Rapport Projet IMI:
DETECTION DE COMMUNAUTES DANS LES GRAPHES

Arthur BRESNU, Michaël LAPORTE, Romain GERARD, Antoine OLIVIER

14/01/2022

Table des matières

1 Présentation générale du projet et revue de la littérature

1.1 Objectif

L'objectif de notre étude est d'identifier la présence de communautés présentes au sein d'un graphe et d'être capable de regrouper ensemble leurs éléments respectifs (les noeuds du graphe). Les seules données que l'on possède pour délimiter ces communautés sont les arêtes. Celles-ci vont nous permettre de déterminer les communautés latentes qui structure le graphe de manière sous-jacente. Qualitativement, une "communauté" d'un graphe se caractérise par un nombre élevé de connections entre les noeuds qui la constituent. Nous serons donc amenés à étudier la modélisation mathématique de ce problème ainsi que les conditions permettant d'estimer les communautés cachées, d'en parcourir les outils et propriétés, et enfin de s'intéresser aux algorithmes utilisés pour les applications informatiques.

1.2 Modèle SBM (Stochastic Block Model)

Le SBM est un modèle de génération aléatoire de graphe contenant des communautés. Son principe est le suivant :
Soit n le *nombre de sommets*, k le *nombre de communautés*, $\Pi = (\pi_i)_{i\in[k]}$ un *vecteur de probabilité sur* $[k]$ (vérifiant ainsi $\sum_{i=1}^{k} \pi_i = 1$) et $W \in M_{k,k}([0,1])$ (l'ensemble des matrices de taille $k \times k$ à valeur dans $[0,1]$) une *matrice symétrique*, dite *de connectivité*. La paire (X, G) suit la loi $\mathrm{SBM}(n, \Pi, W)$, si X est *un vecteur aléatoire à composantes I.I.D. suivant la loi* Π, G est un *graphe simple non orienté à n sommets où les sommets i et j (avec $i \neq j$) sont connectés avec une probabilité* W_{X_i,X_j} On définit A la *matrice $n\times n$ d'adjacence associée à G* telle que

$$A_{i,j} := \begin{cases} 1 & si \;\; i \text{ et } j \text{ sont connectés} \\ 0 & sinon \end{cases}$$

Pour tout $i \in [n]$, on définit d_i le *degré du noeud i* tel que

$$d_i = \sum_{j=1}^{n} A_{i,j}$$

Ainsi, A est une matrice aléatoire telle que pour $i \neq j$, $A_{i,j}$ suit une loi de Bernoulli de paramètre W_{X_i,X_j}.

Modèle SSBM On peut également définir le *modèle SSBM (Symetric SBM)* comme un modèle SBM où $\Pi = (\frac{1}{k})_{i\in[k]}$ (vecteur de probabilité uniforme) et où W est la matrice symétrique prenant les valeurs a sur la diagonale et b pour tous les autres coefficients.

Concrètement, a représente la *probabilité que deux membres* (noeuds du graphe) *d'une même communauté soient reliés entre eux* (et ce indépendamment de leur communauté d'appartenance, car la valeur a est présente sur toute la diagonale), tandis que b est *la probabilité que deux membres de communautés différentes soient reliés*. Dans la pratique, on aura $a > b$: il est plus probable d'être relié à un noeud appartenant à sa propre communauté qu'à un noeud qui ne l'est pas (cadre *assortatif*).

Ce cas particulier du modèle SBM est intéressant car il est tout d'abord facile à construire (le vecteur Π est uniforme et W est entièrement déterminée par les deux coefficients a et b), mais il présente aussi des propriétés intéressantes qui permettent une interpretation intuitive des phénomènes étudiés (cf par exemple le paragraphe *"Cas du SSBM"* ci-dessous).

On notera ainsi :

$$(X, G) \sim \mathrm{SSBM}(n, k, a, b)$$

un graphe à n noeuds, k communautés, généré selon la distribution $\mathrm{SBM}(n, \frac{1}{k}\mathbb{1}_k, W)$ où $\mathbb{1}_k$ est le vecteur de taille k ayant toutes ses entrées égales à 1 et où W est définie par les entiers, et dont la matrice de connectivité W est générée par les entiers a (sur la diagonale) et b (sur les coefficients non-diagonaux).

1.3 Problèmes de "weak/exact recovery"

"Agreement" On définit l'*"agreement"* comme étant la *fonction qui maximise la correspondance des coordonnées de deux vecteurs $x, y \in [k]^n$* (où k est ici le nombre de communautés et n le nombre total de noeuds) sur l'ensemble des permutations de $[k]$, noté S_k. Ainsi,

$$A(x, y) := \max_{\sigma\in S_k} \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(x_i = \sigma(y_i))$$

Ainsi, s'il existe une permutation σ telle que $\forall i \in [n], x_i = \sigma(y_i)$, alors l'inéquation $A \leq 1$ devient un cas d'égalité. Il existe une interpretation concrète de cette notion qui nous intéresse dans notre étude, celle de l'application au test d'un algorithme cherchant à regrouper les communautés entre elles.

Application au test d'un algorithme Soit $X \in [k]^n$ le *vecteur de communauté* qui associe à un membre i sa communauté d'appartenance $X_i \in [k]$, ainsi que $\widehat{X} \in [k]^n$ la *sortie d'un algorithme cherchant à retrouver les communautés d'un graphe.* Alors $A(X, \widehat{X})$ mesure *l'alignement de l'attribution des communautés par l'aglorithme* $(\widehat{X})$ *par rapport à l'attribution réelle* (X)*.*

L'idée de maximiser cette vraisemblance sur l'ensemble des permutations des communautés S_k prend alors du sens car l'algorithme a pour but d'identifier et rassembler ensemble les membres d'une même communauté, mais l'attribution de l'indice des communautés est purement arbitraire (et s'effectue ainsi à une permutation près). Ainsi, dans le cas idéal où l'algorithme de détection aurait correctement rassemblé tous les membres au sein de même communautés, il existerait une permutation $\sigma_{solution}$ telle que

$$A(X, \widehat{X}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(x_i = \sigma_{solution}(y_i)) = \frac{1}{n} \sum_{i=1}^{n} 1 = 1$$

Exact Recovery A partir de cette définition, on peut maintenant définir la notion d'"exact recovery" dans le cadre d'un graphe SBM. Soit $(X, G) \sim \text{SBM}(n, \pi, W)$ un graphe, alors on dit qu'un algorithme de sortie $\widehat{X}$ a résolu le problème d' "exact recovery" si la probabilité d'avoir correctement partitionné l'intégralité des n éléments tend vers 1 lorsque le nombre de noeuds (de membres) n tend vers $+\infty$, ie si :

$$\mathbb{P}(A(X, \widehat{X}) = 1) = 1 - o_{n \to +\infty}(1)$$

Weak Recovery pour un SBM Le problème de "weak recovery" pour un graphe correspond au *cas où on arrive à formuler une répartition non triviale* (ie meilleure qu'une répartition purement aléatoire) *des communautés.* Cela correspond à vérifier qu'un algorithme qui effectue une partition de $[n]$ en deux parties S et $\overline{S}$ ($\bar{\cdot}$ désignant le complémentaire) est tel que : *il existe deux communautés* $(C_1, C_2) \in [k]^2$ *ainsi qu'un réel* $\epsilon > 0$ tels que "la probabilité d'avoir un *écart* entre les proportions relatives de membres de C_1 et C_2 appartenant à S *supérieur à* ϵ tend vers 1".

