

# Graph Mining & network science

Matrice laplacienne



# Un graphe d-régulier

- un graphe  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$  est **d-régulier** si chaque noeud est de degré **d**
- supposons **G** connexe et non-orienté

Quelles valeurs propres et quels vecteurs propres de la matrice d'adjacence ?

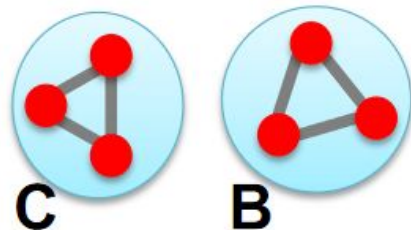
$$\mathbf{A} \cdot \mathbf{x} = \lambda \mathbf{x}$$

- $\mathbf{x} = (1, 1, \dots, 1)^T$  est un vecteur propre, et  $\lambda = \mathbf{d}$  est sa valeur propre associée
- **d** est la valeur propre la plus grande de **A**

# Un graphe à 2 composants

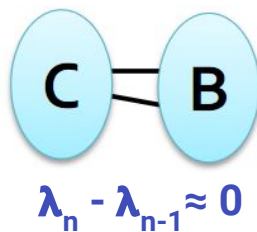
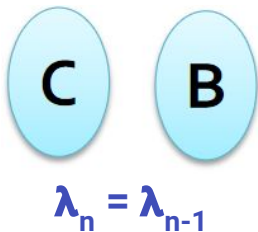
Et si  $G$  est non-connexe ?

- supposons que  $G$  a 2 composants, chacun d'eux  **$d$ -régulier**



Quels vecteurs propres ?

- $\mathbf{x} = (1 \text{ sur tous les noeuds de C et } 0 \text{ ailleurs})$ , et vice-versa
- $\mathbf{x}_1 = (1, \dots, 1, 0, \dots, 0)^T$  et  $\mathbf{A} \cdot \mathbf{x}_1 = (d, \dots, d, 0, \dots, 0)^T$
- $\mathbf{x}_2 = (0, \dots, 0, 1, \dots, 1)^T$  et  $\mathbf{A} \cdot \mathbf{x}_2 = (0, \dots, 0, d, \dots, d)^T$
- dans les 2 cas, la valeur propre est  **$\lambda = d$**



la deuxième plus grande valeur propre  $\lambda_{n-1}$  est très proche de  $\lambda_n$

## Et si G n'est pas régulier ?

- plus compliqué !
- la matrice d'adjacence **n'a pas le spectre le plus riche**
- quelques applications : centralité eigenvector

## “Spectral Graph Theory” - Fan Chung :

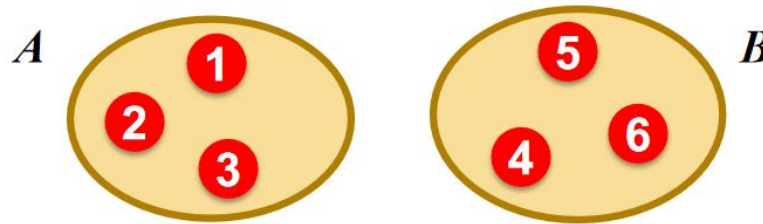
- étude du spectre d'une matrice représentant **G**
- **spectre d'une matrice** : l'ensemble des vecteurs propres  **$\mathbf{v}_i$**  d'une matrice, ordonnés par la grandeur de leurs valeurs propres correspondantes :

$$\Lambda = \{\lambda_1, \lambda_2, \dots, \lambda_n\} \quad \text{avec} \quad \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$$

**Cherchons une matrice plus intéressante que A !**

## Motivation : bipartition d'un graphe

- un graphe **non-orienté**  $G = (V, E)$
- tâche de bipartition :
  - diviser les sommets en deux ensembles disjoints **A**, **B** :

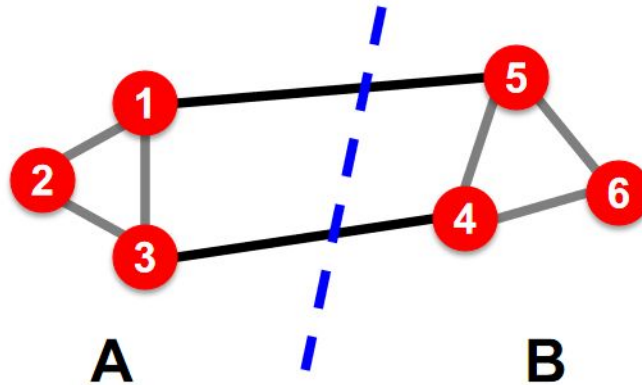


## Questions :

- comment définir ce qu'est une "bonne" bipartition de **G** ?
- comment identifier efficacement une telle partition ?

## Comment définir ce qu'est une bonne bipartition ?

- maximiser le nombre de connexions intra-groupe
- minimiser le nombre de connexions inter-groupe

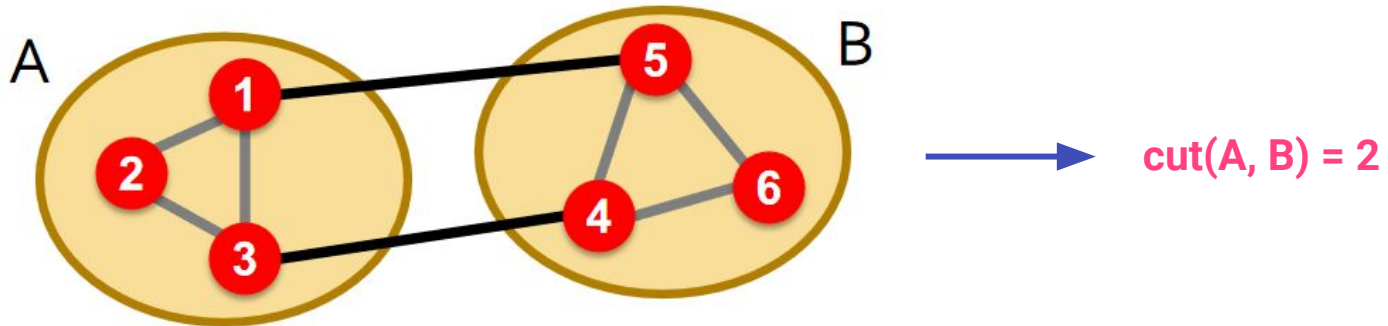


# Coupe ou “cut” d’un graphe

Exprimer l’objectif de partitionnement comme une fonction de l’edge cut :

- un cut ou une coupe :  $\{\{u_i, u_j\} \in E \mid u_i \in A, u_j \in B\}$
- abus de langage : **poids du cut**  $\approx$  cut

$$cut(A, B) = \sum_{i \in A, j \in B} w_{ij}$$

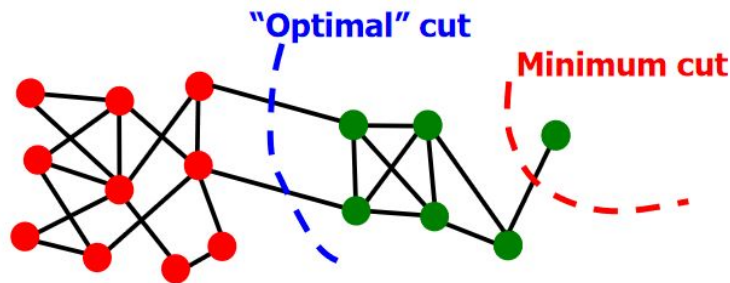


# Critère de coupe minimale

- minimiser le poids des connexions entre les deux groupes :

$$\operatorname{argmin}_{A,B} \operatorname{cut}(A, B)$$

- cas dégénérés :



## Problèmes :

- ne considère que les connexions extérieures
- très sensible au bruit



## Critère : conductance

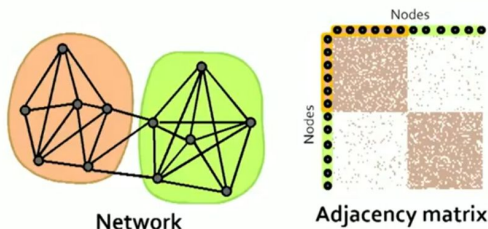
- connexions entre les groupes rapportées à la taille de chaque groupe :

$$\phi(A, B) = \frac{cut(A, B)}{min(vol(A), vol(B))}$$

**vol(A)** = somme des degrés des noeuds de A

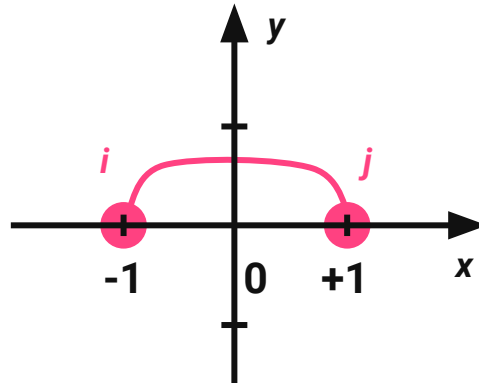
## Avantage :

- produit des partitions beaucoup plus équilibrées



## Labels des noeuds

- on associe à chaque noeud  $u_i$  un label  $-1$  ou  $+1$  en fonction de leur appartenance à **A** ou **B**
- on représente ces labels par un vecteur  $\mathbf{x} \in \{-1, +1\}^n$  avec  $x_i = \text{label de } u_i$
- $\text{cut}(\mathbf{A}, \mathbf{B}) = \text{nombre de liens coupés par l'axe vertical } y$



## Matrice “edge-laplacienne” :

$$L^{ij} = \begin{bmatrix} & \textcolor{red}{i} & & \textcolor{red}{j} & \\ & & \dots & & \\ & 1 & & -1 & \\ \dots & & \dots & & \dots \\ & -1 & & 1 & \\ & & \dots & & \end{bmatrix}$$

**on remarque :**

$$x^T L^{ij} x = (x_j - x_i)^2$$

Supposons  $i \sim j$  :

$$x^T L^{ij} x = \begin{cases} 0 & \text{si } x_i = x_j \\ 4 & \text{si } x_i \neq x_j \end{cases}$$

Donc la formule ci-dessus compte (à un facteur 4 près) le nombre d'edges coupés par la partition  $\mathbf{x}$ .

