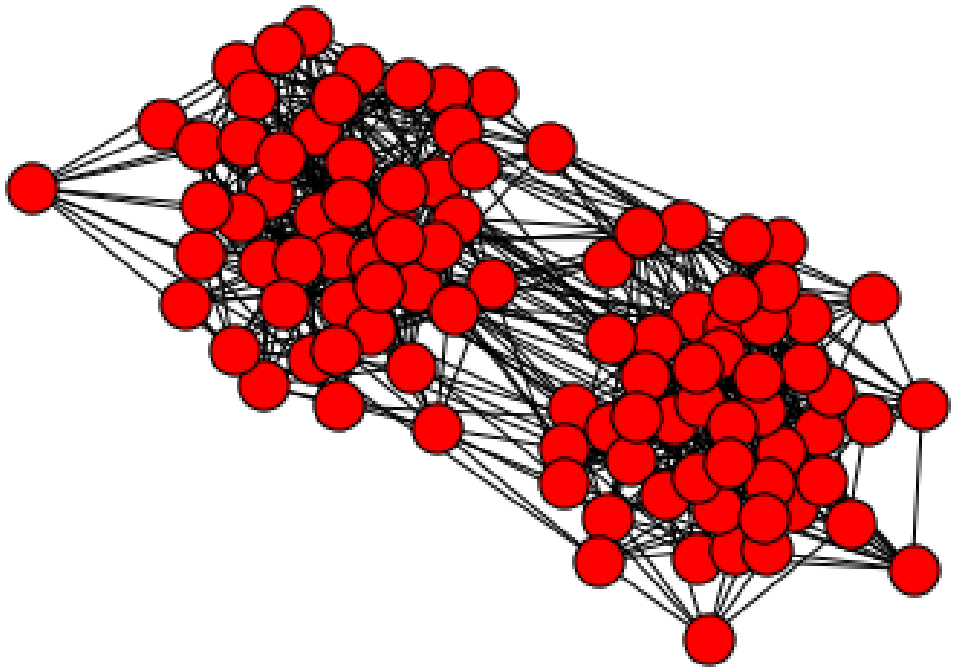
Identification de communautés

Ragel Vincent, Benmahjoub Amir

June 6, 2017



Contents

Introduction

A travers le sujet du modal nous nous sommes intéressés à l'identification de communauté, plus couramment appelé "clustering". L'idée est la recherche d'une segmentation, c'est-à-dire d'une répartition d'individus à travers plusieurs communautés. Ces problématiques sont couramment utilisées dans différents domaines. Par exemple les réseaux d'amitiés dans les réseaux sociaux donnent des exemples très imagés et les entreprises de data-mining regroupent les consommateurs en différents ensembles pour déterminer les publicités à afficher. Le clustering est également utilisé dans de nombreux autres domaines comme la banque, la biologie, etc..

Le clustering fait parti des méthodes de classification non-supervisées. Dans le cadre de notre projet nous allons regrouper les individus en fonction des liens les reliant. Dans la totalité de notre projet nous considérons uniquement deux clusters pour simplifier. Dans le cas d'un graphe non-orienté cela représente un nombre de $\frac{n(n+1)}{2}$ liens. L'augmentation se fait de manière quadratique, si bien que les algorithmes classiques de clustering, comme l'algorithme de k-means ou l'algorithme clustering hiérarchique, sont inadaptés pour le problème. Nous utiliserons par conséquent un algorithme appelé clustering spectral. La méthode consiste à récupérer les vecteurs propres de la matrice d'adjacence pour déterminer les clusters. Deux options sont possibles. Nous verrons dans la suite les avantages et inconvénients de ces deux options.

Notre projet s'intéresse précisément aux probabilités de mauvaises clusterisation de cet algorithme. Malgré la précision des méthodes de clusterisation, des éléments peuvent être mal clusterisés sous certaines conditions. En fonction des propriétés du graphe et des paramètres de départ, la probabilité pour qu'un ou plusieurs éléments se retrouvent assignés dans le mauvais cluster peut être très faible. Nous appliquerons les méthodes vues -en particulier la méthode sampling- dans le cours afin d'étudier ces phénomènes.

Nous verrons dans un premier temps le fonctionnement du clustering spectral et ses dépendances par rapport aux différents paramètres du problème. Dans un second temps, nous nous intéresserons aux différentes façons qui permettent de calculer les probabilités de mauvaises clusterisation. Nous essayerons plus précisément d'adapter l'importance sampling à notre problème .

1 Clustering spectral

1.1 Modélisation du problème

On s'intéresse ici au clustering d'un ensemble $V = (v_1, v_2, ..., v_n)$ d'individu intégrés dans un réseau d'amitiés : on définit alors un graphe $G = (V, E)$ avec E l'ensemble des liens d'amitiés (i.e. $E \subset V \times V$). On supposera que le graphe est non orienté et sans poids : $(v_1, v_2) \in E \Leftrightarrow (v_2, v_1) \in E$. Toute l'information contenue dans ce graphe G est contenue dans la matrice d'adjacence du graphe $A = \left[\mathbf{1}_{(i,j)\in E}\right]_{i,j=1}^{n}$. Ainsi, chaque donnée v_i représentant un individu dans le réseau constitue un vecteur d'attribut à n coefficients qui représente ses liens avec les autres individus. Dans la matrice A, ces vecteurs sont soit les lignes soit les colonnes de la matrice puisqu'elle est symétrique. Plus spécifiquement, on s'intéressera à l'aide de l'observation de la matrice A au clustering en deux classes d'individus C_0 et C_1. Ces deux classes formeront donc deux communautés distinctes. Chaque communauté lie des individus ayant plus de liens entre eux qu'avec ceux de la communauté voisine.

1.2 Stochastic Block Model

Avant de développer la théorie du clustering spectral et de simuler des clustering, il faut pouvoir générer des graphes dont on connait le partitionnement. Ces graphes constitueront donc une base de test fiable pour nos algorithmes puisqu'ils nous permettront de confronter nos résultats au vrai partitionnement. Pour ce faire, nous utilisons ici le Stochastic Block Model.

On considère trois éléments : l'ensemble des individus $V = (v_1, v_2, ..., v_n)$, des clusters souhaités C_0 et C_1 et une matrice carrée symétrique de taille 2 à coefficients dans $[0, 1]$. On note G le graphe tel que :

$$\forall (v, w) \in V \quad \forall (i, j) \in \{0, 1\}^2 \mid (v, w) \in C_i \times C_j \Rightarrow \mathbb{P}((v, w) \in E) = B_{i,j}$$

Notons que les arrêtes existent de façon indépendante. Ainsi, on prend B tel que :

$$B = \begin{pmatrix} p_{in} & p_{out} \\ p_{out} & p_{in} \end{pmatrix} \mid p_{in} > p_{out}$$

Cela signifie la probabilité qu'il existe une arrête entre deux points est plus importante s' ils appartiennent à la même classe que s'ils appartiennent à deux classes différentes. En mettant en place une probabilité d'amitié intra-classe plus grande que la probabilité d'amitié inter-classe on réussit à simuler un ensemble de vecteurs représentés par la matrice d'adjacence A de G et fidèles à la catégorisation de départ. On fera l'hypothèse que le nombre d'amis de chaque individu reste raisonnable même lorsque n reste très grand (i.e. $n \gg 1$ et $p_{in}, p_{out} \sim \frac{1}{n}$). Par la suite on utilisera la notation suivante : $p_{in} = \frac{c_{in}}{n}, p_{out} = \frac{c_{out}}{n}$. Notons que la modélisation proposée ici ne prétend pas simuler des réseaux fidèles à ceux qu'on trouve dans la réalité mais fournit une base de départ pour l'étude des algorithmes de clustering. Plusieurs recherches tentent de trouver la matrice B idéale afin de coller au mieux aux réseaux que l'on trouve dans la réalité [KN11].

1.3 Théorie du clustering spectral et de sa fiabilité

Clustering

Le clustering dans notre cas consiste à diviser $V = \{v_1, v_2, ..., v_n\}$ en 2 groupes homogènes :

$$X = C_0 \cup C_1, \quad C_0 \cap C_1 = \varnothing$$

Notre méthode devra fournir une fonction d'étiquetage $l(.) : x \to \{0, 1\}$ qui indique l'appartenance de v_i au groupe $C_{l(v_i)}$. Le principe du clustering repose sur la minimisation d'une fonction coût. Une fonction coût générique dans notre cas s'écrit comme la somme des coûts pour chaque groupe.

$$e_2(V, C_0, C_1) = e_1(C_0) + e_1(C_1)$$

avec $e_1(C)$ la fonction de coût pour un simple groupe. On peut également associer à chaque groupe C_i un centre $c_i = c(C_i)$ de ce cluster. Ces deux centres permettent de définir la distance entre une donnée v_i et un cluster. On a donc : $D(v_i, G) = D(v_i, c(G))$. On peut ainsi définir le coût pour un simple groupe :

$$e_1(C, c) = \sum_{v \in C} D(v, c)$$

On obtient donc pour le coût total :

$$e_2(V, C_0, C_1) = \sum_{v \in C_0} D(v, c(C_0)) + \sum_{v \in C_1} D(v, c(C_1))$$

On peut aussi définir le coût de manière totalement équivalent en se donnant $\Omega = \{c_1, c_2\}$:

$$e_2(V, \Omega) = \sum_{i=1}^{n} min(D(v_i, c_0), D(v_i, c_1))$$

Le but du clustering est de trouver le couple de centre Ω qui minimise la fonction coût. Il faudra aussi vérifier que la minimisation donne des bonnes partitions.

k-means

Parmis les différentes méthodes de clustering, la méthode des k-moyennes (k-means) est l'une des plus répandue. Elle consiste à minimiser la distance au carré des données à leur centre le plus proche.

$$e_2(V, \Omega) = \sum_{i=1}^{n} min(\|v_i - c_0\|^2, \|v_i - c_1\|^2)$$

On obtient une expression simple du centre de chaque groupe. En effet il s'agit du centre de masse :

$$c(C) = \underset{c}{argmin} \sum_{v \in C} \|v - c\|^2 = \frac{1}{|C|} \sum_{v \in C} v$$

Au final, le coût global correspond à la somme des variances non normalisée de chaque cluster. Le problème devient donc intuitif puisqu'on cherche à trouver les centres tel que la dispersion des points autour de leur centre soit la plus petite.