Mathématiquement, pour une partition S de $[n]$ issue d'un algorithme portant sur le graphe G, cela se traduit sous la forme suivante :

$$\exists (i, j) \in [k]^2, \exists\ \epsilon > 0\ /\ \mathbb{P}\left(\frac{|\Omega_i \cap S|}{|\Omega_i|} - \frac{|\Omega_j \cap S|}{|\Omega_j|} \geq \epsilon \right) = 1 - o_{n \to +\infty}(1)$$

où Ω_i correspond à l'ensemble des membres appartenant à la communauté i : $\Omega_i = \{m \in [n] | X_m = i\}$ et où $|\cdot|$ correspond au cardinal de l'esemble auquel il s'applique.

On comprend intuitivement qu'atteindre un pareil critère requiert une certaine information sur la structure des communautés du graphe G dans la mesure où une répartition S aléatoire (qui pour chaque sommet allouerait une communauté $i \in [k]$ avec probabilité uniforme $1/k$) aurait tendance à contenir des proportions relatives ($\frac{|\Omega_i \cap S|}{|\Omega_i|}$, $i \in [k]$) égales de membres des différentes communautés.

Cette intuition se démarque d'autant plus dans le...

Cas du SSBM En effet, le critère pour un output $\widehat{X}$ d'un algorithme portant sur un graphe $(X, G) \sim \text{SSBM}(n, k, a, b)$ se traduit ici sous la forme :

$$\exists\, \epsilon > 0 \ / \ \mathbb{P}\left(A(X, \widehat{X}) \geq \frac{1}{k} + \epsilon \right) = 1 - o_{n \to +\infty}(1)$$

Cela peut s'interpréter de la façon suivante : lorsqu'un algorithme affecte à un sommet une communauté aléatoire uniformément, il y a une probabilité de $\frac{1}{k}$ (où k est le nombre de communautés) d'avoir affecté la bonne communauté à ce sommet. On cherche donc ici à avoir un agreement entre X et $\widehat{X}$ strictement supérieur à $\frac{1}{k}$. Le régime du graphe caractérise la quantité relative de connexions dans le graphe.

1.4 Cas sparse, cas dense

Régime d'un graphe Pour un graphe donné, *l'évolution des coefficients de sa matrice de connectivité*, que l'on appelle *régime du graphe*, joue un rôle important dans la mesure où elle peut déterminer si l'on peut espérer résoudre le problème d'*exact recovery* ou non.

Considérons une suite de graphes $(X_n, G_n)_{n \in \mathbb{N}}$ telle que :

$$\forall n, (X_n, G_n) \sim \text{SBM}(n, \pi, \alpha_n W_0)$$

où α_n est le coefficient d'évolution et W_0 est la matrice de connectivité initiale.
Si α_n est indépendant de n, on parle de régime dense.
Si $\alpha_n \sim \frac{ln(n)}{n}$, on parle de régime relativement sparse.
Si $\alpha_n \sim \frac{1}{n}$, on parle de régime sparse.

Le problème d'"exact recovery" pour un modèle $SSBM(n, k, a\frac{ln(n)}{n}, b\frac{ln(n)}{n})$ est résolvable de manière efficace (i.e. en *temps polynomial* en la taille du graphe) si

$$\frac{1}{k}(\sqrt{a} - \sqrt{b})^2 > 1.$$

Sinon il n'est pas résolvable.

Pour le problème de "weak recovery" pour un modèle $SSBM(n, k, \frac{a}{n}, \frac{b}{n})$, on s'intéresse au terme $SNR = \frac{(a-b)^2}{k(a+(k-1)b)}$.

-Pour $k \geq 2$, le problème est résolvable efficacement si et seulement si $SNR > 1$.
-Pour $k \geq 4$, le problème est résolvable en temps non polynomial pour certaines valeurs de SNR inférieure à 1.

Connaitre le régime d'un graphe peut alors s'avérer essentiel car on ne peut espérer résoudre le problème d'*"exact recovery"* pour un graphe sparse. En effet, le caractère connexe du graphe est une condition nécessaire pour pouvoir espérer rentrer dans le cadre de l'*exact recovery*, condition qui est presque surement non satisfaite lorsque le graphe est à régime sparse. [1]

1.5 Aperçu des différentes approches

1ère approche : le minicut Une première approche correspondrait à trouver une partition en k communautés $(C_1, ..., C_k)$ qui viserait à avoir le moins de liaisons entre les communautés et leur complémentaire.

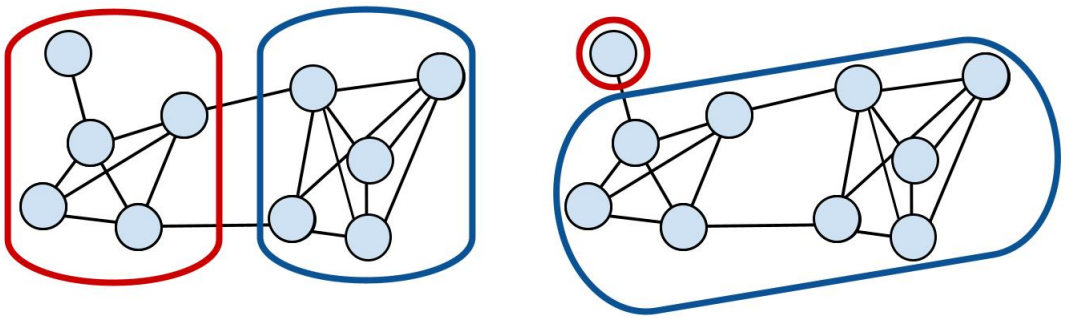


FIGURE 1 – Partition attendue vs partition minimisant la fonction cut

On peut ainsi définir la fonction *cut* qui à une partition $(C_1, ..., C_k)$ de l'ensemble $[n]$ associe le nombre total de liaisons existant entre des noeuds répartis dans deux communautés différentes, soit encore :

$$cut(C_1, ..., C_k) \ := \ \frac{1}{2} \sum_{l=1}^{k} L(C_l)$$

où

$$L(C_l) = \sum_{i \in C_l, \ j \in \overline{C_l}} A_{i,j}$$

est la fonction qui à une communauté C_l associe le nombre de liaisons des membres de cette communauté avec des membres de communautés extérieures ($\overline{C_l}$). Il est par ailleurs logique d'ajouter un facteur $\frac{1}{2}$ étant donné que l'on ne veut pas compter deux fois les mêmes liaisons.