## Matrice laplacienne

- on définit **L** la matrice laplacienne :

$$L = \sum_{\{i,j\} \in E} L^{ij} = \sum_{\{i,j\} \in E} (x_j - x_i)^2$$

- Donc  **$\mathbf{x}^T \mathbf{L} \mathbf{x}$**  est proportionnel à **cut(A, B)**

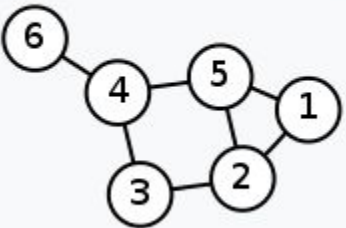
## Propriétés immédiates :

- **L** est **symétrique semi-définie positive** : valeurs propres réelles positives et vecteurs propres orthogonaux
- **1** est vecteur propre associé à la valeur propre  **$\lambda_1 = 0$**  :  $L \cdot \mathbf{1} = 0$
- la multiplicité de **0** correspond au nombre de composants connexes

## Matrice laplacienne

- $L = D - A$

avec **D** la matrice des degrés et **A** la matrice d'adjacence

Graphe non orienté	Matrice des degrés	Matrice d'adjacence	Matrice laplacienne
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

## Bipartition

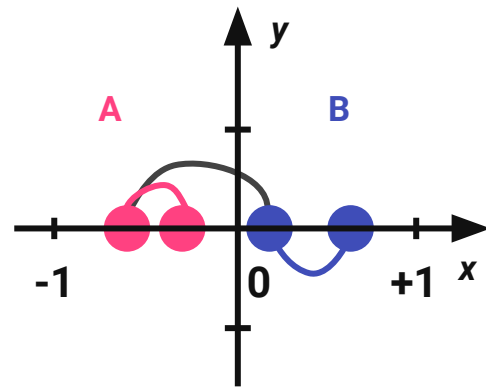
- **cut équilibré**, donc il y a autant de noeuds à “gauche” (-1) qu’à “droite” (+1) :

$$\sum_i x_i = 0 \quad \text{équivalent à} \quad x \perp \mathbf{1} \quad (\text{et évite les cas triviaux})$$

- on cherche :  $\operatorname{argmin}_{x \perp \mathbf{1}} x^T L x \Rightarrow \text{un problème NP-difficile !}$

## Assouplissement des conditions :

- on remplace  $x \in \{-1, +1\}^n$  par  $x \in \mathbb{R}^n$
- on ajoute la contrainte  $\|x\|_2 = 1$  pour garantir l'unicité de la solution
- on cherche alors :  $\operatorname{argmin}_{\substack{x \perp \mathbf{1} \\ \|x\|=1}} x^T L x$



## Diagonalisation de L

- **L** est diagonalisable :

$$L = U\Lambda U^T$$

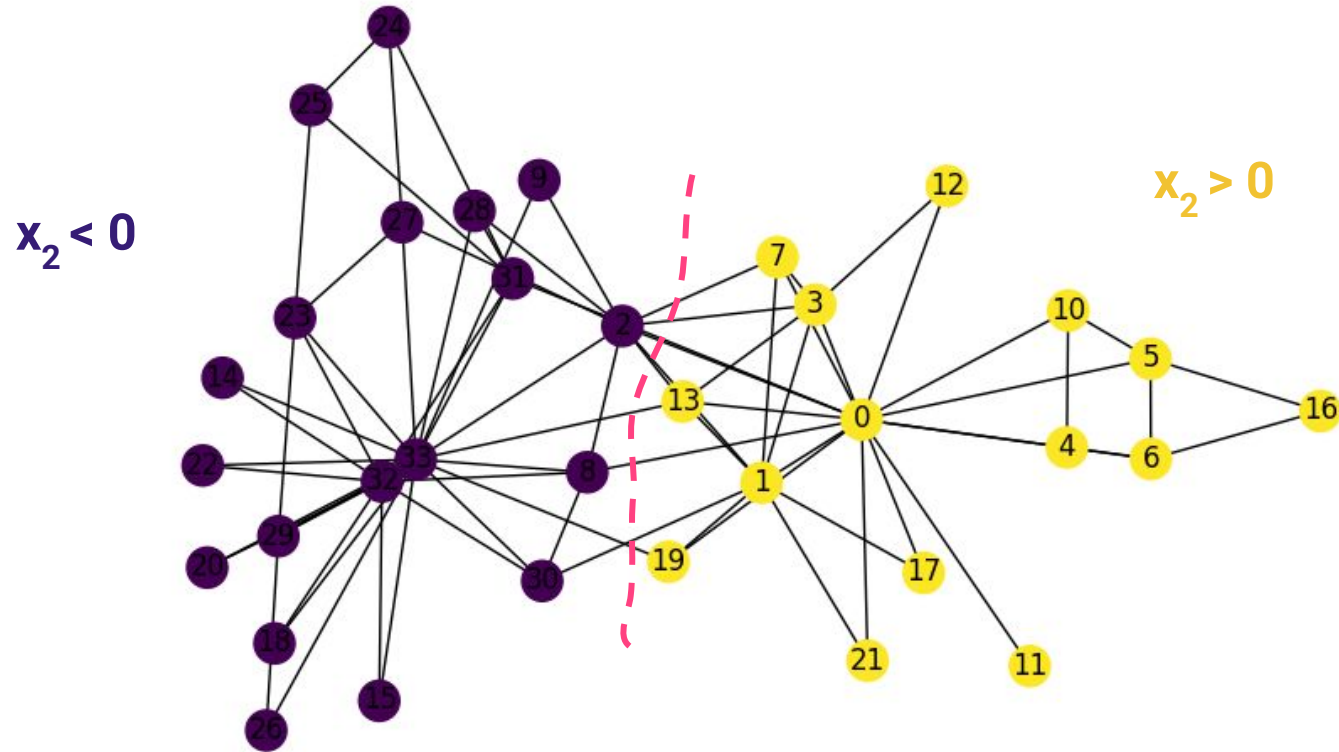
avec **U** orthonormale et  **$\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$**

- on remarque :

$$\begin{aligned}x^T L x &= x^T U \Lambda U^T \\&= (U^T x)^T \Lambda (U^T x) \\&= u^T \Lambda u \\&= \sum_i \lambda_i u_i^2 \geq \lambda_2 \sum_i u_i^2 = \lambda_2\end{aligned}$$

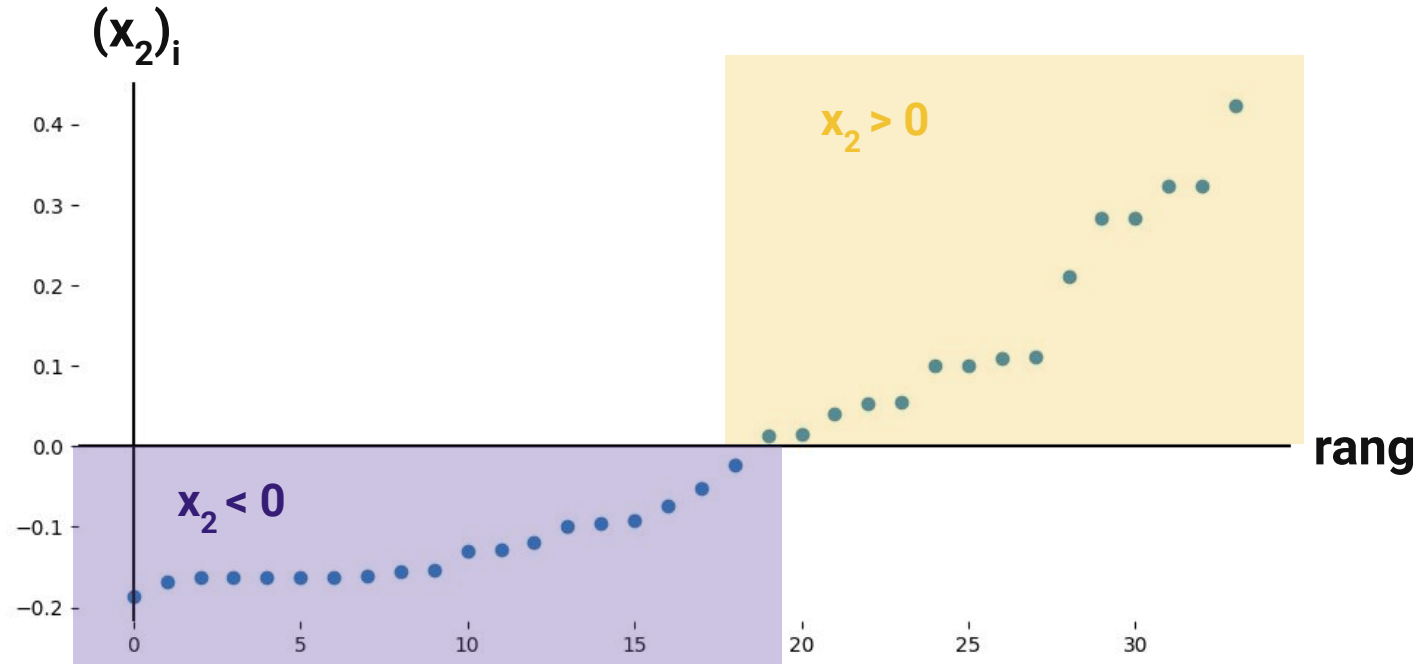
- le vecteur propre  **$x_2$**  associé à  **$\lambda_2$**  est notre solution !

## Exemple : Karate Club

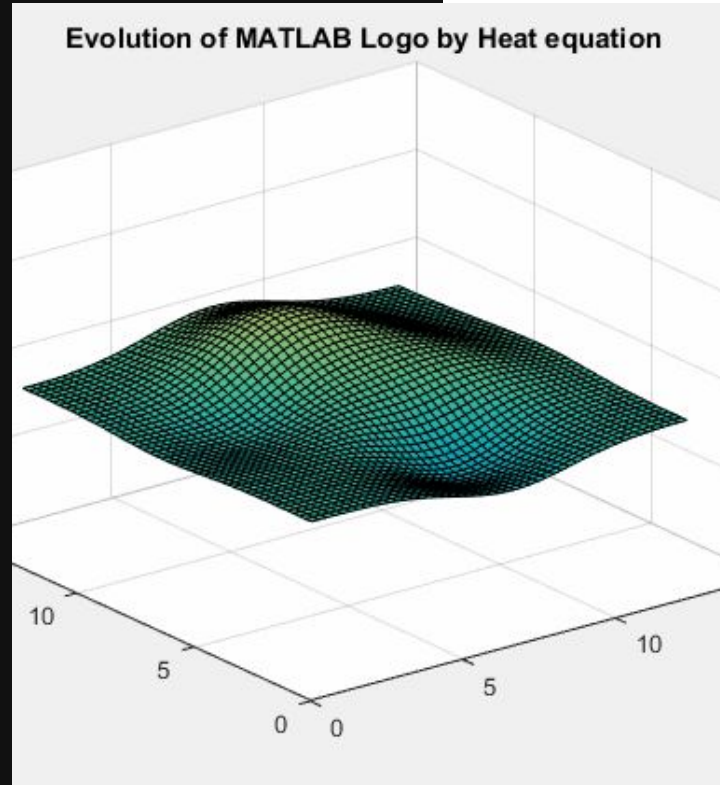




## Exemple : Karate Club



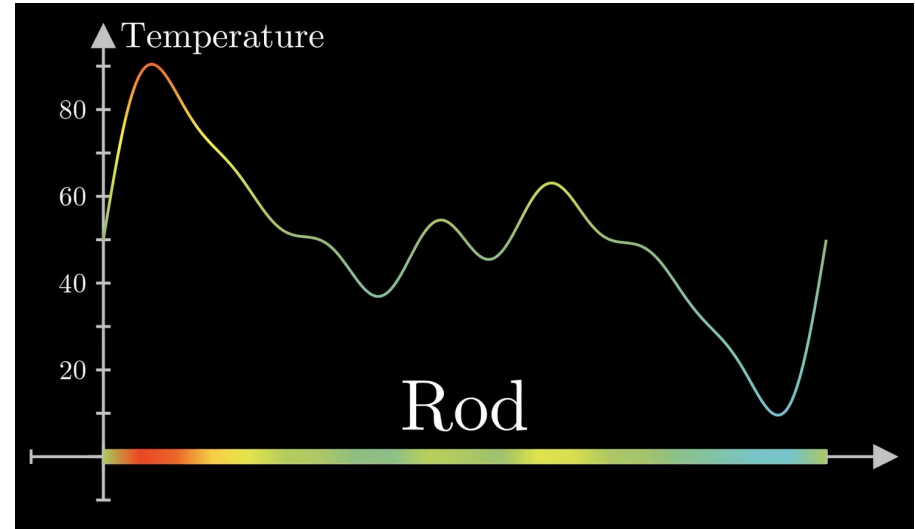
Pourquoi  
“laplacien” ?