Le problème de minimisation de l'algorithme k-means s'ecrit donc :

$$\underset{\Omega}{min}\ e_2(V, \Omega)$$

L'algorithme se base sur l'heuristique locale suivante :

Algorithm 1 K-Means

```
procedure KMeans(A)
    Initialisation des centres c_0 , c_1 par une procédure au choix
    while Pas de convergence do
        for i de 1 à n do
            l(v_i) = argmin(||v − c_0||^2 , ||v − c_1||^2)
        end for
        for i de 1 à 2 do
            c_i = (Σ_{v∈C_i} v) / |C_i|
        end for
    end while
    Return l, c_0, c_1
end procedure
```

On peux prouver que cette heuristique converge vers un minimum local en un nombre fini d'itération.

Il y a plusieurs manière d'initialiser l'algorithme. Dans notre cas, la bibliothèque scikit learn utilise par défaut l'initialisation probabiliste des k-means++. Cela revient à choisir c_0 de départ aléatoirement uniformément dans V puis c_2 avec une probabilité :

$$p(v) = \frac{\|v - c_0\|^2}{\sum_x \|x - c_1\|^2}$$

On pondère ainsi la probabilité de v en fonction de la distance de v au cluster déjà choisi c_0.

Clustering spectral

L'algorithme k-means tel qu'il a été présenté ne fonctionne bien que pour des dimensions des données raisonnables. Cependant, dans notre cas, chaque individu possède n attributs et nous avons n individus. Pour des valeurs de n très grandes , les algorithmes de clustering tel que k-means sont inadaptés. Il faut absolument réduire la dimension pour se ramener à un problème moins complexe. C'est à ce moment que le clustering spectral intervient. Nous allons montrer que l'on peut passer d'un problème à n attributs à un problème à 2 attributs.

L'algorithme de clustering spectrale se présente comme suit :

Algorithm 2 Spectral Clustering

procedure SpectralClustering(A)
    Trouver les 2 vecteurs propres X_1 et X_2 de A associés aux deux plus grandes valeurs propres
    Former la matrice $U = [X_1, X_2] \in \mathbb{R}^{n \times 2}$ en plaçant les vecteurs propres dans les colonnes.
    En considérant chaque ligne comme un vecteur dans $\mathbb{R}^2$ à deux attributs, faire $KMeans(U)$
    Return $KMeans(U)$
end procedure

Notons que l'ordre sur les valeurs propres est bien défini. En effet, la matrice A étant symétrique, d'après le théorème spectrale , elle est diagonalisable et ses valeurs propres sont réelles.

Nous avons aussi la variante suivante :

Algorithm 3 Spectral Clustering BIS

procedure SpectralClusteringBis(A)
    On construit la matrice diagonale D tel que : $D_{i,i} = \sum_{j=1}^{n} A_{i,j}$
    On calcule : $M = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$
    Return $SpectralClustering(M)$
end procedure

Ainsi on remarque que l'algorithme de clustering spectral permet de passer d'un problème en n dimensions à un problème en 2 dimensions qui est beaucoup plus facile à résoudre pour l'algorithme des k-moyennes. Nous n'exposerons par toute la théorie démontrant le lien entre les deux vecteurs propres associés aux valeurs propres les plus grandes et l'information de partitionnement contenue dans le graphe. Cependant, nous allons tenter de comprendre intuitivement d'où viens cette technique ainsi que son domaine de fiabilité dans le cas d'une matrice générée par le stochastic block model. Enfin, nous allons comparer l'algorithme 2 et 3.

On suppose avoir généré par le stochastic block model le graphe A tel que :

$$B = \begin{pmatrix} p_{in} & p_{out} \\ p_{out} & p_{in} \end{pmatrix} \mid p_{in} > p_{out}$$

De plus, on prendra deux clusters de taille égale et on supposera que n est pair. Quitte à réarranger les points on prend :

$$C_0 = \left\{v_1, .., v_{\frac{n}{2}}\right\} \; C_1 = \left\{v_{\frac{n}{2}+1}, .., v_n\right\}$$

.

On obtient donc :

$$\begin{cases} \mathbb{E}A_{i,j} = p_{in} \;\; si \;\; v_i \; et \; v_j \; appartiennent \; au \; mme \; cluster \; C \\ \mathbb{E}A_{i,j} = p_{out} \;\; sinon \end{cases}$$

Ainsi :

$$\mathbb{E}A = \begin{pmatrix} p_{in} & \cdots & p_{in} & p_{out} & \cdots & p_{out} \\ \vdots & \ddots & \vdots\vdots & & \ddots & \vdots \\ p_{in} & \cdots & p_{in} & p_{out} & \cdots & p_{out} \\ p_{out} & \cdots & p_{out} & p_{in} & \cdots & p_{in} \\ \vdots & \ddots & \vdots\vdots & & \ddots & \vdots \\ p_{out} & \cdots & p_{out} & p_{in} & \cdots & p_{in} \end{pmatrix}$$

Le rang de la matrice vaut 2.Les valeurs et vecteurs propres sont :

$$\lambda_1 = \frac{n(p_{in}+p_{out})}{2}, X_1 = \begin{pmatrix} 1 \\ \vdots \\ \vdots \\ \vdots \\ 1 \end{pmatrix}, \quad \lambda_2 = \frac{n(p_{in}-p_{out})}{2}, \quad X_2 = \begin{pmatrix} 1 \\ \vdots \\ 1 \\ -1 \\ \vdots \\ -1 \end{pmatrix}$$

La première valeur propre nous donne l'expression de l'espérance du degré d'un individu v_i quelconque. La seconde valeur propre nous permet de détecter si il y a présence d'une structure de communauté. En effet, si $\lambda_2 = 0$ alors les deux probabilités sont égales et il n'y a pas de structure de communauté. Enfin, le deuxième vecteur propre nous donne exactement la clusterisation souhaitée. On remarque que faire un k-means sur les points (X_1, X_2) reviens trivialement à séparer dans un plan $2D$ les points $(1, 1)$ et $(-1, 1)$. Puisqu'on ne connait pas $\mathbb{E}A$, on doit espérer que A n'est pas très loin en norme de $\mathbb{E}A$. La théorie de la perturbation nous montre dans ce cas que les valeurs et vecteurs propres de $\mathbb{E}A$ et A ne sont pas très éloignées. Dans ce cas, les deux plus grands vecteurs propres (associés aux plus grandes valeurs propres) de A seraient une bonne approximation de ceux de $\mathbb{E}A$, ce qui nous permettrait de réaliser la clusterisation comme précédemment. On obtiendrai cette fois ci deux nuages de points autour de $(1, 1)$ et $(-1, 1)$.

On peux démontrer [Ver16]. que :

$$\mathbb{E}\,\|A - \mathbb{E}A\| \leq \sqrt{\frac{n(p_{in}+p_{out})log(n)}{2}} + log(n)$$

Ainsi, en supposant que la matrice n'est pas trop creuse (i.e. $d = \frac{n(p_{in}+p_{out})}{2} \gg log(n)$ on obtient :

$$\|A - \mathbb{E}A\| \leq \sqrt{dlog(n)}$$

Puisque : $\|\mathbb{E}A\| = \lambda_1 = d$ alors on a :

$$\mathbb{E}\,\|A - \mathbb{E}A\| \ll \|\mathbb{E}A\|$$

Si la matrice n'est trop creuse on peut donc montrer que A approxime bien $\mathbb{E}A$.

Une analyse plus poussée montre que si :

$$c_{in} - c_{out} \gg \sqrt{(c_{in}+c_{out})log(n)}$$

alors avec une probabilité élevé, l'algorithme de clustering spectrale trouve les communautés. Cette condition nous permet d'affirmer à quel moment on peux dire que A approxime bien $\mathbb{E}A$. A ce stade on a donc :

$$\|A - \mathbb{E}A\| \leq \sqrt{\frac{(c_{in}+c_{out})}{2}log(n)} \ll c_{in} - c_{out}$$

La condition peut être améliorée dans certains cas (matrice dense) et d'autres méthodes ont été développées pour les matrices très creuses.

Intéressons nous à la différence entre les algorithmes 2 et 3 pour finir. De manière générale, les algorithme de clustering spectral utilisent l'information contenue dans les vecteurs propres d'une matrice (laplacienne) construite à partir de la matrice d'adjacence. Notons que la matrice d'adjacence est plus généralement une matrice d'affinité mesurant la similarité entre deux individus

et dépendant du type de donnée et de problème. Dans l'algorithme 2, on ne me modifie pas la matrice d'adjacence tandis que dans l'algorithme 3, cette dernière est modifiée et on appelle la matrice résultante matrice Laplacienne. Il existe plusieurs types de matrice Laplacienne en fonction de la structure de donnée que l'on traite mais le dénominateur commun entre ces matrices est la présence de la matrice D. La matrice Laplacienne est obtenue dans notre cas en multipliant A de part et d'autre par la matrice $D^{-\frac{1}{2}}$. Cette matrice est le résultat de la normalisation de A sur les lignes et les colonnes. On remarque qu'appliquer ce Laplacien à la matrice A met un poids aux différentes arrêtes du graphe. En effet on a :

$$(D^{-\frac{1}{2}}AD^{-\frac{1}{2}})_{i,j} = w_{i,j}A_{i,j} \; avec \; w_{i,j} = \frac{1}{\sqrt{d_i}\sqrt{d_j}}$$

Notons que d_i représente le nombre de lien que possède l'individu i avec le reste du réseau (le nombre d'amis). En mettant un poids sur cette données, cela permet d'augmenter l'efficacité de l'algorithme sur certaines structures de données. Prenons par exemple une communauté de très grande taille avec une communauté de très petite taille. Un poids plus élevé sera donné aux éléments de la petite communauté car le nombre d'amis de chaque individus de cette dernière est petit et donc le poids élevé. Cela permettra donc intuitivement de mieux clusteriser cette petite communauté si l'on réfléchit en terme de coupe minimisant le poids des arrêtes inter-clusters. L'algorithme 3 serait donc très efficace pour la détection de petits clusters vs grands clusters. On peux supposer que l'algorithme 2 ne parvient pas à bien clusteriser dans cette situation au vu de la non pondération.

