

# Graph Mining

L. Rouvière

*laurent.rouviere@univ-rennes2.fr*

OCTOBRE 2023

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Définitions - vocabulaire sur les graphes</b>	<b>4</b>
<b>3</b>	<b>Statistiques descriptives sur les graphes</b>	<b>7</b>
3.1	Caractéristiques générales . . . . .	7
3.2	Importance des nœuds . . . . .	10
<b>4</b>	<b>Modèles et construction de graphes</b>	<b>12</b>
4.1	Quelques modèles de graphes . . . . .	12
4.2	Construire un graphe . . . . .	15
<b>5</b>	<b>Détection de communautés</b>	<b>17</b>
5.1	L'edge betweenness . . . . .	17
5.2	La modularité . . . . .	20
5.3	Clustering spectral . . . . .	27
<b>6</b>	<b>Bibliographie</b>	<b>31</b>

## Présentation

- *Objectifs* : comprendre, décrire, interpréter les problèmes associés à des graphes.
- *Pré-requis* : théorie des probabilités, modélisation statistique, R (niveau avancé).
- *Enseignant* : Laurent Rouvière *laurent.rouviere@univ-rennes2.fr*
  - *MCF* à l'Université Rennes 2, *PCC* à l'Ecole Polytechnique.
  - *Recherche* : statistique non paramétrique, apprentissage statistique
  - *Enseignements* : statistique et probabilités (Université, école d'ingénieur et de commerce, formation continue).
  - *Consulting* : énergie, finance, marketing, sport.

## Programme

- *Matériel* : slides + tutoriel R (compléments de cours et exercices). Disponible à l'url : [https://lrouviere.github.io/page\\_perso/graphes.html](https://lrouviere.github.io/page_perso/graphes.html)
- *4 parties* :
  1. Définitions - vocabulaire sur les graphes
  2. Statistiques descriptives sur les graphes
  3. Construction de graphes - modèles de graphes
  4. Détection de communautés

## Table des matières

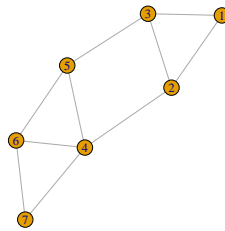
# 1 Introduction

- Le *graph mining* ou *fouille de graphes* correspond à la fouille de données spécifiques aux graphes.

## Graphe

Objet mathématique utilisé pour modéliser des **connexions ou interactions** entre **individus ou entités** :

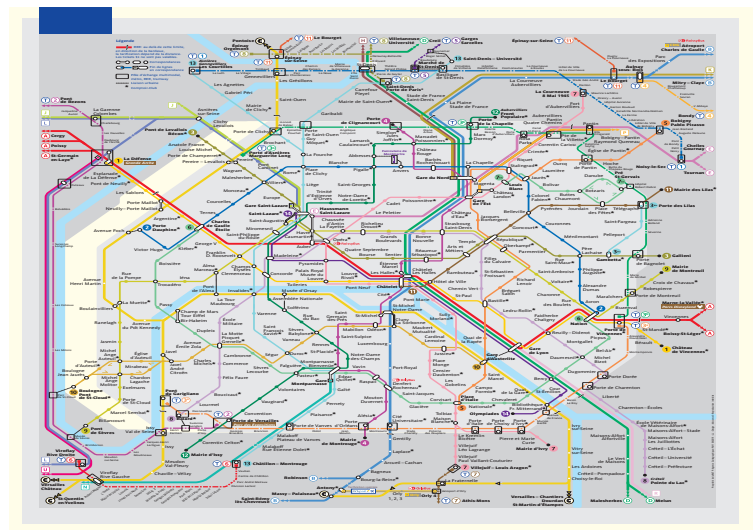
- les entités sont appelées *nœuds* ou *sommets* ;
- une relation entre deux entités est modélisée par une *arête*.



## Nombreuses applications

- *Réseaux routiers* entre villes, *réseaux aériens* entre aéroports...
- *Réseaux électriques* (cables reliant des prises)
- *Internet* (routeurs et ordinateurs connectés par ethernet ou wifi)
- *Réseaux d'amis* Facebook
- *Communication* : personnes avec qui on communique (téléphone par exemple)
- *World wide web* (les nœuds sont les pages internet et les arêtes sont les hyperliens)
- *Réseaux de régulation* entre gènes
- *Systèmes de recommandation*...

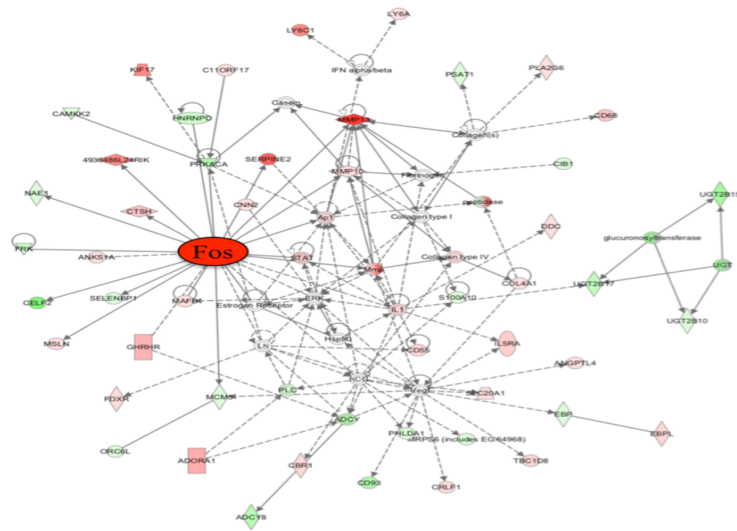
## Métro parisien



## Réseaux sociaux

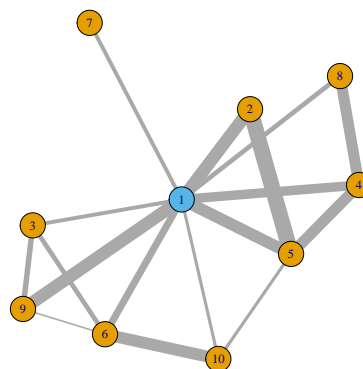


## Réseaux moléculaires



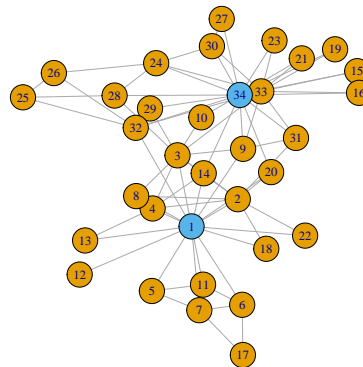
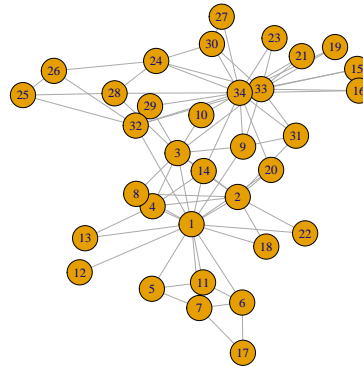
## Communications

— *Objectif* : visualiser les communications d'un individu sur une période donnée.



## Karate

- *Nœuds* : membres d'un club de karaté universitaire ;
- *Arêtes* : lien d'amitié calculé en fonction du nombre d'activités communes.



## Plusieurs problématiques

### Analyse exploratoire

Comprendre la *structure d'interaction* entre entités en analysant la topologie du graphe.

1. *Construction* d'un graphe : définition des *nœuds* et des *arêtes*.
2. *Visualisation* : comment représenter et dessiner un graphe ?
3. *Détection de communautés* : identifier des sous-groupes de nœuds très connectés.

### Inférence sur les graphes

Mettre des *lois de probabilités* sur la structure du graphe.

1. Modèles de *graphes aléatoires*.
2. *Prédiction de connexions* pour des *nouveaux nœuds*.

## 2 Définitions - vocabulaire sur les graphes

### Graphe

Un *graphe*  $G = (V, E)$  est composé :

- d'un ensemble  $V$  de **nœuds ou sommets** (vertices) qui représentent les individus ou entités qui interagissent entre eux,
- et d'un ensemble  $E$  **d'arêtes** (edges) qui indiquent la présence d'une **interaction** ou **connexion entre deux nœuds** :

$$\{i, j\} \in E \text{ si il y a une arête entre } i \text{ et } j \text{ dans } G.$$

## Définitions

- Le nombre de nœuds  $|V|$  est l'**ordre** du graphe. Le nombre d'arêtes  $|E|$  est la **taille** du graphe.
- Un graphe est **dirigé** (ou **orienté**) lorsque ses arêtes le sont. Il est **non dirigé** sinon.
- Les graphes peuvent être **binaires** (arête présente ou absente), ou **valués** (arêtes munies d'un poids positif).
- Un graphe est dit **simple** s'il n'y a pas de boucles :  $(i, i)$  n'est jamais une arête.
- Le graphe **complet** ou **clique** est le graphe (non dirigé) qui contient toutes les arêtes possibles entre les sommets ( $C_{|V|}^2$  arêtes).