## Conduction thermique

- soit  $T(x,t)$  la température d'un cylindre (représenté par un segment 1D)
- l'équation de la chaleur est :

$$\frac{\partial T}{\partial t} = C \frac{\partial^2 T}{\partial x^2}$$



### Conduction thermique

- plus généralement, pour une dimensionnalité supérieure à 1 :

$$\frac{\partial T}{\partial t} = C \Delta T$$

- avec :

$$\Delta f = \operatorname{div}(\operatorname{grad}(f)) = \nabla \cdot \nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \dots$$

- Imaginons qu'une substance occupe chaque noeud d'un graphe, et qu'à chaque timestep, elle se diffuse le long des edges :

$$\phi_i(t + \partial t) = \phi_i(t) + C \cdot \sum_{i \sim j} (\phi_j(t) - \phi_i(t)) \partial t$$

- Donc :

$$\begin{aligned} \frac{\partial \phi_i}{\partial t} &= C \cdot \sum_j A_{ij} (\phi_j - \phi_i) \\ &= C \left( \sum_j A_{ij} \phi_j - \sum_j A_{ij} \phi_i \right) \\ &= C \left( \sum_j [A_{ij} \phi_j] - k_i \phi_i \right) \\ &= C \cdot \sum_j (A_{ij} - k_i \delta_{ij}) \phi_j \\ &= -C \cdot \sum_j L_{ij} \phi_j \end{aligned}$$

- on peut représenter une fonction sur  $V$  par un vecteur  $\phi$  de taille  $n$
- on en déduit :

$$\frac{\partial \phi}{\partial t} = -CL \cdot \phi$$

- par analogie avec l'équation de la chaleur :

$$\Delta \equiv -L$$

- solution fondamentale de l'équation de la chaleur :

$$\phi(t) = \phi_0 e^{-CLt}$$

- on peut simplifier la fonction  $\phi$  en diagonalisant  $L$

$$\phi(t) = \phi_0 e^{-CLt} = \phi_0 e^{-CU\Lambda U^T t} = \phi_0 U e^{-C\Lambda t} U^T$$

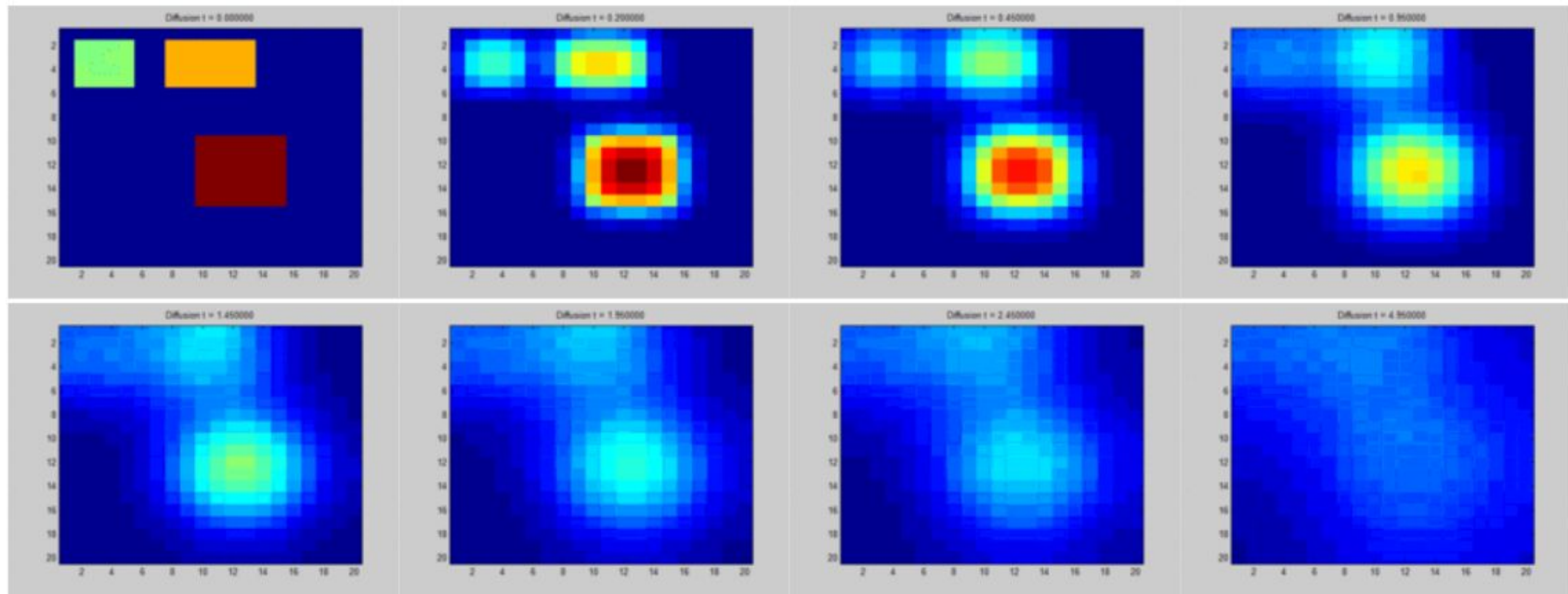
- on appelle le noyau de la chaleur d'un graphe (heat kernel) la fonction :

$$H(t) = e^{-Lt} = U e^{-\Lambda t} U^T$$

avec :

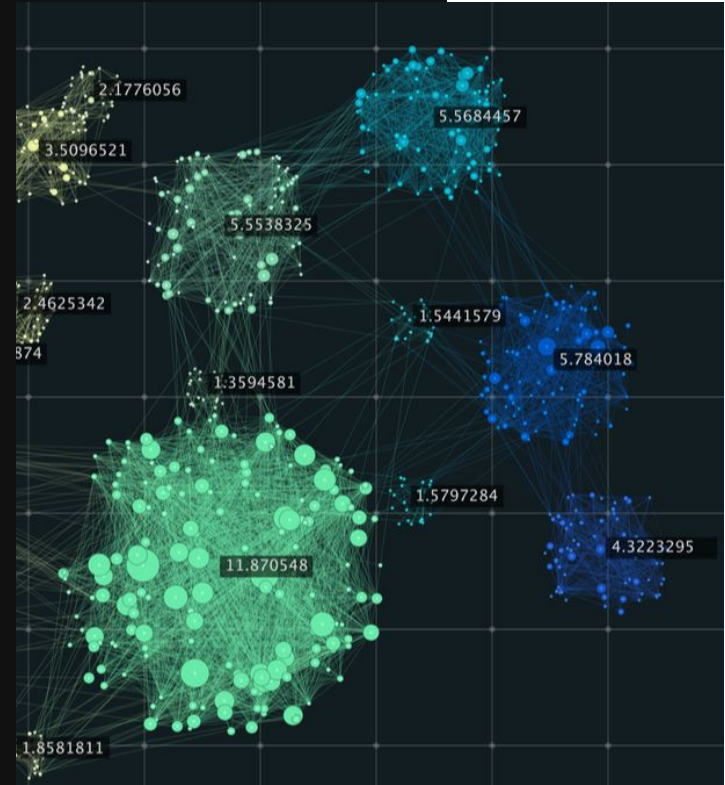
$$e^{-\Lambda t} = \text{diag}(e^{-\lambda_1 t}, e^{-\lambda_2 t}, \dots, e^{-\lambda_n t})$$

# Diffusion dans un graphe





# Applications : Spectral Clustering



## Rappel

- $\mathbf{x}_2$  minimise une version simplifiée de la conductance :

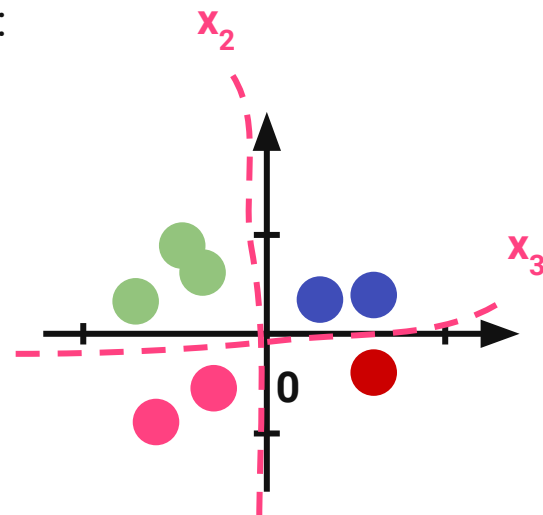
$$\min_{\substack{\|x\|=1 \\ x \perp x_1}} x^T L x = \lambda_2$$

- on peut aller plus loin :

$$\min_{\substack{\|x\|=1 \\ x \perp (x_1, x_2)}} x^T L x = \lambda_3$$

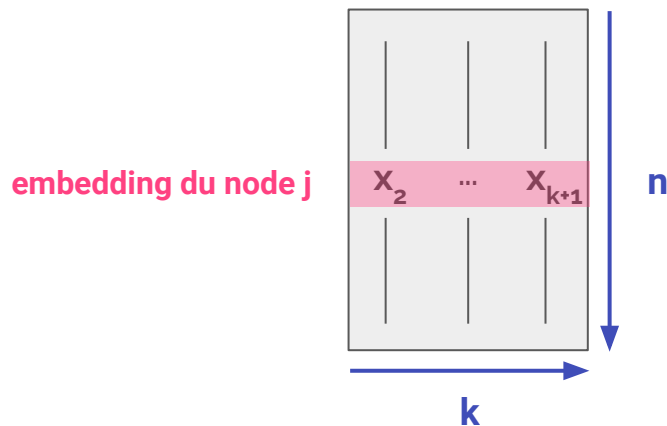
$\mathbf{x}_3$  minimise le “edge-cut” d’une façon  
indépendante (orthogonale) à  $\mathbf{x}_2$

- et ainsi de suite avec  $\mathbf{x}_4, \mathbf{x}_5, \dots$



# Embedding spectral

- embedding = associer à chaque noeud un vecteur  $\mathbf{u} \in \mathbb{R}^k$  de dimension  $k$
- en pratique, on extrait les  $k+1$  vecteurs propres associés aux valeurs propres les plus petites, et on ignore le premier :