1.4 Algorithmes

Nous avons fournit un fichier StochaBM.py qui permet de :

- Générer des matrices selon le stochastic block model : $stocha(V, C, c_in, c_out)$
- Réaliser le clustering spectrale cette matrice : $clusteringspectral(A)$
- Calculer le taux de réussite du clustering $\frac{J}{n}$ $avec$ $J = Card\{v \in V \setminus cluster(v) \; correct\}$: $tauxreussite(C, B)$
- Afficher le graphe de deux manières différentes : En pointant les points non nuls de la matrice d'adjacence et en dessinant le graphe complètement (cf page de garde). Il faudra dé-commenter certaines partie du code pour tracer ou obtenir des résultats.
- Afficher les clusters issus des vecteurs propres correspondant aux deux plus grandes valeurs propres. Cette affichage permet de mieux comparer les options 1 et 2 et de mieux comprendre le clustering spectral en pratique.

Il est important de noter certains choix stratégiques au niveau du code. Pour le calcul des deux plus grandes valeurs propres et vecteurs propres associés nous avons opté pour un passage aux matrices creuses ce qui réduit le temps calcul et optimise le code. En effet, la fonction de recherche des valeurs propres sous Python propose une méthode spéciale pour les matrices symétriques et permet de trouver directement les deux plus grandes valeurs propres (sans tri) : sparse.linalg.eigsh

Le plus délicat dans ce code reste le calcul du taux de réussite. La fonction calculant le clustering spectrale assigne à chaque individu un 0 ou un 1 en fonction de son appartenance au cluster C_0 ou C_1. Cependant, il se peut que la fonction switch les 0 ou les 1 (par rapport à la référence) et fournit quand même le bon partitionnement. Dans ce cas, il faut à chaque fois associé un cluster calculé à son cluster de référence non pas en regardant les 0 et les 1 mais plutôt en comparant les individus dans les clusters. C'est la matrice de confusion qui permet de réaliser cela ($Confusionmatrix(C1, C2, B1, B2)$, cf Table 1). Ainsi, en notant M cette matrice, si $argmin(M_{0,0}, M_{0,1}) = 1$ alors le cluster calculé $B1$ correspond au cluster de référence $C2$ et le cluster calculé $B2$ correspond au cluster de référence $C1$. Dès qu'on connait cette association, il suffit de récupérer le nombre d'erreur associé au niveau des éléments de la matrice puis de calculer le taux de réussite. Notons qu'il existe plusieurs autres indicateurs qui jugent de la bonne clusterisation d'un graphe mais nous avons décider de nous focaliser sur ce dernier qui semble par la suite bien décrire le comportement des algorithmes.

Table 1: Confusion Matrix

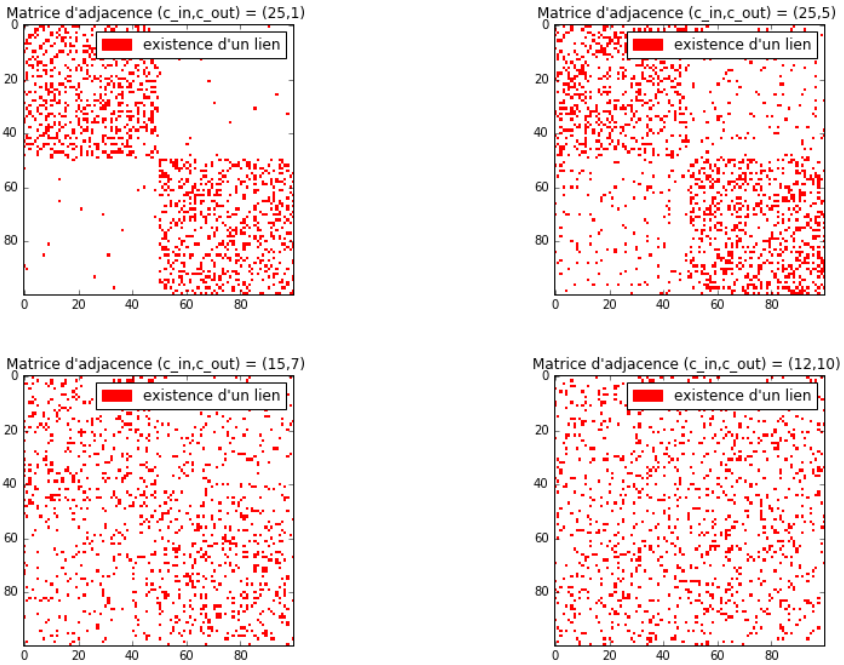
	Cluster de référence 1 (C1)	Cluster de référence 2(C2)
Clusteur caculé 1 (B1)	Nb dissimilarités B1/C1	Nb dissimilarités B1/C2
CLuster calculé 2 (B2)	Nb dissimilarités B2/C1	Nb dissimilarités B2/C2

1.5 Simulations

Dans cette partie nous allons mettre en avant certains résultats théoriques et certaines observations intéressantes à travers nos simulations Pyhton .

Stochastic Block Model

On affiche la matrice d'adjacence A (imshow) de la façon suivante : les points contenant des 1 sont colorés en rouge et les points contenant des 0 en blanc. Voici quelques résultats pour $n = 100$ et des clusters de départ de taille égale:



On voit bien que plus on resserre les probabilité inter et intra cluster plus on a du mal a distinguer les communautés dans la matrice symétrique A. Cela donne une première explication intuitive de la condition de bon clustering en $c_{in} - c_{out}$

On peut aussi visualiser les graphes en les dessinant à l'aide de la bibliothèque networkx :

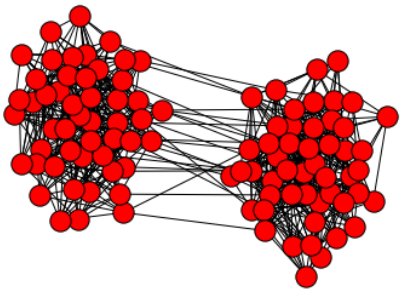


Figure 1: cin = 25, cout = 1

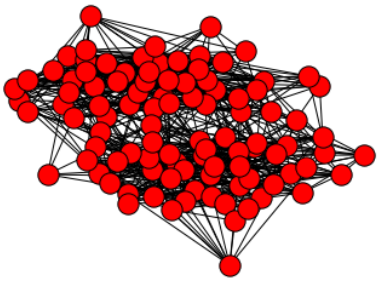


Figure 2: cin = 25, cout = 5

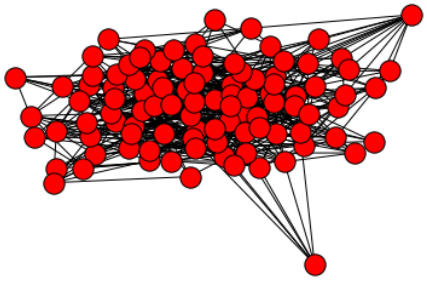


Figure 3: cin = 15, cout = 7

Pour la figure 1, on voit très bien où placer la droite correspondant au mincut. Plus on resserre la condition $c_{in} - c_{out}$ plus il est dur de placer cette droite et de distinguer les clusters.

Mise en avant du clustering spectral

Nous avons vu que le clustering spectral permet de passer d'un espace à n dimension à un espace à 2 dimensions. Nous allons montrer que l'on a bien une situation de clustering dans un plan 2D. Pour cela on trace le second vecteur propre X_2 en fonction du premier X_1 . On colore également les points issus du clustering par l'algorihtme 2 en deux couleurs (la comparaison avec l'algorithme 3 se fera après). Voici le résultat pour $n = 100$ et des clusters de départ de taille égale:

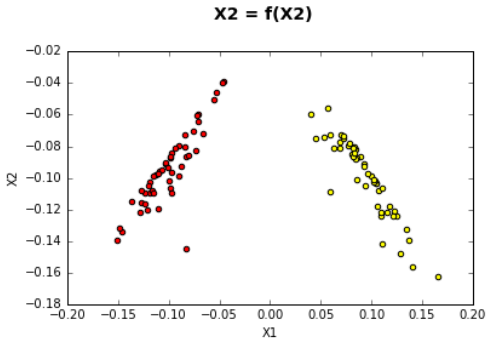


Figure 4: cin = 25, cout = 1

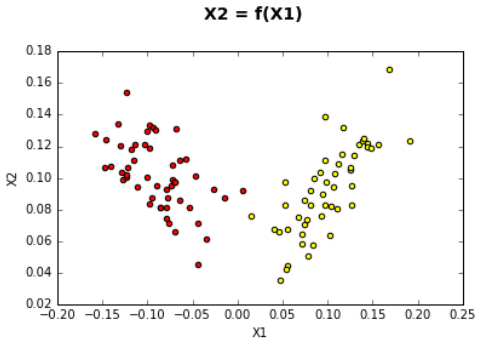


Figure 5: cin = 25, cout = 5

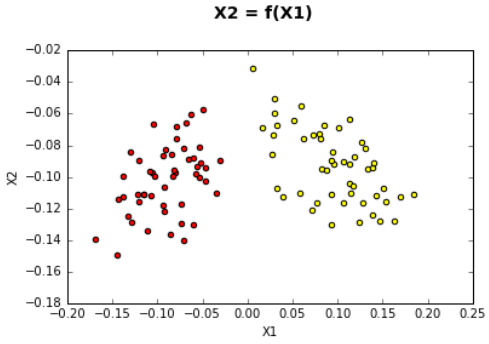


Figure 6: cin = 15, cout = 7

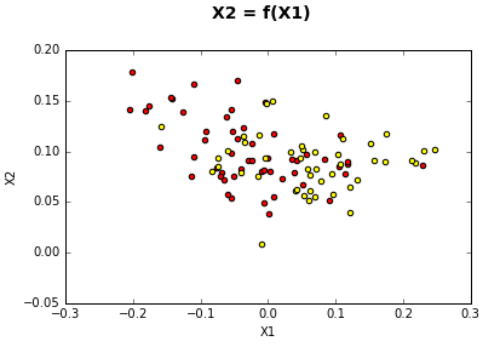


Figure 7: cin = 12, cout = 10

Ainsi le clustering spectral nous donne bien deux ensembles clairement distincts pour les deux premiers jeux de donnée. Le partitionnement en deux groupes dans un plan $2D$ semble trivial. On remarque que plus la condition se resserre plus la variance de chaque cluster augmente (la comparaison du jeu 1 et du jeu 3 le montre très bien). Au final, il est clairement impossible de distinguer les deux clusters pour le dernier jeux de données.