## Question

Comment **stocker** un graphe ?

## Matrice d'adjacence

### Définition

La **matrice d'adjacence** d'un graphe  $G = (V, E)$  binaire est la matrice  $|V| \times |V|$  de terme général

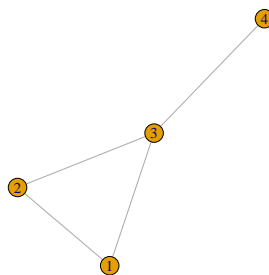
$$A_{ij} = \begin{cases} 1 & \text{si } \{i, j\} \in E \\ 0 & \text{sinon.} \end{cases}$$

### Remarques

- Graphe **non dirigé**  $\implies A$  symétrique.
- Graphe **simple**  $\implies A_{ii} = 0 \ \forall i$ .
- Graphe **valué**  $\implies A_{ij} = w_{ij} \in \mathbb{R}^+$ .

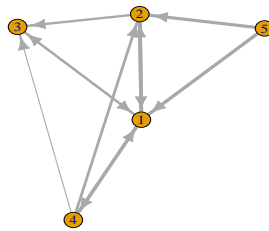
## Exemple : graphe non dirigé

```
> A
##      [,1] [,2] [,3] [,4]
## [1,]    0    1    1    0
## [2,]    1    0    1    0
## [3,]    1    1    0    1
## [4,]    0    0    1    0
> set.seed(1234)
> G <- graph_from_adjacency_matrix(A,
+                               mode='undirected')
> plot(G)
```



## Exemple : graphe dirigé

```
> A
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    4    3    2    0
## [2,]    4    0    3    0    0
## [3,]    1    0    0    0    0
## [4,]    4    3    1    0    0
## [5,]    4    4    0    0    0
> G <- graph_from_adjacency_matrix(A,
+   mode='directed',weighted = TRUE)
> E(G)$width <- E(G)$weight
> plot(G)
```



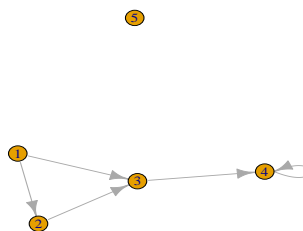
### Remarque

- Pas toujours efficace en terme de stockage :  $O(|V|^2)$ .
- Utiliser des matrices *sparses* si le graphe est très creux.

### Liste d'arêtes

- Il est souvent plus efficace de définir le graphe en donnant une liste d'arêtes.
- *Attention* : penser à donner le *nombre total de nœuds du graphe* pour éviter d'oublier les *nœuds isolés*.

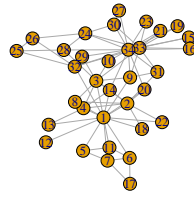
```
> G <- graph(edges=c(1,2,1,3,3,4,4,4,2,3),n=5)
> plot(G)
```



## Visualisation d'un graphe

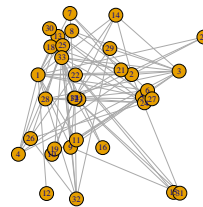
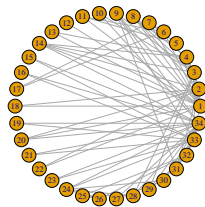
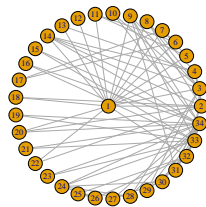
- *Etape importante* : elle permet de comprendre la *structure du graphe* : identification de nœuds importants, très connectés...
- *Différentes représentations* :
  1. en cercle, en étoile...
  2. selon différents algorithmes (voir [Bahoken et al., 2013])mais *attention* : certaines peuvent être *trompeuses* :

```
> plot(kar)
```



## Autres représentations

```
> plot(kar,layout=layout_as_star(kar))  
> plot(kar,layout=layout_circle(kar))  
> plot(kar,layout=layout_randomly(kar))
```



## Packages

- **R** : igraph, visNetwork, GGally.
- **Python** : NetworkX.

## 3 Statistiques descriptives sur les graphes

### Caractéristiques d'un graphe

- De *nombreux indicateurs* permettent de décrire un graphe.

### Objectifs

- Donner une *version résumée* du graphe.
- Décrire et comprendre les *interactions entre entités* :
  - transfert d'information entre deux sommets.
  - importance de certains sommets.
  - sous-structure particulière dans le graphe.
- **Comparer** deux graphes.
- **Comparer** un graphe avec un *modèle de graphe aléatoire*.

### 3.1 Caractéristiques générales

#### Distance - diamètre

- Un *chemin* entre  $i \in V$  et  $j \in V$  est une suite de nœuds et d'arêtes permettant de relier  $i$  et  $j$ .
- *Longueur d'un chemin* entre  $i$  et  $j$  : nombre d'arêtes qui composent ce chemin.
- *Distance  $\ell_{ij}$  entre  $i$  et  $j$*  : longueur du plus court chemin qui les relie. Si les deux nœuds ne sont pas connectés :  $\ell_{ij} = +\infty$ .

- **Diamètre d'un graphe** : plus grande distance entre deux nœuds (quantité définie uniquement pour les graphes connexes).

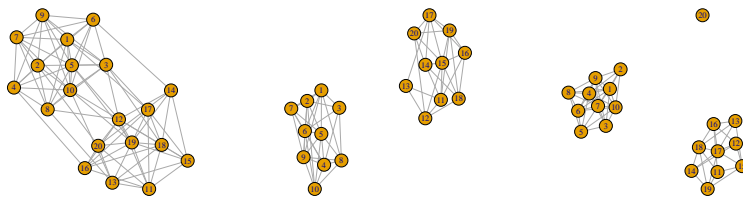
### Transfert d'information

Un **petit diamètre** indique que l'information circule rapidement dans le graphe entier.

### Connexité

- Une **composante connexe** est un sous-ensemble  $C = \{v_1, \dots, v_k\} \subset V$  tel que, pour tout  $v_i, v_j \in C$ , il existe un chemin dans  $G$  de  $v_i$  à  $v_j$ .
- Un nœud qui n'est connecté à aucun autre est dit **isolé**  $\implies$  il forme une composante connexe à lui tout seul.
- Un graphe est dit **connexe** s'il possède une **unique composante connexe**.

### Exemple



### Commentaires

- **Gauche** : graphe connexe.
- **Centre** : 2 composantes connexes.
- **Droite** : 3 composantes connexes, 1 nœud isolé.

### Questions liées à la connexité

1. **Plusieurs composantes connexes** dans le graphe ?
  - présence de **nœuds isolés** (participant inactif) ?
  - 1 composante connexe = **1 groupe d'individus** (ou 1 **cluster**) ?
2. Si non, est-ce "**presque**" le cas ?  $\implies$  Création de nouvelles composantes connexes en supprimant quelques nœuds ou arêtes, voir section 3.2 .
3. Recherche de **communautés** dans les graphes connexes : groupes de sommets **très connectés entre eux** et **peu connectés avec les autres groupes**, voir section 5.

### Densité

- Un graphe simple possède au plus  $|V|(|V| - 1)/2$  arêtes si il est non dirigé et  $|V|(|V| - 1)$  si il est dirigé.

### Densité

- **Graphe non dirigé** :

$$\text{den}(G) = \frac{|E|}{|V|(|V| - 1)/2}.$$

- **Graphe dirigé** :

$$\text{den}(G) = \frac{|E|}{|V|(|V| - 1)}.$$

### Remarque

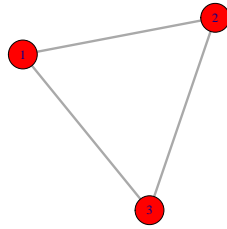
Varie entre 0 (**graphe vide**) et 1 (**graphe complet ou clique**).



## Densité locale

Recherche de *motifs particuliers* dans le graphe.

- Nombre de *triangles* dans un graphe : traduit des relations de *transitivité*

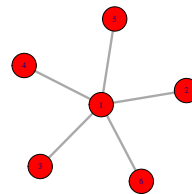
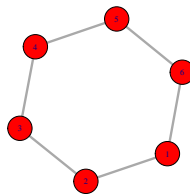
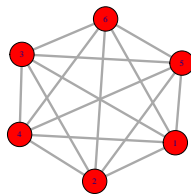


## Intérêt

*Réseaux sociaux* : mes amis sont-ils amis ?