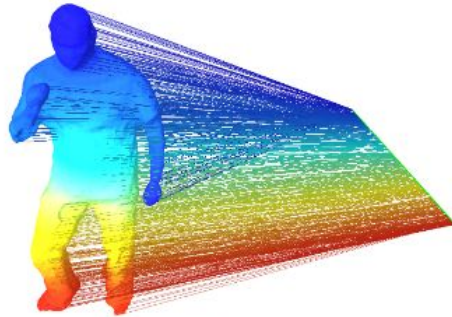


# Embedding spectral

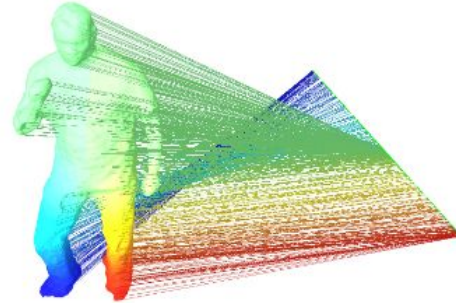
vecteur de "Fielder"



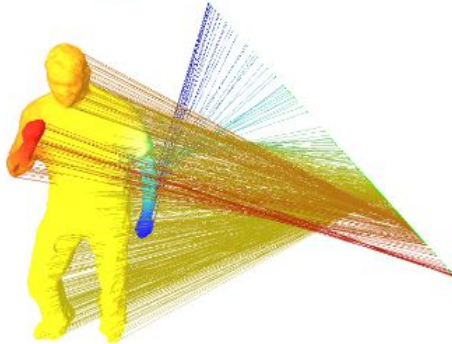
$u_2$



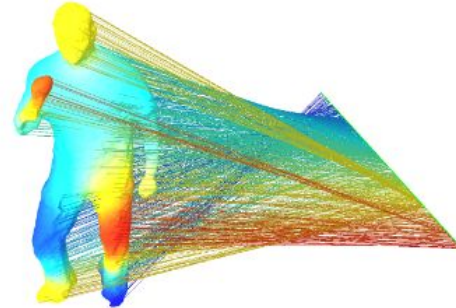
$u_3$



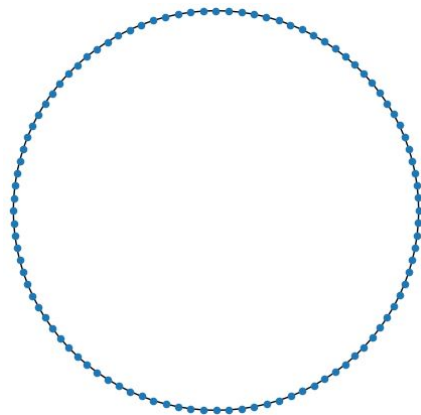
$u_4$



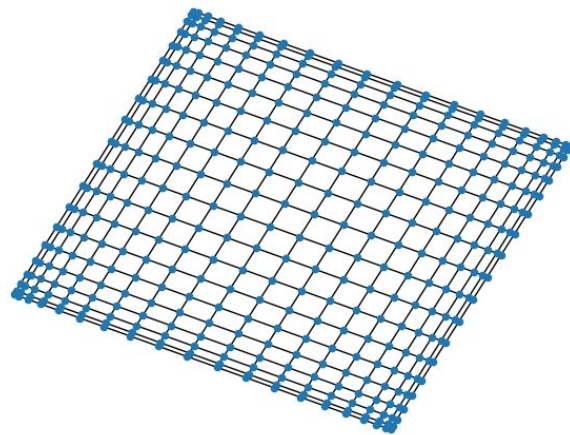
$u_8$



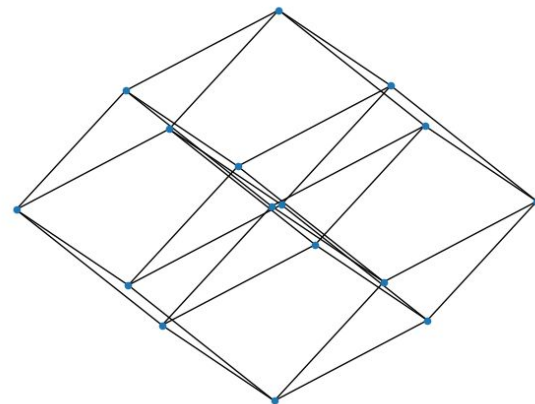
# Embedding spectral comme algorithme de layout en 2 dimensions



layout spectral d'un graphe cycle



layout spectral d'un graphe grid 2D



layout spectral d'un graphe hypercube

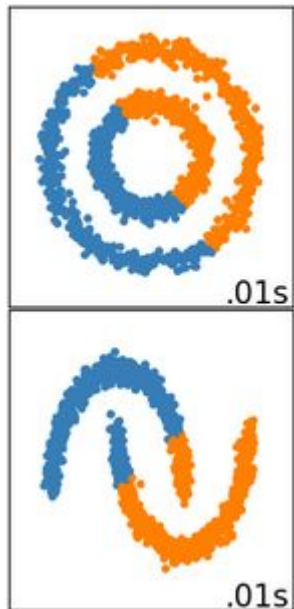
**le layout spectral est une phase d'initialisation d'un grand nombre d'algorithmes**  
(ex : UMAP, Spectral Clustering, ...)

## Clustering spectral

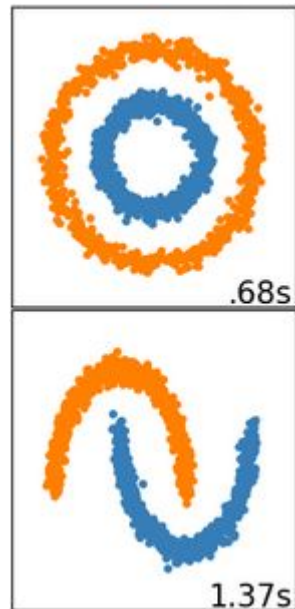
Soit un nuage de points  $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$

- construire une matrice de similarité :  $S_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2)$   
(interpréter la matrice de similarité comme matrice d'adjacence)
- construire la matrice laplacienne du graphe
- construire un embedding spectral
- appliquer l'algorithme **k-means++** sur les embeddings

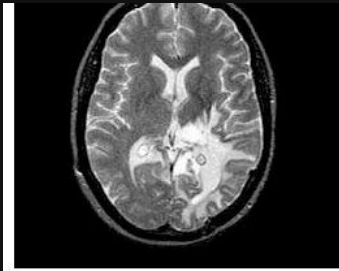
k-means++



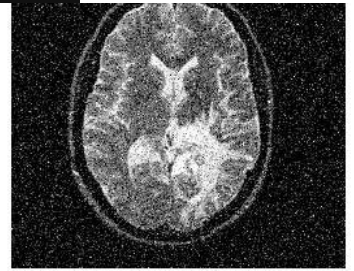
spectral clustering



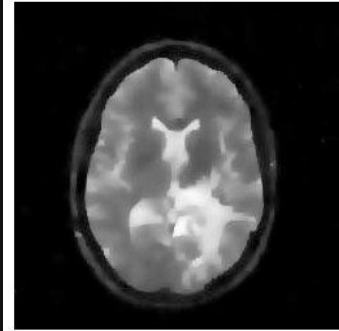
# Applications : Denoising



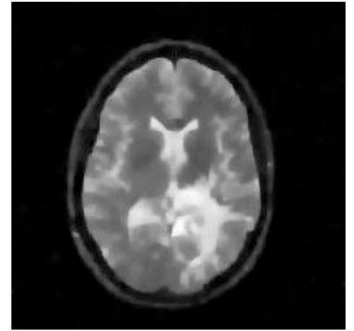
(a) Original image



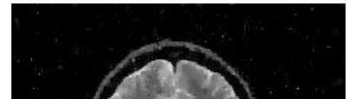
(b) Noisy image SNR = 3.91 db



(c) Huber flow



(d) Entropic flow



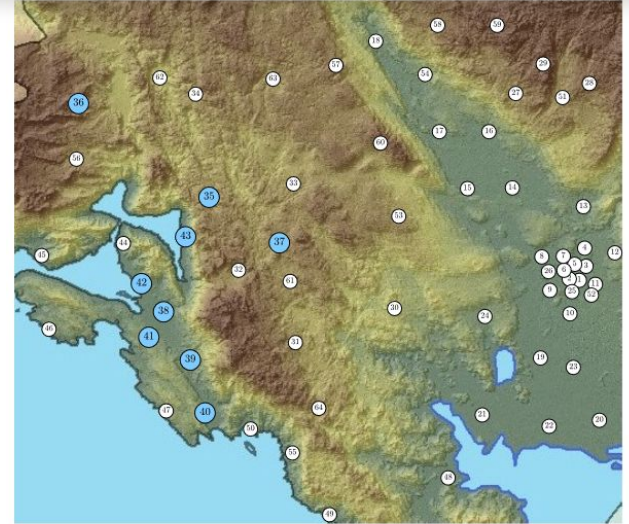


# Contexte du problème

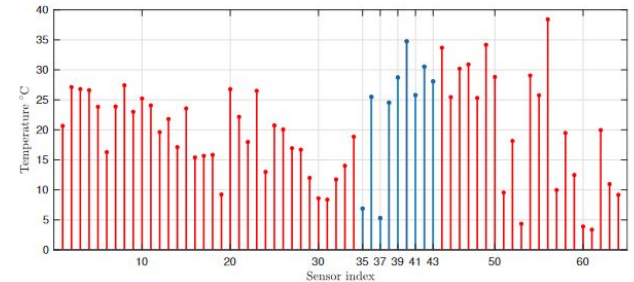
- Imaginons une installation constituée de plusieurs capteurs de température
- Chaque capteur reçoit un signal :
$$x(n) = s(n) + \epsilon(n)$$
où  $s(n)$  est la température réelle et  $\epsilon(n)$  un bruit gaussien

**Le traitement du signal classique requiert les températures rangées sur une ligne**

**Problème : un tel ordre n'existe pas pour notre installation**



a)

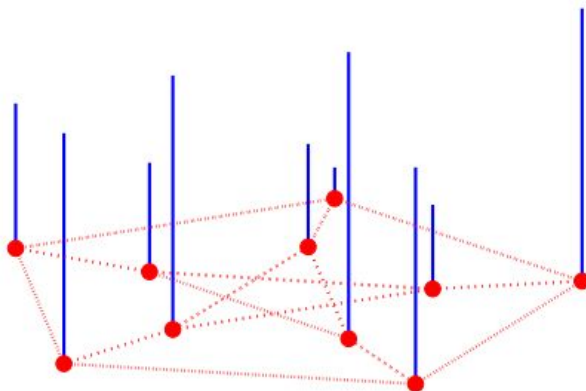


## Signal sur un graphe

- un signal sur un graphe est une fonction sur l'ensemble de ses noeuds
- une fonction sur  $V$  peut-être représentée par un vecteur :

$$f : V \rightarrow \mathbb{R} \equiv X \in \mathbb{R}^n$$

- exemples de fonctions sur les graphes : degré, pagerank, nombre de tweets, etc.