Test de la condition de bonne clusterisation

On souhaite vérifier la condition suivante :

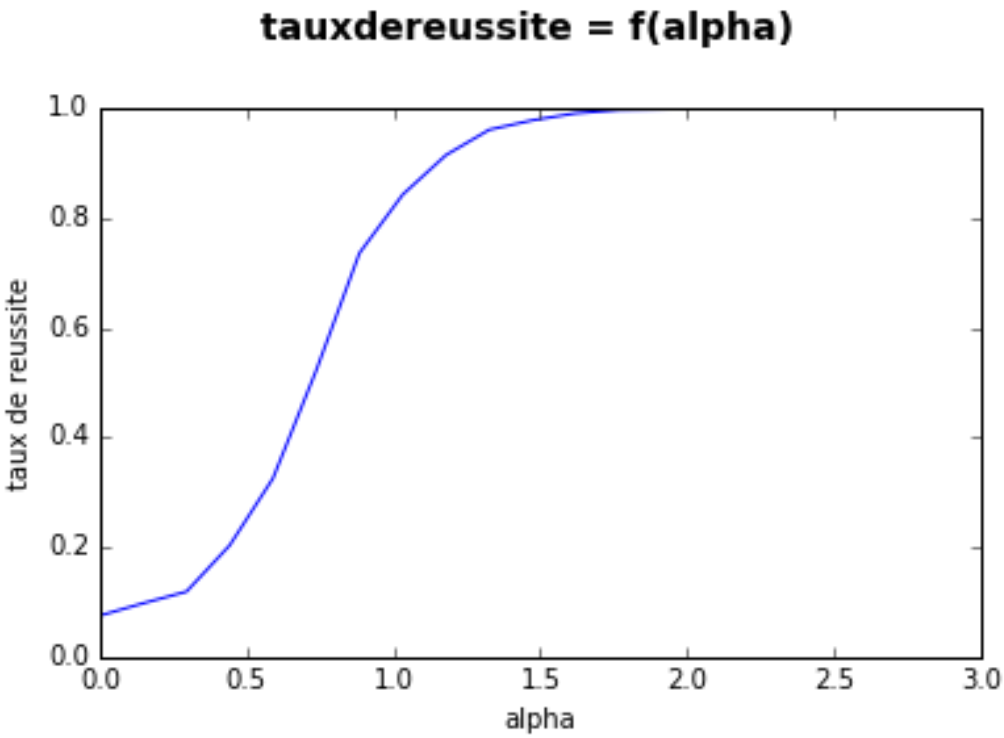
$$c_{in} - c_{out} \gg \sqrt{(c_{in} + c_{out})log(n)}$$

Pour cela, nous allons considérer comme précédemment deux clusters de taille égale puis nous allons faire varier l'écart $c_{in} - c_{out}$ et voir la répercussion sur le taux de réussite. On utilise encore l'algorithme 2 puisqu'on le comparera au 3 qu'à la section suivante.

Posons :

$$\alpha = \frac{c_{in} - c_{out}}{\sqrt{(c_{in} + c_{out})log(n)}}$$

On invitera le lecteur à lire le fichier python qui trace cette courbe (tauxdereussite.py).On obtient le tracé suivant :



On remarque que l'on atteint un taux de réussite de 1 pour un rapport α autour de 1.8 ce qui vérifie bien la condition. De plus, notons la forme spéciale de la courbe avec un véritable saut pour des valeurs de α entre 0.5 et 1.2. Dans cet intervalle, la derivée de la courbe est plus élevé que dans les autres régions et on assiste à un saut. Notons que cette zone va être très utile pour les simulations de la seconde partie du rapport et qu'il faudra tenir compte de cette courbe. Aussi, on remarque que l'augmentation du nombre de point augmente la dérivée au niveau du saut qui devient plus raide et permet à la courbe d'atteindre un taux de réussite de 1 bien avant 1.8 comme le montre cette comparaison :

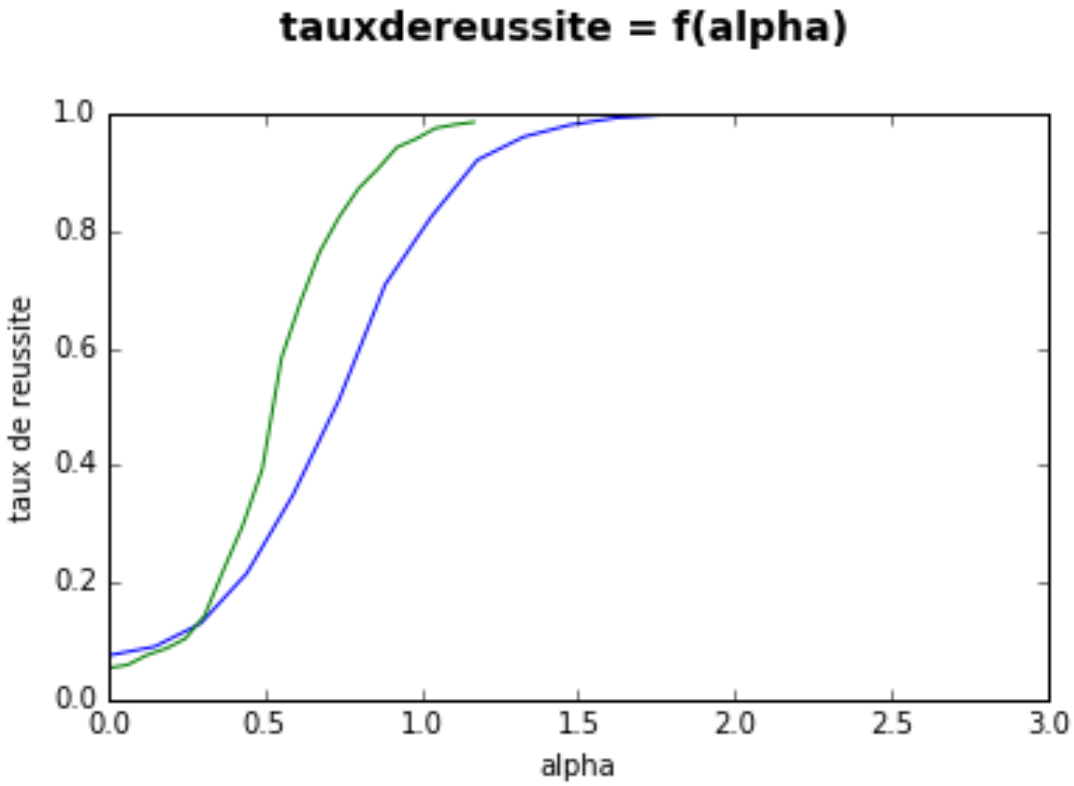
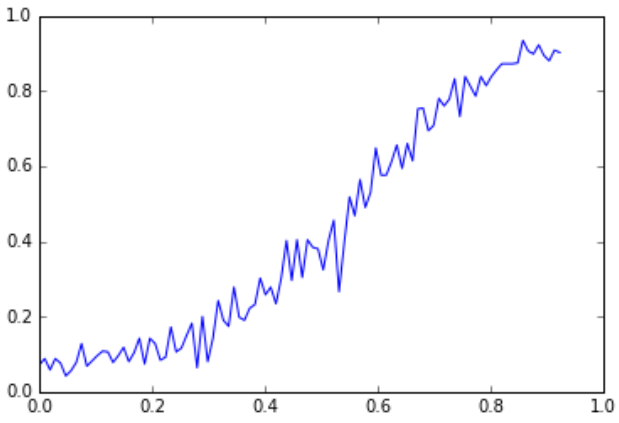