Il s'agirait ainsi *de minimiser la fonction cut sur l'ensemble des partitions des noeuds*. Ce problème du minCut peut être résolu de façon optimale en temps polynomial, mais présente néanmoins un défaut important.

En effet, l'inconvénient majeur de cette approche est qu'elle a tendance à *isoler les noeuds qui sont peu connectés aux autres* (de degré faible). Le schéma ci-après illustre ce phénomène pour 2 communautés : l'image de gauche illustre la partition intuitive attendue, telle que

$$cut(C, \overline{C}) = 2$$

La partition de droite, à priori peu satisfaisante, atteint néanmoins le minimum de la fonction *cut* sur l'ensemble des partitions, de telle sorte que

$$cut(C', \overline{C'}) = 1.$$

Amélioration : le RatioCut Le *RatioCut* consiste à prendre en compte la "taille" des communautés d'une partition de sorte à éviter l'apparition de noeuds isolés comme nous venons de l'observer. Son expression est la suivante :

$$RatioCut(C_1, ..., C_k) = \frac{1}{2} \sum_{l=1}^{k} \frac{L(C_l)}{|C_l|}$$

où $|C_l|$ est le nombre de noeuds contenus dans C_l (son cardinal).

La minimisation du *RatioCut* sur le graphe précédent donne effectivement la partition attendue (de valeur minimale $\text{RatioCut}(C, \overline{C}) = \frac{1}{2}$) !

Malheureusement, ce problème de minimisation n'est plus convexe et est NP-difficile (voir [1]).

Formulation équivalente du RatioCut Afin de grandement simplifier les calculs, nous considérerons dans ce paragraphe le cas où il n'y a que 2 communautés, soit $k = 2$. Nous cherchons ici à trouver une formulation équivalente du problème

$$\min_{C \subset X} RatioCut(C, \overline{C})$$

et de se ramener à un cas convexe qui permette de s'affranchir du caractère NP-difficile.

Laplacien d'un graphe On définit préalablement *la matrice laplacienne* (aussi appelé *Laplacien*) de la matrice d'adjacence d'un graphe (X,G) comme étant la matrice :

$$L = D - A$$

où D est une matrice diagonale dont les coefficients diagonaux sont les degrés de chaque noeud : $\forall i \in [n], D_{i,i} = d_i = \sum_{j=1}^{n} A_{i,j}$.

Le vecteur "f" On définit également le vecteur f tel que :

$$\forall i \in [n], f_i = \begin{cases} \sqrt{\frac{|\overline{C}|}{|C|}}\ , & si\ i \in C \\ -\sqrt{\frac{|C|}{|\overline{C}|}}\ , & sinon \end{cases}$$

qui dépend donc de la partition C (on laissera dans la suite implicite la dépendance de f à une partition C des noeuds du graphe). Ce vecteur vérifie les propriétés $\begin{cases} f \perp \mathbb{1} \\ ||f|| = \sqrt{n} \end{cases}$, où $\mathbb{1} = (1, ..., 1)$.

Reformulation du problème du RatioCut L'intérêt de ces deux outils se dévoile lorsqu'on établit après calcul que :

$$\text{RatioCut}(C, \overline{C}) = \frac{1}{n} f^T L f$$

où L est la matrice laplacienne du graphe et f est le vecteur associé à la partition C que nous venons d'introduire.
On cherche donc une solution à cette nouvelle formulation :

$$\min_{C \subset X} f^T L f \ s.c.$$
$$f_i = \begin{cases} \sqrt{\frac{|\overline{C}|}{|C|}}\ , & si\ i \in C \\ -\sqrt{\frac{|C|}{|\overline{C}|}}\ , & sinon \end{cases},\ f \perp \mathbb{1},\ ||f|| = \sqrt{n}$$

On s'est donc bel et bien ramené à un problème équivalent, qui conserve évidemment le caractère non convexe (qui ici se traduit par le caractère *discret* des coefficients de f, qui ne peuvent prendre que deux valeurs - cf définition de f). Un moyen de facilement relâcher ce problème pour se ramener au cas convexe est de remplacer X (l'ensemble des noeuds) par $\mathbb{R}^n$, ce qui nous donne enfin le problème :

$$\min_{f \subset \mathbb{R}^n} f^T L f$$
$$s.c.\ f \perp \mathbb{1},\ ||f|| = \sqrt{n}$$

Le théorème de Rayleigh-Ritz nous indique que le minimum est atteint en le vecteur propre f associé à la deuxième plus petite valeur propre de la matrice laplacienne L [2]. On peut donc faire l'approximation de la solution du problème de minRatioCut grâce à ce vecteur f. Il s'agit maintenant de regrouper les n noeuds de notre graphe en deux communautés, conformément à notre objectif. Nous avions défini les f comme étant de la forme :

$$\forall i \in [n], f_i = \begin{cases} \sqrt{\frac{|\overline{C}|}{|C|}}\ , & si\ i \in C \\ -\sqrt{\frac{|C|}{|\overline{C}|}}\ , & sinon \end{cases}$$

il suffirait donc de regarder le signe des f_i du vecteur f solution afin de regrouper les membre $i \in [n]$ en deux communautés C_1 et $C_2 = \overline{C_1}$. Concrètement, on regrouperait les $(f_i)_{i\in[n]}$ en deux ensembles A_1 et $A_2 = \overline{A}$, définis tels que

$$A_1 = \{f_i \mid i \in n \text{ et } f_i > 0\} \text{ et } A_2 = \{f_i \mid i \in n \text{ et } f_i < 0\}$$

et on en déduirait :

$$C_1 = \{i \in [n] \mid f_i \in A_1 \ \Leftrightarrow\ f_i > 0\} \text{ et } C_2 = \{i \in [n] \mid f_i \in A_2 \ \Leftrightarrow\ f_i < 0\}$$

Nous avons (enfin) réussi à déterminer une manière efficace de retrouver les communautés voulues ! Enfin presque, car la méthode proposée (se basant sur le signe des f_i) n'est valable que dans le cas (simple) où $k = 2$, c'est-à-dire où l'on considère seulement 2 communautés.

Généralisation pour k > 2 Il est également possible de se ramener à un procédé plus général lorsque l'on considère plus de 2 communautés. On montre que le problème de minRatioCut d'une partition des n noeuds d'un graphe en k communautés $(A_1, A_2, ..., A_k)$ est équivalent au problème de minimisation :

$$\min_{A_1,..,A_k} \operatorname{Tr}(H^T L H) \quad s.c.\ H^T H = I, \ H \text{ définie telle que ci-dessous}$$

où $H \in \mathbb{R}^{n\times k}$ est la matrice définie telle que :

$$\forall j \in [k], \text{ les vecteurs colonnes de } H : h_j = (h_{1,j}, ..., h_{n,j}) \text{ vérifient}$$
$$\forall i \in [n],\ h_{i,j} = \begin{cases} 1/\sqrt{|A_j|} & si\ i \in A_j \\ 0 & sinon \end{cases}$$

(on peut effectivement vérifier que la relation $H^T H = I$ est bien vérifiée avec cette définition de H) [3].