## Autres motifs

1. Cliques de taille  $k$  donnée.
2. Cycles de longueur  $k$  donnée.
3.  $k$  stars...



## Exemple

Comparer le nombre d'occurrences d'un motif à un nombre attendu ou observé.

## Le coin R

La plupart des indicateurs s'obtiennent directement avec *R*

- *Nombre* de nœuds, d'arêtes, composantes connexes, *diamètre*, *densité* :

```
> vcount(kar)
## [1] 34
> ecount(kar)
## [1] 78
> count_components(kar)
## [1] 1
> diameter(kar)
## [1] 5
> edge_density(kar)
## [1] 0.1390374
```

- Nombre de *triangles* :

```
> head(count_triangles(kar))
## [1] 18 12 11 10 2 3
> length(count_triangles(kar))/3
## [1] 45
```

- Nombre de *cliques* de taille 3 à 5 :

```
> count_max_cliques(kar,min=3,max=5)
## [1] 25
```

## 3.2 Importance des nœuds

### Objectif

- Identifier les **nœuds centraux** d'un réseau.
- Rechercher les **nœuds influents**, clés d'un réseau.

### Comment ?

En définissant des *indicateurs de centralité* pour les nœuds d'un graphe.

### Voisins, degré

- Les *voisins* de  $i \in V$  sont les nœuds  $j \in V$  tels que  $i, j \in E$ . On note

$$\mathcal{V}(i) = \{j \in V : \{i, j\} \in E\}.$$

### Degré

- Le **degré**  $d_i$  d'un nœud  $i$  est le nombre de voisins de  $i$  :  $d_i = |\mathcal{V}(i)|$ .
- **Calcul** à partir de la matrice d'adjacence  $A$  :
  - *Graphe non dirigé* :  $d_i = \sum_{j, j \neq i} A_{ij}$ .
  - *Graphe dirigé* : degrés **sortant** et **entrant**

$$d_i^{\text{out}} = \sum_{j, j \neq i} A_{ij} \quad \text{et} \quad d_i^{\text{in}} = \sum_{j, j \neq i} A_{ji}.$$

### Remarque

Notion la plus simple mais qui ne prend *pas nécessairement en compte la structure du graphe*.

### Degré moyen

- *Graphe non dirigé* :  $\bar{d} = \frac{1}{|V|} \sum_{i \in V} d_i$ .
- *Graphe dirigé* :

$$\bar{d}^{\text{out}} = \frac{1}{|V|} \sum_{i \in V} d_i^{\text{out}} \quad \text{et} \quad \bar{d}^{\text{in}} = \frac{1}{|V|} \sum_{i \in V} d_i^{\text{in}}.$$

### Remarque

- **Pas toujours très informatif** car souvent grande variabilité.
- **plus intéressant** : distribution des degrés (avec un histogramme par exemple).

### Centralité de proximité

- **Objectif** : étudier si le sommet est à **proximité** des autres sommets et si il peut **interagir rapidement** avec eux.
- **Idée** : regarder la distance entre le sommet et les autres sommets.

### Définition

Le *degré de centralité de proximité* (closeness centrality) du nœud  $i$  est défini par

$$C_c(i) = \frac{1}{\sum_{j \neq i} \ell_{ij}}$$

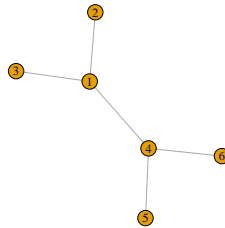
où  $\ell_{ij}$  distance = longueur du plus court chemin entre  $i$  et  $j$ .

### Commentaire

$C_c(i) \nearrow$  si sa **distance** aux autres nœuds est **faible**.

## Exemple

nœud $j$	$\ell_{1j}$	$\ell_{2j}$	nœud $j$	$\ell_{1j}$	$\ell_{2j}$	nœud $j$	$\ell_{1j}$	$\ell_{2j}$
1			1			1		1
2			2	1		2	1	
3			3	1		3	1	2
4			4	1		4	1	2
5			5	2		5	2	3
6			6	2		6	2	3



- *Conclusion* :  $C_c(1) = 1/7$ ,  $C_c(2) = 1/11$ .

```
> closeness(G)
## [1] 0.14285714 0.09090909 0.09090909 0.14285714 0.09090909 0.09090909
```

## Centralité d'intermédierité

*Objectif* : mesurer à quel point

- un nœud est important pour *connecter deux autres nœuds* dans le graphe.
- un nœud sert *d'intermédiaire*.

### Définition

Le *degré de centralité d'intermédierité* (betweenness centrality) du nœud  $i$  est défini par

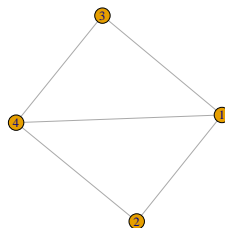
$$C_B(i) = \sum_{j \neq i, k \neq i, j \neq k} \frac{g_{jk}(i)}{g_{jk}}$$

où

- $g_{jk}$  est le *nombre de plus courts chemins* entre  $j$  et  $k$ .
- $g_{jk}(i)$  est le *nombre de plus courts chemins* entre  $j$  et  $k$  qui *passent par  $i$* .

## Exemple

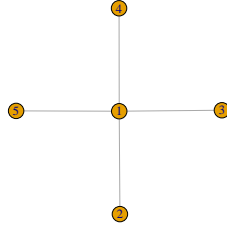
$(j, k)$	$g_{jk}$	$g_{jk}(1)$	$(j, k)$	$g_{jk}$	$g_{jk}(1)$	$(j, k)$	$g_{jk}$	$g_{jk}(1)$
(2, 3)			(2, 3)	2		(2, 3)	2	1
(2, 4)			(2, 4)	1		(2, 4)	1	0
(3, 4)			(3, 4)	1		(3, 4)	0	0



— *Conclusion* :  $C_B(1) = 1/2$ .

```
> betweenness(G1)
## [1] 0.5 0.0 0.0 0.0 0.5
```

Autre exemple : graphe étoilé



```
> betweenness(G2)
## [1] 6 0 0 0 0 0
```

### Conclusion

On retrouve bien que *seul le nœud 1 sert d'intermédiaire*.

### Commentaires

- Un des concepts les plus *importants*.
- $C_b(i) \nearrow$  s'il est *point de passage sur un grand nombre de chemins* entre deux nœuds.
- Mesure de l'utilité du nœud dans la *communication* et le *transfert d'information* dans le graphe.

### Le coin R

Là encore les différents *degrés d'importance* des nœuds s'obtiennent directement avec *R* :

— Degrés et degré moyen :

```
> degree(kar) %>% head()
## [1] 16 9 10 6 3 4
> degree(kar) %>% mean()
## [1] 4.588235
```

— Degrés de proximité et d'intermédierité :

```
> closeness(kar) %>% head() %>% round(3)
## [1] 0.017 0.015 0.017 0.014 0.011 0.012
> betweenness(kar) %>% head() %>% round(3)
## [1] 231.071 28.479 75.851 6.288 0.333 15.833
```

## 4 Modèles et construction de graphes

### 4.1 Quelques modèles de graphes

#### Pourquoi des modèles sur les graphes ?

- *Rappel* : un graphe  $G = (V, E)$ .
- *Modéliser* : trouver des *lois de probabilités* sur la *distribution* des *nœuds* et/ou sur celle des *arêtes*.

But

1. *Générer* des graphes "réalistes".
2. *Comprendre* un graphe réel en *ajustant un (bon) modèle* dessus.
3. Détecter des *communautés* ou *clusters* de nœuds.
4. Faire de la *prévision*...

## Modèle d'Erdős et Rényi

- Modèle le plus *simple*, introduit dans les années 1950.
- En général pour des graphes *non dirigés*.
- *Modèle à deux paramètres* :
  - $n \in \mathbb{N}^*$  : nombre de nœuds ;
  - $p \in [0, 1]$  : probabilité de connexion entre 2 nœuds.

### Graphe d'Erdős et Rényi

Graphe à  $n$  nœuds où la probabilité qu'il y ait une arête entre 2 nœuds est une loi de Bernoulli  $B(p)$ .

### Propriété (évidente)

La variable aléatoire  $D_i$  donnant le *degré du nœud  $i$*  dans un graphe  $G(n, p)$  suit une loi binomiale  $\mathcal{B}(n-1, p)$ .