Figure 8: courbe verte : n = 200 , courbe bleue : n= 100

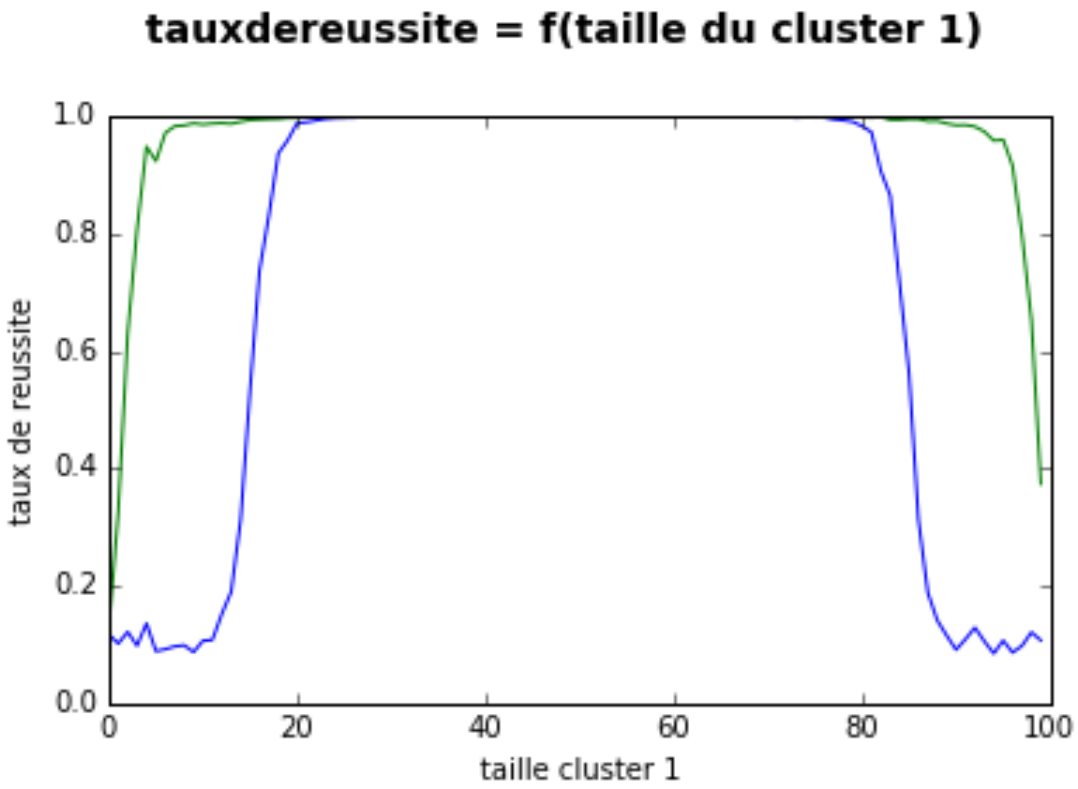
Au niveau de l'implémentation, tracer ce type de courbe n'est pas instantané puisqu'on doit moyenner sur plusieurs points pour chaque α afin d'obtenir une courbe lisse.

Si on ne moyenne pas, la variance fera ressortir le processus aléatoire et on obtient des courbes du type :



Comparaison des algorithme 2 et 3

On se propose de comparer l'efficacité des algorithme 2 et 3. Nous avons intuité lors de l'étude théorique que l'algorithme 3 se comporte très bien pour des petits clusters vs grands clusters. De plus, nous avons supposé également que l'algorithme 2 a du mal fournir de bon résultats pour cette même situation. Nous nous sommes donc placés dans une situation idéale pour la condition de clusterisation : $c_{in} = 50$ et $c_{out} = 10$ pour 100 points. On fait varier la taille du cluster 1 et on observe comment les deux algorithmes se comporte. On invite le lecteur à lire le code python traçant cette courbe (comparaisonmethodes.py) On obtient :



Ainsi l'algorithme 2 fonctionne très bien pour $\frac{Card(C_0)}{Card(C_1)} \in [0.25, 1]$ tandis que l'algorithme 3 fonction très bien sur $\frac{Card(C_0)}{Card(C_1)} \in [0.1, 1]$. En dehors de leurs intervalles de fonctionnement le taux de réussite chute de manière importante. De toute manière il est très rare de se trouver dans des configurations ou il ne faut clusteriser que quelques points. Ainsi, on remarque bien que l'algorithme 3 résiste plus à des clusters de tailles différentes dans le sens où il est largement plus efficace que l'algorithme 2 sur la plage $\frac{Card(C_0)}{Card(C_1)} \in [0.1, 0.25]$. En effet, sur cette plage, son taux de réussite vaut quasiment 1 contre 0.1 pour l'algorithme 2. Notons aussi que l'obtention de cette courbe demande de moyenner pour chaque taille de cluster 1 afin de lisser. On peut également observer la supériorité de l'algorithme 3 sur le 2 en traçant les vecteurs propres issus du clustering spectral pour un rapport $\frac{Card(C_0)}{Card(C_1)} = 0.1$:

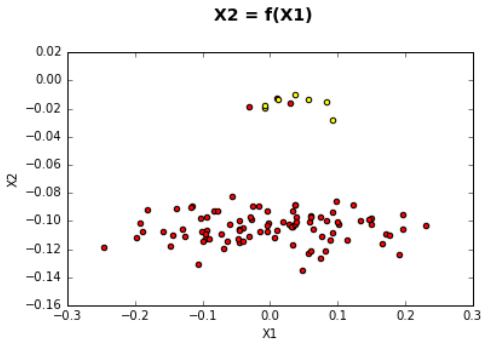


Figure 9: Algorithme 2

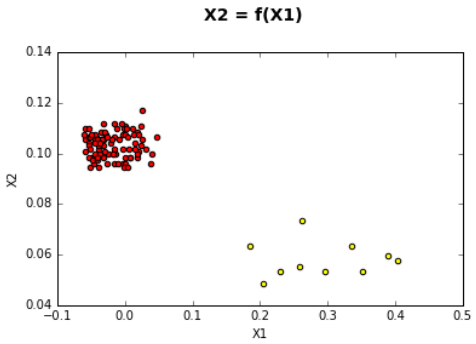


Figure 10: Algorithme 3

On voit bien que l'algorithme 3 fournit des clusters avec des variances beaucoup plus faible et plus distincts entre eux ce qui rend le clustering après plus efficace.

2 Événements rares : Probabilité de mauvaise clusterisation

Après avoir étudié le fonctionnement et les résultats obtenus des deux options de la méthode de clustering nous nous intéressons aux cas pour lesquels des individus ne sont pas assignés au bon cluster.

2.1 Définition de la problématique

Les événements étudiés sont des événements rares. Pour obtenir des simulations concluantes il faut généralement répéter les simulations un grand nombre de fois.

Pour déterminer la probabilité d'obtenir ce type d'événement la méthode naïve consiste à compter le nombre de réalisations concluantes sur le nombre d'essais. Le problème rencontré pour ce type d'événement vient de la faible probabilité d'apparition ainsi le nombre de simulations nécessaires peut être très important. La méthode naïve (méthode de Monte-Carlo) n'est plus adaptée pour ces simulations. Il faut désormais utiliser des méthodes adaptées comme la méthode de l'importance sampling.

2.2 Théorie : Méthode de Monte-Carlo et de l'importance sampling

Méthode de Monte-Carlo

La méthode de Monte Carlo naïve consiste à appliquer la loi des grands nombres. Si $(X_i)_{i\in\mathbb{N}}$ est une suite de variables aléatoires indépendantes de même loi que X sous $\mathbb{P}$ alors d'après la loi des grands nombres :

$$\mathbb{P}(X = a) \approx \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{X_i=a}$$

On approxime donc la quantité $\mathbb{P}(X = a)$ en comptant le nombre de réalisation sur un grand nombre d'événement.
Lorsque l'événement $\{X = a\}$ est rare alors il est très long de simuler une apparition et cette méthode s'avère rapidement très inefficace. Dans ce cas, on effectue un changement de probabilité: importance sampling.

Importance Sampling

On considère une variable aléatoire dans $\mathbb{R}^d$ définie sur espace de probabilité muni de $\mathbb{P}$. Soit $L : \mathbb{R}^d \to \mathbb{R}_+^*$ une fonction mesurable tel que :

$$\mathbb{E}\left[L(X)\right] = 1$$

Notons $\mathbb{Q}$ la loi de probabilité de densité $L(X)$ alors pour toute fonction $g : \mathbb{R}^d \to \mathbb{R}$ mesurable bornée on a :

$$\mathbb{E}\left[g(X)\right] = \mathbb{E}\left[\frac{g(X)}{L(X)}L(X)\right] = \mathbb{E}_{\mathbb{Q}}\left[\frac{g(X)}{L(X)}\right]$$

Ce qui correspond à un changement de probabilité. L'importance sampling consiste donc à faire un changement de loi pour augmenter le taux d'apparition des événements et calculer la probabilité souhaitée.

2.3 Probabilité de mauvaise clusterisation : ensemble de sommets fixés et cardinal faible

Dans cette partie on essaye d'estimer à l'aide de l'importance sampling, la probabilité qu'un ensemble fixé de sommets, de cardinal faible (autour de 2), soit mal clusterisé.Des essais avec la méthode de Monte-Carlo (cf résultats) montrent qu'à moins d'être dans un cas très défavorable, c'est-à-dire dans un cas ou c_{in} est proche de c_{out}. Il faut généralement une dizaine de milliers de simulations afin d'obtenir plusieurs événements

Importance sampling pour notre problème

On montre à l'aide la formule ci dessus de changement de variable le résultat suivant (cf $TD3$): Soient $(X_1, ..., X_n)$ des v.a. indépendantes de loi de Bernoulli $B(p_1), ..., B(p_n)$ et $(Z_1, ..., Z_n)$ des v.a. indépendantes de loi de Bernoulli $B(q_1), ..., B(q_n)$. Pour toute fonction $g : \{0,1\}^n \to \mathbb{R}$ mesurable et bornée alors on a :

$$\mathbb{E}\left[g\left(X_1, ..., X_n\right)\right] = \mathbb{E}\left[g\left(Z_1, ..., Z_n\right)\left(\prod_{i=1}^{n} \mathbf{1}_{Z_i=0}\frac{1-p_i}{1-q_i} + \mathbf{1}_{Z_i=1}\frac{p_i}{q_i}\right)\right]$$

Les variables de Bernoulli qui nous intéressent dans notre problème sont les $(X_{i,j})_{1\leq i\leq j\leq n}$ éléments de la matrice diagonale A générées par la méthode du stochastic block model et que l'on va clusterisé. On a donc :

$$\begin{cases} X_{i,j} \sim B(p_{in}) & si\ (i,j)\ appartiennent\ au\ mme\ cluster\ de\ rfrence \\ X_{i,j} \sim B(p_{out}) & sinon \end{cases}$$

Il y en a $\frac{n*(n+1)}{2}$ car on prend que la partie supérieure de la matrice symétrique

La formule de l'importance sampling nous donne la possibilité de changer les probabilités des événements pour qu'ils apparaissent plus souvent. Par conséquent, au lieu d'utiliser la méthode de Monte-Carlo avec un c_{in} et c_{out} nous prendrons des variables aléatoires avec des probabilités légèrement différentes c'_{in} et c'_{out}afin d'obtenir plus de simulations concluantes.