De manière analogue au cas où $k = 2$, on relache le problème en minimisant plutôt sur $\mathbb{R}^{n\times k}$, de telle sorte à ce que le problème devienne :

$$\min_{H\in\mathbb{R}^{n\times k}} \operatorname{Tr}(H^T L H) \quad s.c.\ H^T H = I$$

où le théorème de Rayleigh-Ritz nous indique que la solution est atteinte pour la matrice H dont les vecteurs colonne sont les k premiers vecteurs propres de la matrice laplacienne L.

Cette fois, nous devons effectuer notre *clustering* sur les vecteurs colonnes de notre matrice solution H, qui se fait ici dans l'espace $\mathbb{R}^k$ (et non dans $\mathbb{R}$ comme pour le cas $k = 2$). Nous décrivons la méthode pour effectuer un tel regroupement (grâce à l'algorithme du *k-means*) dans la deuxième partie.

Conclusion On a donc établit comment approximer les regroupements de noeuds en communautés par le procédé de *spectral clustering*, qui se présente ici comme une *version relâchée* (convexe) *du problème de minRatioCut*. L'objectif est donc de calculer la matrice laplacienne du graphe, de trouver les vecteurs propres qui nous intéressent, puis d'en déduire les communautés de chaque sommet en appliquant un algorithme de clustering sur les lignes de la matrice qui concatène en colonnes les vecteurs propres précédemment sélectionnés.

Nous décrivons ce procédé (ainsi que son implémentation informatique) de manière plus détaillée dans la partie suivante.

2 Méthodes spectrales - Exact recovery

2.1 Description détaillée du spectral clustering

Nous avons décrit dans les grandes lignes le principe du *spectral clustering*, nous en décrivons un procédé algorithmique en plus de détails ici. Le principe général du *spectral clustering* repose sur l'idée suivante : si une matrice A est légèrement perturbée par un ajout d'un bruit B, alors le spectre de la matrice $\widetilde{A} = A + B$ est proche de celui de A. Un graphe $(\widetilde{X}, \widetilde{G})$ structuré en k communautés "presque parfaites" (ie dont certains noeuds ont quelques rares liens avec des noeuds d'autres composantes connexes) peut être considéré comme la perturbation d'un graphe (X,G) avec k vraies communautés, c'est-à-dire k composantes connexes auquel on aurait rajouté quelques arêtes. On s'attend alors à ce que le spectre du graphe perturbé $(\widetilde{X}, \widetilde{G})$ (ou d'une matrice calculée à partir de la matrice d'adjacence de ce graphe) soit proche du spectre d'un graphe idéal (X,G) à k composantes connexes, et à ce que les vecteurs propres associés contiennent ainsi de l'information sur les communautés (ou clusters). [2]

L'algorithme de spectral clustering que nous avons employé au cours de ce projet effectue les opérations suivantes :

Entrée : Un graphe (X,G) à n noeuds
1) Calcul de la matrice "outil" choisie *(Laplacien, nonBacktracking, Bethe-Hessian)* à partir de la matrice d'adjacence du graphe
2) Extraction des valeurs propres caractéristiques de la matrice calculée
3) Regroupement des vecteurs propres entre eux grâce à un algorithme d'approximation du K-means
Sortie : La partition des noeuds de X en k communautés $(A_1, ..., A_k)$; ou bien de manière équivalente, une partition des indices des noeuds de $[n]$: $(C_1, ..., C_k)$

Note : afin d'alléger les notations, nous ferons par la suite l'amalgame de confondre l'élément $x_i \in X$ et son indice $i \in [n]$, rendant ainsi licite l'expression : $x_i \in C_l$ (pour $i \in [n]$ et $l \in [k]$) voulant signifier "le i-ème noeud x_i appartient à la communauté C_l"

2.2 Problème du K-means

Le problème du K-means est un outil crucial dans la détection des communautés et sa résolution (approchée) est utilisé dans la phase finale du spectral clustering. A partir d'un ensemble de points donnés et un nombre de communauté prédéfini, la résolution de ce problème permet de déterminer à quelle communauté chaque point appartient.

Le principe du problème du K-means est, pour $x_1, ..., x_n \in \mathbb{R}^d$ des *points à regrouper* et pour k le *nombre de communautés*, de **trouver une partition** $C_1, ..., C_k$ **des** $(x_i)_{i\in[n]}$ **qui minimise la distance des** x_i **aux barycentres** μ_l **(**$l \in [k]$**) de leur communauté** C_l **d'appartenance** .

Mathématiquement parlant, le problème du K-means se traduit sous la forme d'un problème d'optimisation, qui pour $x_1, ..., x_n \in \mathbb{R}^d$ des *points à regrouper*, k le *nombre de communautés* et pour la distance d, prend la forme suivante :

$$\min_{C_1,...,C_k} D_{tot}(C_1, ..., C_k) = \min_{C_1,...,C_k} \sum_{l=1}^{k} \sum_{x_i \in C_l} d(x_i, \mu_l)$$

où les $(\mu_l)_{l\in[k]}$ sont les *barycentres* des communautés $(C_l)_{l\in[k]}$, ie : $\forall l \in [k],\ \mu_l = \frac{1}{|C_l|} \sum_{x_i \in C_l} x_i$ (on laissera

implicite la dépendance des μ_l vis-à-vis des C_l par la suite)

Pour la norme euclidienne (que nous adopterons dans le reste de notre étude), le problème du K-means se reformule de la manière suivante :

$$\min_{C_1,\dots,C_k} \sum_{l=1}^{k} \sum_{x_i \in C_l} ||x_i - \mu_l||^2$$

2.3 Algorithme de Lloyd

Le problème du K-means étant un problème d'optimisation non convexe, il faut faire appel à des algorithmes de résolution approchés pour le résoudre. Pour tenter de se rapprocher de la résolution du problème du K-means vue précédemment, nous avons donc decidé de ce concentrer sur un algorithme : l'algorithme de Lloyd.

Ci-dessous le pseudo-code de l'algorithme de Lloyd :

Algorithm 1 Algorithme d'approximation du problème K-means : Algorithme de Lloyd

Require: $k > 0$ entier représentant le nombre de communautés, la matrice $M \in \mathbb{R}^{n \times d}$ dont les lignes représentent les n points $(x_i)_{i \in [n]} \in (\mathbb{R}^d)^n$ à regrouper.
On pioche au hasard k points différents, que l'on décide ici de choisir dans M (donc k lignes de M). On appelle $barycentre_{ancien}$ cette liste de points

On associe à chaque point de depart l'ensemble C_j des points les plus proches de $barycentre_{ancien}[j]$ (au sens de la norme euclidienne)

for $j \in [1, k]$ **do**
  On calcule le barycentre B_j de C_j
  $barycentre_{nouveau}[j] \leftarrow B_j$
end for

while $barycentre_{nouveau} \neq barycentre_{ancien}$ **do**
  $barycentre_{ancien} \leftarrow barycentre_{nouveau}$
  for $j \in [1, k]$ **do**
    $C_j \leftarrow$ ensemble des points les plus proches de $barycentre_{nouveau}[j]$
    On calcule le barycentre B_j de C_j
    $barycentre_{nouveau}[j] \leftarrow B_j$
  end for
end while
return $(C_j)_{j \in [k]}$

Note : si un point x_l est équidistant de deux barycentres différents μ_i et μ_j (où l'on considère sans perte de généralité que $i < j$), on choisira par convention de le considérer comme étant le plus proche du barycentre d'indice minimal, ici μ_i (car $i < j$).