- la forme quadratique d'un signal sur un graphe :

$$E_x = xLx^T = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n A_{ij} (x(i) - x(j))^2$$

- la forme quadratique mesure la “**smoothness**” d'un signal :

le vecteur constant est toujours le signal le plus lisse :  **$\mathbf{1}^T \mathbf{L} \mathbf{1} = 0$**

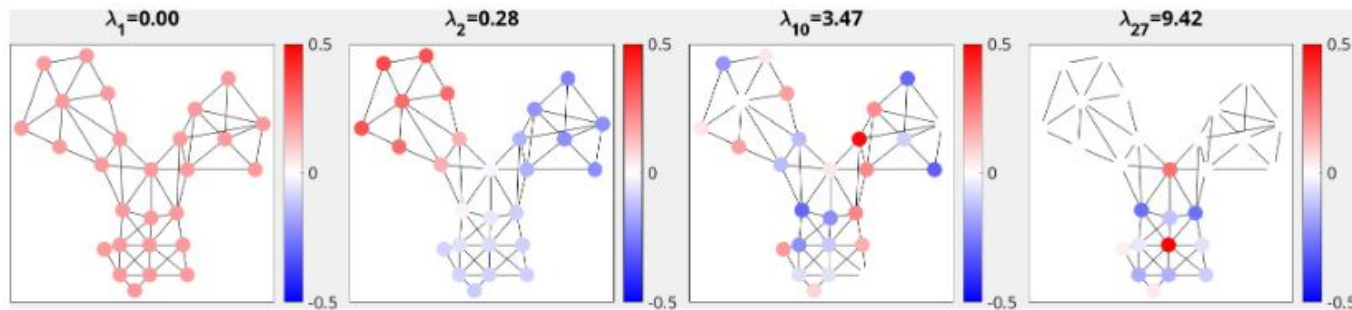
- on peut assimiler les vecteurs propres aux harmoniques fondamentales en traitement du signal classique. Soit  **$\mathbf{L} = \mathbf{U} \mathbf{\Lambda} \mathbf{U}^{-1}$**  et  **$\mathbf{U}$**  orthonormale :
  - $\mathbf{X} = \mathbf{U}^{-1} \mathbf{x}$**  est la transformée de Fourier du signal sur le graphe :  
elle exprime le signal dans la base des harmoniques
  - $\mathbf{x} = \mathbf{U} \mathbf{X}$**  est la transformée de Fourier inverse :  
elle ramène le signal de la base harmonique vers la base initiale

# Transformée de Fourier sur les graphes

- les premières valeurs propres de  $L$  correspondent aux basses fréquences. Avec  $x_i$  le  $i^{\text{ème}}$  vecteur propre :

$$\frac{x_i^T L x_i}{x_i^T x_i} = \lambda_i$$

- une valeur propre élevée implique donc une valeur élevée de la forme quadratique, ie. une fonction qui varie rapidement entre voisins



- $\mathbf{x}^T \mathbf{L} \mathbf{x}$  est une mesure du “lissage” du signal
- on peut donc minimiser sa valeur en lui retranchant une fraction de sa dérivée  $2\mathbf{L}\mathbf{x}$ . On obtient ainsi la procédure itérative :

$$x_{p+1} = x_p - \alpha L x_p$$

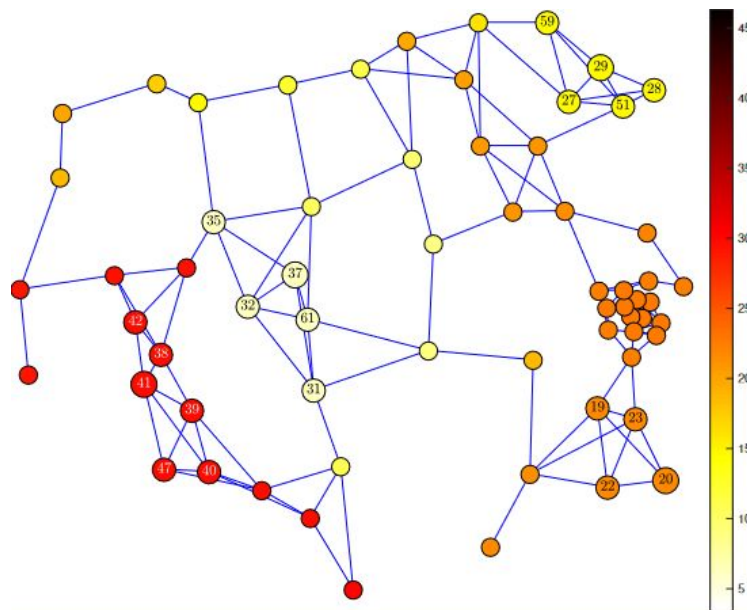
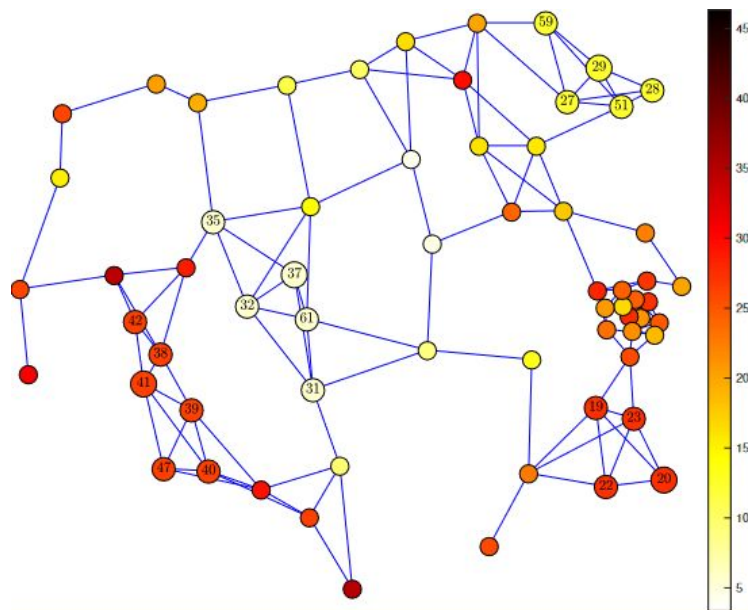
- pour éviter de converger vers une valeur constante, on alterne avec :

$$x_{p+2} = x_{p+1} + \beta L x_{p+1}$$

- **on obtient finalement l'algorithme de Taubin :**

$$x_{p+2} = (I + \beta L)(I - \alpha L)x_p$$

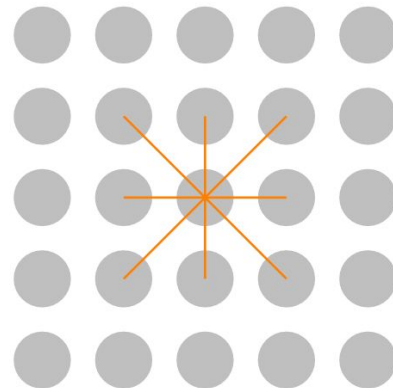
# Algorithme de Taubin



- une image peut être interprétée comme un graphe : chaque pixel possède 8 pixels voisins (sauf les bords)
- chaque edge  $(u, v)$  est pondéré par la similarité entre les deux couleurs de **u** et **v**
- appliquer l'algorithme de Taubin

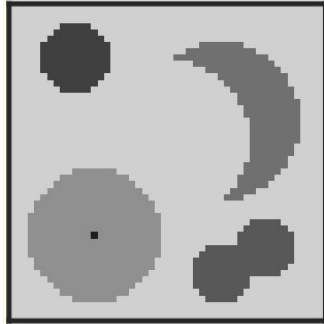
Digital Image

8-neighborhood :  $3 \times 3$

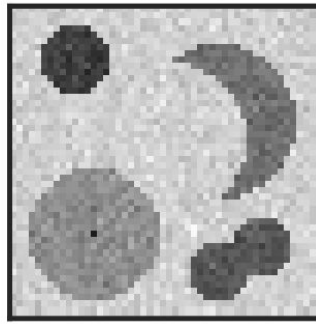


# Image denoising

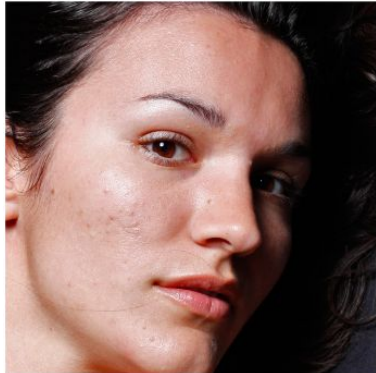
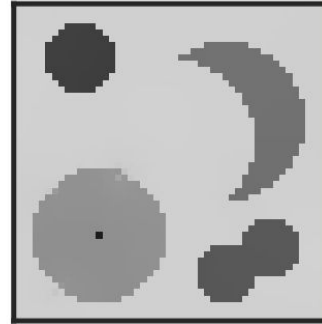
Original image



Noisy image



Graph filtered image





# Applications : GraphWave (role identification)

## Learning Structural Node Embeddings via Diffusion Wavelets

Claire Donnat, Marinka Zitnik, David Hallac, Jure Leskovec

Stanford University

{cdonnat,marinka,hallac,jure}@stanford.edu

### ABSTRACT

Identifying different parts of a graph can have similar structural roles within their local network topology. The identification of these roles provides key insight into the organization of networks and can be used for a variety of machine learning tasks. However, learning structural representations of nodes is a challenging problem and it has typically involved manually specifying and learning topological features for each node. In this paper, we develop GraphWave, a method that represents each node's network neighborhood via a low-dimensional embedding by leveraging heat diffusion patterns. Instead of training on hand-selected topological features, GraphWave learns these embeddings in an unsupervised manner. We mathematically prove that nodes with similar network neighborhoods will have similar GraphWave embeddings even if these nodes may reside in very different parts of the network. GraphWave runtime scales linearly with the number of nodes. Experiments in a variety of different settings demonstrate GraphWave's real-world potential for capturing structural information about the nodes can be used for a variety of tasks, including as input to machine learning problems, or even to identify key nodes in a system (principal "influencers" in a social network, critical hubs in contagion graphs, etc.).

### CONCEPTS

**Tasks** → Topology analysis and generation; • **Computational methodologies** → Kernel methods; Learning latent representations; **Spectral methods**; Cluster analysis; Motif discovery; **Application systems** → Clustering; Nearest-neighbor search; Recommendation systems

### Reference Format:

Donnat, Claire, Marinka Zitnik, David Hallac, Jure Leskovec. 2018. Learning Node Embeddings via Diffusion Wavelets. In *KDD '18: The 24th ACM SIGMOD Conference on Knowledge Discovery & Data*, August 19–23, 2018, London, United Kingdom. ACM, New York, NY, USA. <https://doi.org/10.1145/3219819.3220025>

### INTRODUCTION

Learning structural representations of nodes is a challenging problem and it has typically involved manually specifying and learning topological features for each node. In this paper, we develop GraphWave, a method that represents each node's network neighborhood via a low-dimensional embedding by leveraging heat diffusion patterns. Instead of training on hand-selected topological features, GraphWave learns these embeddings in an unsupervised manner. We mathematically prove that nodes with similar network neighborhoods will have similar GraphWave embeddings even if these nodes may reside in very different parts of the network. GraphWave runtime scales linearly with the number of nodes. Experiments in a variety of different settings demonstrate GraphWave's real-world potential for capturing structural information about the nodes can be used for a variety of tasks, including as input to machine learning problems, or even to identify key nodes in a system (principal "influencers" in a social network, critical hubs in contagion graphs, etc.).

