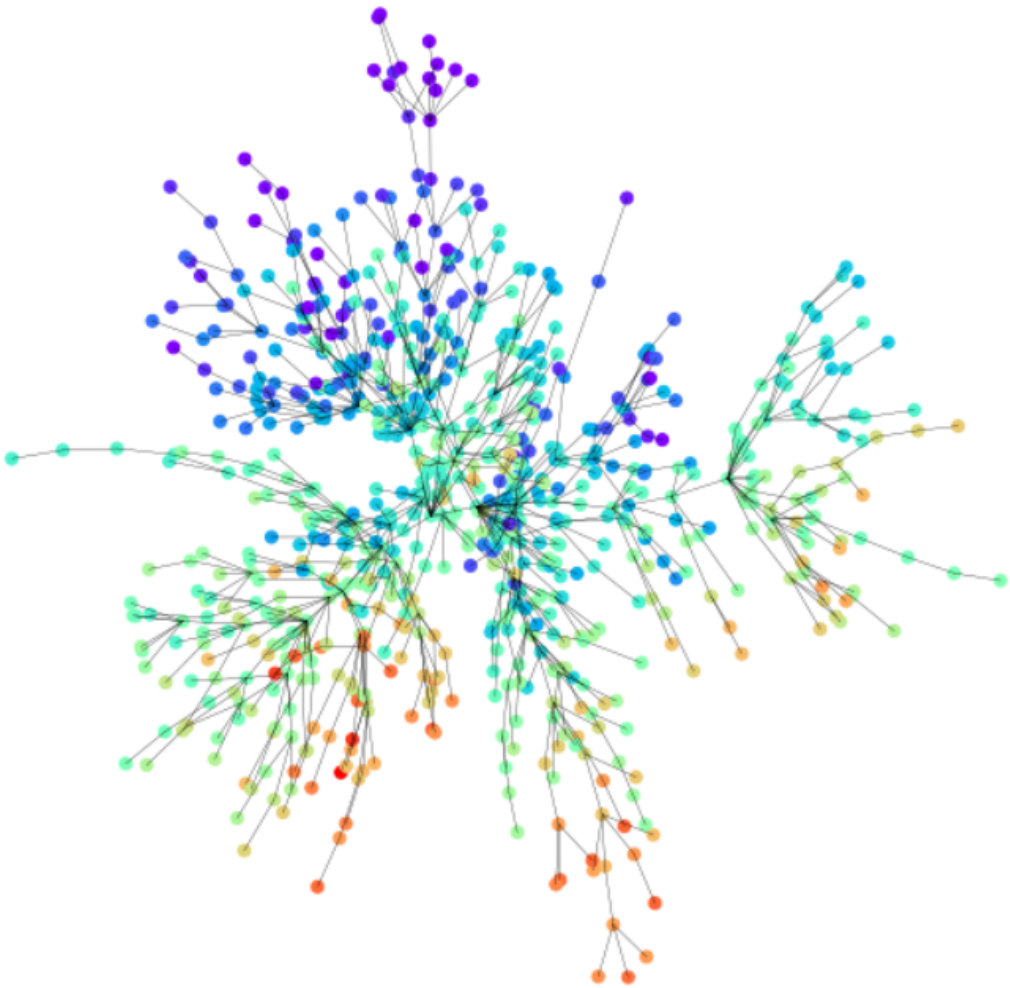
ÉCOLE POLYTECHNIQUE
UNIVERSITÉ PARIS-SACLAY

MAP572 : Projet Final–Graphes (modèles et algorithmes)

Enlin ZHU , Te SUN

Décembre 2018

Code address : https ://github.com/sun-te/ProjetMAP572



1 Simulation de graphes aléatoires : graphe par attachement préférentiel

1.1 Question 1.1

Écrire un script qui simule le graphe G_n. Pour l'instant on représente un graphe par sa matrice d'adjacence $n \times n$ à valeurs dans $\{0, 1\}$ définie par Adjacence(i,j) = 1 si i et j sont voisins dans le graphe.

Réponse : D'abord, on crée un objet python qu'il s'appelle graghG. On stocke le nombre des sommets (self.n), la matrice d'adjacence (self.adjacence), et la matrice de la distance minimale entre deux sommets (self.distance) pour question 2. De plus, on choisit les positions de chaque sommet (self.pos) aléatoirement entre $\{0, 1\} \times \{0, 1\}$. Après, on crée la fonction pour calculer la matrice adjacente.