On illustre cet algorithme par l'exemple suivant :

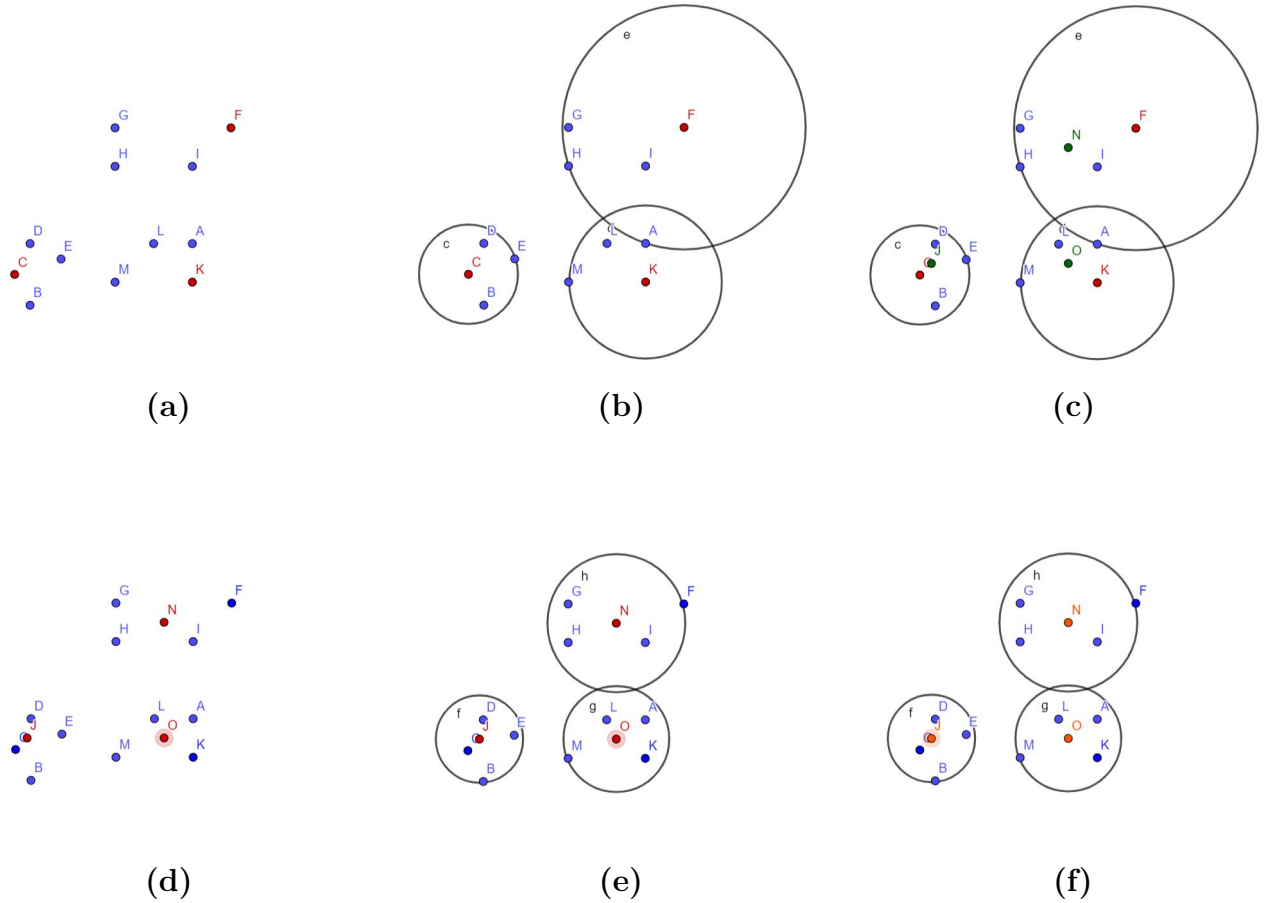


FIGURE 2 – **(a)** Choix (aléatoire) des points pivots
(b) Recherche des points les plus proches du pivot
(c) Calcul du barycentre des points les plus proches de chaque pivot
(d) Les barycentres calculés deviennent les nouveaux pivots
(e) Recherche des points les plus proches des nouveaux pivots
(f) On recalcule les barycentres - ces derniers n'ont pas été modifié, l'algorithme est fini et on a trouvé toutes les communautés

Il est à noter que l'output $(C_1, ..., C_k)$ de cet algorithme est fortement dépendant du choix des barycentres initiaux. De fait, un "mauvais" choix de barycentres initiaux pourrait conduire à un partition $(C_1, ..., C_k)$ peut satisfaisante, c'est-à-dire conduisant à une valeur $D_{tot}(C_1, ..., C_k) = \sum_{l=1}^{k} \sum_{x_i \in C_l} d(x_i, \mu_l)$ bien supérieure à la valeur optimale $D_{tot}(C_1^*, ..., C_k^*)$ du problème du K-means. Pour palier à cette limitation, on exécute l'algorithme plusieurs fois (donc avec des initialisations qui varient aléatoirement).

Si l'on effectue n itérations de l'algorithme, on choisira alors la partition $(C_1, ..., C_k)_i$ (pour $i \in [n]$) telle que

$$D_{tot}(C_1, ..., C_k)_i = \min_{j \in [n]} \{D_{tot}(C_1, ..., C_k)_j\}$$

c'est-à-dire la partition qui minimise la fonction D_{tot} sur l'ensemble des partitions obtenues avec des initialisations différentes.

2.4 Propriétés du Laplacien

Nous avons montré dans la partie **1.5** que le problème de minRatioCut pouvait être reformulé en utilisant une formule faisant appel au *Laplacien* de la matrice d'adjacence. Il semblerait donc naturel de chercher à appliquer le procédé de *spectral clustering* en choisissant la matrice laplacienne du graphe comme la matrice "outil" (étape **1)** du procédé de la section **2.1**). Pour se faire, une étude préalable de la matrice laplacienne s'impose, afin d'en déduire des propriétés utiles et de bien comprendre comment on peut exploiter les vecteurs propres du Laplacien pour la détection de communautés.

Premières propriétés utiles Soit (X,G) un graphe de matrice d'adjacence A. En reprenant la définition du Laplacien introduite dans la partie **1.5**, on peut établir la propriété suivante :

$$\forall f \in \mathbb{R}^n, \quad f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} A_{i,j}(f_i - f_j)^2$$

On peut alors directement en déduire deux propriétés supplémentaires :

1) L est *symétrique* et *semi-definie positive* ,

2) 0 est une valeur propre de L associée au vecteur propre $\mathbb{1} = (1, ..., 1)$.