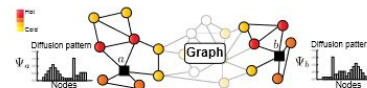


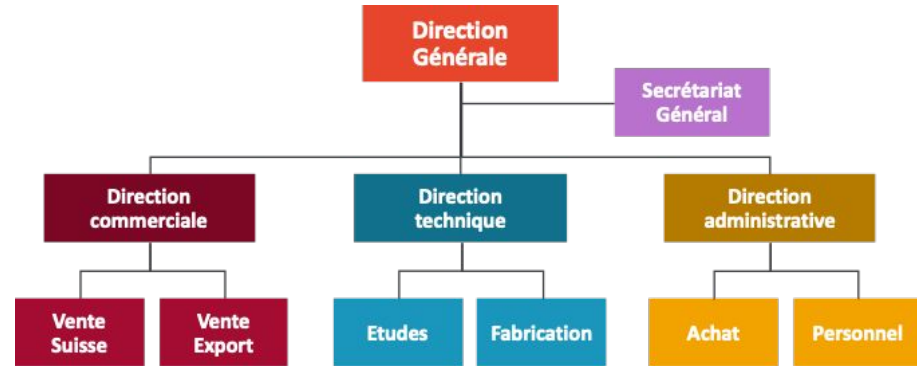
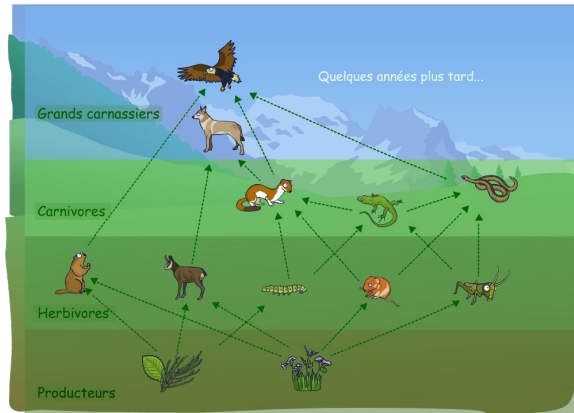
Figure 1: Nodes  $a$  and  $b$  have similar structural roles even though they are distant in the graph. While the raw spectral graph wavelets of  $a$  and  $b$  might be very different, we treat them as probability distributions and prove that the distributions of wavelet coefficients of structurally similar nodes are indeed similar.

notions [9, 12–14, 20, 24, 26, 35], which assume some measure of “smoothness” over the graph and thus consider nodes residing in close network proximity to be similar. Such structural role information about the nodes can be used for a variety of tasks, including as input to machine learning problems, or even to identify key nodes in a system (principal “influencers” in a social network, critical hubs in contagion graphs, etc.).

When structural roles of nodes are defined over a discrete space, they correspond to different topologies of local network neighborhoods (e.g., node on a chain, center of a star, a bridge between two clusters). However, such discrete roles must be pre-defined, requiring domain expertise and manual inspection of the graph structure. A more powerful and robust method for identifying structural similarity involves learning a continuous vector-valued structural embedding  $\chi_a$  of each node  $a$  in an unsupervised way. This motivates a natural definition of structural similarity in terms of closeness of topological embeddings: For any  $\epsilon > 0$ , nodes  $a$  and  $b$  are defined to be  $\epsilon$ -structurally similar with respect to a given distance if  $\text{dist}(\chi_a, \chi_b) \leq \epsilon$ . Thus, a robust approach must introduce both an appropriate embedding and an adequate distance metric.

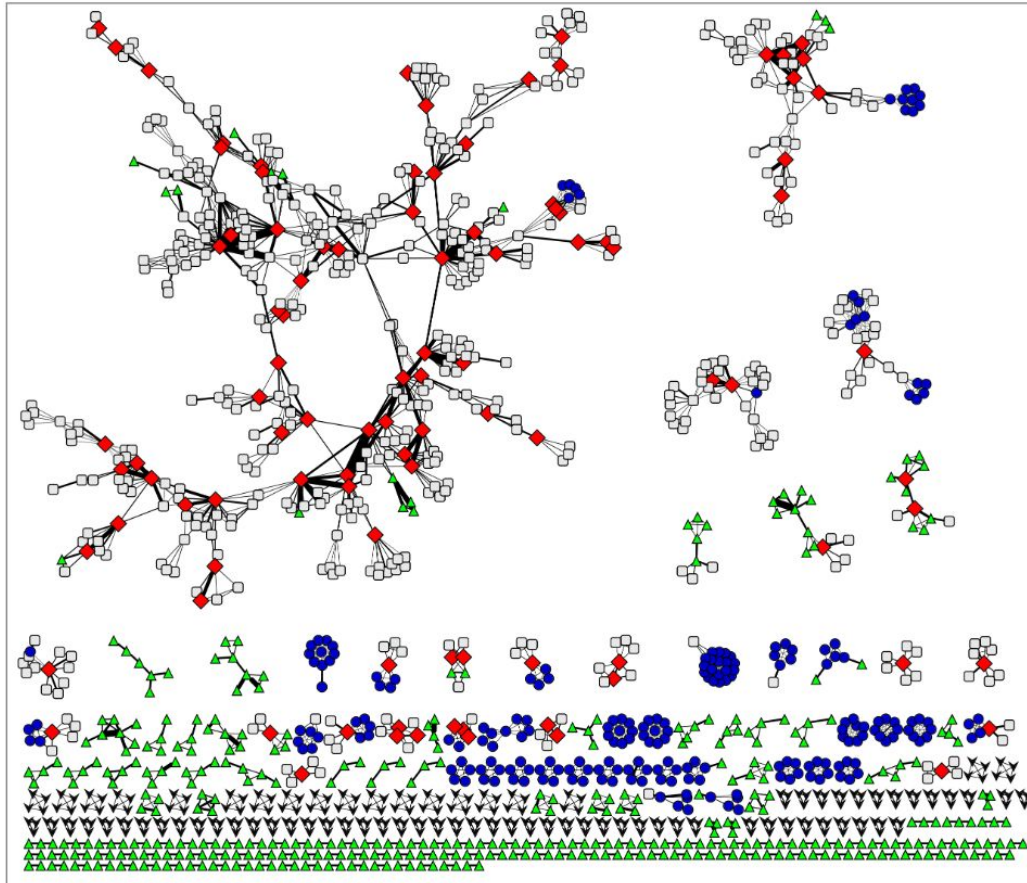
# Que sont les rôles ?

- Les rôles sont les “fonctionnalités” structurelles des noeuds d’un réseau :
  - le rôle des espèces dans les écosystèmes
  - le rôle des individus au sein d’une entreprise



- Les rôles se mesurent structurellement :
  - centres d'étoiles, membres d'une clique, noeuds périphériques, etc.

# Que sont les rôles ?



## Réseau de citations scientifiques

Légende :

- ◆ centres d'étoiles
- membres d'une clique
- ▲ noeuds périphériques

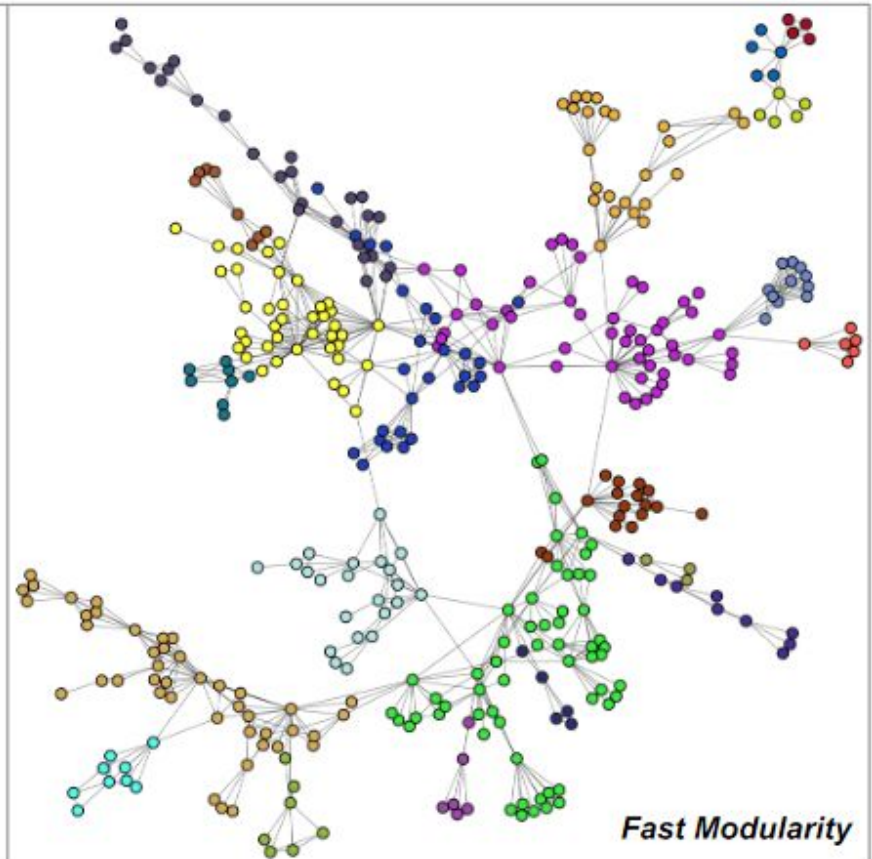
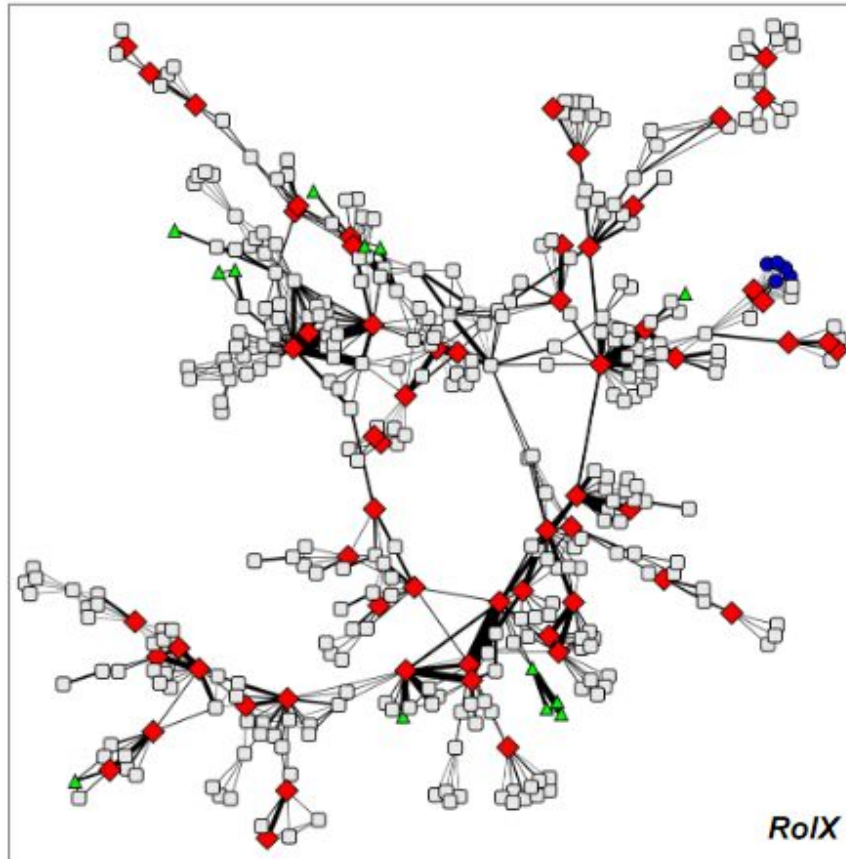
# Les rôles ne sont pas des groupes

- un rôle est une collection de noeuds qui ont des positions similaires dans un réseau :
  - les rôles sont basés sur la similarité topologique
  - ils sont différents des communautés qui sont basés sur l'adjacence, la proximité ou l'atteignabilité
- les rôles et les communautés sont complémentaires :
  - rôles : élèves, enseignants, administration
  - communautés : IASD, MIAGE, I2D, ...

