1   0   0   0   0   0   0   0   0   0   0   0   0
## s03   0   0   0   0   0   1   1   1   1   0   0   0   0   0   0   0   0
## s04   0   0   0   0   0   0   0   0   1   1   1   0   0   0   0   0   0
##     U18 U19 U20
## s01   0   0   0
## s02   0   0   1
## s03   0   0   0
## s04   0   0   0
```

links2 est une matrice d'adjacence d'un réseau biparti. Les réseaux bipartis, appelés aussi bimodaux ou 2-mode, représentent les liens entre deux catégories d'acteurs différents. Ce jeu de données donne les liens entre des médias et des personnes.

```
links2 <- as.matrix(links2)
dim(links2)
dim(nodes2)
```

4 Visualiser des réseaux avec igraph

Nous commençons par convertir les données en objet `igraph`.

4.1 Transformer le réseau en objet igraph

Jeu de données 1 : liste de liens

Pour convertir nos données en objet `igraph`, nous utilisons la fonction `graph_from_data_frame()` qui prend deux data.frame comme arguments : `d` et `vertices`.

- `d` décrit les liens du réseau. Ses deux premières colonnes contiennent les identifiants des origines et des destinations de chaque lien. Les colonnes suivantes contiennent les attributs des liens (intensité, type, nom, etc.). Si les sommets n'ont pas d'attributs, il est possible de générer un objet `igraph` avec ce seul data.frame.

- `vertices` a pour première colonne les identifiants des sommets. Les colonnes suivantes sont considérées comme des attributs des sommets.

```
library('igraph')
net <- graph_from_data_frame(d=links, vertices=nodes, directed=TRUE)
net
```

```
## IGRAPH DNW- 17 49 --
## + attr: name (v/c), media (v/c), media.type (v/n), type.label
## | (v/c), audience.size (v/n), type (e/c), weight (e/n)
## + edges (vertex names):
##  [1] s01->s02 s01->s03 s01->s04 s01->s15 s02->s01 s02->s03 s02->s09
##  [8] s02->s10 s03->s01 s03->s04 s03->s05 s03->s08 s03->s10 s03->s11
## [15] s03->s12 s04->s03 s04->s06 s04->s11 s04->s12 s04->s17 s05->s01
## [22] s05->s02 s05->s09 s05->s15 s06->s06 s06->s16 s06->s17 s07->s03
## [29] s07->s08 s07->s10 s07->s14 s08->s03 s08->s07 s08->s09 s09->s10
## [36] s10->s03 s12->s06 s12->s13 s12->s14 s13->s12 s13->s17 s14->s11
## [43] s14->s13 s15->s01 s15->s04 s15->s06 s16->s06 s16->s17 s17->s04
```

La description d'un objet igraph commence par des lettres :

- D ou U, indique si le réseau est orienté (*D* comme *directed*) ou non (*U* comme *undirected*)
- N indique que les sommets ont un nom pour attribut
- W indique un graphe valué (les liens ont une intensité - *W* comme *weight*)
- B indique que le graphe est biparti.

Les deux nombres suivants (17 49) donnent le nombre de sommmets et le nombre de liens du réseau. La description liste également les attributs éventuels des sommets et des liens. Par example :

- (g/c) - attribut du graphe de type caractère
- (v/c) - attribut des sommets (*v* comme *vertices*) de type caractère
- (e/n) - attrbut des liens (*e* comme *edge*) de type numérique

Nous avons accès facilement aux sommets, aux liens et à leurs attributs avec les commandes.

```
E(net)       # les liens de l'objet "net"
V(net)       # les sommets de l'objet "net"
E(net)$type  # attribut des liens "type"
V(net)$media # attribut des sommets "media"
```

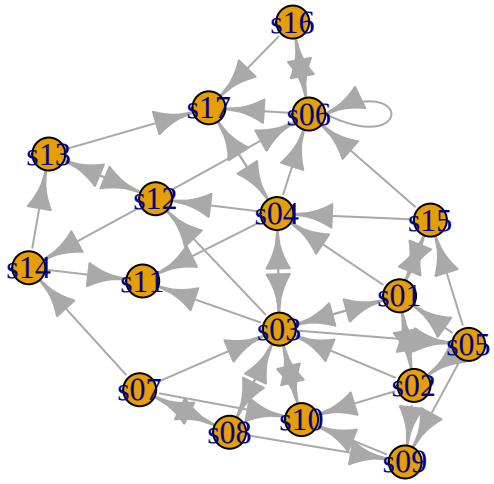
```r
# Rechercher des sommets et des liens en fonction des attributs
# Résultats de forme numéro des sommets/numéro des liens
V(net)[media=="BBC"]
E(net)[type=="mention"]

# On peut également examiner la matrice directement
net[1,]
net[5,7]
net[2:5, 2:5]
```

Il est facile d'extraire une liste de liens ou une matrice d'un objet `igraph`.

```r
# Obtenir une liste de liens ou une matrice
as_edgelist(net, names=TRUE)
as_adjacency_matrix(net, attr="weight")

# ou des dataframe décrivant sommets et liens
as_data_frame(net, what="edges")
as_data_frame(net, what="vertices")
```

Maintenant que nous avons notre objet `igraph`, voici la visualisation par défaut du package.

```r
plot(net) # pas top...
```



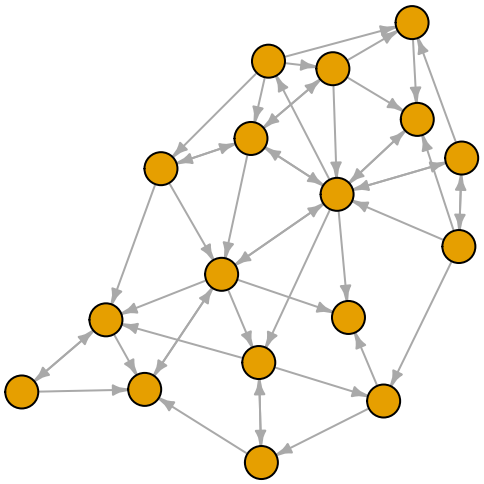
Commençons par supprimer les boucles (i.e. lien d'un lien vers lui-même).

```r
net <- simplify(net, remove.multiple = FALSE, remove.loops = TRUE)
```

Il aurait été possible d'utiliser `simplify` pour agrréger les liens multiples en sommant les intensités avec la commande `simplify(net, edge.attr.comb=list(Weight="sum","ignore"))`. Ici, cela aurait conduit à combiner des liens de différents types ("hyperlinks" and "mentions").

Réduisons la taille des fléches et supprimons les labels des sommets (utilisation de NA).

```
plot(net, edge.arrow.size=.4,vertex.label=NA)
```



Jeu de données 2 : matrice

Comme indiqué plus haut, les liens du second réseau sont en format matriciel. On peut les transformer en objet `igraph` avec la fonction `graph_from_incidence_matrix()`. Dans `igraph`, les réseaux bipartis ont un attribut sur les sommets appelé `type` qui est `FALSE` (ou 0) pour les sommets d'un ensemble d'acteurs et `TRUE` (ou 1) pour les sommets de l'autre ensemble d'acteurs.

```
head(nodes2)
head(links2)

net2 <- graph_from_incidence_matrix(links2)
table(V(net2)$type)
```

La fonction `graph_from_adjacency_matrix()` permet de transformer la matrice d'adjacence d'un réseau one-mode.

4.2 Paramètres graphiques

`igraph` propose de nombreux paramètres pour visualiser les réseaux. Les options concernant les sommets commencent par `vertex.`, les options concernant les liens par `edge.`. La liste ci-dessous n'est pas exhaustive, voir `?igraph.plotting` pour plus d'informations.

Principaux paramètres graphiques disponibles dans `igraph`

Sommets

`vertex.color`	Couleur du sommet
`vertex.frame.color`	Couleur de la bordure du sommet
`vertex.shape`	Choisir entre “none”, “circle”, “square”, “csquare”, “rectangle” “crectangle”, “vrectangle”, “pie”, “raster” ou “sphere”
`vertex.size`	Taille du sommet (15 par défaut)
`vertex.size2`	Taille 2 du sommet (pour un rectangle par exemple)
`vertex.label`	Vecteur de type caractère utilisé pour nommer les sommets
`vertex.label.family`	Police des labels (e.g.“Times”, “Helvetica”)
`vertex.label.font`	Fonte : 1 normal, 2 gras, 3, italique, 4 italique gras, 5 symbole
`vertex.label.cex`	Taille de la police (facteur multiplicatif, dépendant de l’OS)
`vertex.label.dist`	Distance entre le label et le sommet
`vertex.label.degree`	Position du label par rapport au sommet, où 0 est à droite, “pi” à gauche, “pi/2” en dessous et “-pi/2” au dessus

Liens

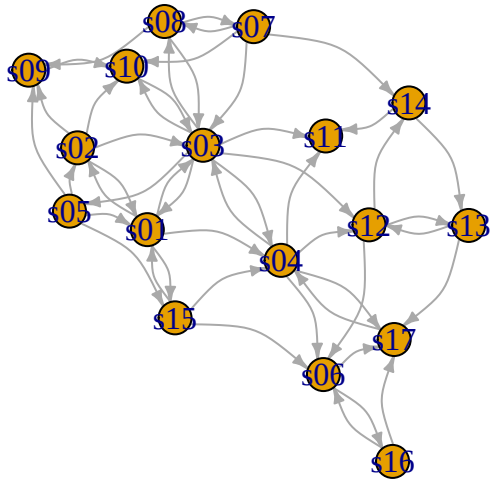
`edge.color`	Couleur du lien
`edge.width`	Épaisseur du lien, 1 par défaut
`edge.arrow.size`	Taille des fléches, 1 par défaut
`edge.arrow.width`	Épaisseur des flèches, 1 par défaut
`edge.lty`	Type de ligne, dont 0 ou “blank”, 1 ou “solid”, 2 ou “dashed”, 3 ou “dotted”, 4 ou “dotdash”, 5 ou “longdash”, 6 ou “twodash”
`edge.label`	Vecteur de type caractère utilisé pour nommer les liens
`edge.label.family`	Police des labels (e.g.“Times”, “Helvetica”)
`edge.label.font`	Fonte : 1 normal, 2 gras, 3, italique, 4 italique gras, 5 symbole
`edge.label.cex`	Taille de la police pour les labels des liens
`edge.curved`	Courbure des liens, varie de 0 (FALSE) à 1
`arrow.mode`	Vecteur indiquant si les liens doivent avoir des flèches : 0 pas de flèche, 1 avant, 2 arrière, 3 les deux

Autres

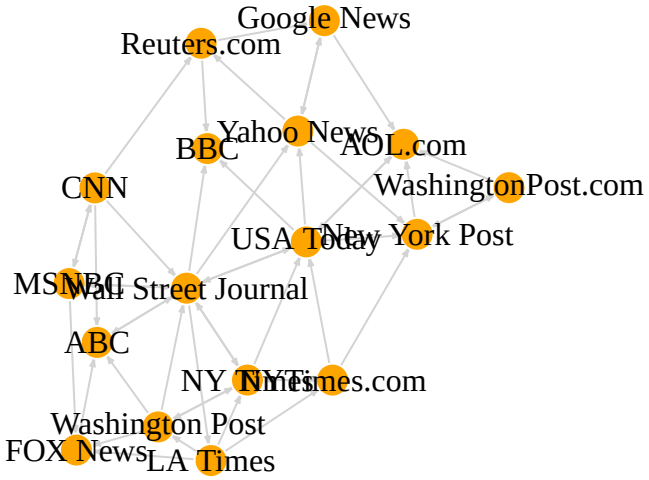
`margin`	Marges autour du graphique, vecteur de longueur 4
`frame`	si TRUE, le graphique sera encadré
`main`	Titre du graphique
`sub`	Sous-titre du graphique
`asp`	Numérique, the aspect ratio of a plot (y/x).
`palette`	Gamme de couleurs à utiliser pour les sommets
`rescale`	Coordonnées à [-1,1]. TRUE par défaut.

Il est possible d’utiliser les options de deux manières. La première est de les préciser dans la fonction `plot` comme dans les exemples suivants.