Ces trois résultats nous permettent enfin d'arriver au théorème suivant :

Valeurs propres du Laplacien Soit (X, G) un graphe unidirectionnel de matrice d'adjacence A, dont les coefficients $A_{i,j}$ sont tous positifs ou nuls. Soit L le Laplacien associé à A (tel que défini précédemment). Alors :

1) *la multiplicité k de la valeur propre* 0 *est égale au nombre de composantes connexes $A_1, ..., A_k$ du graphe*

2) *le sous-espace propre associé à la valeur propre* 0 *est engendré par les vecteurs $\mathbb{1}_{A_1}, ..., \mathbb{1}_{A_k}$ représentant ces composantes*

Démonstration : Soit f un vecteur propre associé à 0. Alors :

$$f^T L f = 0 = \frac{1}{2} \sum_{i,j=1}^{n} A_{i,j}(f_i - f_j)^2$$

Or : $\forall (i,j) \in [n]^2, \ A_{i,j} \geq 0$
donc la somme est à termes positifs et ne peut être nulle que ssi tous ses termes sont nuls, ie ssi

$$\forall (i,j) \in [n]^2, \quad A_{i,j} = 0 \text{ ou } A_{i,j} > 0 \text{ et } f_i = f_j$$

Cela implique entre autre que si les noeuds i et j sont reliés par une arrête de poids non nul (ce que se traduit par $A_{i,j} > 0$) alors nécessairement $f_i = f_j$, donc par transitivité, tous noeuds i et j pour lesquels il existe un chemin composé d'arrêtes de poids non nuls ont pour conséquence le fait que $f_i = f_j$.

On montre maintenant que le nombre de composantes connexes est exactement égal à k. En effet, supposons qu'il y ait $m \in \mathbb{N}^*$ composantes connexes, alors (quitte à réordonner les indices des noeuds) on pourrait écrire la matrice Laplacienne de la forme diagonale par bloc :

$$L = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_m \end{pmatrix}$$

Or, L étant diagonale par bloc (sous-espaces stables), le sous-espace propre associé à 0 pour la matrice L est l'espace vetoriel engendré par les vecteurs propres de chaque sous-matrice Laplacienne L_i ($i \in [m]$) complété avec des 0 ailleurs, soit ici les $\mathbb{1}_{A_i} = (0, ..., 0, \underbrace{1, ..., 1}_{L_i}, 0, ...)$ (à noter que le vecteur $\mathbb{1}$ est bien contenu dans ce sous-espace car $\mathbb{1} = \sum_{i=1}^{m} \mathbb{1}_{A_i}$). C'est donc un sous-espace propre (associé à 0) de dimension m, dont la dimension coincide avec la multiplicité de 0, c'est-à-dire k.

Donc *la multiplicité k de la valeur propre 0 est égale au nombre de composantes connexes.*

- Pour $k = 1$: Si $k = 1$ (0 est de multiplicité 1), alors le vecteur propre lui étant associé est nécessairement $\mathbb{1}$, comme nous l'avons vu dans la propriété précédente. Or $\mathbb{1} = (1, ..., 1)$ et est caractérisé par le fait que $\forall (i, j) \in [n]^2, \mathbb{1}_i = \mathbb{1}_j = 1$, ce qui implique donc que le graphe est connexe.

- Pour $k \geq 2$: De manière analogue au raisonnement par l'absurde ci-dessus, on pourrait (quitte à permuter les indices des noeuds) à nouveau représenter L sous forme diagonale par bloc (avec cette fois $m = k$). Dans ce cas, les vecteurs propres sont alors les $(\mathbb{1}_{A_i})_{i \in [n]}$ tels que définis ci-dessus (dont l'espace vectoriel engendré contient toujours $\mathbb{1}$ pour la même raison que citée précédemment).

2.5 Lien avec la méthode de spectral clustering

Considérons le cas idéal où l'on aurait un graphe (X, G) comportant k composantes connexes (communautés) isolées, c'est-à-dire de matrice d'adjacence (et donc de laplacien) de la forme :

$$A = \begin{pmatrix} D_1 & & \\ & \ddots & \\ & & D_k \end{pmatrix} \quad \text{et} \quad L = D - A = \begin{pmatrix} L_1 & & \\ & \ddots & \\ & & L_k \end{pmatrix}$$

où les $(D_l)_{l \in [k]}$ sont des matrices carrées de dimensions respectives notées $(n_l)_{l \in [k]}$. D'après ce que nous avons précédemment établi, les partitions de noeuds $(x_i)_{i \in [1,\ n_1\ ]}$, ... , $(x_i)_{i \in [(n - n_k + 1),\ n]}$ forment des communautés $(A_1, ..., A_k)$ associées aux vecteurs propres $(\mathbb{1}_{A_i})_{i \in [k]}$.

On essayerait donc de faire un clustering sur les vecteurs propres du laplacien associés à la valeur propre 0, ie avec $(\mathbb{1}_{A_i})_{i \in [k]}$ associés aux différentes communautés (ici interpétées comme étant les composantes connexes de A). On appliquerait alors l'algorithme de Lloyd à la matrice :

$$M = (\mathbb{1}_{A_1} | ... | \mathbb{1}_{A_k}) = \begin{array}{c} \\ A_1 \\ \\ \\ \\ \\ A_k \\ \\ \end{array} \begin{array}{c} \\ \left\{ \begin{array}{l} x_1 \\ \vdots \\ x_{n_1} \end{array} \right. \\ \vdots \\ \vdots \\ \left\{ \begin{array}{l} x_{(n - n_k + 1)} \\ \vdots \\ x_n \end{array} \right. \end{array} \begin{bmatrix} 1 & | & (0)_{k-2} & | & 0 \\ \vdots & | & (0)_{k-2} & | & \vdots \\ 1 & | & (0)_{k-2} & | & 0 \\ 0 & | & \cdots & | & \vdots \\ \vdots & | & \cdots & | & \vdots \\ 0 & | & (0)_{k-2} & | & 1 \\ \vdots & | & (0)_{k-2} & | & \vdots \\ 0 & | & (0)_{k-2} & | & 1 \end{bmatrix} \in \mathbb{R}^{n \times k}$$

Les n points $(x_i)_{i \in [n]}$ seraient alors des vecteurs de $\mathbb{R}^k$ dont les coefficients seraient tous nuls, sauf pour le coefficient dont l'indice $l \in [k]$ est également l'indice de la composante connexe A_l à laquelle x_i appartient. Ainsi, pour tout $i \in [n]$:

$$\forall\ l \in [k],\ (x_i)_l = \begin{cases} 1 & \text{si } x_i \in A_l \\ 0 & \text{sinon} \end{cases}$$

Par exemple, on a dans notre cas ici considéré : $x_1 \in A_1$ donc $x_1 = (1, 0, ..., 0)$; ou encore : $x_n \in A_k$ donc $x_n = (0, ..., 0, 1)$.