Le code est comme suivant :

```
def Matrix_delta(n,delta=0):
    n=int(n)
    ans = np.zeros((n,n))
    ans[0,0] = 1;
    degree = np.array([1])
    for i in range(1,n):
        proba=np.cumsum((degree+delta)/(2*i-1+i*delta))
        flag=np.random.uniform(0,1)
        for j in range(len(proba)):
            if(flag<=proba[j]):
                v=j
                break
        ans[i,v],ans[v,i] = 1,1
        degree=np.sum(ans,axis=0)
    return ans
```

1.2 Question 1.2

L'une des caractéristiques de ce modèle est que les degrés des sommets suivent approximativement une loi de puissance (power-law) : si s est un sommet tiré uniformément dans G_n,

$$\mathbb{P}(deg(s) = k) \approx^{kgrand} ck^{-\alpha}$$

où c, α sont des constantes positives. Pouvez-vous illustrer ce phénomène (et évaluer le coefficient α) sur une simulation de G_n ?

Réponse : Il y a deux façons pour illustrer le résultat de cette question.

Premier : On ne simule un graphe qu'une fois avec un grand n. Après, on cumule le degré de chaque sommet. Pour vérifier une loi de puissance, on trace le **fréquence-degré pré-formalisé par la "log" fonction**. Si ils ont la loi de puissance, tous les points seront aligné par une droite.

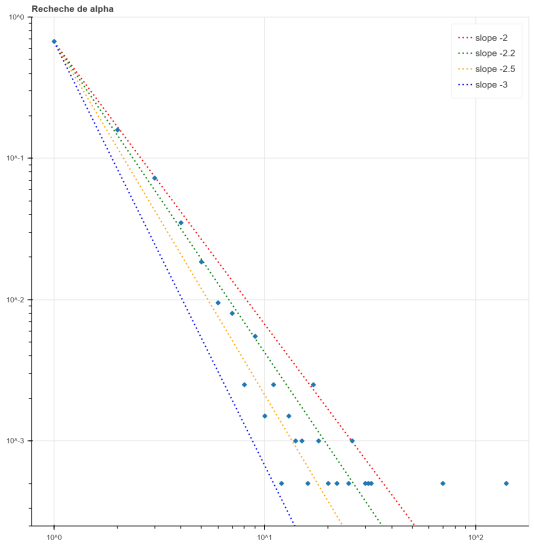
```
n=2000
M=1
dict_counter={}
for i in range(M):
    graph=Graph.grapheG()
    graph.reload(Adjacency.Matrix_delta(n,0))
    Adj=graph.adjacence
    degree=np.sum(Adj,1)
    degree.sort()
    degrees=np.unique(degree)

    for d in degree:
        if(d in dict_counter.keys()):
            dict_counter[d]+=1.0
        else:
            dict_counter[d]=1.0
df_res=pd.DataFrame(dict_counter, index=['num']).T
print(df_res)
degrees=np.array(df_res.index.tolist())
total=df_res['num'].sum()
count_d=np.array((df_res['num']/total).tolist())
```

Deuxième : On a observé que le calcul d'une matrice adjacente de taille N a la complexité n^2. Une idée est de faire échantillons sur plusieurs graphes de la taille presque-grande. Dans notre programme, on change les paramètres. Maintenant, n = 100, m = 40.