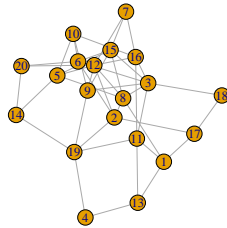
- La loi binomiale est une loi à *queue légère* (peu de valeurs extrêmes).
- Or, très souvent, dans les réseaux réels, la distribution des degrés est plutôt à *queue lourde* : un petit nombre de nœuds ont un degré élevé.

⇒ modèle souvent un peu *trop simple* pour des *graphes réels*.

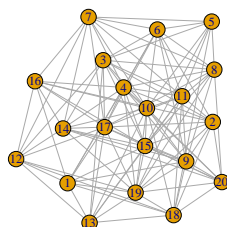
## Le coin R

- On peut générer des graphes  $G(n, p)$  avec la fonction *sample\_gnp*.

```
> set.seed(1234)
> G1 <- sample_gnp(n=20, p=0.2)
> plot(G1)
```



```
> set.seed(1234)
> G2 <- sample_gnp(n=20, p=0.6)
> plot(G2)
```



## Stochastic Bloc Model (SBM)

### Idée

- Existence de **groupes de nœuds**.
- **Fortes connexions** entre les nœuds d'un **même groupe**.
- **Faibles connexions** entre les nœuds de **groupes différents**.

### Modélisation

Représenter ces connexions à l'aide d'une **matrice**  $K \times K$  où  $K$  est le nombre de groupes.

### Graphe SBM

- Paramètres :  $K$  nombre de groupes,  $n$  nombre de nœuds,  $\pi = (\pi_1, \dots, \pi_K)$ ,  $\Lambda = (\lambda_{k\ell})_{1 \leq k, \ell \leq K}$  matrice  $K \times K$ .
- $Z_1, \dots, Z_n$  i.i.d. avec  $Z_i \sim \mathcal{M}(n, \pi)$  (variable **latente** qui représente le **groupe du nœud**  $i$ ).
- Une arête entre les nœuds  $i$  et  $j$  est représentée par une variable aléatoire  $A_{ij}$  telle que

$$A_{ij} | (Z_i = k, Z_j = \ell) \sim B(\lambda_{k\ell}).$$

### Interprétation

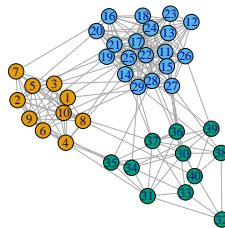
La **distribution des arêtes** entre

- les nœuds d'un même groupe est la même.
- les nœuds de deux groupes différents est identique aussi.

### Exemple

On peut générer des **graphes SBM** avec la fonction **sample\_sbm**.

```
> set.seed(1234)
> n <- 40
> eff <- rmultinom(n=40, size=1, prob=c(0.3, 0.4, 0.3)) %>% apply(1, sum)
> eff
## [1] 10 19 11
> bern.mat <- matrix(c(0.9, 0.05, 0.05, 0.05, 0.8, 0.05, 0.05, 0.05, 0.7), ncol=3)
> G1 <- sample_sbm(n, bern.mat, eff)
> gr <- rep(1:3, eff)
> plot(G1, vertex.color=gr)
```



## SBM et détection de communautés

### Idée

Utiliser les SBM pour **détecter des communautés** (alternative aux techniques qui seront vues en section 5) en

- **estimant le nombre de groupes** d'un SBM ;
- **estimant les groupes de chaque nœud**.
- Il existe différentes approches pour estimer ces quantités, notamment basées sur les **modèles de mélange** (voir [Daudin et al., 2008]).
- Par exemple les méthodes **VEM** (approche variationnelle de l'algorithme EM) pour  $\pi$  et  $\Lambda$ .
- et l'**ICL** (Integrated classification likelihood) pour le **nombre de groupes**  $K$ .
- Sur **R**, on pourra utiliser **BM\_bernoulli** du package **blockmodels**.

## 4.2 Construire un graphe

- Dans de nombreuses applications où on souhaite avoir une approche par graphe, ce dernier peut *ne pas être spécifié*.
- On dispose, de manière classique, d'individus  $x_1, \dots, x_n$  avec  $x_i \in \mathbb{R}^p$ .

### Objectifs

Définir un **graphe**  $G = (V, E)$  où

1. les sommets sont les individus  $V = \{1, \dots, n\}$ ;
2. les arêtes sont définies à partir de la **proximité** ou la **similarité** entre les individus.

### Un exemple

- On considère un sous-échantillon des *iris de Fisher*.

```
> data(iris)
> set.seed(12345)
> donnees <- iris[sample(nrow(iris), 30),]
> head(donnees)
##      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 142             6.9         3.1         5.1         2.3 virginica
## 51              7.0         3.2         4.7         1.4 versicolor
## 58              4.9         2.4         3.3         1.0 versicolor
## 93              5.8         2.6         4.0         1.2 versicolor
## 75              6.4         2.9         4.3         1.3 versicolor
## 96              5.7         3.0         4.2         1.2 versicolor
```

### Objectif

Construire un **graphe** en utilisant **uniquement les variables continues**.

### Matrice de distance ou similarité

- Les arêtes vont être définies à partir de la *proximité-similarité* entre individus.
- Il faut donc définir une **matrice de distance**  $D = (d_{ij})$ ,  $1 \leq i, j \leq n$  tel que  $d_{ij}$  est *petit* si  $i$  et  $j$  sont *proches*
- ou une **matrice de similarité**  $S = (s_{ij})$ ,  $1 \leq i, j \leq n$  tel que  $s_{ij}$  est *grand* si  $i$  et  $j$  sont *similaires*

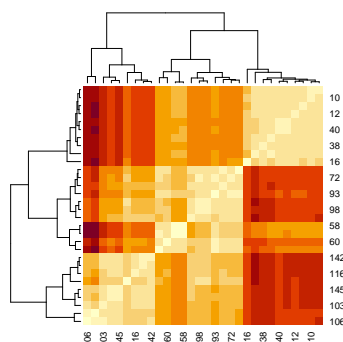
### Exemple

- Sur les iris on peut calculer la *distance euclidienne* entre iris sur les 4 variables quantitatives.

```
> D <- as.matrix(dist(donnees[, -5]))
```

- Que l'on peut visualiser à l'aide d'un *heatmap* :

```
> heatmap(D)
```



## Neighborhood graph

- Une fois  $D$  ou  $S$  calculée, on définit des *arêtes* en fonction de la *proximité entre individus*.

### $\varepsilon$ -neighborhood graph

Soit  $\varepsilon > 0$ , on appelle  $\varepsilon$ -neighborhood graph le graphe associé à la matrice d'adjacence

$$A_{ij} = \begin{cases} 1 & \text{si } d_{ij} \leq \varepsilon \\ 0 & \text{sinon.} \end{cases}$$

### Choix de $\varepsilon$

- Il influence le *nombre d'arêtes* :  $\varepsilon \nearrow \implies |E| \nearrow$ .
- A faire en fonction de l'*analyse du graphe* (nombre de communautés par exemple...).

## Graphe par plus proches voisins

- *Idée* : définir une arête entre  $i$  et  $j$  si  $i$  appartient aux kppv de  $j$  et/ou  $j$  appartient aux kppv de  $i$ .

### Graphes de plus proches voisins.

Soit  $k \leq n$ .

- *graphe des  $k$  plus proches voisins* : matrice d'adjacence

$$A_{ij} = \begin{cases} 1 & \text{si } i \text{ est parmi les kppv de } j \text{ ou } j \text{ est parmi les kppv de } i \\ 0 & \text{sinon.} \end{cases}$$

- *graphe des  $k$  plus proches voisins mutuels* : matrice d'adjacence

$$A_{ij} = \begin{cases} 1 & \text{si } i \text{ est parmi les kppv de } j \text{ et } j \text{ est parmi les kppv de } i \\ 0 & \text{sinon.} \end{cases}$$

## Remarques

- Définis ainsi les graphes par  $k$  ppv sont *non orientés* mais il est facile d'obtenir un *graphe orienté* pour les  $k$  plus proches voisins (*non mutuels*).
- *Choix de  $k$*  : il influence encore le nombre d'arêtes du graphe  $k \nearrow \implies |E| \nearrow$ .
- Sur *R*, on peut utiliser la fonction *nng* du package *cccd*.