Il est à noter que dans ce cas idéal, il ne serait pas nécessaire d'effectuer un clustering par approximation du K-means, dans la mesure où les points $(x_i)_{i\in[n]}$ sont égaux lorsqu'ils appartiennent à une même communauté (tous les vecteurs x_i de $\mathbb{R}^k$ appartenant à une même communauté coincident déjà en un même point de $\mathbb{R}^k$, nul besoin de s'embêter à leur appliquer l'algorithme de Lloyd pour les regrouper ensemble alors qu'ils sont déjà égaux !).

Cependant, on ne rencontrera pas dans la pratique de tels cas idéaux où les communautés sont parfaitement séparées comme ici. La matrice d'adjacence du graphe ressemblerait plutôt à

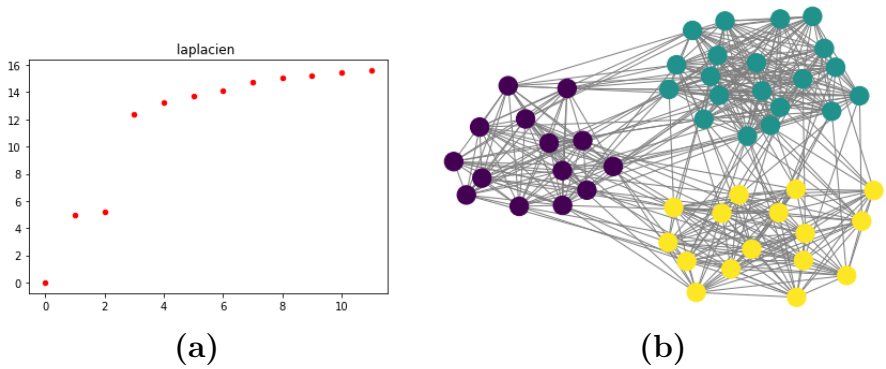
$$\widetilde{A} = A + B$$

où A est la matrice diagonale par bloc introduite dans le cas idéal, et où B est une matrice dite *"de perturbation"* (ou encore *"de bruit"*), donc les coefficients sont arbitrairement petits devant 1. Ce bruit a pour conséquence de transformer la multiplicité k de la valeur propre 0 en k *valeurs propres de norme proche de* 0, que l'on note $(\epsilon_i)_{i\in[k]}$, et qui sont associées à des vecteurs de la forme $\widetilde{x}_i = x_i + \tau_i$ (où x_i est le vecteur précédemment défini et $||\tau_i||_\infty \ll 1$). Bien entendu, *plus le bruitage sera grand* (ie plus les coefficients de la matrice B seront de norme élevée), *plus les vecteurs propres* $(x_l)_{l\in[k]}$ *auront tendance à se disperser dans* $\mathbb{R}^k$ (car $||\tau_i||_\infty$ ne sera plus nécessairement $\ll 1$), ce qui peut rendre leur *clustering* plus difficile -voire même complètement faussé. Plus grave encore, il se peut que l'on ne soit même plus en mesure d'identifier les valeurs propres qui contiennent des informations non-triviales sur la structure des clusters. Nous verrons plus loin que dans le cas sparse, les k valeurs propres du Laplacien qui nous intéressent (qui sont alors non nulles) peuvent être "noyées" parmis les autres valeurs non-informatives. Pour plus de détails sur l'étude (outrement dense et complexe) de l'influence de la norme du bruitage B sur la restitution des communautés, voir [1].

L'application de l'algorithme de Lloyd prend alors tout son sens ici, car il s'agit de regrouper k ensembles de vecteurs de même communautés, qui bien que non égaux reste néanmoins proches entre eux. L'algorithme du *spectral clustering* pour la matrice Laplacienne devient alors :

Entrée : Un graphe (X,G) à n noeuds
1) Calcul de la matrice Laplacienne à partir de la matrice d'adjacence du graphe
2) Extraction des k plus petites valeurs propres et de leurs vecteurs propres associés
3) Regroupement des vecteurs propres entre eux grâce à un algorithme d'approximation du K-means
Sortie : La partition des noeuds de X en k communautés $(A_1, ..., A_k)$

Nous avons appliqué cet algorithme à un graphe (X,G) $\sim$ SSBM($n = 50,\ k = 3,\ a = 0.9,\ b = 0.1$) (on s'est arrangé de sorte à ce que l'on puisse bien appliquer le cadre d'*"exact recovery"* à partir des paramètres du graphe), et il en a résulté :



(a) **(b)**

FIGURE 3 – **(a)** Spectre du Laplacien du graphe **(b)** Restitution des communautés

Notre algorithme a également calculé un *agreement* égal à 1 (restitution parfaite) pour un tel clustering sur ce cas simple.

On peut déjà facilement observer un "saut" entre la 3ème et la 4ème valeur propre du Laplacien, ainsi que la fameuse valeur propre 0 (associée au vecteur $\mathbb{1}$). Plus généralement, ce "saut" entre la k-ème et la $(k+1)$-ème valeur propre est présent lorsque les paramètres du graphe satisfont les conditions du cadre de l'*exact recovery*.

2.6 Calcul de l'erreur

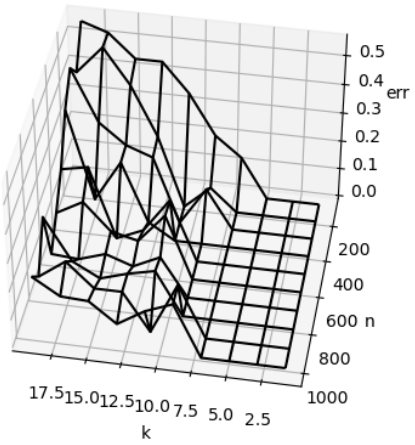


FIGURE 4 – Calcul de l'erreur en fonction de n et k pour $a = 0.9$ et $b = 0.2$

Dans ce cas dense (a et b constants), on observe que quand n croît l'erreur diminue et que l'on atteint même une erreur nulle pour n suffisamment grand - ce qui est logique car on s'est placé dans un cas ou l'exact recovery est possible.

On peut également donner une explication de la croissance de l'erreur en fonction de k. En effet, nous avions évoqué partie **1.4** qu'un critère pour s'inscrire dans le cadre d'*exact recovery* pour un SSBM($n, k, a\frac{ln(n)}{n}, b\frac{ln(n)}{n}$) était d'avoir

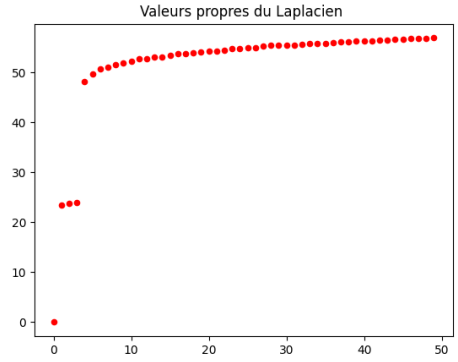
$$\frac{1}{k}(\sqrt{a} - \sqrt{b})^2 > 1$$

Ainsi, lorsque k devient trop grand (pour a et b fixés), on finit éventuellement par sortir du cadre dense et par perdre le caractère connexe du graphe, pourtant nécessaire à la résolution du problème d'*exact recovery*.