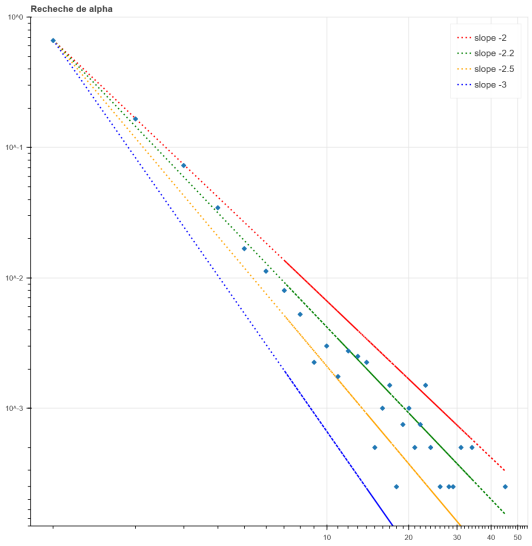
Commentaire : Bien que l'on prenne un grand n, **la fréquence de grand degré est creuse**. Par exemple, dans une expérience avec 1000 sommets, degré 101, 78, 65, 53, 51 apparaît une fois. Dans une autre expérience avec 1000 sommets, degré 201, 99, 73, 60, 54 n'apparaît qu'une fois. La fréquence du grand degré dans une seule expérience est toujours **quelques 1 avec pleins de 0**. Afin de mieux représenter la distribution de la fréquence de grand degré, on considère de **faire échantillons sur plusieurs graphes** dont résultat est plût proche d'une droite. Les dessins sont comme suivant. Selon le résultat de la première méthode, on pense que α est plus proche de 2. Selon le résultat de la deuxième méthode, on est sûr que α **est entre 2.2 et 2.5**. Tous les cas preuve que la distribution suit une loi puissance et la coefficient est entre -2.5 et -2.2. Comme la vie économique, **le plus riche que les gens sont, le plus rapide qu'ils agrandissent leur richesse**. À la fin, la distribution de la richesse suit une loi puissance.

1.3 Question 1.3

[**Théorique**] Dans l'article original, les auteurs prétendent que leur modèle reproduit "a rich-gets-richer phenomenon that can be easily detected in real networks". Pour illustrer le fait que G_n est



Échantillons sur une grande graphe



Échantillons sur plusieurs graphes

Figure 1 – La courbe de la loi puissance de la distribution du degré

extrêmement non-homogène, estimer l'espérance du degré du sommet l dans G_n, et comparer avec le degré moyen d'un sommet. Indication : On pourra calculer $E[X' + 1|X]$où X' est le degré du sommet 1 lorsque le l-ème sommet apparaît dans le graphe.

Réponse : Supposons que X_l est le degrée du sommet 1 lorsque le l-ème sommet apparaît dans le graphe

$$\begin{aligned}
\mathbb{E}[X_{l+1}|X_l] &= X_l + \frac{X_l}{2l-1} \\
&= X_l \frac{2l}{2l-1} \\
\mathbb{E}(\mathbb{E}[X_{l+1}|X_l]) = \mathbb{E}[X_{l+1}] &= X_1 \prod_{i=1}^{l} \frac{2l}{2l-1} \\
&= \frac{(l!)^2 2^{2l}}{(2l)!} \\
&\approx^{\text{l-est-grand}} \frac{(\sqrt{2\pi l}(\frac{l}{e})^l)^2\, 2^{2l})}{\sqrt{2\pi 2l}\frac{2l}{l}^{2l}} \\
&= \sqrt{\frac{\pi}{2}l}
\end{aligned}$$

Cependant, **le degré moyen est présenté comme** $\frac{2l+1}{l+1} \approx 2$**. On a constaté donc ce phénomène de non-homogénéité**.

2 Visualisation d'un graphe : un problème d'optimisation

2.1 Question 2.1

Tracer le graphe de votre choix de la façon suivante :

1. Tirer au sort n points uniformes indépendants $(X_1, Y_1), ..., (X_n, Y_n)$ dans le carré [0,1]
2. Si i,j sont voisins dans G_n, tracer un segment entre $(X_i, Y_i) et (X_j, Y_j)$.

Réponse : Pour commencer, on a tracé un graphe avec une initialisation aléatoire dans un plan 2D. (Fig 2)

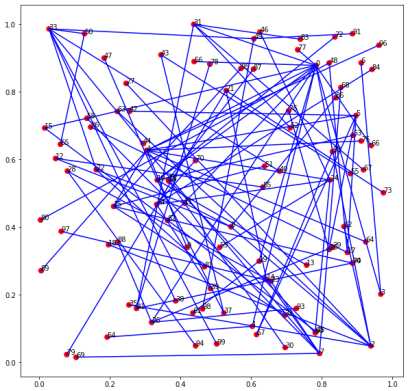


Figure 2

Á partir d'ici, on cherche à optimiser le graphe de fa con que la distance physique sur un graphe soit plus représentative pour une distance topologique. Pour ce faire, on défini une énergie à minimiser :

$$E = \sum_{i,j} \frac{(\frac{1}{\sqrt{\text{dimension}}}||M_i - M_j|| - D^*_{i,j})^2}{(D^*_{i,j})^2}$$

où

$$D^*_{i,j} = \frac{Distance(i,j)}{\max_{i,j} Distance(i,j)}$$

2.2 Question 2.2

Tracer le graphe G_n de façon à minimiser le plus possible l'énergie E. On pourra utiliser des méthodes de type gradient. Vous pouvez aussi modifier le choix de E pour obtenir des graphes plus harmonieux. Enfin, on peut se poser la question de représenter le graphe en dimension 3, voire d'utiliser des couleurs.