## Le coin R

```
> library(cccd)
> gppv2 <- as.undirected(nng(dx=D,k=2,mutual=FALSE))
> gppv20 <- as.undirected(nng(dx=D,k=20,mutual=FALSE))
> plot(gppv2)
> plot(gppv20)
> ecount(gppv2)
## [1] 44
> ecount(gppv20)
## [1] 354
```

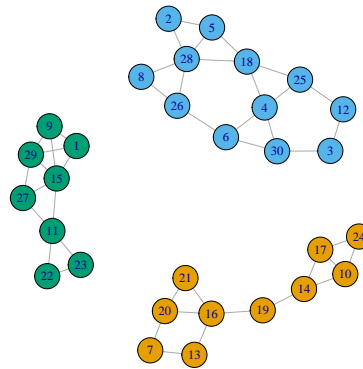




### Remarque

Le graphe à 2 plus proches voisins permet d'identifier parfaitement les espèces.

```
> gppv2_bis <- gppv2
> V(gppv2_bis)$color <- donnees[,5]
> plot(gppv2_bis)
```



## 5 Détection de communautés

### But

- Partitionner les nœuds du graphe en un nombre fini de groupes.
- Trouver des groupes de nœuds homogènes, des individus au comportement similaire.

### Idéal

- Beaucoup de connexions entre les nœuds d'une même communauté.
- Peu de connexions entre les nœuds de communautés différentes.

### Remarque

Thème très proche du *clustering*.

### Comment ?

*Différentes méthodes pour détecter des communautés :*

- techniques basées sur la modularité.
- *clustering spectral*.
- méthodes probabilistes (modèles SBM).
- ...

### 5.1 L'edge betweenness

#### Idée

- Définir l'équivalent du *node betweenness* pour les arêtes.
- Identifier des arêtes qui relient des "groupes", puis les supprimer.
- Critère élevé pour des arêtes qui relient des groupes.

## Définition

L'*edge betweenness* d'une arête  $e$  est défini par

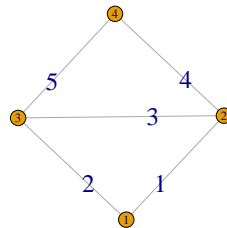
$$E_B(e) = \sum_{j,k,j \neq k} \frac{g_{jk}(e)}{g_{jk}}$$

où

- $g_{jk}$  est le *nombre de plus courts chemins* entre  $j$  et  $k$ .
- $g_{jk}(e)$  est le *nombre de plus courts chemins* entre  $j$  et  $k$  qui *empruntent l'arête  $e$* .

## Exemple

$(j, k)$	$g_{jk}$	$g_{jk}(1)$	$(j, k)$	$g_{jk}$	$g_{jk}(1)$	$(j, k)$	$g_{jk}$	$g_{jk}(1)$
(1, 2)			(1, 2)	1		(1, 2)	1	1
(1, 3)			(1, 3)	1		(1, 3)	1	0
(1, 4)			(1, 4)	2		(1, 4)	2	1
(2, 3)			(2, 3)	1		(2, 3)	1	0
(2, 4)			(2, 4)	1		(2, 4)	1	0
(3, 4)			(3, 4)	1		(3, 4)	1	0

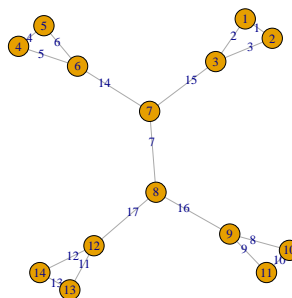


- *Conclusion* :  $E_B(1) = 1.5$ .

```
> edge_betweenness(G)
## [1] 1.5 1.5 1.0 1.5 1.5
```

## EB pour clustering

- $E_b(e)$  *élevé* si  $e$  *relie des groupes* plutôt que des arêtes internes au groupe.



```
> edge_betweenness(G)
## [1] 1 12 12 1 12 12 49 12 12 1 12 12 1 33 33 33 33
```

## Classification par EB

- *Idée : enlever les arêtes à fort EB les unes après les autres.*

### Algorithme [Girvan and Newman, 2002]

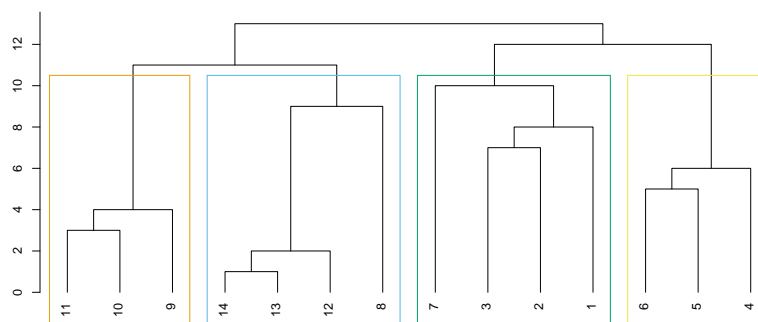
1. Calculer l'EB de toutes les arêtes.
2. Retirer l'arête avec le plus grand EB.
3. Recalculer l'EB du nouveau graphe.
4. Retirer l'arête...
5. Jusqu'à isoler tous les nœuds.

### Remarque

- Processus *similaire à la CAH*.
- On peut visualiser l'algorithme avec un *dendrogramme*.

## Le dendrogramme

```
> clust.EB <- cluster_edge_betweenness(G)
> dendPlot(clust.EB)
```



### Question

Où couper le dendrogramme pour obtenir les groupes ?

### Couper le dendrogramme

- On se donne un *critère* qui mesure la qualité d'une *partition* associée à une coupure.
- On choisit la partition qui *optimise le critère choisi*.
- *Critère usuel* : la *modularité*

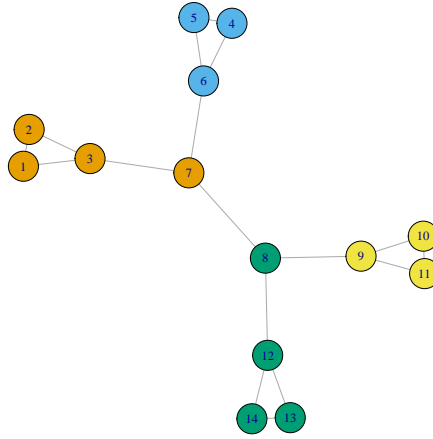
```
> clust.EB$modularity %>% round(3)
## [1] -0.074 -0.022 0.074 0.126 0.223 0.272 0.372 0.420 0.521 0.543
## [11] 0.566 0.503 0.441 0.000
```

### Conclusion

On retiendra une partition en *4 groupes*.

## Visualisation des groupes

```
> plot(G, vertex.color=clust.EB$membership)
```



## 5.2 La modularité

- *Rappel* :  $G = (V, E)$  un graphe,  $A$  sa matrice d'adjacence.
- On cherche une *partition de  $V$*  l'ensemble des nœuds en  $K$  groupes ou communautés  $\mathcal{C}_1, \dots, \mathcal{C}_K$ .

### Idée

1. Se donner un critère qui mesure la performance d'une partition.
2. Choisir la partition qui optimise le critère choisi.

### Modularité

- Un des critères les *plus utilisés*.
- *Idée* : comparer la performance de la partition *sur le graphe* à sa performance sur un *graphe aléatoire*.
- Soit  $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_K\}$  une partition des nœuds de  $G = (V, E)$ .

#### Modularité de $\mathcal{C}$

$$\mathcal{M}(\mathcal{C}) = \frac{1}{2m} \sum_{1 \leq i, j \leq n} (A_{ij} - P_{ij}) \delta(\mathcal{C}(i), \mathcal{C}(j))$$

où

- $m = |E|$ .
- $\delta(\mathcal{C}(i), \mathcal{C}(j)) = 1$  si  $i$  et  $j$  sont dans le même élément de la partition, 0 sinon
- $P_{ij}$  représente l'espérance du nombre d'arêtes entre  $i$  et  $j$  sous un modèle nul (graphe aléatoire) à définir.

### Interprétation

- $-1 \leq \mathcal{M}(\mathcal{C}) \leq 1$  ;
- $\mathcal{M}(\mathcal{C}) \nearrow$  *plus d'arêtes* dans les communautés que le modèle nul (*bonnes communautés*) et réciproquement lorsque  $\mathcal{M}(\mathcal{C}) \searrow$ .