On considère donc les variables $(Z_{i,j})_{1\leq i\leq j\leq n}$ tel que :

$$\begin{cases} Z_{in} = Z_{i,j} \sim B(p'_{in}) & si\ (i,j)\ appartiennent\ au\ mme\ cluster\ de\ rfrence \\ Z_{out} = Z_{i,j} \sim B(p'_{out}) & sinon \end{cases}$$

De plus on a , en notant *nprime* le nombre d'éléments dans le cluster 1 :

$$D = Card\left\{(i,j) \mid (i,j) \in mme\ cluster\right\} = \frac{nprime*(nprime+1)}{2} + \frac{(n-nprime)*(n-nprime+1)}{2}$$

$$Card\left\{(i,j) \mid (i,j) \in pas\ mme\ cluster\right\} = \frac{n(n+1)}{2} - D$$

En appliquant la formule du début on obtient :

$$\mathbb{E}\left[g\left(X_1, ..., X_{\frac{n(n+1))}{2}}\right)\right] = \mathbb{E}\left[g\left(Z_1, ..., Z_{\frac{n(n+1))}{2}}\right)\prod_{i=1}^{D}\left(\mathbf{1}_{Z_{in}=0}\frac{1-p_{in}}{1-p'_{in}} + \mathbf{1}_{Z_{in}=1}\frac{p_{in}}{p'_{in}}\right)\right.$$
$$\left.\prod_{i=1}^{D-\frac{n(n+1))}{2}}\left(\mathbf{1}_{Z_{out}=0}\frac{1-p_{out}}{1-p'_{out}} + \mathbf{1}_{Z_{out}=1}\frac{p_{out}}{p'_{out}}\right)\right]$$

Il reste à déterminer la fonction g. Rappelons ce qu'on cherche: on essaye d'estimer la probabilité qu'un ensemble fixé de sommets, de cardinal faible (autour de 2), soit mal clusterisé. On se fixe donc deux points (on peut généraliser à plusieurs points tant que le cardinal est faible) : (i_1, j_1). On obtient donc la fonction indicatrice $g : \{0,1\}^{\frac{n(n+1)}{2}} \to \{0,1\}$ tel que :

$$g((X_1, ..., X_{\frac{n(n+1))}{2}}) = \begin{cases} 1\ si\ (i_1,j_1)\ mal\ clusterises \\ 0\ sinon \end{cases}$$

Il faut donc comprendre g comme une fonction indicatrice qui incorpore la clusterisation spectrale d'une matrice du stochastic block model et son évaluation pour vérifie qu'elle a bien clusterisé ou pas nos sommets sélectionnés.

On obtient donc bien :

$$\mathbb{E}\left[g\left(X_1, ..., X_{\frac{n(n+1))}{2}}\right)\right] = \mathbb{P}(i_1\ et\ j_1 mal\ clusturis)$$

La fonction g présente dans le fichier sampling-initial.py est très simple Cette fonction prend en paramètre une matrice d'adjacence, les paramètres de la simulation comme par exemple, le nombre d'individus à considérer dans chaque cluster C_0 et C_1 et la taille du premier cluster. La première étape consiste à appliquer l'algorithme de clustering spectral et ensuite de vérifier si les individus considérés sont assignés dans les bons clusters. Dans la majorité des cas nous avons considérés les deux premiers individus du cluster 1. Néanmoins, notre algorithme permet de considérer des individus dans les deux clusters et voir si ces individus sont bien clusterisés. En considérant toujours la même problématique de switch entre les 0 et 1 au niveau du clustering spectrale on doit encore une fois utiliser une matrice de confusion (cf code). De plus, on peut changer la taille des clusters. Rappelons que notre fonction indicatrice renvoie 0 si tous les individus choisis (cardinal faible) sont bien clusterisés et 1 sinon .

Implémentation de l'algorithme - Première Partie

Au début de l'implémentation de la méthode de l'importance sampling nous avons décidé de changer la totalité des probabilités assignées aux individus. Pendant l'état de la création de la matrice d'adjacence les probabilités deviennent donc p'_{in} et p'_{out} pour tous les éléments et sont différentes de p_{in} et p_{out} .

Avant de parler des résultats obtenus. Nous présentons ci-dessous l'algorithme utilisé.

Algorithm 4 Importance Sampling

procedure IMPORTANCE SAMPLING($M, p_{in}, p_{out}, p_{in}{}', p_{out}{}'$)
    Initialisation des vecteurs Ech = Ech_0 ,.., Ech_M pour stocker les résultats.
    for i de 1 à M **do**
        Générer A une matrice d'adjacence avec l'algorithme Matrice Adjacence avec les paramètres p_{in}' et p_{out}'
        Appliquer à A la méthode Indicatrice (fonction g)
        Calcul du coefficient de vraisemblance P
        $Ech_i \leftarrow g(A) * P$
    end for
    La probabilité demandée vaut la moyenne du tableau Ech
end procedure

Une fois l'algorithme fini et les premiers résultats obtenus nous nous sommes rendu compte qu'il est difficile d'interpréter ces résultats. Les résultats n'ont pas été satisfaisants.

Résultats de l'algorithme Voici les résultats obtenues grâce à notre première méthode sampling pour 100 simulations avec $c_{in} = 25$ et $c_{out} = 5$
Probabilité obtenue = 5.34113115434e-51
Ecart type = 5.27622774765e-50
Demi largeur = 1.03414063854e-50
Intervalle de confiance : [-5.00027523106e-51 , 1.56825375397e-50]

Bien évidemment, les résultats donnés ne sont pas intéressants. La demi-largeur trouvée est d'un ordre plus grand que la probabilité obtenue. De plus, il nous a semblé que la probabilité est anormalement très faible.

Notons également que puisqu'on modifie toutes les probabilités, le calcul du coefficient de vraissemblance peut exploser ou être très petit et donc sous python amener à des résultats non exploitable.

Nous avons donc décidé d'implémenter une deuxième méthode de calcul afin de valider notre algorithme. Nous avons décidé d'implémenter une méthode naïve avec le principe de Monte-Carlo (Methode-Naive.py)

Algorithm 5 Méthode Naïve : Méthode de Monte-Carlo

procedure MÉTHODE NAÏVE(M, p_{in}, p_{out})
    Initialisation des vecteurs Ech = Ech_0 ,.., Ech_M pour stocker les résultats.
    for i de 1 à M **do**
        Générer A une matrice d'adjacence avec l'algorithme Matrice Adjacence avec les paramètres P_{in} et P_{out}
        Appliquer à A la fonction g ou fonction indicatrice
        $Ech_i \leftarrow g(A)$
    end for
    La probabilité demandée vaut la moyenne du tableau Ech
end procedure

Les résultats obtenus à l'aide la méthode naïve nous donne des résultats complètement différents de la méthode sampling.

Résultats obtenus avec la méthode naïve:
Probabilité obtenue = 0.004
Ecart type = 0.0631189353522
Demi largeur = 0.00391215214428
Intervalle de confiance : [8.78478557193e-05 , 0.00791215214428]

Nous décidons alors de regarder en détails le tableau Echantillon obtenu par la méthode Sampling.

Ech = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.483673610470445e+81, 0.0, 3.6048413262723854e+93, 4.332390992236853e+81, 2.483673610470445e+81, 0.0, 0.0, 0.0, 6.8341400099225961e+92, 0.0, 0.0, 0.0, 0.0, 2.5649633701852028e+97, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.5167110236553906e+90, 0.0, 0.0, 0.0, 2.0982938156651714e+83, 0.0, 0.0, 0.0, 0.0, 1.3606941442834981e+75, 1.1495617201573723e+77, 0.0, 0.0, 2.6640653810887371e+90, 5.4948609744615478e+88, 0.0, 3.129939386007785e+75, 0.0, 8.473493362995133e+96, 0.0, 0.0, 1.6089376861574175e+73, 1.2889439182628106e+92, 0.0, 0.0, 0.0, 0.0, 0.0, 4.598246806294893e+77, 1.2916149084938315e+92, 0.0, 0.0, 5.8380910418405651e+84, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.4834978972106665e+88, 0.0, 0.0, 0.0, 4.0692508433947771e+85, 1.3529553014415398e+98, 1.3080302633988032e+105, 0.0, 0.0, 0.0, 0.0, 3.8300135257252362e+89, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.5555127232765297e+80, 1.3722941238789894e+88, 0.0, 0.0, 0.0, 0.0, 3.2785840869768303e+81]

Le vecteur Ech nous montre que les grandeurs atteintes sont soit des valeurs extrêmement grandes soit extrêmement petites. Cela vient de la vraisemblance de la formule de sampling. La multiplication de petits nombres donnent des nombres extrêmement petits pour un nombre de multiplication important. Respectivement la multiplication de grands nombres donnent des nombres extrêmement grands pour un nombre de multiplication importants

Lorsque le nombre de simulation devient trop grand la vraisemblance diverge si les coefficients sont différents de 1. La formule de la vraisemblance, visible ci-dessous, nous que montre des coefficients légèrement supérieur (respectivement inférieur) à 1 amènent à des grandeurs extrêmement grandes (respectivement petite). Le risque est d'atteindre la limite de sotckage de python. Le produit suivant devient alors nul .

$$\left(\frac{1-p_{in}}{1-p_{in}'}\right)^{Card(Z_{in}=0)} \left(\frac{p_{in}}{p_{in}'}\right)^{Card(Z_{in}=1)} \left(\frac{1-p_{out}}{1-p_{out}'}\right)^{Card(Z_{out}=0)} \left(\frac{p_{out}}{p_{out}'}\right)^{Card(Z_{out}=1)}$$

Implémentation de l'algorithme - Deuxième Partie

Afin d'améliorer notre algorithme nous avons décidé de modifier la façon de générer les matrices d'adjacences. Dans le cadre de la question 2.1 nous voulons seulement savoir si un petit nombre d'individus est mal clusterisé. Il n'est donc pas nécessaire d'utiliser p_{in}' et p_{out}' pour chaque individu. Nous allons uniquement changer les probabilités pour les individus concernés (c'est-à-dire la probabilité des arrêtes entre les individus concernés et le reste du monde), ils sont stockés dans les listes notées list1 et list2 présentes dans la fonction indicatrice-générale-Q21 de clustering.py et dans la fonction nécessaire pour générer la matrice stochastique.