```
# Liens courbes (edge.curved=.1) et taille de flèches réduite :
# L'utilisation des liens courbes permet de voir les liens multiples et/ou mutuels
plot(net, edge.arrow.size=.4, edge.curved=.4)
```



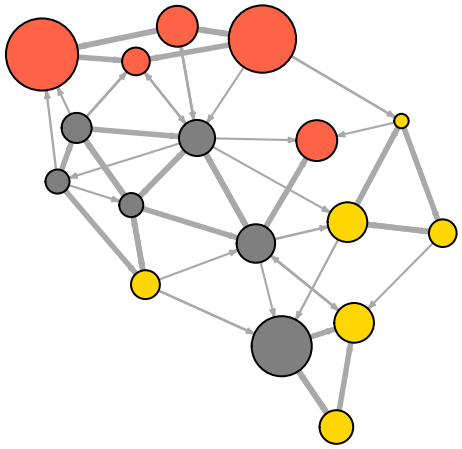

```
# Couleur des liens en gris pale, sommet en orange
# Utilisation des noms "media" comme label des sommets
plot(net, edge.arrow.size=.2, edge.color="lightgrey",
     vertex.color="orange", vertex.frame.color="#ffffff",
     vertex.label=V(net)$media, vertex.label.color="black")
```



La seconde méthode pour personnaliser les graphiques est d'ajouter les attributs à l'objet `igraph`. On peut par exemple colorer les sommets selon le type de média, faire varier la taille en fonction de la centralité de degré et faire varier l'épaisseur des liens selon leur intensité.

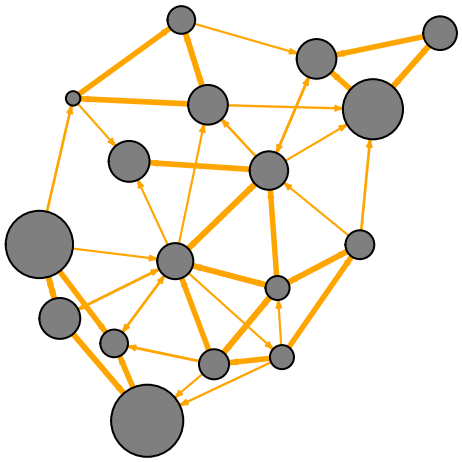
```
# Attribuer des couleurs selon le type de média
colrs <- c("gray50", "tomato", "gold")
V(net)$color <- colrs[V(net)$media.type]

# Calculer le degré et faire varier la taille des sommets
deg <- degree(net, mode="all") # mode="all" compte les degrés entrants et sortants
V(net)$size <- deg*3
# On pourrait également utiliser l'audience
V(net)$size <- V(net)$audience.size*0.6

# Les labels sont pour le moment les identifiants
```

```r
# Leur attribuer NA permet de les neutraliser
V(net)$label <- NA

# Épaisseur des liens fonction de l'intensité
E(net)$width <- E(net)$weight/6

# Changer la taille des flèches et la couleur des liens
E(net)$arrow.size <- .2
E(net)$edge.color <- "gray80"
E(net)$width <- 1+E(net)$weight/12
plot(net)
```



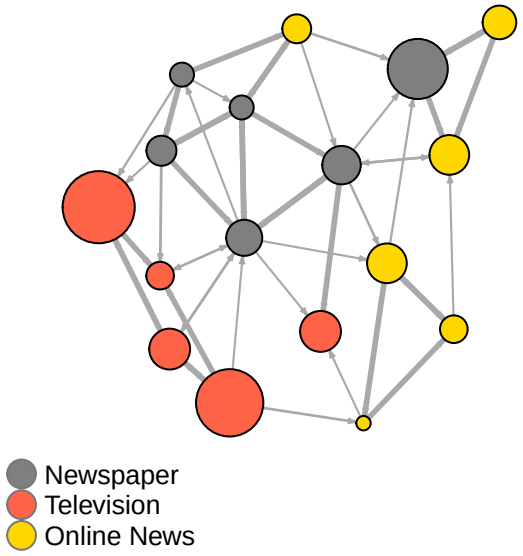
On peut écraser temporairement ces attributs en changeant les arguments de la fonction `plot`.

```r
plot(net, edge.color="orange", vertex.color="gray50")
```



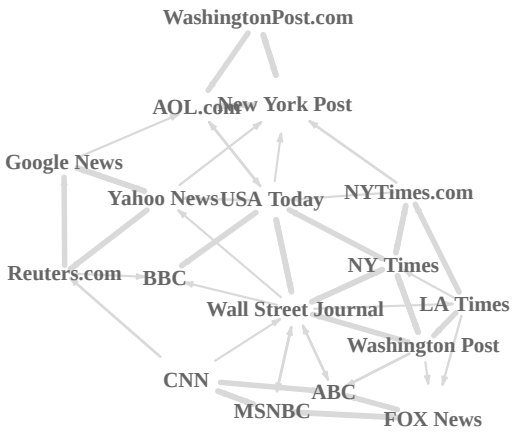
Il est utile (voire indispensable) d'ajouter une légende expliquant le sens des couleurs utilisées.

```r
plot(net)
legend(x=-1.5, y=-1.1, c("Newspaper","Television", "Online News"), pch=21,
       col="#777777", pt.bg=colrs, pt.cex=2, cex=.8, bty="n", ncol=1)
```



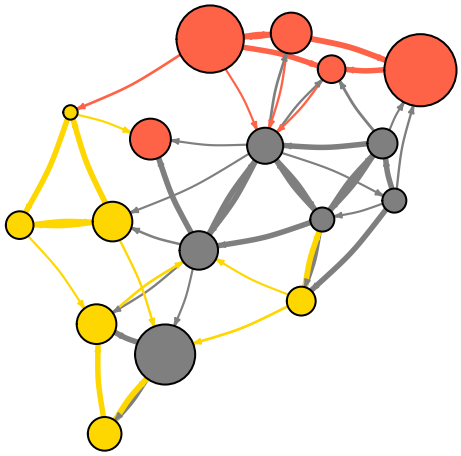
Parfois, notamment avec les réseaux sémantiques, on peut vouloir ne représenter que les labels des sommets.

```
plot(net, vertex.shape="none", vertex.label=V(net)$media,
     vertex.label.font=2, vertex.label.color="gray40",
     vertex.label.cex=.7, edge.color="gray85")
```



Colorons les liens selon le sommet émetteur. On peut obtenir le sommet émetteur pour chaque lien avec la fonction `ends()`. Elle renvoie le sommet d'origine et d'arrivée des liens listés dans le paramètre `es`. Le paramètre `names` permet d'obtenir soit le nom du lien soit son identifiant.

```
edge.start <- ends(net, es=E(net), names=F)[,1]
edge.col <- V(net)$color[edge.start]

plot(net, edge.color=edge.col, edge.curved=.1)
```

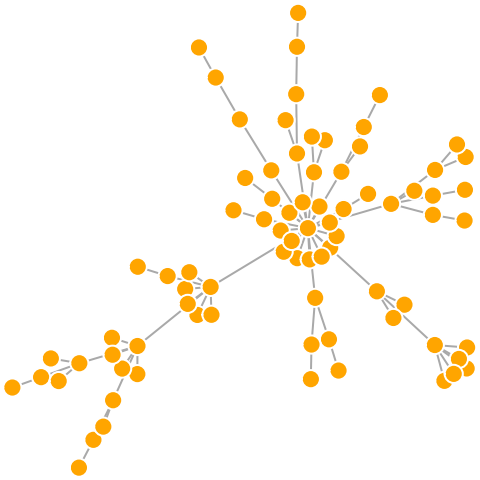


4.3 Spatialisations

Les algorithmes de spatialisation de réseaux permettent d'attribuer des coordonnées à chacun des sommets.

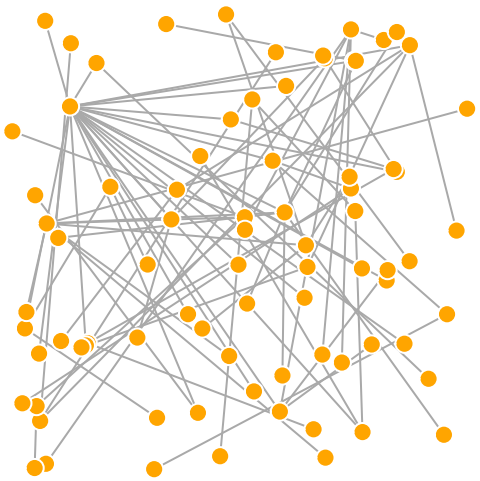
Afin de montrer les différents algorithmes, nous allons générer un réseau un peu plus grand (80 sommets). La fonction `sample_pa()` geénère un graphe simple, composé d'un seul sommet au départ, puis les ajouts de sommets et de liens suivent la logique de l'attachement préférentiel (modèle de Barabasi-Albert).

```
net.bg <- sample_pa(80)
V(net.bg)$size <- 8
V(net.bg)$frame.color <- "white"
V(net.bg)$color <- "orange"
V(net.bg)$label <- ""
E(net.bg)$arrow.mode <- 0
plot(net.bg)
```



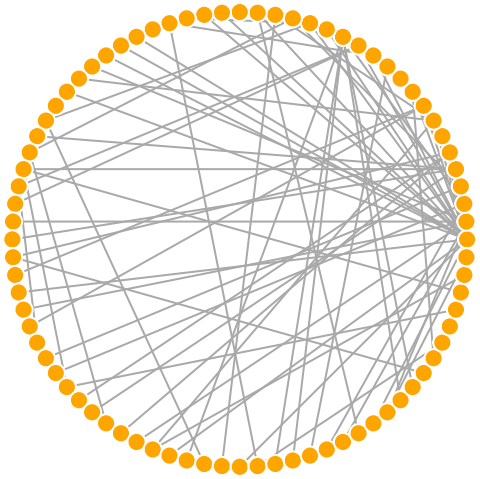
On peut choisir la spatialisation dans la fonction `plot`.