**Les noeuds qui ont un même rôle ne sont pas nécessairement en contact direct : ils peuvent être à des endroits diamétralement opposés dans un réseau.**

**C'est cette propriété qui explique la difficulté de leur identification.**

## Les rôles ne sont pas des groupes



## Pourquoi les rôles sont importants ?

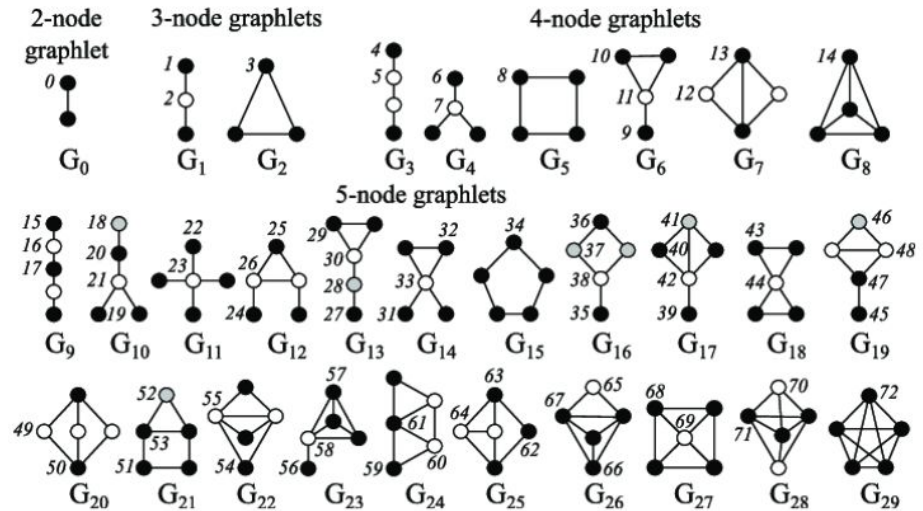
- **Requêter un rôle** : identifier les individus qui ont un comportement similaire
- **Outliers** : identifier les individus ayant un comportement inhabituel
- **Dynamique** : identifier des changements inhabituels dans les comportements
- **Résolution d'identité** : désanonymiser des individus dans un autre réseau
- **Comparaison entre réseaux** : calculer la similarité entre deux réseaux

# Méthode traditionnelle

- Créer un vecteur pour chaque noeud **u** contenant à l'index **i** le nombre du **i<sup>ème</sup>** graphlet que **u** touche
- Problème : **NP-difficile**

Les graphlets sont l'ensemble exhaustif des sous-graphes non-isomorphes d'une taille donnée.

Pour  $n = 3, 4, 5, \dots, 10$ , il y a 2, 6, 21, ..., 11716571 graphlets



## Intuition :

- imaginons que l'on chauffe un noeud dans un réseau (non-orienté)
- la chaleur va se répandre progressivement dans le graphe
  - les noeuds seront chauffés avec une intensité variable
  - la température va évoluer dans le temps différemment pour chaque noeud

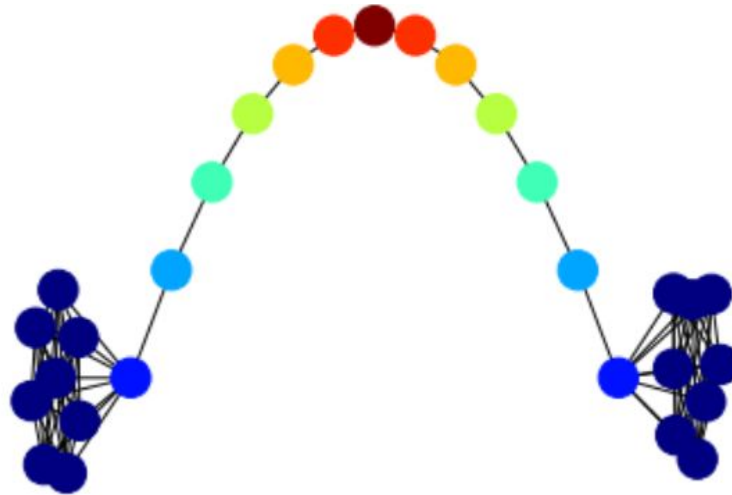
## Hypothèse :

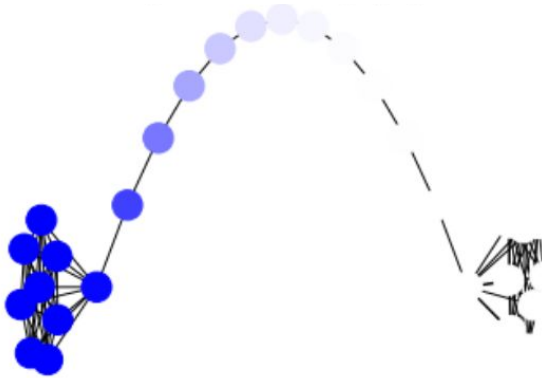
**Les noeuds qui ont des rôles similaires chauffent leur entourage de façon similaire**



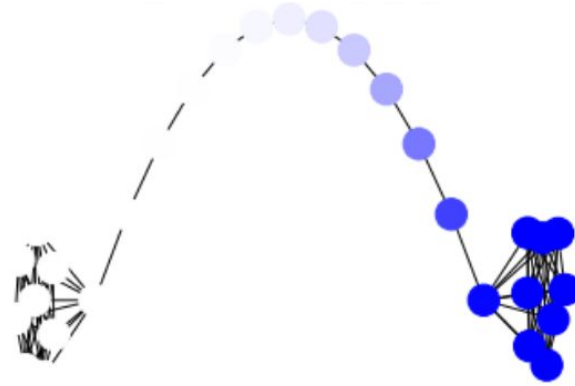
## Barbell Graph :

- deux graphes complets joints par une chaîne
- les noeuds sont colorés en fonction de leur rôle structurel (par symétrie)



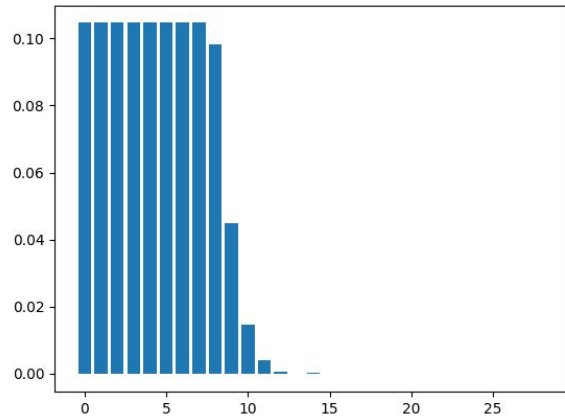


origine de la diffusion de chaleur :  
un noeud en bas à gauche



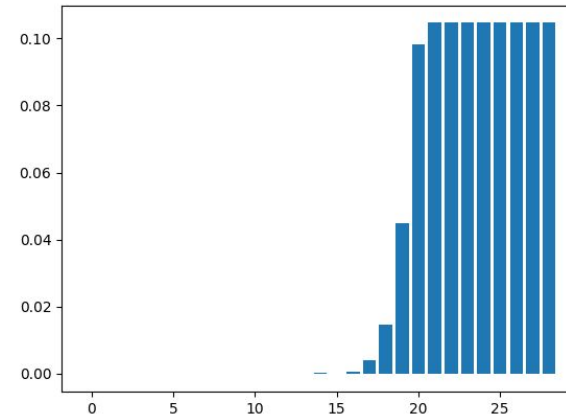
origine de la diffusion de chaleur :  
un noeud en bas à droite

**histogramme de la chaleur  
dans le réseau à  $t=1$**



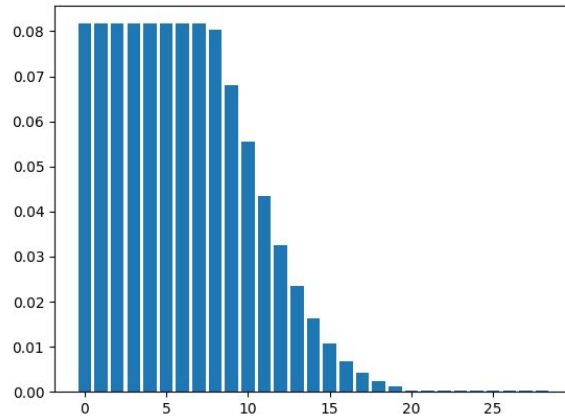
**origine de la diffusion de chaleur :  
noeud en bas à gauche**

**histogramme de la chaleur  
dans le réseau à  $t=1$**



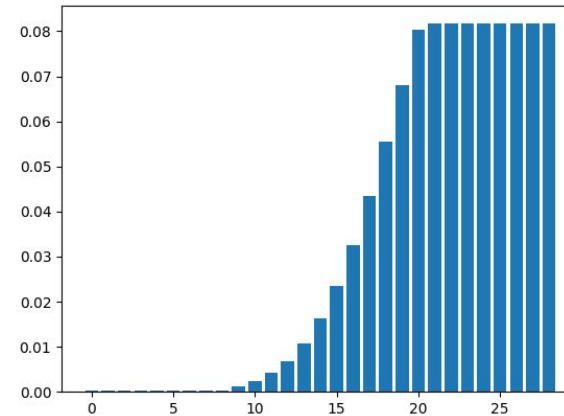
**origine de la diffusion de chaleur :  
noeud en bas à droite**

histogramme de la chaleur  
dans le réseau à  $t=10$



origine de la diffusion de chaleur :  
noeud en bas à gauche

histogramme de la chaleur  
dans le réseau à  $t=10$



origine de la diffusion de chaleur :  
noeud en bas à droite

## Similarité structurelle :

- deux noeuds sont structurellement similaires **si leurs histogrammes ont les mêmes valeurs** (*note : nul besoin de relever les valeurs à plusieurs temps  $t$ , car il n'y a pas/peu de gain d'information*)
- on souhaite créer un embedding pour chaque noeud qui respecte cette propriété