### Le modèle nul

- Il peut être spécifier de *plusieurs façons* (voir [Fortunato, 2010]).
- *Première approche* : les  $m$  arêtes sont *distribuées uniformément* entre les paires de nœuds :

$$P_{ij} = \frac{2m}{n(n-1)}, \quad 1 \leq i, j \leq n.$$

- *Seconde approche* : générer aléatoirement les arêtes en conservant les *degrés de centralité* des nœuds :

$$P_{ij} = \frac{d_i d_j}{2m}, \quad 1 \leq i, j \leq n.$$

On a alors

$$\mathcal{M}(\mathcal{C}) = \frac{1}{2m} \sum_{1 \leq i, j \leq n} \left( A_{ij} - \frac{d_i d_j}{2m} \right) \delta(\mathcal{C}(i), \mathcal{C}(j))$$

- C'est souvent la *seconde approche* qui est utilisée.
- Le terme

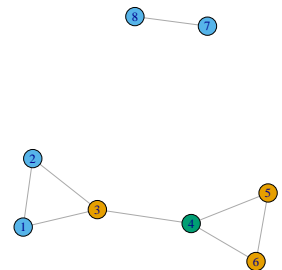
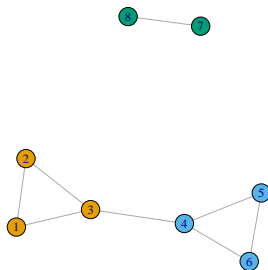
$$A_{ij} - \frac{d_i d_j}{2m}$$

correspond à la *différence de lien* entre le graphe considéré et un graphe aléatoire dont la contrainte est la conservation des degrés de sommets.

- Si on a *beaucoup d'arêtes connectés dans les communautés* alors  $A_{ij}$  sera souvent *plus grand* que  $\frac{d_i d_j}{2m} \implies \mathcal{M}(\mathcal{C}) \nearrow$ .
- Sous *R*, on utilise la fonction *modularity*.

## Le coin R

- On considère les *2 partitions* suivantes pour le même graphe.



- On obtient les *modularités* avec

```
> modularity(G, cl1)
## [1] 0.4765625
> modularity(G, cl2)
## [1] 0.0078125
```

## Maximisation de la modularité

- *Idée* : Considérer *toutes les partitions* et choisir celle qui *maximise la modularité*.

### Approche exhaustive

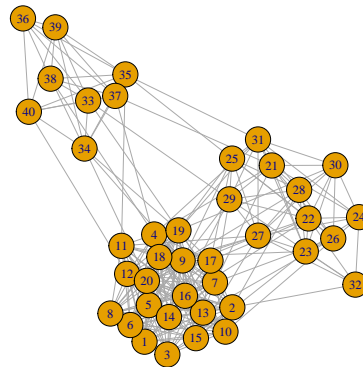
1.  $\mathcal{V}$  : ensemble des partitions des nœuds de  $G = (V, E)$ .
2. Pour chaque  $\mathcal{C} \in \mathcal{V}$  calculer  $\mathcal{M}(\mathcal{C})$ .
3. Renvoyer

$$\operatorname{argmax}_{\mathcal{C} \in \mathcal{V}} \mathcal{M}(\mathcal{C}).$$

- Sur *R* on utilise *cluster\_optimal*.

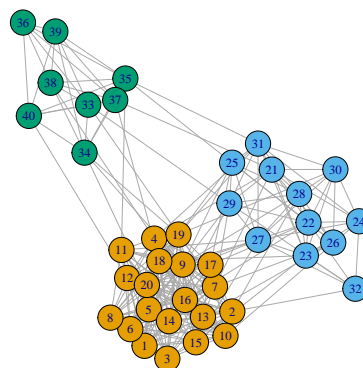
## Le coin R

```
> plot(G1)
```



## Le coin R

```
> cl.exh <- cluster_optimal(G1)
> V(G1)$color <- membership(cl.exh)
> G1$palette <- categorical_pal(length(cl.exh))
> plot(G1)
```



## Complexité

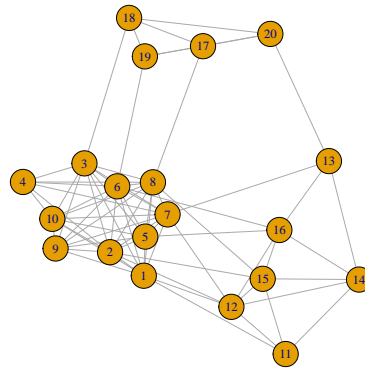
- Problème *NP complet*.
- Si  $|V|$  est "grand", *impossible dans un temps raisonnable*.
- *Solution* : utiliser des *algorithmes itératifs* qui vont converger vers des *maxima locaux*.

## La méthode de Louvain

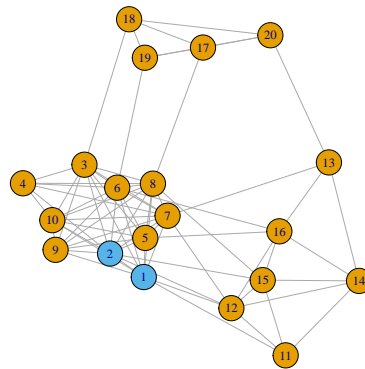
- Proposée par des chercheurs français qui se sont retrouvés à Louvain [Blondel et al., 2008].
- Elle repose sur *deux phases distinctes*, répétées itérativement.
- On pourra consulter l'url suivante pour plus de détails :  
<http://cedric.cnam.fr/vertigo/Cours/RCP216/coursFouilleGraphesReseauxSociaux2.html>

### Passe 1 - phase 1, itération 1

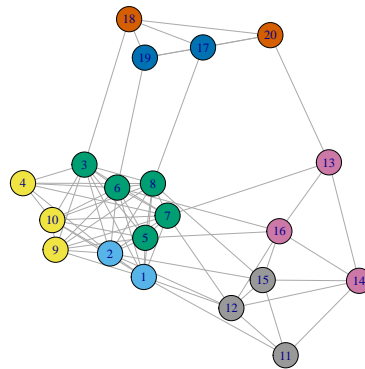
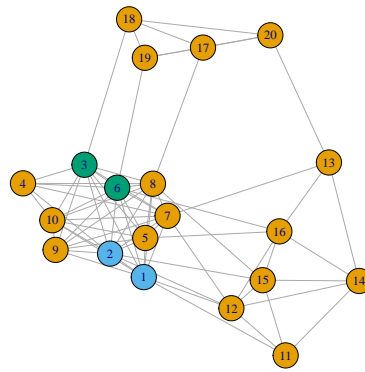
- Chaque *nœud* forme *une communauté*.



- Pour chaque nœud  $i$ , on place  $i$  dans la communauté de chacun de ses *voisins*  $j$ .
- Calcul du *gain de modularité*.
- On place  $i$  dans la *communauté où le gain est maximum*.



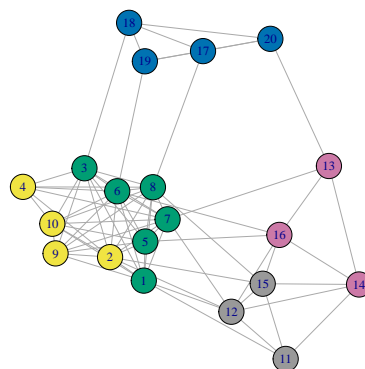
- Même procédé pour tous les nœuds suivants.



*Fin de l'itération 1*

**Itération 2, ...**

- L'itération 1 est *répétée* sur ce nouveau graphe...
- et ainsi de suite jusqu'à ce qu'il n'y ait *plus d'amélioration*.
- On obtient un *maximum local* de la modularité.





## Fin de la phase 1

- A la fin de la phase 1, on obtient un *graphe valué*  $G1 = (V1, E1)$
- *muni de communautés*  $\mathcal{C}_1 = \{C_{1,1}, \dots, C_{1,p_1}\}$ .

## Sur l'exemple

On a *5 communautés*.