```
plot(net.bg, layout=layout_randomly)
```



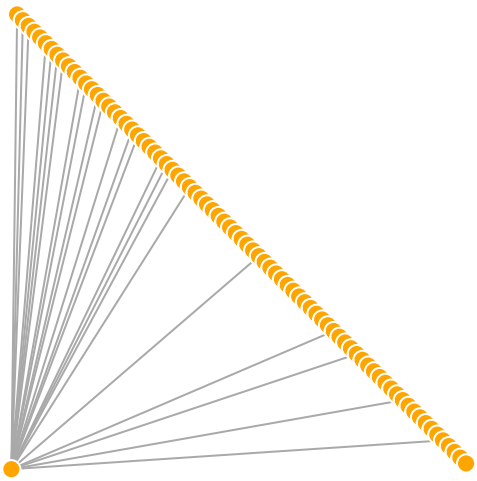
Ou alors calculer au préalable les coordonnées des sommets.

```
l <- layout_in_circle(net.bg)
plot(net.bg, layout=l)
```



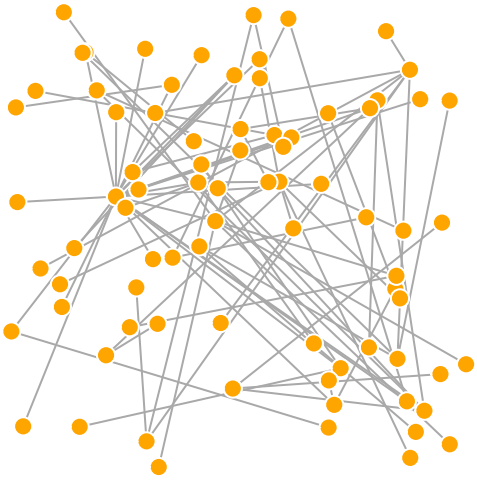
`l` est simplement une matrice de coordonnées x,y de dimension N x 2 pour les N sommets du graphe. Il est aisé d'en générer soi-même.

```
l <- cbind(1:vcount(net.bg), c(1, vcount(net.bg):2))
plot(net.bg, layout=l)
```

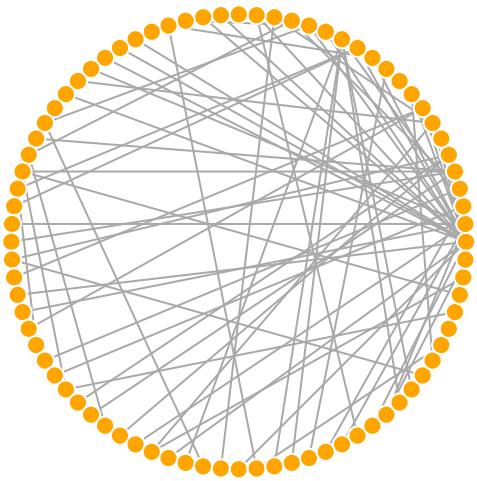


Cette spatialisation est juste un exemple d'une utilité relative... Heureusement, `igraph` propose un certain nombre de spatialisations.

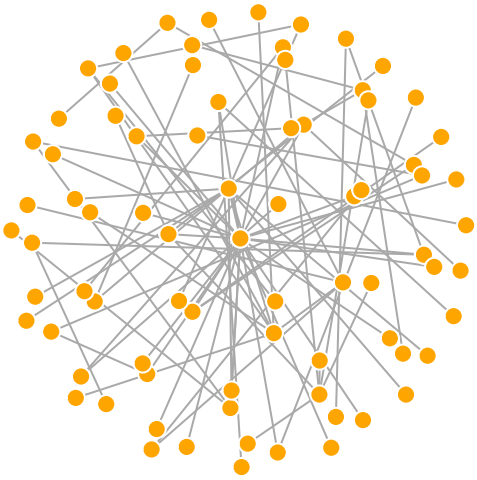
```
# Placement aléatoire
l <- layout_randomly(net.bg)
plot(net.bg, layout=l)
```



```
# Placement circulaire
l <- layout_in_circle(net.bg)
plot(net.bg, layout=l)
```



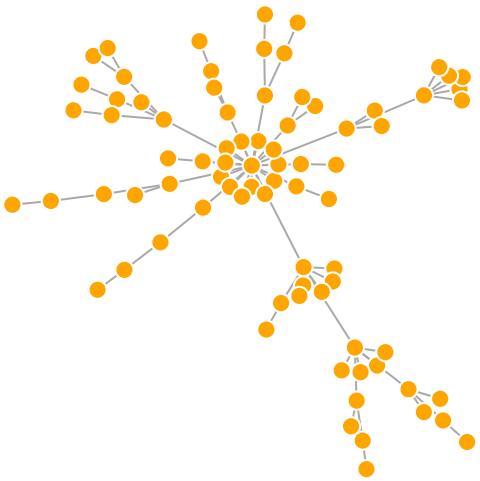
```
# Spatialisation 3D sphérique
l <- layout_on_sphere(net.bg)
plot(net.bg, layout=l)
```



Fruchterman-Reingold est l'une des spatialisations *force-directed* les plus utilisées.

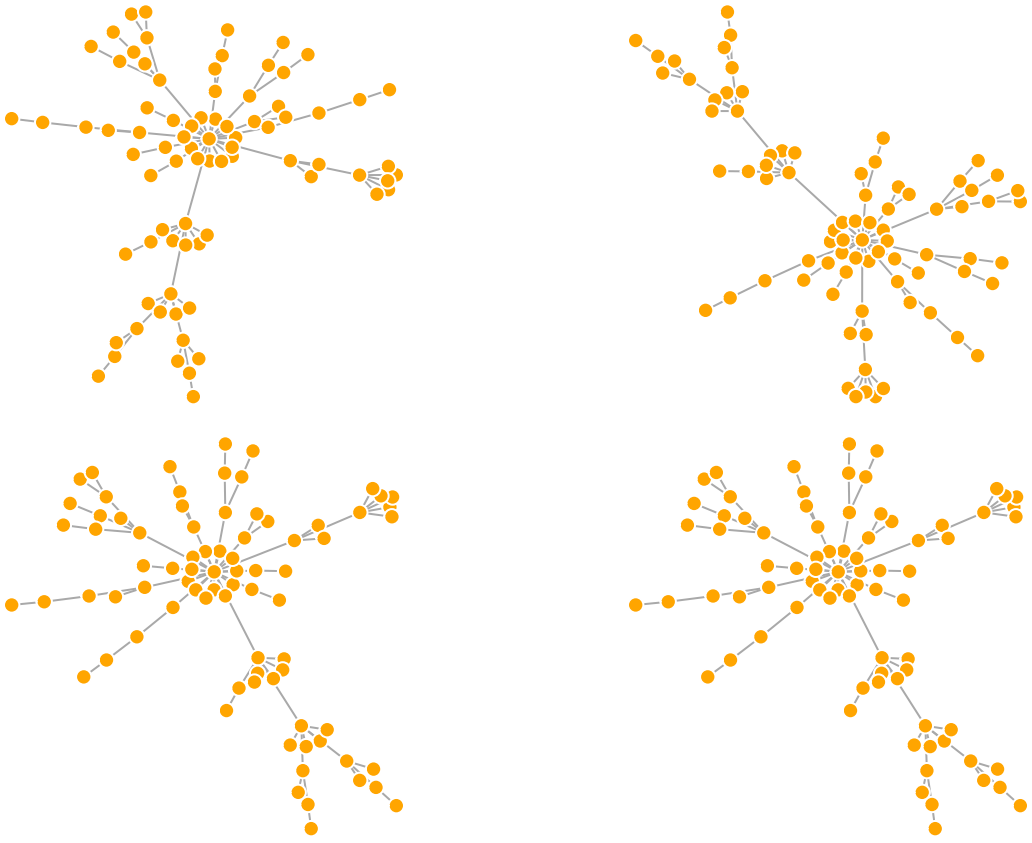
Les spatialisations *force-based* tentent de générer un graphe élégant où les liens sont de longueur similaire et se croisent aussi peu que possible. Ils modélisent le réseau en un système physique. Les sommets sont chargés d'électricité et se repoussent lorsqu'ils sont trop proches. Les liens agissent comme des ressorts qui rapprochent les sommets connectés. Au final, les sommets sont disposés régulièrement sur le plan et les sommets partageant des connexions sont proches les uns des autres. L'inconvénient de ces algorithmes est qu'ils sont assez lents donc peu souvent utilisés pour des graphes de plus de ~1000 sommets. Il est possible de régler le paramètre `weight` qui augmente les forces d'attraction entre les sommets connectés par des liens plus intenses.

```
l <- layout_with_fr(net.bg)
plot(net.bg, layout=l)
```



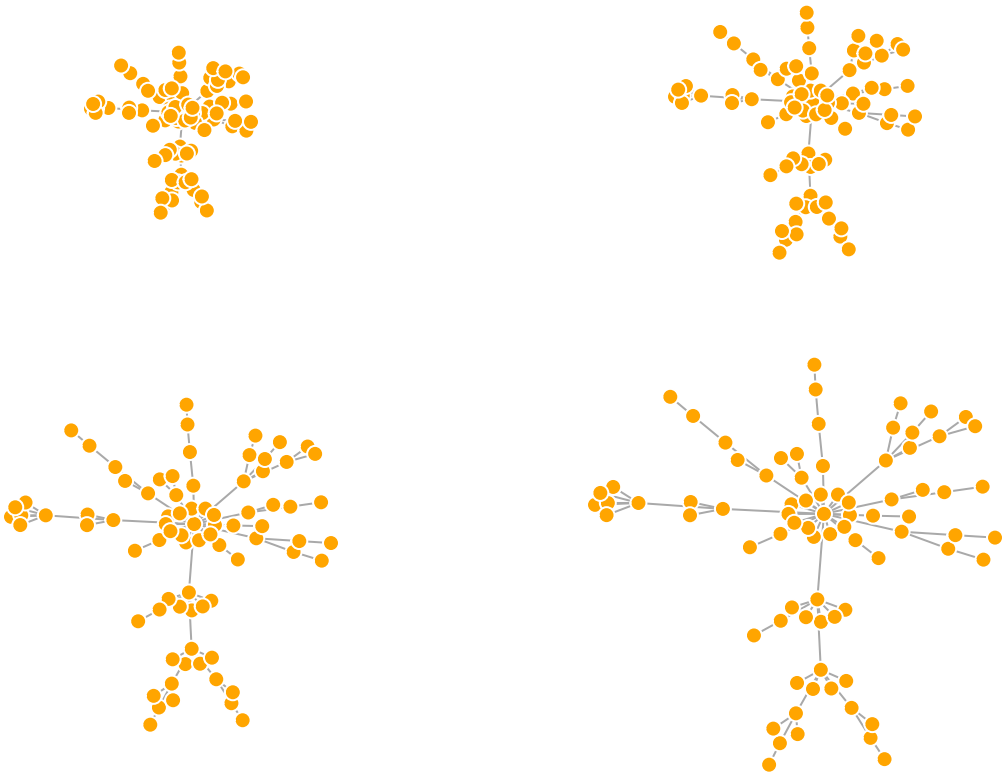
Cette spatialisation n'est pas déterministe : différentes exécutions donneront des résultats légèrement différents. Sauvegarder les coordonnées dans `l` permet de garder la même représentation graphique (position des sommets), ce qui peut être utile pour représenter l'évolution d'un graphe ou différents types de relations entre les sommets.