Le changement est important. Initialement le nombre de variable aléatoire est de $\frac{n(n+1)}{2}$. Au lieu de modifier la probabilité pour les $\frac{n(n+1)}{2}$ nous ne changeons la probabilité que pour $2n-1$ pour le cas de deux individus par exemple. En plus d'être plus intuitif, cette méthode réalise des petites perturbations ciblées et en même temps permet de ne pas saturer le facteur de vraisemblance puisqu'une bonne partie des coefficients du facteurs deviennent égaux à 1.

Reprenons l'algorithme de Stochastic Block Model avec les modifications nécessaires pour prendre en compte les différents types d'individus. (cf samplingGeneral.py et clustering.py)

Notons également que p_1 (respectivement p_2) correspond au nombre d'individu considérés pour faire parti de l'ensemble test pour le cluster C_1 (respectivement C_2). Ainsi, nous allons considérer la liste $list_1 = [v_0, ..v_{p_1-1}]$ (respectivement $list_2 = [v_{n'}, ..v_{n'+p_2}]$) qui représente la liste des éléments considérés (n' est le nombre d'éléments dans le cluster 1).

Algorithm 6 Stochastic Block Model -version sampling

procedure STOCHASTIC BLOCK MODEL($c_{in}, c_{out}, c_{in}', c_{out}', p_1, p_2, nprime$)
    Initialisation des deux listes $list_1$ et $list_2$
    for i de 1 à n **do**
        choix du couple de probabilité à utiliser
        if $v_i \in list_1$ ou $list_2$ **then**
            Utiliser le couple de probabilités (p_{in}', p_{out}')
        else
            Utiliser le couple de probabilités (p_{in}, p_{out})
        end if
        for j de 1 à n **do**
            Création du lien entre i et j en fonction de la Bernoulli correspondante
        end for
    end for
    Tous les liens sont créés

    Retourner la matrice d'adjacence créée
end procedure

Notons également qu'il faut modifier le code de la fonction indicatrice g pour l'adapter à cette nouvelle méthode. Le lecteur pourra lire ce code a nouveau sur fichier python : clustering.py Les résultats obtenus avec les modifications apportées à l'algorithme sont de meilleur qualité et sont cohérents avec les résultats obtenus avec la méthode naïve.

Sampling Généralisé

Données du problème
Nombre d'individus : 50
Nombre de simulations : 5000
$C_{in} = 22.0$ $C_{out} = 7.0$
$p_{in} = 0.44$ $p_{out} = 0.14$
$C_{in}' = 18.0$ $C_{out}' = 9.0$

$p_{in}{}' = 0.36$ $p_{out}{}' = 0.18$

Condition pour un clustering fonctionnel :
$15.0 > 10.65$ Condition vérifiée
$9.0 < 10.27$ Condition non vérifiée

Résultats
Probabilité obtenue = 0.0263207413444
Écart type = 0.119379317043
Demi largeur = 0.00330902584489
Intervalle de confiance : [0.0230117154995 , 0.0296297671892]

La méthode naïve permet de valider la méthode sampling. Remarquons que cette méthode donne des résultats que dans des cas favorables, c'est-à-dire dans des cas où l'algorithme de clustering peut faire des erreurs.

Méthode Naïve
Données du problème
Nombre d'individus : 50
Nombre de simulations : 5000
$c_{in} = 22.0$ $c_{out} = 7.0$
$p_{in} = 0.44$ $p_{out} = 0.14$

Résultats
Probabilité obtenue = 0.0296
Écart type = 0.169481090391
Demi largeur = 0.00469777614929
Intervalle de confiance : [0.0249022238507 , 0.0342977761493]

Nos deux méthodes nous donnent des résultats similaires ce qui permet de valider notre algorithme de sampling. Nous remarquons également que la demi-largeur trouvée est légèrement plus faible dans le cas de l'algorithme de sampling mais l'amélioration n'est pas réellement significative. L'importance sampling n'améliore pas réellement la précision mais permet d'obtenir des résultats dans les cas où le clustering est très favorable (c'est-à-dire dans des cas où la probabilité est très faible). Notons d'ailleurs que dans la simulation d'événement rare il n'est pas incohérents de trouver des demi-largeurs du même ordre de grandeur ou d'un ordre plus faible que le résultat.

Les résultats sont satisfaisants toutefois nous n'avons pas de stratégie pour déterminer les $c_{in}{}'$ et $c_{out}{}'$. Nous verrons un moyen pour déterminer ces paramètres dans la partie 2.5.

Après avoir vérifié les résultats donnés par l'algorithme de sampling. Nous allons tester notre algorithme de sampling dans des conditions où la clusterisation est très favorable.

Sampling Généralisé

Données du problème
Nombre d'individus :
Nombre de simulations :
$c_{in} = 42.0$ $c_{out} = 7.0$
$p_{in} = 0.84$ $p_{out} = 0.14$
$c_{in}{}' = 24.0$ $c_{out}{}' = 10.0$
$p_{in}{}' = 0.48$ $p_{out}{}' = 0.2$

Condition pour un clustering fonctionnel :
$35.0 > 13.8451842626$ Condition vérifée
$14.0 > 11.5329433444$ Condition vérifée

Résulats

Probabilité obtenue = 1.98153114134e-10
Écart type = 2.91148418627e-09
Demi largeur = 1.80455659444e-10
Intervalle de confiance : [1.76974546908e-11 , 3.78608773578e-10]

Parallèlement l'algorithme naïf ne permet plus d'obtenir de valeur pour ces valeurs de c_{in} et c_{out}. Notons que l'on a une demi-largeur d'un ordre de grandeur plus faible que le résultat ce qui est encourageant.

Approfondissement

Pour approfondir notre compréhension de l'algorithme il est possible d'étudier les probabilités en fonction d'autres paramètres. On pourra par exemple modifier la taille des clusters et voir l'incidence sur les probabilités. On pourra également faire varier la répartition des individus choisis à travers les clusters et voir l'incidence sur la probabilité. Notre algorithme prend en compte toutes ces possibilités car on peut à la fois modifier la taille des clusters avec nprime et modifier la répartition des individus ciblés à travers les deux clusters.

2.4 Probabilité de mauvaise clusterisation : Grand nombre de sommets

Contrairement à la partie précédente, nous nous intéressons maintenant à la probabilité qu'un grand nombre d'individus soit mal clusterisés.

Cet algorithme est différent de l'algorithme précédent; la première différence apparaît dans la génération de la matrice d'adjacence. Dans cet algorithme la totalité des probabilités des variables aléatoires est modifiée. Nous avons d'ailleurs implémenté une nouvelle fonction, appelée *matrice Adjacence Sampling Q2.2* dans le fichier *clustering.py* qui permet de générer les matrices d'adjacence pour la méthode sampling grand nombre.

L'algorithme sampling grand nombre (indicatrice-grdNombres-Q22 dans clustering.py) est également légèrement différent. La fonction de vraisemblance est différente de cette utilisée pour le sampling général avec un petit ensemble d'individus. Nous avons décidé de raisonner en terme de pourcentage de données mal clusterisée. Le coefficient multiplicatif utilisé est également différent (cf: l'implémentation du coefficient dans le fichier *SamplingGrandNombre.py*) Enfin, on se place dans des conditions $c_{in} - c_{out}$ resserrées sinon on se retrouve avec des probabilités trop faibles pour être repérées.

Sampling Généralisé
Données du problème
Nombre d'individus : 50
Nombre de simulations : 1000
Pourcentage de données mal clusterisées : 40
c_{in} = 19.0 c_{out} = 8.0
p_{in} = 0.38 p_{out} = 0.16
$c_{in}{}'$ = 12.0 $c_{out}{}'$ = 9.0
$p_{in}{}'$ = 0.24 $p_{out}{}'$ = 0.18

Condition pour un clustering fonctionnel :
11.0 > 10.2773839641 Condition vérifiée
3.0 < 9.06380069915 Condition non vérifiée

Résulats
Probabilité obtenue = 2.70468883531e-09
Ecart type = 3.35863230576e-08
Demi largeur = 2.08170187021e-09

Intervalle de confiance : [6.22986965103e-10 , 4.78639070552e-09]

On voit donc que déjà pour 40 pourcent de données mal clusterisées on se retrouve avec des probabilités basses. Le lecteur pourra en testant l'algorithme vérifier que l'augmentation du pourcentage de données mal clusterisées réduit considérablement la probabilité.

2.5 Optimisation des paramètres $c_{in}{}'$ et $c_{out}{}'$

Problématique : Nous voulons déterminer les paramètres $c_{in}{}'$ et $c_{out}{}'$ qui donnent le résultat le plus précis possible. C'est-à-dire les paramètres $c_{in}{}'$ et $c_{out}{}'$ qui minimise l'erreur: la demi-largeur de l'échantillon.