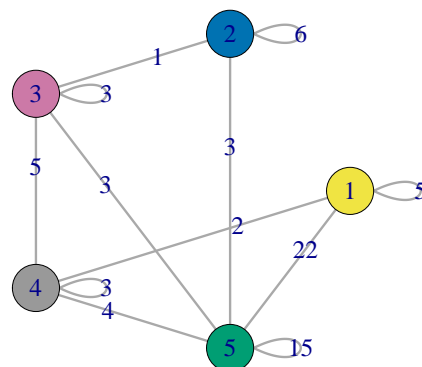
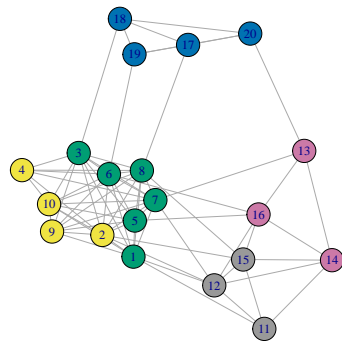
## Passe 1 - phase 2

### Construction d'un nouveau graphe

- *1 nœud* = 1 communauté.
- *Arêtes valuées* : le poids correspond au nombre de liens entre les nœuds de chaque communauté.
- Ce nouveau graphe peut s'obtenir en calculant sa *matrice d'adjacence*.
- *Sur l'exemple, on a*

```
> A
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    5    0    0    2   22
## [2,]    0    6    1    0    3
## [3,]    0    1    3    5    3
## [4,]    2    0    5    3    4
## [5,]   22    3    3    4   15
```

## Obtention du nouveau graphe

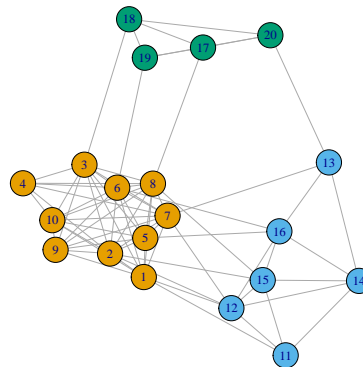


### Fin de la phase 2 et de la passe 1

- La passe suivante consiste à *ré-appliquer les deux phases* au graphe obtenu en fin de pass1.
- Les passes sont *répétées* jusqu'à atteindre *un maximum de modularité*.
- Sur R, on utilise *cluster\_louvain*.

```
> cl.louv <- cluster_louvain(g)
> V(g)$color <- cl.louv$membership
> plot(g)
```

### Représentation des "communautés Louvain"

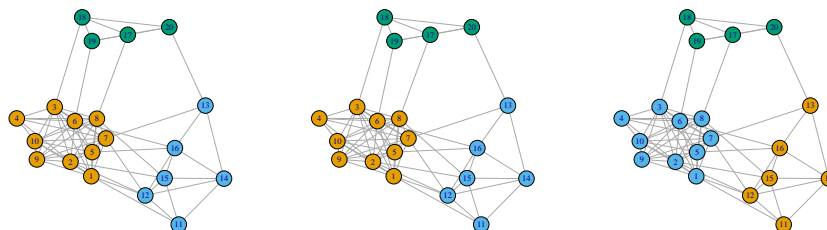


### Quelques remarques

- Le nombre de communautés *diminue à chaque passe*.
- Nombre de passes *généralement faible* (moins de 10).
- Nœuds numérotés au hasard au début de la première passe  $\Rightarrow$  *algorithme aléatoire*.
- Il existe d'autres *procédures itératives* que l'algorithme de Louvain.

### Autres méthodes

```
> c11 <- cluster_edge_betweenness(g)
> c12 <- fastgreedy.community(g)
> c13 <- cluster_walktrap(g)
```



### 5.3 Clustering spectral

- *Cadre identique* :  $G = (V, E)$  un graphe et on veut trouver une partition de  $V$  en *clusters* ou *communautés*.
- Approche basée sur la *décomposition spectrale du Laplacien du graphe*.
- Approche utilisée dans un *cadre plus large* :
  - *Problème* : clustering sur un jeu de données standards  $n \times p$  ;
  - L'approche peut être appliquée à une *matrice de similarité*.
- On pourra consulter [von Luxburg, 2017] dont cette partie est fortement inspirée.

#### Notations

- $G = (V, E)$  un graphe *non dirigé valué* avec  $n = |V|$ .
- $w_{ij} \geq 0$  *poids de l'arête* entre  $i$  et  $j$  et  $W = (w_{ij})_{1 \leq i, j \leq n}$  la *matrice d'adjacence*.
- $d_i = \sum_{j \neq i} w_{ij}$  *degré* du nœud  $i$  et  $D = \text{diag}(d_i)_{1 \leq i \leq n}$  la *matrice des degrés*.

#### Laplacien non normalisé

Le *Laplacien non normalisé* de  $G$  est la matrice  $n \times n$  définie par :

$$L = D - W.$$

#### Quelques propriétés

Les deux propositions suivantes sont *fondamentales* pour l'algorithme de *clustering spectral*.

##### Proposition 1

1. Pour tout vecteur  $f \in \mathbb{R}^n$  on a

$$f' L f = \frac{1}{2} \sum_{1 \leq i, j \leq n} w_{ij} (f_i - f_j)^2.$$

2.  $L$  est symétrique et semi définie positive.
3. La *plus petite valeur propre* de  $L$  est 0, le vecteur propre correspondant est  $\mathbf{1}_n$ .
4.  $L$  a  $n$  valeurs propres non nulles  $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ .

#### Valeurs propre et nombre de compo. connexes

##### Proposition 2

Soit  $G$  un graphe *non dirigé*. Alors

1. le *degré de multiplicité*  $k$  de la valeur propre 0 de  $L$  est égal au *nombre de composantes connexes*  $A_1, \dots, A_k$  dans  $G$ .
2. l'*espace propre* associé à la valeur propre 0 est engendré par les vecteurs d'indicatrices  $\mathbf{1}_{A_1}, \dots, \mathbf{1}_{A_k}$ .

#### Conséquence importante

Le *spectre de  $L$*  permet d'identifier *les composantes connexes de  $G$* .

- *En pratique* : 1 communauté n'est pas forcément égale à une composante connexe.
- On peut par exemple vouloir *extraire des communautés* dans un graphe *à une composante connexe*.

#### Idée

Considérer les  *$k$  plus petites valeurs propres* du Laplacien.

## Spectral clustering non normalisé

### Algorithme

Entrées : un graphe non dirigé  $G$ ,  $k$  le nombre de clusters.

1. Calculer le Laplacien non normalisé  $L$  de  $G$ .
2. Calculer les  $k$  premiers vecteurs propres  $u_1, \dots, u_k$  de  $G$ .
3. On note  $U$  la matrice  $n \times k$  qui contient les  $u_k$  et  $y_i$  la  $i^e$  ligne de  $U$ .
4. Faire un  $k$ -means avec les points  $y_i, i = 1, \dots, n \implies A_1, \dots, A_k$ .

Sortie : clusters  $C_1, \dots, C_k$  avec

$$C_j = \{i | y_i \in A_j\}.$$

### Remarque

- Si  $G$  ne possède pas  $k$  composantes connexes alors  $U$  n'est pas composé que de 1 et de 0.
- On ne peut donc pas extraire directement les composantes à cette étape.
- Mais si il existe (presque)  $k$  composantes, alors les  $y_i \in \mathbb{R}^k$  risquent de se rapprocher de cette configuration 0-1.
- C'est pourquoi on fait un  $k$ -means en 4.
- Il existe plusieurs versions d'algorithme de clustering spectral.
- Les plus utilisées s'appliquent à une version normalisée du Laplacien, par exemple :

$$L_{\text{norm}} = I - D^{-1/2} W D^{-1/2}.$$

- Les propriétés de  $L_{\text{norm}}$  sont proches de celles de  $L$ . On a par exemple la propriété suivante.

### Proposition 3

Soit  $G$  un graphe non dirigé. Alors

1. le degrés de multiplicité  $k$  de la valeur propre 0 de  $L_{\text{norm}}$  est égal au nombre de composantes connexes  $A_1, \dots, A_k$  dans  $G$ .
2. l'espace propre associé à la valeur propre 0 est engendré par les vecteurs d'indicatrices  $D^{1/2} \mathbf{1}_{A_1}, \dots, D^{1/2} \mathbf{1}_{A_k}$ .

## Clustering spectral normalisé

- On déduit de cette propriété la version la plus courante de clustering spectral du à [Ng et al., 2002].

### Algorithme

Entrées : un graphe non dirigé  $G$ ,  $k$  le nombre de clusters.

1. Calculer le Laplacien normalisé  $L_{\text{norm}}$  de  $G$ .
2. Calculer les  $k$  premiers vecteurs propres  $u_1, \dots, u_k$  de  $G$ . On note  $U$  la matrice  $n \times k$  qui les contient.
3. Calculer  $T$  en normalisant les lignes de  $U$  :  $t_{ij} = u_{ij} / (\sum_{\ell} u_{i\ell}^2)^{1/2}$ .
4. Faire un  $k$ -means avec les points  $y_i, i = 1, \dots, n$  ( $i^e$  ligne de  $T$ )  $\implies A_1, \dots, A_k$ .

Sortie : clusters  $C_1, \dots, C_k$  avec

$$C_j = \{i | y_i \in A_j\}.$$

### Remarques

- Algorithme quasi similaire au clustering spectral non normalisé.
- Une étape de normalisation en plus.
- Cette étape se justifie par la théorie de la perturbation du spectre d'une matrice.
- On pourra consulter [von Luxburg, 2017] pour des justifications.

## Choix de $k$

- Comme souvent en *clustering*, cet algorithme nécessite de connaître le *nombre de groupes*.
- Utilisation de *connaissances métier* pour ce choix
- ou étude des *valeurs propres* du Laplacien.