```
par(mfrow=c(2,2), mar=c(0,0,0,0))    # 4 figures, 2 lignes, 2 colonnes, marges à zéro
plot(net.bg, layout=layout_with_fr)
plot(net.bg, layout=layout_with_fr)
plot(net.bg, layout=l)
plot(net.bg, layout=l)
```



Par défaut, les coordonnées des graphiques sont graduées sur l'intervalle [-1,1] pour les x et les y. Il est possible de les modifier avec le paramètre `rescale = FALSE` et de graduer son graphique manuellement en multipliant les coordonnées par un scalaire. On peut utiliser `norm_coords` pour normaliser le graphique aux dimensions désirées. On peut ainsi créer des spatialisations plus compactes ou plus amples.

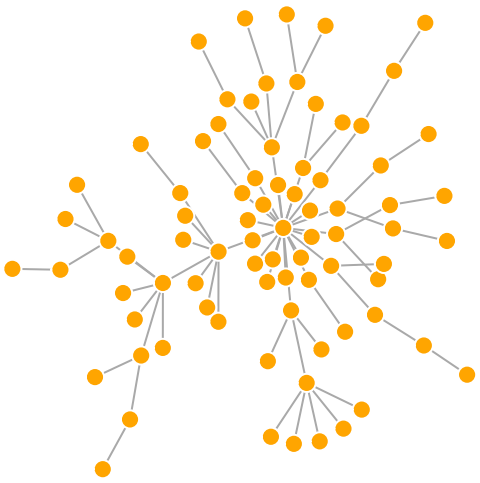
```
l <- layout_with_fr(net.bg)
l <- norm_coords(l, ymin=-1, ymax=1, xmin=-1, xmax=1)

par(mfrow=c(2,2), mar=c(0,0,0,0))
plot(net.bg, rescale=F, layout=l*0.4)
plot(net.bg, rescale=F, layout=l*0.6)
plot(net.bg, rescale=F, layout=l*0.8)
plot(net.bg, rescale=F, layout=l*1.0)
```



```
dev.off()
```

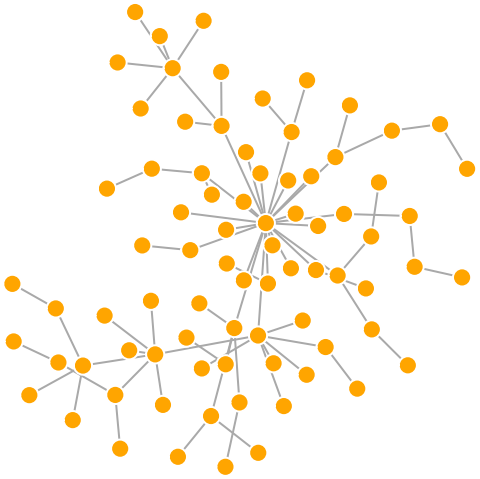
Un autre algorithme *force-directed* produisant de beaux résultats est appelé Kamada Kawai. Comme Fruchterman Reingold, il vise à minimiser l'énergie dans un système basé sur des ressorts.

```
l <- layout_with_kk(net.bg)
plot(net.bg, layout=l)
```



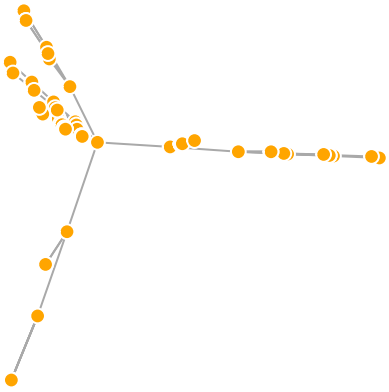
L'algorithme LGL est conçu pour de grands graphes connexes. Il est possible de spécifier une racine, c'est-à-dire qu'un sommet sera placé au centre du graphe.

```
plot(net.bg, layout=layout_with_lgl)
```



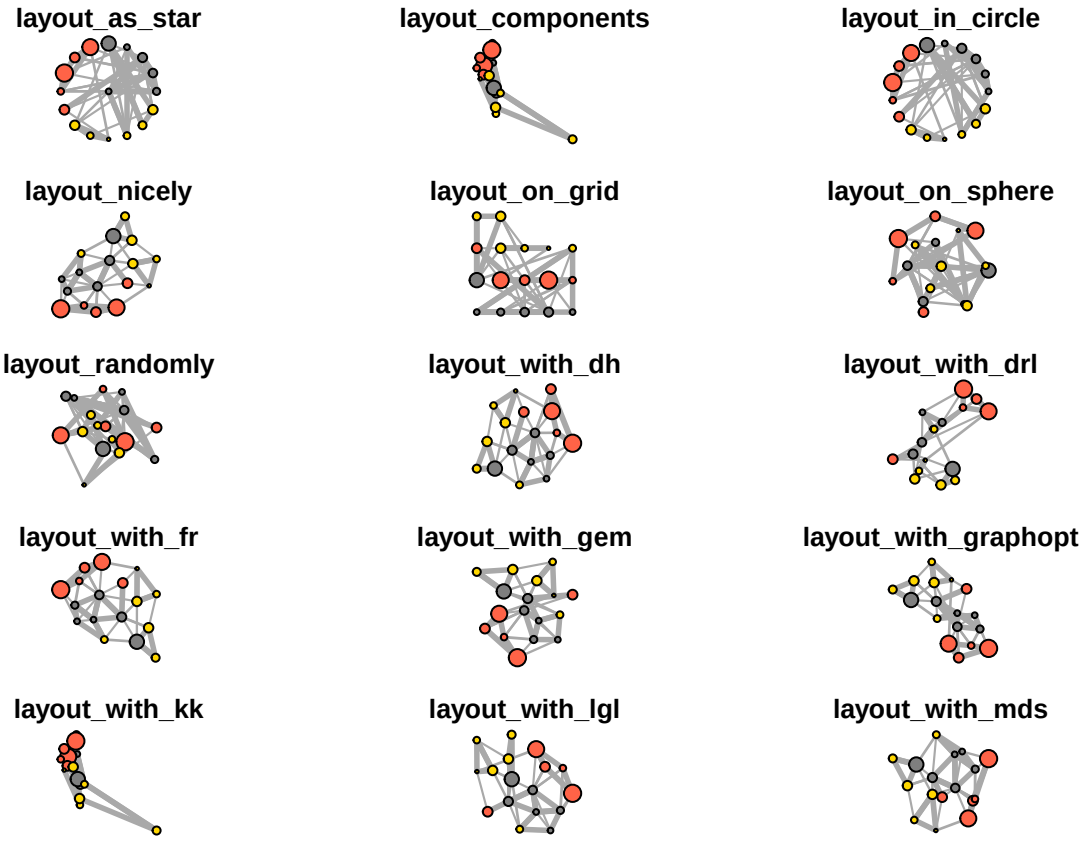
L'algorithme MDS (*multidimentional scaling*) cherche à placer les sommets en fonction d'une mesure de similarité, considérée comme une distance, entre eux. Les sommets les plus semblables seront placés très près les uns des autres. Par défaut, la distance utilisée est le plus court chemin entre sommets. On peut utiliser une autre matrice de distance (qu'il faut donc définir) avec le paramètre `dist`. Les spatialisations MDS sont intéressantes car positions et distances s'interprétent clairement. Par contre, le résultat est peu satisfaisant au niveau esthétique : les sommets se superposent souvent ou se retrouvent empilés les uns au dessus des autres.

```
plot(net.bg, layout=layout_with_mds)
```



Regardons l'ensemble des spatialisations disponibles dans igraph :

```
layouts <- grep("^layout_", ls("package:igraph"), value=TRUE)[-1]
# Suppression des spatialisations non adaptées à ce graphe
layouts <- layouts[!grepl("bipartite|merge|norm|sugiyama|tree", layouts)]

par(mfrow=c(5,3), mar=c(1,1,1,1))
for (layout in layouts) {
  print(layout)
  l <- do.call(layout, list(net))
  plot(net, edge.arrow.mode=0, layout=l, main=layout) }
```

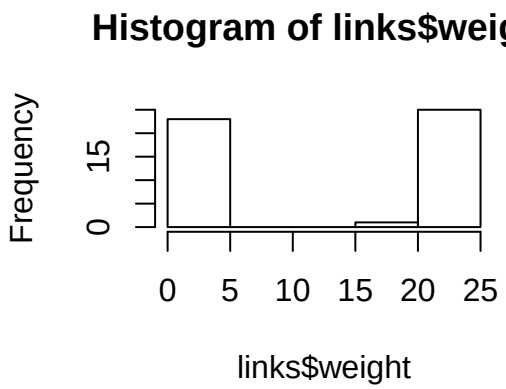


```
dev.off()
```

4.4 Souligner certains aspects du réseau

Nos visualisations de réseau restent peu utiles. On parvient à identifier la taille et le type des sommets mais percevoir la structure est difficile en raison de la densité des liens. Une option possible est de filtrer les liens pour ne garder que les plus intenses.