Réponse : Étant donné que la fonction énergie est quadratique, donc convexe, on peut appliquer la méthode de gradient sans s'inquiéter une solution du minimum local. Nous avons implémenté trois

type de méthode de gradient : **Gradient à Pas Fixe, Gradient à Pas dit "Armijo" et Gradient à Pas dit "Barzilai-Borwein"** :

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla E(\mathbf{x}_n),\ n \geq 0.$$

La différence entre ces trois méthode se lie aux choix de la longueur γ du pas pendant chaque itération :

1. Gradient à Pas Fixe (script : `TreePy->Graph->GPF`)

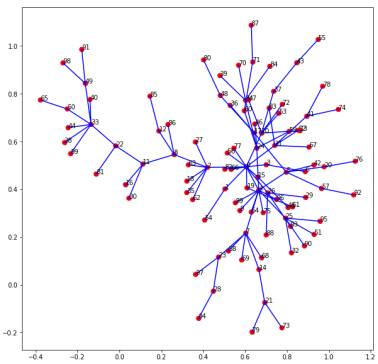
$$\gamma_n = \text{constante}$$

2. Gradient à Pas dit "Barzilai-Borwein" (script : `TreePy->Graph->GPO_d`) :

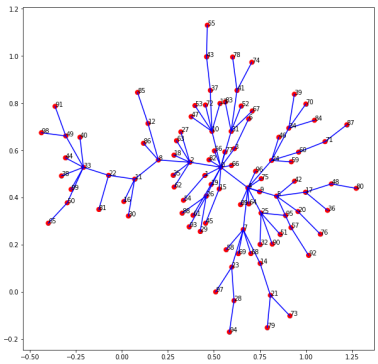
$$\gamma_n = \frac{(\mathbf{x}_n - \mathbf{x}_{n-1})^T [\nabla E(\mathbf{x}_n) - \nabla E(\mathbf{x}_{n-1})]}{||\nabla E(\mathbf{x}_n) - \nabla E(\mathbf{x}_{n-1})||^2}$$

3. Gradient à Pas dit "Armijo" (script :`TreePy->Graph->GPO_Armijio`) :
   `while` $E(\mathbf{x}_n - \gamma_n \nabla E(\mathbf{x}_n)) > E(\mathbf{x}_n) - \alpha\gamma_n \times ||\nabla E||^2$:
   $\gamma_n := \gamma_n \times \beta$
   où α et β est de petites constantes dont le choix dépendent le système étudié.

Pour calculer le gradient de la fonction énergie, nous avons utilisé l'approximation par la différence finie. (script :`TreePy->Graph->Gradient_1d`). L'optimisation du graphe est présentée ci-dessous :



Graph obtenu par `GPO_d`



Graph obtenu par `GPO_Armijio`

Figure 3 – Deux images obtenues après optimisations

Selon le Figure 3, **nous pouvons constater que la méthode `GPO_Armijio` nous donne un graphe plus "harmonieux". En effet, par même nombre d'itérations, l'énergie finale obtenue par `GPO_Armijio` est plus petite.** Cela est du fait que : 1) le critère d'arrêt du `GPO_Armijio` est imposé sur la norme du gradient, alors que `GPO_d` va s'arreter lorsque la valeur de fonction $E(.)$ converge avec certaine tolérance. 2)la longueur du pas `GPO_Armijio` est décroissante, ce qui s'assurer la ralentissement de la recherche à la fin. `GPO_d` risque de se tomber dans une piège aller-retour sur la fonction $E(.)$.

Pourtant, **`GPO_Armijio` a besoin plus de temps au début**, car il s'agit une recherche linéaire de γ_n et que l'état initial est trop aléatoire que l'on ne soit pas assez audacieux à descendre (avec un pas plus grande).