## Généralisation

### Remarque importante

- L'algorithme *n'utilise pas* nécessairement la *structure du graphe*.
- Il est *entièrement basé sur la matrice (d'adjacence)  $W$*  des poids qui contient des arêtes.
- Cette matrice peut également être vue comme une *matrice de similarité*.

### Conséquence

- On peut donc *généraliser cet algorithme* à n'importe quel problème où on *possède une matrice de similarité*.
- *Exemple* : problème de *clustering standard* sur des données  $n \times p$  (il "suffit" de construire une matrice de similarité).

## Clustering spectral sur un tableau de données

- *Données* : tableau  $n \times p$   $n$  individus,  $p$  variables.
- *Problème* : classification non supervisée des  $n$  individus.
- *Méthodes classiques* :  $k$ -means, CAH...

### Alternative : clustering spectral

1. construire un *graphe de similarité* ;
2. lancer l'algorithme de *clustering spectral* sur *ce graphe* (ou plutôt sur sa *matrice de similarité*).

## Construction du graphe de similarités

- On peut utiliser les techniques vues dans la *section 4* :  *$\varepsilon$ -neighborhood graph* ou *plus proches voisins (mutuels ou non)*.
- De façon plus générale, la matrice de similarités s'obtient souvent à partir d'un *noyau  $K$*  :

$$K : \mathbb{R}^p \times \mathbb{R}^p \rightarrow \mathbb{R} \\ (x, y) \mapsto \langle \Phi(x), \Phi(y) \rangle_{\mathcal{H}}$$

où  $\Phi : \mathbb{R}^p \rightarrow \mathcal{H}$  est une fonction qui *plonge* les observations dans un espace de Hilbert  $\mathcal{H}$  appelé *feature space*.

## Exemples de noyau

- *Linéaire (vanilladot)* :  $K(x, y) = \langle x, y \rangle$ .
- *Gaussien (rfbdot)* :  $K(x, y) = \exp(-\sigma \|x - y\|^2)$ .
- *Polynomial (polydot)* :  $K(x, y) = (\text{scale} \langle x, y \rangle + \text{offset})^{\text{degree}}$ .
- ...

## Références

On pourra trouver dans exemples de noyau dans [Karatzoglou et al., 2004].

## Matrice de similarités avec un noyau

- Etant données  $n$  observations  $x_i \in \mathbb{R}^p$  et un *noyau  $K$*
- on peut construire une *matrice de similarité*, par exemple pour un *noyau Gaussien* :

$$w_{ij} = \begin{cases} \exp(-\sigma \|x_i - x_j\|^2) & \text{si } i \neq j \\ 0 & \text{sinon.} \end{cases}$$

### Clustering spectral

Le *clustering spectral* consiste à appliquer l'algorithme vu précédemment en calculant le *Laplacien normalisé* à partir de cette *matrice de similarités* (voir [Ng et al., 2002, Arias-Castro, 2011]).

## Clustering spectral sur des données $n \times p$

### Algorithme

**Entrées** : tableau de données  $n \times p$ ,  $K$  un noyau,  $k$  le nombre de clusters.

1. Calculer la matrice de *similarités*  $W$  sur les données avec le *noyau*  $K$ .
2. Calculer le *Laplacien normalisé*  $L_{\text{norm}}$  à partir de  $W$ .
3. Calculer les  $k$  *premiers vecteurs propres*  $u_1, \dots, u_k$  de  $G$ . On note  $U$  la matrice  $n \times k$  qui les contient.
4. Calculer  $T$  en *normalisant les lignes* de  $U$  :  $t_{ij} = u_{ij} / (\sum_{\ell} u_{i\ell}^2)^{1/2}$ .
5. Faire un *k-means* avec les points  $y_i, i = 1, \dots, n$  ( $i^{\text{e}}$  ligne de  $T$ )  $\Rightarrow A_1, \dots, A_k$ .

**Sortie** : clusters  $C_1, \dots, C_k$  avec

$$C_j = \{i | y_i \in A_j\}.$$

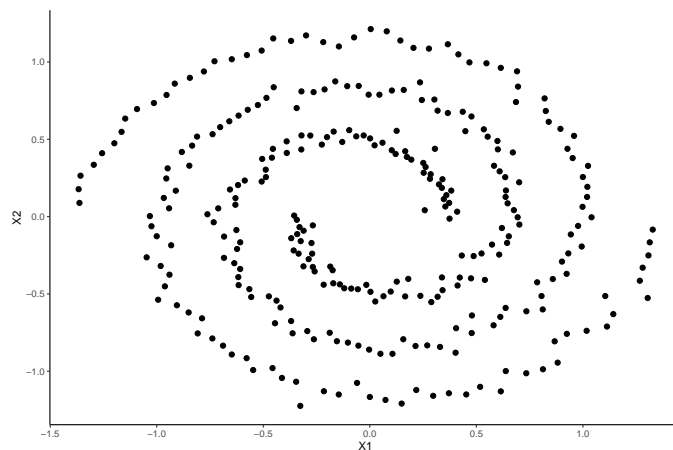
### Le coin R

- La fonction *specc* du package *kernelab* permet de faire le clustering spectral.
- *Exemple* : données *spirals*

```
> library(kernelab)
> data(spirals)
> spirals1 <- data.frame(spirals)
> head(spirals1)
##           X1           X2
## 1  0.8123568 -0.98712687
## 2 -0.2675890 -0.32552004
## 3  0.3739746 -0.01293652
## 4  0.2576481  0.04130805
## 5 -0.8472613  0.32939461
## 6  0.4097649  0.03205686
```

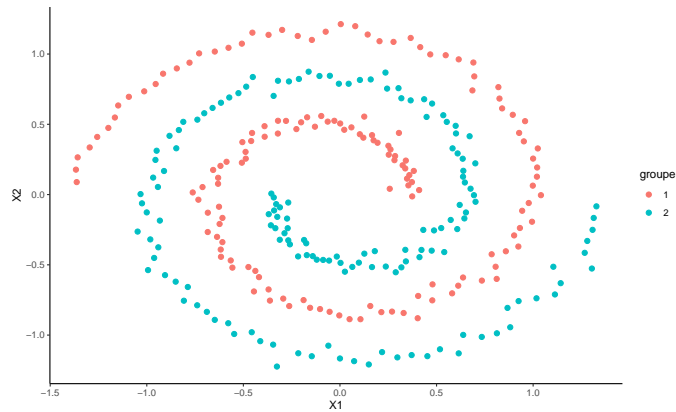
### Visualisation du nuage de points

```
> ggplot(spirals1)+aes(x=X1,y=X2)+geom_point()+theme_classic()
```



### Le clustering spectral

```
> groupe <- specc(spirals,centers=2,kernel="rbfdot")
> head(groupe)
## [1] 2 2 1 1 2 1
> spirals1 <- spirals1 %>% mutate(groupe=as.factor(groupe))
> ggplot(spirals1)+aes(x=X1,y=X2,color=groupe)+geom_point(size=2)+theme_classic()
```



## 6 Bibliographie

### Références

- [Arias-Castro, 2011] Arias-Castro, E. (2011). Clustering based on pairwise distances when the data is of mixed dimensions. *IEEE Transaction on Information Theory*, 57(3) :1692–1706.
- [Bahoken et al., 2013] Bahoken, F., Beauguitte, L., and Lhomme, S. (2013). *La visualisation des réseaux. Principes, enjeux et perspectives*. halshs-00839905.
- [Blondel et al., 2008] Blondel, V. D., Guillaume, J., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics : Theory and Experiment*.
- [Daudin et al., 2008] Daudin, J.-J., Picard, F., and Robin, S. (2008). A mixture model for random graphs. *Statistics and computing*, 18 :173–183.
- [Fortunato, 2010] Fortunato, S. (2010). Community detection in graphs. *Physics report*, 486 :75–174.
- [Girvan and Newman, 2002] Girvan, M. and Newman, M. E. J. (2002). Community structure in social and biological networks. *Proc. Natl. Acad. Sci.*, pages 7821–7826.
- [Karatzoglou et al., 2004] Karatzoglou, A., Hornik, K., Smola, A., and Zeileis, A. (2004). kernlab – an s4 package for kernel methods in r. *Journal of Statistical Software*, 11(9).
- [Ng et al., 2002] Ng, A., Jordan, M., and Weiss, Y. (2002). On spectral clustering analysis. In *Advances in Neural Information Processing Systems (NIPS)*, volume 14, pages 849–856.
- [von Luxburg, 2017] von Luxburg, U. (2017). A tutorial on spectral clustering. *Statistics and computing*, 17 :395–416.