```
hist(links$weight)
```




```
mean(links$weight)
```

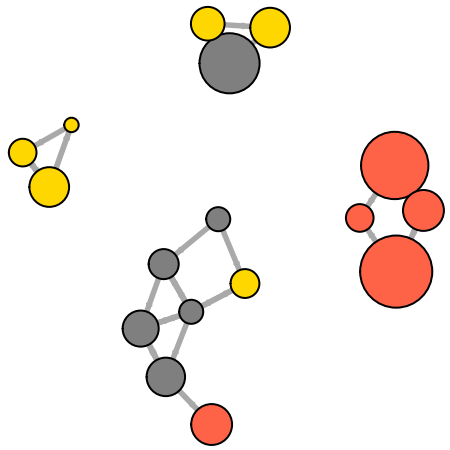
```
## [1] 12.40816
```

```
sd(links$weight)
```

```
## [1] 9.905635
```

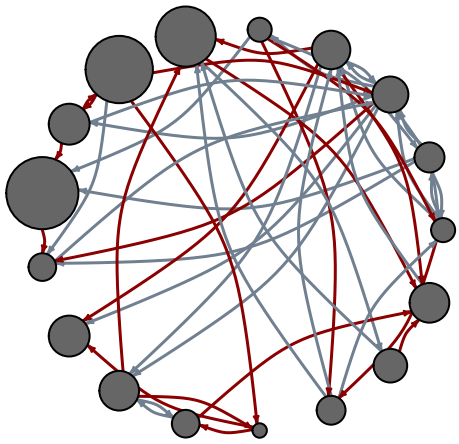
Il existe des moyens plus rigoureux pour sélectionner les principaux liens mais dans le cadre de ce tutoriel, nous allons seulement ceux dont l'intensité est supérieure à la moyenne. Dans `igraph`, supprimer des liens se fait avec la fonction `delete_edges(net, edges)` :

```
cut.off <- mean(links$weight)
net.sp <- delete_edges(net, E(net)[weight<cut.off])
plot(net.sp)
```



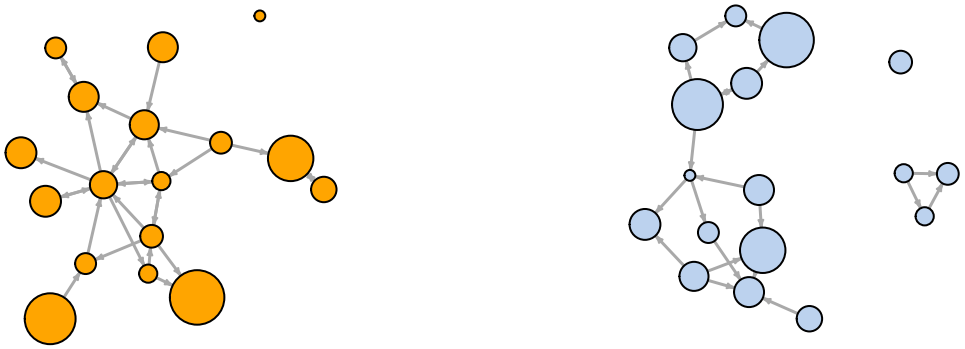
On peut choisir de représenter les liens de nature différente par des liens de couleurs différentes.

```
E(net)$width <- 1.5
plot(net, edge.color=c("dark red", "slategrey")[(E(net)$type=="hyperlink")+1],
      vertex.color="gray40", layout=layout_in_circle, edge.curved=.3)
```

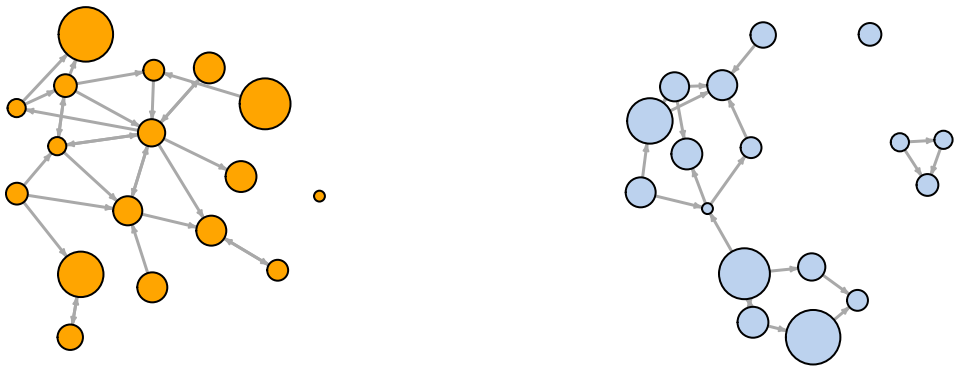


Une autre option est de représenter les deux types de liens (hyperliens et mentions) dans deux graphes différents.

```
net.m <- net - E(net)[E(net)$type=="hyperlink"] # autre moyen de sélectionner des liens
net.h <- net - E(net)[E(net)$type=="mention"]   # avec l'opérateur moins

# Deux types de liens, deux graphes différents
par(mfrow=c(1,2))
plot(net.h, vertex.color="orange")
plot(net.m, vertex.color="lightsteelblue2")
```



```
# Attribuer les mêmes coordonnées aux sommets facilite la comparaison
l <- layout_with_fr(net)
par(mfrow=c(1,2))
plot(net.h, vertex.color="orange", layout=l) # hyperliens
plot(net.m, vertex.color="lightsteelblue2")  # mentions
```

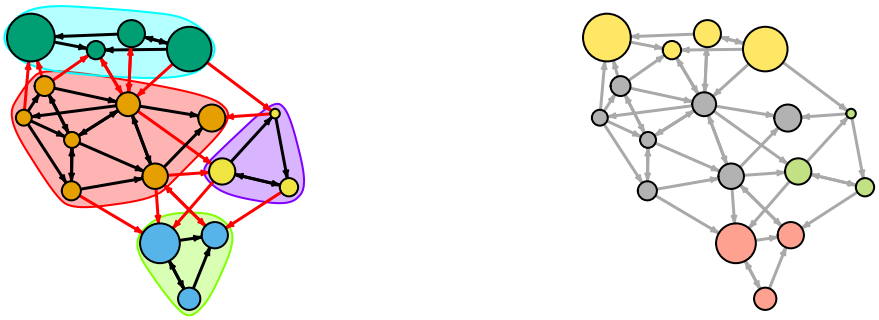


On peut également rendre la représentation plus lisible en montrant les communautés du graphe[2] :

```
par(mfrow=c(1,2))

# Détection de communautés basée sur la propagation des labels
clp <- cluster_label_prop(net)
class(clp)
```

```
## [1] "communities"
```

```
# La détection de communautés renvoie un objet de classe "communities"
# qu'igraph sait comment représenter
l <- layout_with_fr(net)
plot(clp, net, layout = l)

# On peut aussi colorer les sommets selon les communautés
V(net)$community <- clp$membership
colrs <- adjustcolor( c("gray50", "tomato", "gold", "yellowgreen"), alpha=.6)
plot(net, vertex.color=colrs[V(net)$community], layout=l)
```



```
dev.off()
```

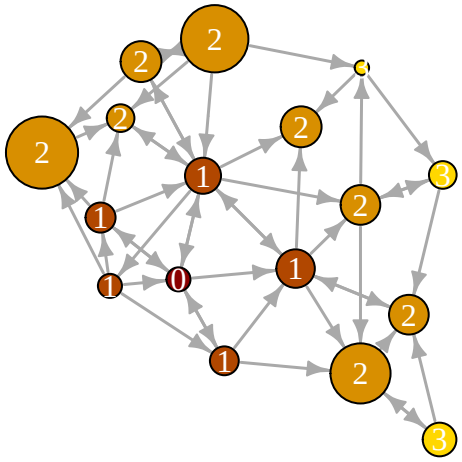
4.5 Mettre en évidence des sommets ou des liens spécifiques

Nous désirons parfois centrer la visualisation sur un sommet ou un ensemble particulier de sommets. Dans notre réseau de médias, nous pouvons examiner la diffusion d'information à partir de certains acteurs. Par exemple, représentons les distances à partir du *New York Times*.

[2]Petit rappel : 'les communautés du graphe est une formulation raccourcie de l'expression 'les communautés du graphe décelées par tel ou tel algorithme'. Comme tous les bons logiciels d'analyse de réseaux, 'igraph' propose plusieurs algorithmes de détection de communautés.

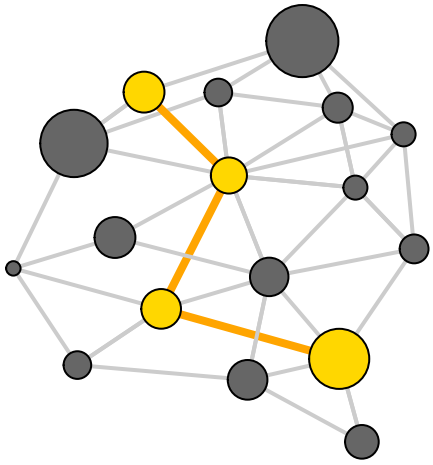
La fonction `distances` renvoie une matrice des plus courts chemins depuis le(s) sommet(s) listé(s) dans le paramètre `v` jusqu'aux sommets inclus dans le paramètre `to`.

```
dist.from.NYT <- distances(net, v=V(net)[media=="NY Times"],
                           to=V(net), weights=NA)

# Attribuer des couleurs aux distances
oranges <- colorRampPalette(c("dark red", "gold"))
col <- oranges(max(dist.from.NYT)+1)
col <- col[dist.from.NYT+1]

plot(net, vertex.color=col, vertex.label=dist.from.NYT, edge.arrow.size=.6,
     vertex.label.color="white")
```



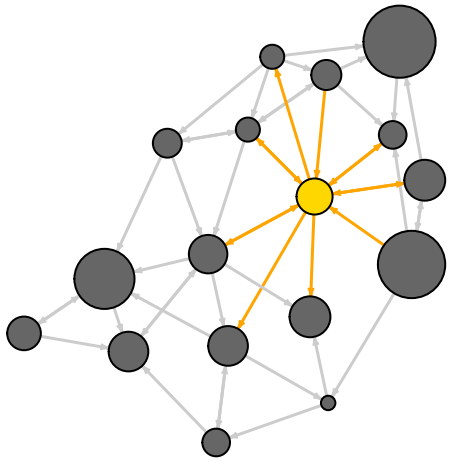
Il est également possible de mettre en évidence un chemin dans le réseau :

```
news.path <- shortest_paths(net,
                            from = V(net)[media=="MSNBC"],
                             to  = V(net)[media=="New York Post"],
                             output = "both") # both pour surligner les sommets
                                              # et les liens du chemin
# Générer une couleur de liens pour souligner le chemin
ecol <- rep("gray80", ecount(net))
ecol[unlist(news.path$epath)] <- "orange"
# Générer une épaisseur de liens pour le chemin
ew <- rep(2, ecount(net))
ew[unlist(news.path$epath)] <- 4
# Générer une couleur de sommet pour le chemin
vcol <- rep("gray40", vcount(net))
vcol[unlist(news.path$vpath)] <- "gold"

plot(net, vertex.color=vcol, edge.color=ecol,
     edge.width=ew, edge.arrow.mode=0)
```



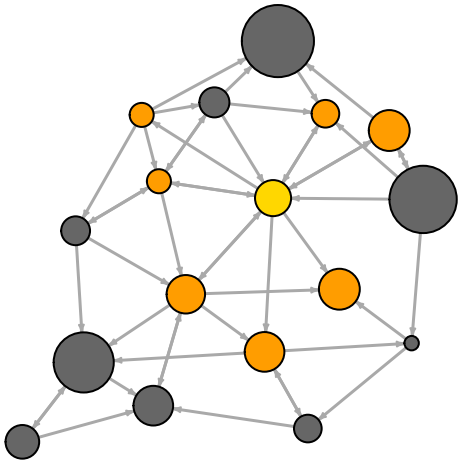
On peut mettre en évidence les liens entrants et/ou sortants d'un sommet, par exemple le *Wall Street Journal*. Pour un seul sommet, utiliser la fonction `incident()` ; pour plusieurs sommets, utiliser la fonction `incident_edges()`.