La demi largeur correspond à $1.96.\frac{\sigma_\theta}{\sqrt{n}}$ avec :

$$\sigma_\Theta = \sqrt{Var_\Theta\left(\frac{X}{L}\right)} = \sqrt{\mathbb{E}_\Theta\left(\left(\frac{X}{L}\right)^2\right) - \left(\mathbb{E}_\Theta\left(\frac{X}{L}\right)\right)^2}$$

Or, par le changement de probabilité :

$$\mathbb{E}_\Theta\left(\frac{X}{L}\right) = \mathbb{E}(X)$$

Donc il s'agit de trouver les paramètres $c_{in}{}'$ et $c_{out}{}'$ qui au final minimisent $\mathbb{E}_\theta\left(\left(\frac{X}{L}\right)^2\right)$

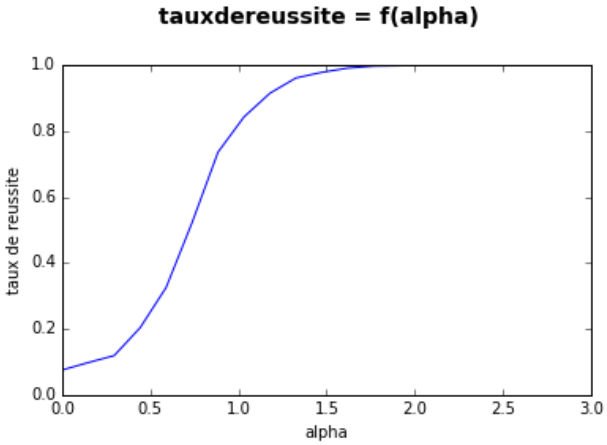
La première technique consiste à calculer pour plusieurs valeurs de $c_{in}{}'$ et $c_{out}{}'$ sur une grille la valeur de de cette quantité obtenue après plus simulations. Nous verrons que cette technique est très coûteuse en temps mais nous essayons toutes les possibilités.

La deuxième technique consiste à paramétrer le problème par rapport à la différence des deux paramètres $c_{in}{}'$ et $c_{out}{}'$. En effet, nous avons remarqué précédemment que la qualité du clustering dépend essentiellement de cet écart. Cette deuxième technique consiste donc à rapprocher au fur et à mesure les deux indices pour déterminer le couple qui minimise $\mathbb{E}_\theta\left(\left(\frac{X}{L}\right)^2\right)$.

Bien entendu pour chaque couple $(c_{in}{}', c_{out}{}')$ la probabilité de l'événement dépend de plusieurs processus aléatoires. Afin d'atténuer le bruit aléatoire de la simulation il faut moyenner sur un grand nombre d'expérience.

Toutes ces expériences sont donc très coûteuses en temps. Si l'on considère un incrément de 0,1 et une dizaine de steps. Il faut faire 100 fois la partie simulation. D'après nos expériences la fluctuation est très importante. Il faut simuler au minimum 1000 matrices d'adjacence afin d'obtenir une valeur avec le bruit le plus faible.

Afin de réduire le temps de calcul, on peux essayer de cibler la bonne plage de variation pour $(c_{in}{}', c_{out}{}')$. L'idée est de choisir (c_{in}, c_{out}) tel qu'une légère modification de ces derniers en les rapprochant va induire une grande modification du taux de réussite de l'algorithme. Cela permettra d'augmenter la probabilité d'échec sans trop faire varier les paramètres. Pour cibler ce type de point il suffit de reprendre le graphe suivant :



Il faudra donc prendre (c_{in}, c_{out}) au niveau de $\alpha = 1.5$ puis choisir $c_{in}{}', c_{out}{}'$ en partant de (c_{in}, c_{out}) et en les rapprochant petit à petit. On tombera au niveau du saut et on bénéficiera d'une bonne variation du taux de réussite.

Les deux listes $List_{C_{in}{}'}$ et $List_{C_{out}{}'}$ introduites dans le pseudocode ci-dessous correspondent aux valeurs de $C_{in}{}'$ et $C_{out}{}'$ a testé. Et la formule Indic(A) correspond à la fonction qui envoie 1 si les individus tests sont mal clusterisés et 0 sinon.

Algorithm 7 Optimisation des paramètres $C_{in}{}'$ et $C_{out}{}'$ -version grille

procedure OPTIMISATION DES PARAMÈTRES($C_{in}, C_{out}, nrpime$)
    Initialisation du vecteur Min $=\left[EsperanceCarre, C_{in}{}', C_{out}{}'\right]$
    Initialisation d'un vecteur Echantillon $=[Ech_0, .., Ech_M]$
    for $C_{in}[i]' \in List_{C_{in}{}'}$ **do**
        for $C_{out[j]}{}' \in List_{C_{out}{}'}$ **do**
            for j de 1 à M **do**
                Générer une matrice aléatoire
                Etape de clustering
                Echantillon.append(Indic(A)*P^2)
            end for
            $EsperanceCarre \leftarrow np.means(Echantillon)$
            if EsperanceCarre $<$ Min[0] **then**
                Nous avons trouvé un meilleur couple de $\left(EsperanceCarre, C_{in}{}', C_{out}{}'\right)$
                $Min \leftarrow \left[EsperanceCarre, C_{in}[i]', C_{out[j]}{}'\right]$
            else
                Le meilleur couple est inchangé
            end if
        end for
    end for
    Print le meilleur couple c'est-à-dire l'espérance au carré minimale obtenue pour le couple $(Min[1], Min[2])$
end procedure

Algorithm 8 Optimisation des paramètres $C_{in}{}'$ et $C_{out}{}'$ -version unidimensionnelle

procedure OPTIMISATION DES PARAMÈTRES($C_{in}, C_{out}, M, NbreSteps$)
  Initialisation du vecteur Min $=\left[EsperanceCarre, C_{in}{}', C_{out}{}'\right]$
  Initialisation d'un vecteur Echantillon $=[Ech_0, .., Ech_M]$
  while i $<$ Nbre Steps **do**
    for j de 1 à M **do**
      Générer une matrice aléatoire A
      Simulation d'une clustérisation
      $Echantillon[j] \leftarrow Indic(A) * P^2$
    end for
    $EsperanceCarre \leftarrow np.means(Echantillon)$
    if EsperanceCarre $<$ Min[0] **then**
      Nous avons trouvé un meilleur couple de $\left(EsperanceCarre, C_{in}{}', C_{out}{}'\right)$
      $Min \leftarrow \left[EsperanceCarre, C_{in}[i]', C_{out[j]}{}'\right]$
    else
      Le meilleur couple est inchangé
    end if
    $C_{in}{}' \leftarrow C_{in}{}'$ - incrément
    $C_{out}{}' \leftarrow C_{out}{}'$ + incrément
  end while
  Print le meilleur couple c'est-à-dire l'espérance au carré minimale minimale obtenue pour le couple $(Min[1], Min[2])$
end procedure

L'algorithme précédent nous permet d'obtenir deux graphes (on traite le cas de la mauvaise clusterisation de deux points) comme suit :

Données du problème

Nombre d'individus : 50

Nombre de simulations : M = 1000 (premier graphe), M = 5 000 (deuxième graphe)

C_{in} = 35.0 C_{out} = 16.0

$C_{in}{}'$ = [35,..,33] $C_{out}{}'$ = [16,..,18]

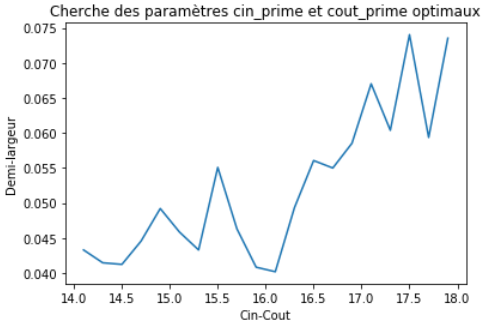


Figure 11: M=1000 simulations

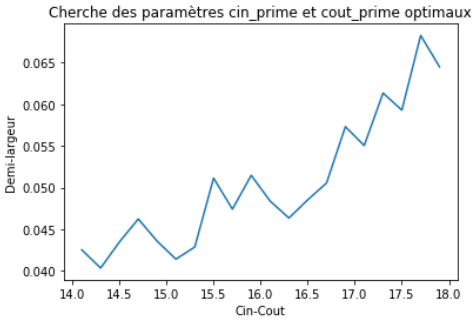


Figure 12: M=5000 simulations

Les deux graphes nous montre l'importance du bruit dans les processus aléatoires (la figure 11 plus que la 12). Nous voyons très bien la présence d'une tendance. Ce type de graphe est également un bon moyen afin de vérifier que le minimum obtenu ne résulte pas que du bruit aléatoire et que le résultat obtenu correspond bien à une valeur cohérente. Dans ce cas là par exemple on obtient pour valeurs optimales : $(C_{in}{}', C_{out}{}') = (33.1, 18.8)$. On remarque que la valeur ici correspond à l'écart le plus faible entre $C_{in}{}'$ et $C_{out}{}'$. On peut penser qu'en continuant à tracer on trouvera un

meilleur couple de paramètres. Néanmoins la tendance à la baisse de la valeur de la demi-largeur est réelle.

Conclusion

Pour conclure, nous avons réussi à implémenter tous les algorithmes demandés. Toutefois une réflexion reste à faire au niveau de la recherche des $c_{in}{}'$ et $c_{out}{}'$ optimaux. Dans cette recherche, nous avons mis en place un algorithme d'optimisation et nous avons donné une méthode intuitive pour choisir le bon couple de départ en fonction du graphique de taux de réussite. Dans notre recherche, nous avons surtout été confrontés à la problématique du temps de calcul. En effet, le calcul des vecteurs propres de la matrice ainsi que sa génération aléatoire prennent un temps important pour de grandes données. Un autre problème est celui du caractère aléatoire qui impose de moyenner sur plusieurs simulations pour pouvoir tracer des graphes sans bruits et exploitables. Une des grandes difficultés était également l'absence de données pour vérifier les résultats obtenus. Afin d'augmenter le temps de calcul on pourrait penser à paralléliser nos algorithmes sous MPI car ces derniers sont parfaitement parallélisables. Par exemple, la génération de la matrice du stochastic block model peut être partagée sur un cluster de machine sans aucune difficulté. Enfin, il y a énormément de paramètre à exploiter. On pourrait également essayer d'affiner les recherches pour voir l'effet du placement des individus et de la taille des clusters sur les probabilités trouvées.

References

[KN11] Brian Karrer and M. E. J. Newman. Stochastic blockmodels and community structure in networks. *Phys. Rev. E*, 83:016107, Jan 2011.

[Ver16] R. Vershynin. Four lectures on probabilistic methods for data science. *2016 PCMI Summer School*, Dec 2016.