**Problème : comment comparer deux histogrammes ? Comment les “compresser” dans un vecteur de taille fixe ?**

**~~Solution : faire la moyenne des valeurs~~ non ! Car ils émettent tous une même quantité de chaleur, et la chaleur se conserve dans le temps**

## Fonction caractéristique :

- interpréter les valeurs de chaleur comme le résultat d'une variable aléatoire
- la fonction caractéristique permet de **caractériser parfaitement une distribution** (elle contient l'ensemble de ses moments statistiques)
- dans le cas discret, la fonction caractéristique vaut :

$$\phi_X(t) = \mathbb{E}[e^{iXt}] = \sum_x p(X = x) e^{ixt}$$

- dans le cas de la distribution de la chaleur, c'est une simple moyenne :

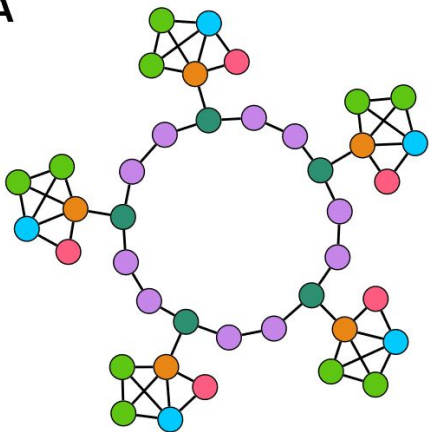
$$\phi_X(t) = \frac{1}{n} \sum_{k=1}^n e^{ix_k t}$$

**attention : pas le même t ! Ici t est le paramètre de la fonction caractéristique**

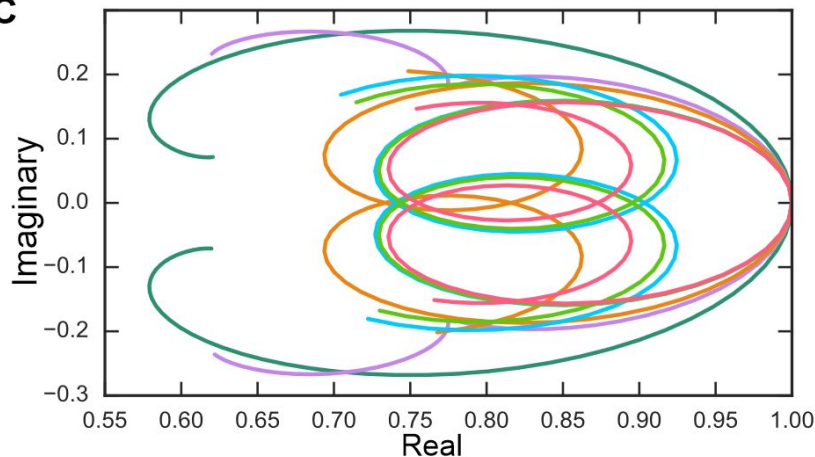
# Une astuce qui devient très populaire !

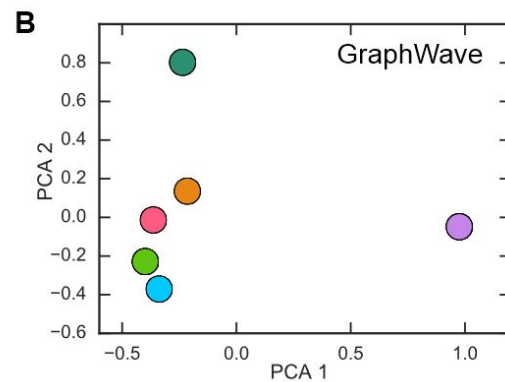
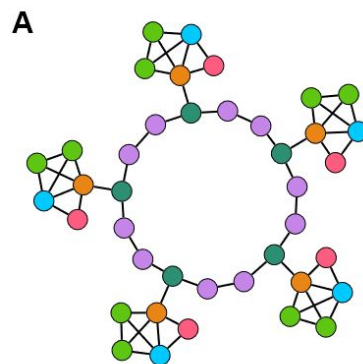
- la fonction caractéristique de chaque noeud **a sa propre trajectoire dans le plan complexe**
- une même trajectoire indique **un même rôle structural**
- on crée un embedding en échantillonnant la fonction caractéristique à différents **t**, et **en concaténant les parties réelles et imaginaires** :  $[\text{Re}(\varphi(t_k)), \text{Im}(\varphi(t_k))]$
- pour **d** échantillons, le vecteur est donc de taille **2\*d**

A

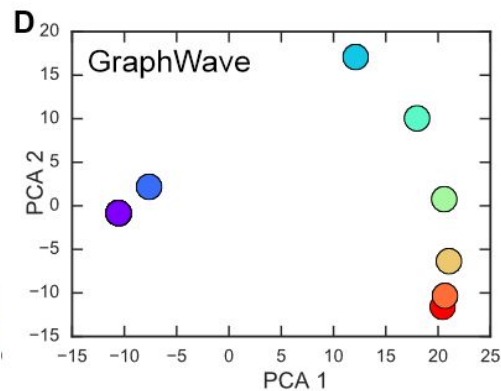
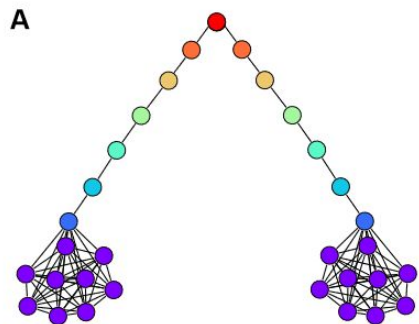


C





Projection en 2D  
des embeddings via PCA





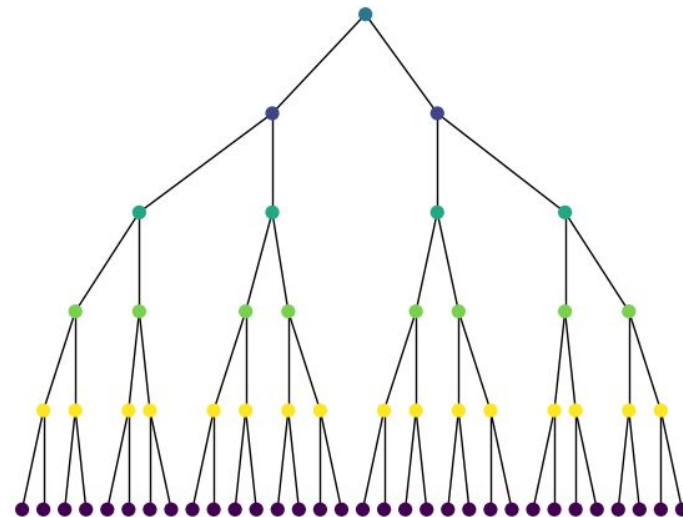
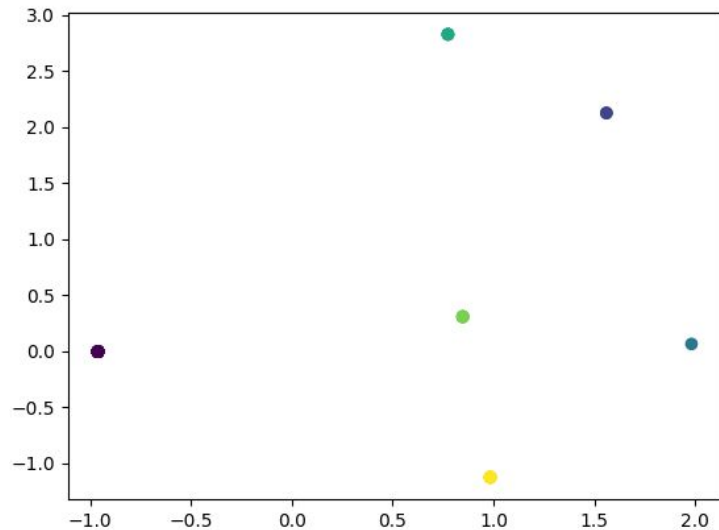
## GraphWave est à l'état de l'art mais...

- ...ne fonctionne que sur des graphes non-orientés (laplacien symétrique)
- ...est assez coûteux à calculer (diagonalisation)
- ...n'a aucune implémentation publique correcte (à ma connaissance)

## Ma proposition pour l'étendre à des graphes orientés : “rolewalk” ?

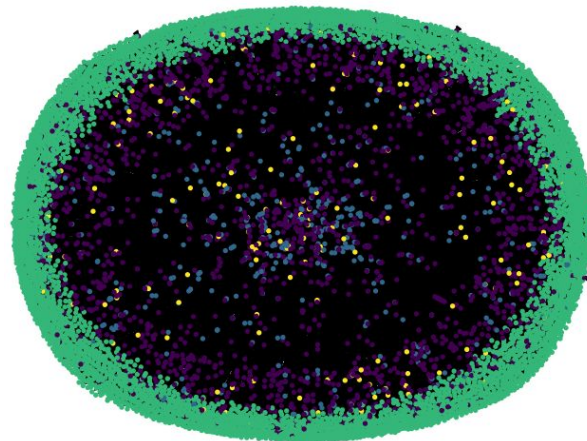
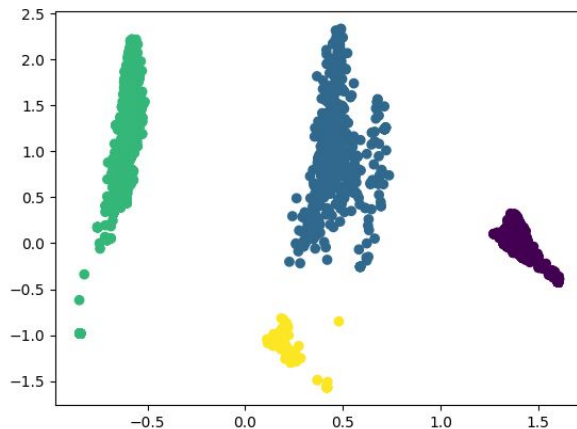
- se baser sur les random walks : établir un histogramme de la probabilité de visiter un noeud du graphe après une marche aléatoire (avec boucle) de longueur  $k$
- “compresser” l'histogramme avec la même astuce (échantillonnage de la fonction carac.)
- avantages :
  - calcul théorique très rapide (ne requiert que le calcul des puissances de  $\mathbf{A}$ )
  - calcul approximé parallélisable (via simulation réelle de marches aléatoires)

# “Rolewalk”



1 rôle identifié pour chaque “étage” d’un arbre binaire

# “Rolewalk”



## 4 rôles identifiés sur le graphe des retweets Twitter (requête “Intelligence artificielle”)

- **les super-influenceurs** : sont très retweetés, retweetent occasionnellement (ex: microsoft, elonmusk)
- **les créateurs d’information** : sont souvent retweetés, ne retweetent jamais (ex: AFPfr, BFM)
- **les relais** : transmettent l’information (ex: bfmbusiness, inria\_bordeaux)
- **les consommateurs d’information** : reçoivent l’information mais ne sont jamais retweetés

(calcul 270 fois plus rapide qu’avec GraphWave)

**GraphWave et Rolewalk : implémentons ensemble ces deux méthodes !**