```
inc.edges <- incident(net,  V(net)[media=="Wall Street Journal"], mode="all")

# Choix de la couleur pour les liens sélectionnés
ecol <- rep("gray80", ecount(net))
ecol[inc.edges] <- "orange"
vcol <- rep("grey40", vcount(net))
vcol[V(net)$media=="Wall Street Journal"] <- "gold"
plot(net, vertex.color=vcol, edge.color=ecol)
```



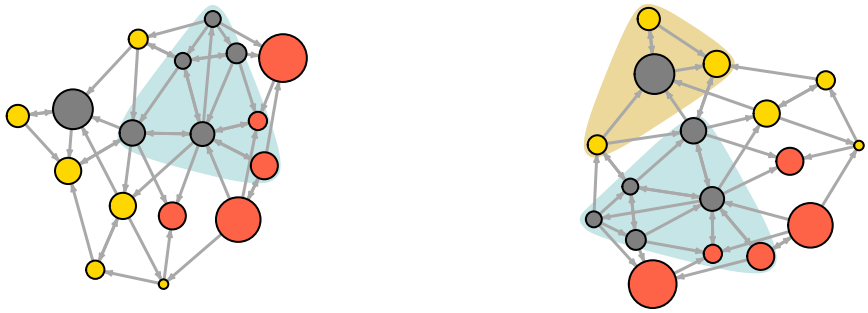
On peut également s'intéresser aux voisins d'ordre 1 d'un sommet. La fonction `neighbors` trouve l'ensemble des voisins d'ordre 1 d'un sommet donné. Pour trouver les voisins de plusieurs sommets, utiliser la fonction `adjacent_vertices()`. Pour les voisins d'ordre supérieur à 1, utiliser la fonction `ego()` avec le paramètre `order`.

```
neigh.nodes <- neighbors(net, V(net)[media=="Wall Street Journal"], mode="out")

# Choix de la couleur pour les voisins d'ordre 1 - degré sortant
vcol[neigh.nodes] <- "#ff9d00"
plot(net, vertex.color=vcol)
```



Un moyen d'attirer l'attention sur un groupe de sommets (nous l'avons vu avec les communautés) est de les entourer.

```
par(mfrow=c(1,2))
plot(net, mark.groups=c(1,4,5,8), mark.col="#C5E5E7", mark.border=NA)

# Entourer différents ensembles
plot(net, mark.groups=list(c(1,4,5,8), c(15:17)),
          mark.col=c("#C5E5E7","#ECD89A"), mark.border=NA)
```

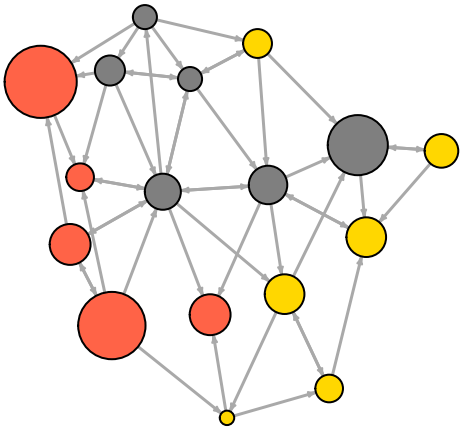


```
dev.off()
```

4.6 Visualisation interactive avec tkplot

R et `igraph` permettent la visualisation interactive des réseaux. Cela peut servir pour bidouiller légèrement la spatialisation d'un petit graphe. Après avoir fait les ajustements manuels, on peut récupérer les coordonnées des sommets et s'en resservir ensuite.

```
tkid <- tkplot(net)          # tkid est l'identifiant de ce que va ouvrir tkplot
l <- tkplot.getcoords(tkid) # récupérer les coordonnées depuis tkplot
plot(net, layout=l)
```

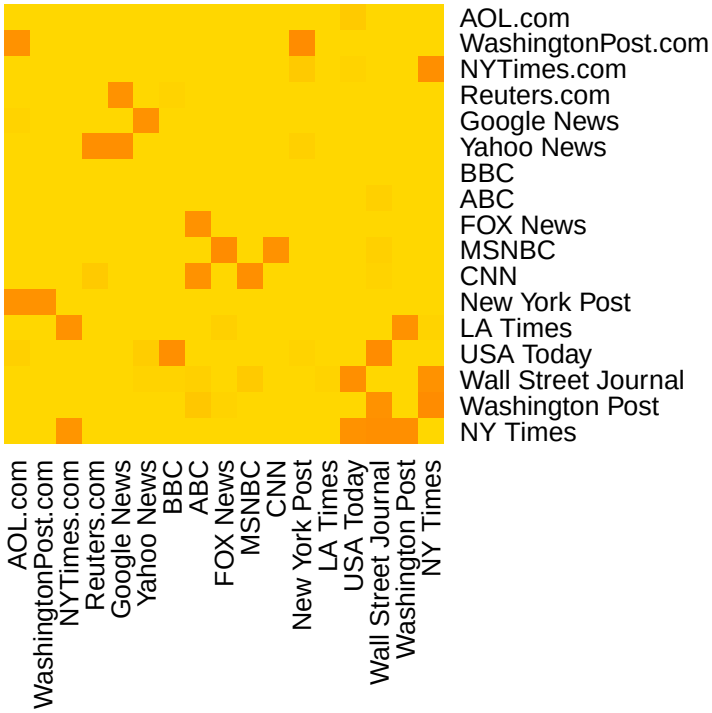


4.7 Autres modes de représentation

Il est utile de rappeller que d'autres modes de représentation des réseaux existent et que le plat de nouilles n'est pas la seule option.

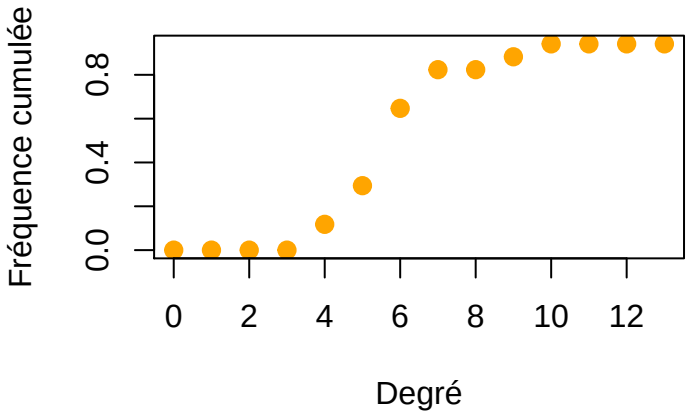
Voici par exemple la `heatmap` du réseau 2-mode.

```
netm <- as_adjacency_matrix(net, attr="weight", sparse=FALSE)
colnames(netm) <- V(net)$media
rownames(netm) <- V(net)$media

palf <- colorRampPalette(c("gold", "dark orange"))
heatmap(netm[,17:1], Rowv = NA, Colv = NA, col = palf(100),
        scale="none", margins=c(10,10))
```



En fonction des propriétés du réseau, des sommets ou des liens qui vous semblent essentielles, de simples graphiques peuvent être plus parlants qu'un graphe.

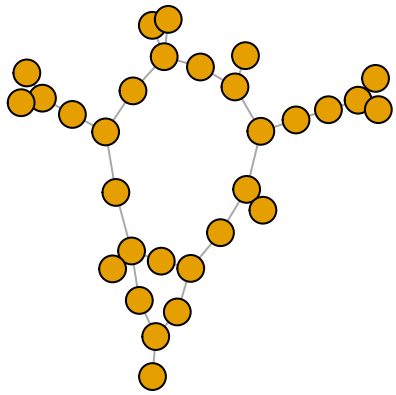
```
# Courbe de la distribution des degrés
deg.dist <- degree_distribution(net, cumulative=TRUE, mode="all")
plot( x=0:max(deg), y=1-deg.dist, pch=19, cex=1.2, col="orange",
      xlab="Degré", ylab="Fréquence cumulée")
```



4.8 Représenter les réseaux 2-mode

Notre deuxième jeu de données est un réseau biparti de liens entre des médias et des personnes.

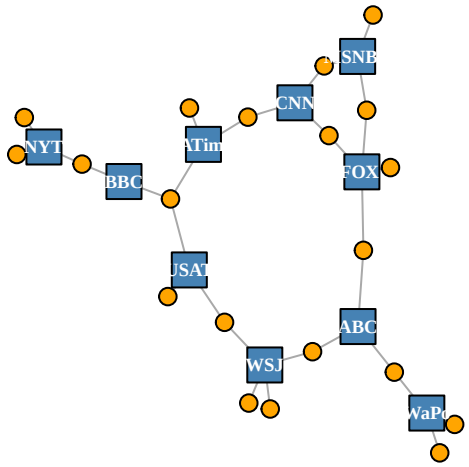
```
head(nodes2)
head(links2)

net2
plot(net2, vertex.label=NA)
```



Comme pour un réseau one-mode, nous pouvons modifier l'objet `igraph` pour ajouter des propriétés visuelles qui seront ensuite utilisées par défaut pour le représenter. Cette fois, nous allons également changer la forme des sommets: les médias seront des carrés et leurs consommateurs des ronds.

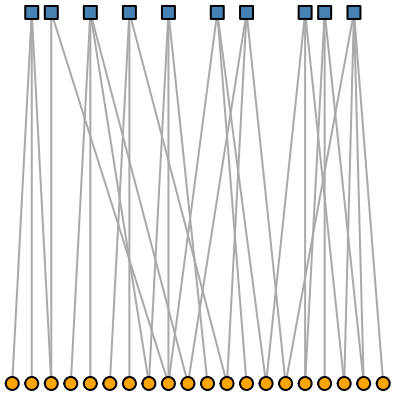
```
# Les médias sont des carrés bleus, l'audience des ronds orange
V(net2)$color <- c("steel blue", "orange")[V(net2)$type+1]
```

```
V(net2)$shape <- c("square", "circle")[V(net2)$type+1]