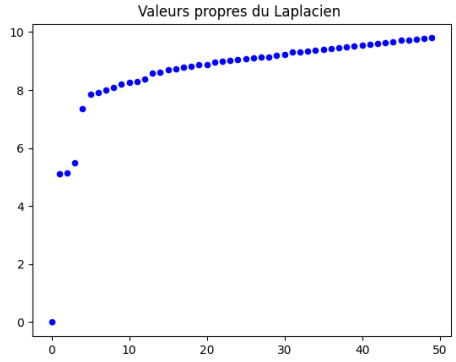
3 Méthodes spectrales - Weak recovery

3.1 Limites de l'utilisation du Laplacien

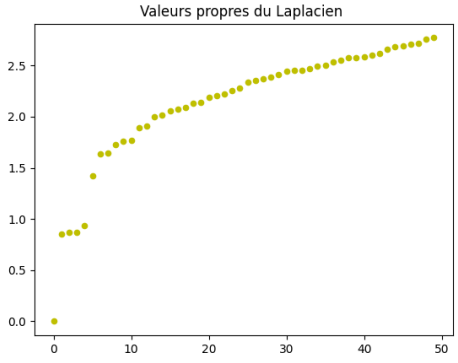
Affaissement des valeurs propres Dans le cas sparce, les valeurs propres du Laplacien sont plus petites et se mélangent aux valeurs dont les vecteurs propres associés contiennent de l'information sur la structure communautaire latente du graphe. L'utilisation du Laplacien est infructueuse pour l'approche par clustering spectrale précédemment introduite.



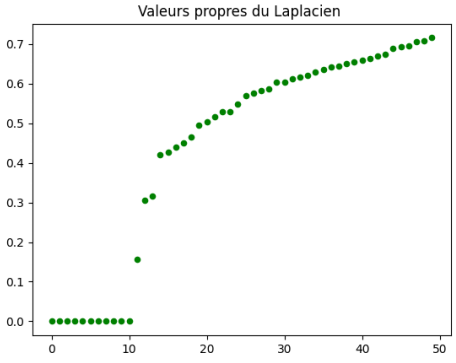
(a) $a = \frac{9}{20}$, $b = \frac{1}{20}$



(b) $a = \frac{9}{40}$, $b = \frac{1}{40}$



(c) $a = \frac{9}{60}$, $b = \frac{1}{60}$



(d) $a = \frac{9}{80}$, $b = \frac{1}{80}$

FIGURE 5 – Représentation des 50 plus petites valeurs propres du Laplacien pour des graphes générés selon une $SSBM(n, k, a, b)$ pour $n = 1000$, $k = 4$ et différentes valeurs de a et b

La figure montre bien que pour a et b suffisamment grands les 4 valeurs propres importantes sont simples à déterminer et que quand a et b diminuent, le palier des valeurs propres sans information s'affaisse ce qui à un certain point empêche de déterminer les valeurs propres qui nous intéressent.

Qualitativement, cela s'explique par la présence presque sûre de noeuds fortement connectés (caractéristique du cas sparse) qui viennent interférer avec les valeurs propres du Laplacien par le phénomène que nous avons illustré. [1]

L'utilisation du spectral clustering sur le Laplacien n'étant pas satisfaisant dans le cas sparce, on s'intéresse à d'autres matrices.

3.2 Non-Backtracking

La matrice Non-Backtracking est une matrice de taille $|\mathcal{E}_d| \times |\mathcal{E}_d|$ où $\mathcal{E}_d$ est l'ensemble des arêtes dirigées du graphe généré. On note (ij) l'arête qui relie le nœud i au nœud j.
On a

$$\forall (ij),(kl) \in \mathcal{E}_d,\ B_{(ij),(kl)} = \delta_{jk}(1-\delta_{il}) = \begin{cases} 1 & si\ \ j = k\ et\ i \neq l \\ 0 & sinon \end{cases}$$

Comme pour le Laplacien, certains de ses vecteurs propres portent de l'information sur les communautés de graphe. On remarque que la taille de la matrice dépend du nombre d'arêtes. De plus, sa construction fait qu'elle ne subit pas "l'affaissement des valeurs propres" que l'on rencontre quand on passe du cas dense au cas sparse et que son spectre est peu sensible aux nœuds ayant beaucoup de connections. Comme son non l'indique ("backtracking" = "revenir sur ses pas"), la matrice nonBacktracking est construite de telle sorte à ce que lorsqu'élevée à la puissance $m-1$, les coefficients de B^{m-1} soient égaux au nombre de marches sur les arêtes de longueur m qui ne bouclent pas. On s'affranchit alors du problème de noeuds fortement connectés qui ont tendance à créer des boucles sur eux-mêmes et qui sont à l'origine du fait que la matrice laplacienne devient inutilisable dans le cas sparse. [1]

La matrice Non-Backtracking semble donc avantageuse dans le cas sparse. Cependant, il y a généralement plus d'arêtes que de nœuds. La matrice Non-Backtracking est donc plus grande, en plus d'être non symétrique ce qui rend la recherche de vecteurs propres plus coûteuse d'un point de vue computationel. Cela nous pousse à nous intéresser à une autre matrice appelée la Bethe-Hessian.

3.3 Bethe-Hessian

La Bethe-Hessian est une matrice de taille $n \times n$ qui dépend d'un paramètre $r \in \mathbb{R}$. Elle est définie par

$$H_r = (r^2-1)I_n + D - rA$$

où $D = Diag((d_i)_{1\leq i\leq n}))$ avec les d_i les degrés des nœuds définis plus tôt et A la matrice d'adjacence.

L'intérêt de H_r est sa symétrie et les relations qui existe entre ses valeurs propres et celles de la matrice Non-Backtracking.
On s'intéresse à différentes valeurs de r pour construire une matrice M à partir de vecteurs propres comme pour le laplacien. Remarquons tout d'abord que pour $r = 1$, on retombe sur la définition du Laplacien classique. Il est donc important de choisir une valeur de r judicieuse afin de ne pas retomber dans le cas de la matrice laplacienne, inutilisable dans le cas sparse. La méthode que l'on a utilisée construit M à partir des vecteurs associées aux valeurs propres négatives de H_r en $r = r_c = \frac{1}{n}\sum_{k=1}^{n} d_k$ et en $r = -r_c$, comme proposé dans [4].

4 Bibliographie

Références

[1] Emmanuel Abbe. *Community Detection and Stochastic Block Models : Recent Developments*. 2017.

[2] Tabea Rebafka. *Analyse Statistique de Graphes*. 2021.

[3] Ulrike von Luxburg. « A Tutorial on Spectral Clustering ».
In : *Statistics and Computing* 17 (2007).

[4] Lorenzo Dall'Amico. *Spectral Methods for Graph Clustering [English]*.
Université Grenoble Alpes, 2021.