# On affiche seulement les labels des médias
V(net2)$label <- ""
V(net2)$label[V(net2)$type==F] <- nodes2$media[V(net2)$type==F]
V(net2)$label.cex=.6
V(net2)$label.font=2

plot(net2, vertex.label.color="white", vertex.size=(2-V(net2)$type)*8)
```



Dans `igraph`, il y a une spatialisation dédiée aux graphes bipartis[3].

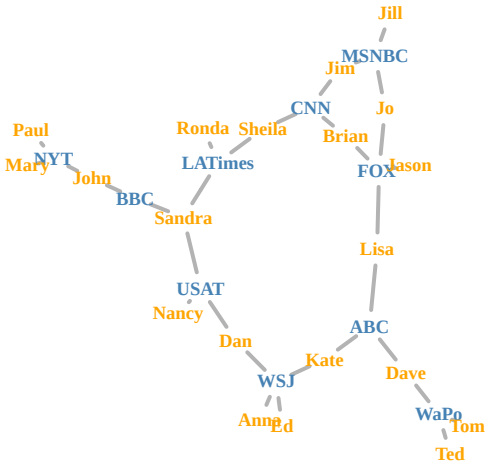
```
plot(net2, vertex.label=NA, vertex.size=7, layout=layout.bipartite)
```



Utiliser du texte à la place des sommets peut parfois être utile.

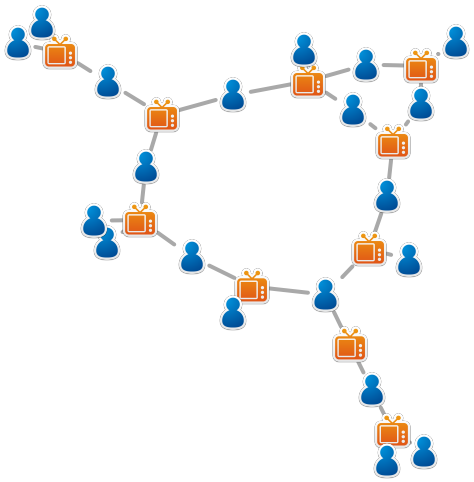
```
plot(net2, vertex.shape="none", vertex.label=nodes2$media,
     vertex.label.color=V(net2)$color, vertex.label.font=2,
     vertex.label.cex=.6, edge.color="gray70",  edge.width=2)
```

[3] *Elle obéit aux règles canoniques de représentation des réseaux 2-mode mais est d'une efficacité très discutable...*



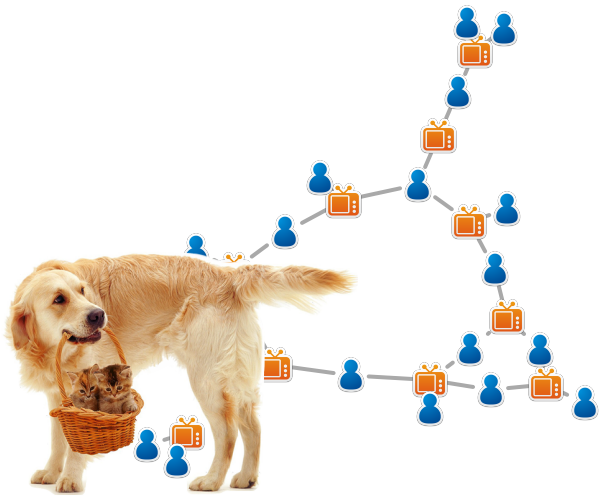
Dans cet exemple, nous allons également expérimenter l'utilisation d'images pour les sommets. Il est nécessaire d'installer le package `png` à l'aide de la fonction `install.packages('png')`.

```
#install.packages('png')
library(png)

img.1 <- readPNG("Data/Images/news.png")
img.2 <- readPNG("Data/Images/user.png")

V(net2)$raster <- list(img.1, img.2)[V(net2)$type+1]

plot(net2, vertex.shape="raster", vertex.label=NA,
     vertex.size=16, vertex.size2=16, edge.width=2)
```



Il est possible d'ajouter n'importe quelle image. Si vous ne pouvez vous empêcher d'ajouter des images d'animaux partout...

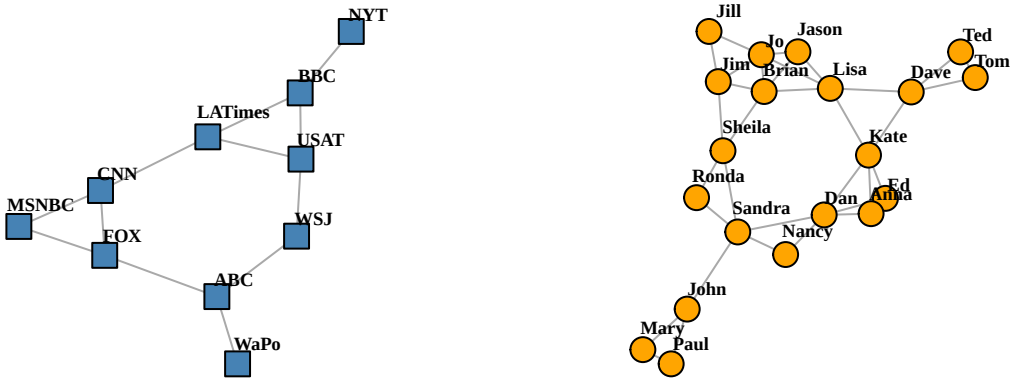
```
plot(net2, vertex.shape="raster", vertex.label=NA,
     vertex.size=16, vertex.size2=16, edge.width=2)

img.3 <- readPNG("Data/Images/puppy.png")
rasterImage(img.3,  xleft=-1.7, xright=0, ybottom=-1.2, ytop=0)
```



```
# Les nombres après l'image sont les coordonnées
# Les limites de votre figure s'obtiennent avec par()$usr
```

Nous pouvons également transformer le graphe biparti en deux graphes one-mode : ils s'obtiennent en multipliant la matrice du graphe 2-mode par sa transposée (et vice-versa) ou à l'aide de la fonction `bipartite.projection()`.

```
par(mfrow=c(1,2))

net2.bp <- bipartite.projection(net2)

plot(net2.bp$proj1, vertex.label.color="black", vertex.label.dist=1,
     vertex.label=nodes2$media[!is.na(nodes2$media.type)])

plot(net2.bp$proj2, vertex.label.color="black", vertex.label.dist=1,
     vertex.label=nodes2$media[ is.na(nodes2$media.type)])
```




```
dev.off()
```

C'est une bonne pratique de détacher les packages quand on n'en a plus besoin. C'est notamment le cas avec `igraph` et les packages statnet qui peuvent poser des problèmes s'ils sont chargés ensemble.

```
detach('package:png')
detach('package:igraph')
```

5 Exemple rapide avec le package network

Visualiser avec le package **network** est très proche de de que nous avons vu avec **igraph**. La syntaxe est légèrement différente, tout comme le nom des fonctions. Ce package permet moins l'incorporation de paramètres visuels dans l'objet **network** et demande plus de paramètres explicites dans la fonction de visualisation.

Voici un court exemple sur le réseau médiatique. Nous commençons par convertir les données en objet **network**, format utilisé par les packages Statnet (**network**, **sna**, **ergm**, **stergm**, etc.).

Comme dans **igraph**, nous pouvons créer un objet **network** à partir d'une liste de liens, d'une matrice d'adjacence ou d'une matrice d'incidence (voir **?edgeset.constructors**). Comme avec **igraph** ci-dessus, nous utilisons la liste de liens et le tableau des attributs des sommets pour créer l'objet **netwok**. Il faut faire attention au paramètre **ignore.eval** : il est TRUE par défaut et dans ce cas, le poids des liens n'est pas pris en compte.

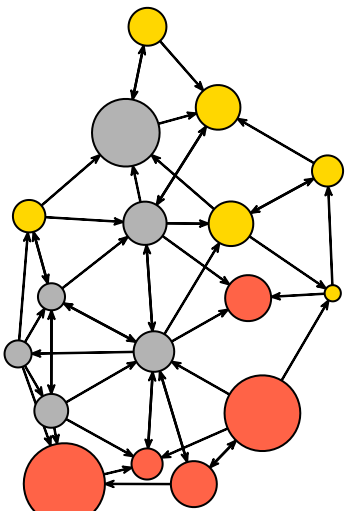
```
library('network')

net3 <- network(links,  vertex.attr=nodes, matrix.type="edgelist",
                loops=F, multiple=F, ignore.eval = F)
```

Ici aussi, il est possible facilement d'accéder aux sommets, aux liens et à la matrice d'adjacence.

```
net3[1:3, 1:3]
net3 %n% "net.name" <- "Media Network" # Attribut du réseau
net3 %v% "media"                       # Attribut du sommet
net3 %e% "type"                        # Attribut du lien
```

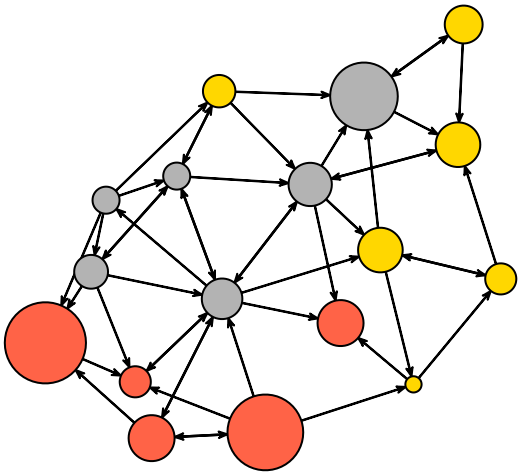
Visualisons ce réseau une fois encore.

```
net3 %v% "col" <- c("gray70", "tomato", "gold")[net3 %v% "media.type"]
plot(net3, vertex.cex=(net3 %v% "audience.size")/7, vertex.col="col")
```

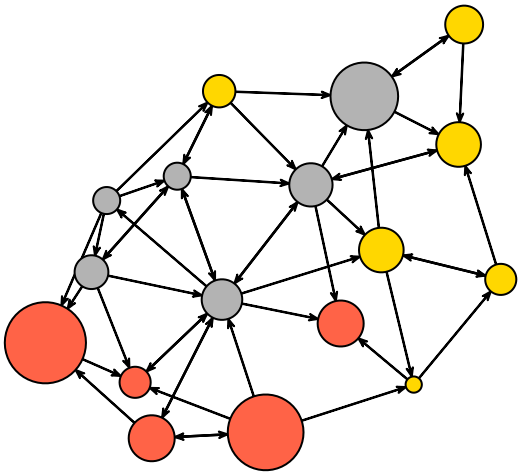


Comme dans `igraph`, les coordonnées des sommets peuvent être récupérées et réutilisées avec le paramètre `coord`.

```
l <- plot(net3, vertex.cex=(net3 %v% "audience.size")/7, vertex.col="col")
```



```
plot(net3, vertex.cex=(net3 %v% "audience.size")/7, vertex.col="col", coord=l)
```



Le package `network` permet également de réaliser des graphes interactifs à l'aide du paramètre `interactive = TRUE`.

```
plot(net3, vertex.cex=(net3 %v% "audience.size")/7, vertex.col="col", interactive=TRUE)
```

```
detach('package:network')
```

Pour connaître l'ensemble des paramètres disponibles, voir `?plot.network`.

6 Visualisations dynamiques et interactives (à venir)

Je n'ai pas encore traduit cette partie. La version anglaise est accessible ici

7 Cartographie de flux

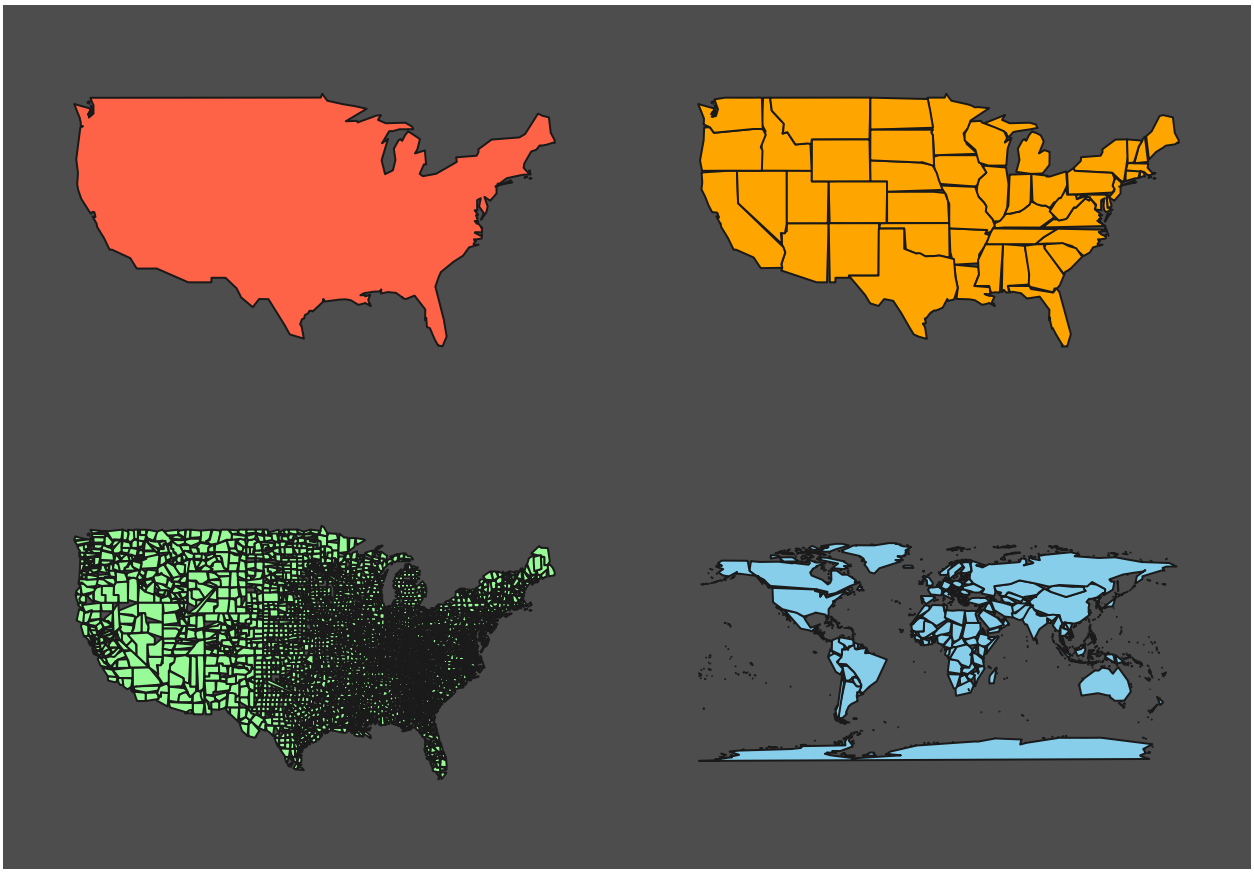
L'exemple présenté ici nécessite uniquement R base et deux packages de cartographie. Si vous connaissez ggplot2, il vous permet de faire à peu près la même chose. Le code utilisant la fonction `ggplot` sera similaire à ce qui suit, il faudra simplement utiliser `borders()` pour le fond de carte et `geom_path()` pour les liens.

Pour cartographier, quelques packages sont nécessaires : `maps` permet de visualiser le fond de carte, `geosphere` de représenter les liens.

```
#install.packages('maps')
#install.packages('geosphere')

library('maps')
library('geosphere')
```

Voici quelques exemples. Les paramètres principaux de la fonction `maps()` sont `col` pour la couleur de fond des unités spatiales, `border` pour la couleur des limites des unités spatiales et `bg` pour la couleur du fond.

```
par(mfrow = c(2,2), mar=c(0,0,0,0))

map("usa", col="tomato",  border="gray10", fill=TRUE, bg="gray30")
map("state", col="orange",  border="gray10", fill=TRUE, bg="gray30")
map("county", col="palegreen",  border="gray10", fill=TRUE, bg="gray30")
map("world", col="skyblue",  border="gray10", fill=TRUE, bg="gray30")
```



```
dev.off()
```

Les données utilisées concernent les vols aériens entre aéroports étatsuniens. Le fichier `airport` contient les coordonnées (latitude et longitude). Si elles ne sont pas dans vos données, vous pouvez les récuperer à partir d'une adresse à l'aide de la fonction `geocode()` du package ggmap.

```
airports <- read.csv("Data/Dataset3-Airlines-NODES.csv", header=TRUE)
flights <- read.csv("Data/Dataset3-Airlines-EDGES.csv", header=TRUE, as.is=TRUE)
head(flights, 4)
```

```
##   Source Target Freq
## 1      0    109   10
## 2      1     36   10
## 3      1     61   10
## 4      2    152   10
```

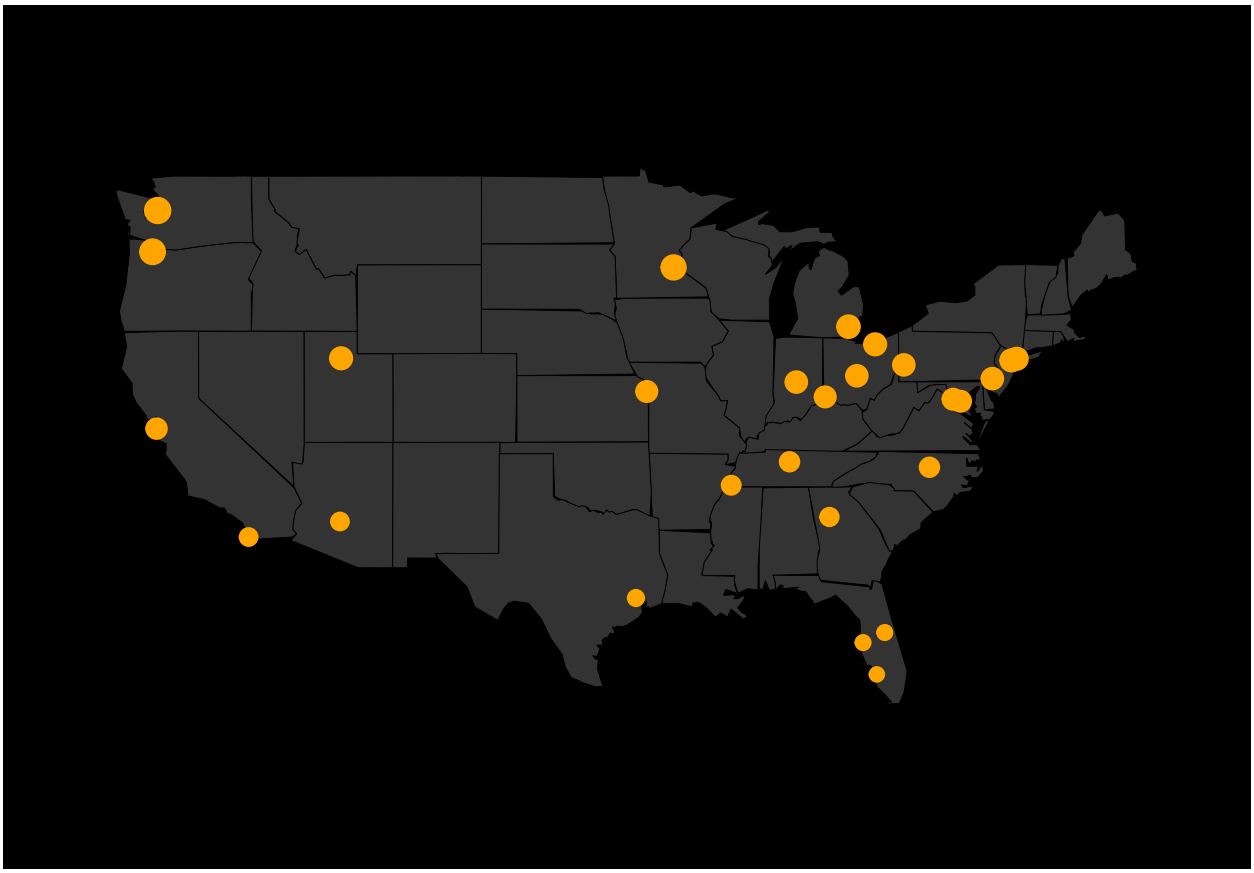
```
head(airports, 4)
```

```
##   ID                 Label Code             City latitude longitude ToFly
## 1  0   Adams Field Airport  LIT  Little Rock, AR 34.72944 -92.22444     0
## 2  1 Akron/canton Regional  CAK Akron/Canton, OH 40.91611 -81.44222     0
## 3  2  Albany International  ALB           Albany 42.73333 -73.80000     0
## 4  3             Albemarle  CHO  Charlottesville 38.13333 -78.45000     1
##   Visits
## 1    105
## 2    123
## 3    129
## 4    114
```

```
# Sélection des principaux aéroports (plus de 10 connections)
tab <- table(flights$Source)
big.id <- names(tab)[tab>10]
airports <- airports[airports$ID %in% big.id,]
flights  <- flights[flights$Source %in% big.id &
                    flights$Target %in% big.id, ]
```

Afin de créer notre figure, nous allons d'abord ajouter une carte des États-Unis. Puis nous ajouterons des points pour représenter les aéroports.

```
# Visualiser la carte des États-Unis
map("state", col="grey20", fill=TRUE, bg="black", lwd=0.1)

# Ajouter un point sur la carte pour chaque aéroport
points(x=airports$longitude, y=airports$latitude, pch=19,
       cex=airports$Visits/80, col="orange")
```



Puis nous générons un gradient de couleurs pour colorer les liens : plus le lien est fort, plus la couleur est lumineuse.

```
col.1 <- adjustcolor("orange red", alpha=0.4)
col.2 <- adjustcolor("orange", alpha=0.4)
edge.pal <- colorRampPalette(c(col.1, col.2), alpha = TRUE)
edge.col <- edge.pal(100)
```

Pour chaque vol dans nos données, nous utilisons `gcIntermediate()` pour obtenir les coordonnées de l'arc le plus court connectant le point de départ et le point d'arrivée. Puis nous projetons chaque arc sur la carte avec la fonction `lines()`.

```
for(i in 1:nrow(flights))  {
    node1 <- airports[airports$ID == flights[i,]$Source,]
    node2 <- airports[airports$ID == flights[i,]$Target,]

    arc <- gcIntermediate( c(node1[1,]$longitude, node1[1,]$latitude),
                           c(node2[1,]$longitude, node2[1,]$latitude),
                           n=1000, addStartEnd=TRUE)
    edge.ind <- round(100*flights[i,]$Freq / max(flights$Freq))

    lines(arc, col=edge.col[edge.ind], lwd=edge.ind/30)
}
```

Si vous cartographiez un réseau à l'échelle mondiale, certains liens sortiront à droite de l'écran pour entrer à gauche (liens transpacifiques pour une projection de Mercator centrée sur l'Europe par exemple). Afin d'éviter ça, on peut utiliser la fonction `greatCircle()`. Une autre option est de changer le mode de projection.