La stratégie qu'on prend est d'utiliser `GPO_d` pour commencer pendant 50 itérations et puis d'utiliser le `GPO_Armijio` afin d'obtenir un résultat plus satisfaisant.

Après les premiers testes sur les scripts, on a implementé (`TreePy->Graph.py->Gradient_exact`) finalement **une fonction qui calcule le gradient exact de E.** L'expression est donnée par :

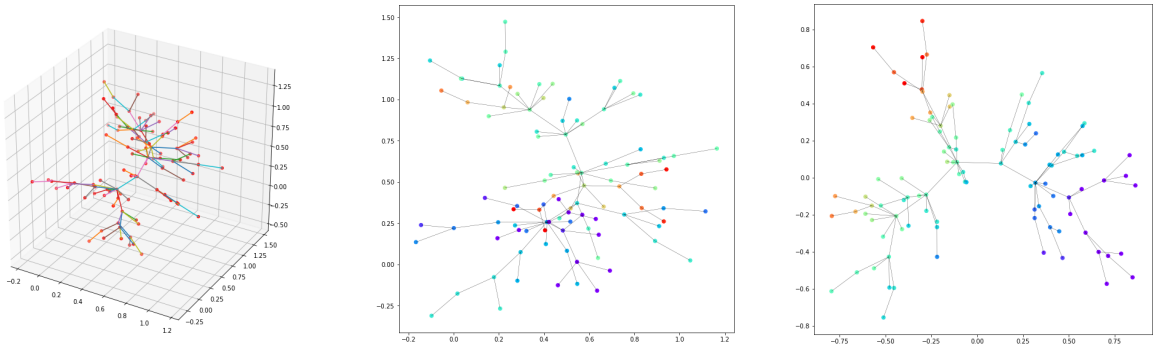
$$\frac{\partial}{\partial x_i}E(x_1,y_1,x_2,y_2,...,x_n,y_n) = 2\sqrt{2}\sum_{j\neq i}\frac{1}{D^*_{ij}}\left(\frac{1}{\sqrt{2}D^*_{ij}} - \frac{1}{||M_i - M_j||}\right)(x_i - x_j)$$

et

$$\frac{\partial}{\partial y_i}E(x_1,y_1,x_2,y_2,...,x_n,y_n) = 2\sqrt{2}\sum_{j\neq i}\frac{1}{D^*_{ij}}\left(\frac{1}{\sqrt{2}D^*_{ij}} - \frac{1}{||M_i - M_j||}\right)(y_i - y_j)$$

On remarque que la vitesse d'optimisation par gradient exact est beaucoup plus vite que celle par différence finie. En calculant le gradient approché, d'une part, pour chaque point (x_i, y_i) on doit calculer une énergie associé et une autre pour les positions aprés incrément $(x_i + \delta, y_i)$ et $(x_i, y_i + \delta)$. D'ici, on a une complexité $\mathcal{O}(n)$ pour cette procédure ; d'autre part, pour calculer l'énergie, il s'agit d'une complexité de l'ordre $O(n^2)$. Le coût total de la différence finie est donc $\mathcal{O}(n^3)$. Mais, le calcul exact du gradient coûte $O(n^2)$, ce qui accélère notre optimisation.

Pour illustrer cette optimisation en 3D, on a effectué une réalisation pour un système de 3D. (Fig 13)



Présentation en 3D après l'optimisation | Projection en 2D sur $Vect(x, y)$ | Projection en 2D par PCA

Figure 4

La représentaion du graphe 3D sur un papier 2D conduit à une perte d'information. Même en représentant l'axe z par la couleur, le graphe n'est pas si harmonieux. **Pour éviter les chevauchements entre les arrêtes, une solution possible c'est d'effectuer un PCA (principal component analysis) pour trouver le plan sur lequel l'on peut garder plus d'information après la projection**. Effectivement, l'image à gauche retrouve l'harmonie qu'on a perdu avant.

2.3 Question 2.3

On peut généraliser le modèle du graphe à attachement préférentiel de la façon suivante. Soit $\delta > -1$ un paramètre fixé, on remplace l'équation (*) par

$$\mathbb{P}'(v_{k+1} = i) = \frac{\deg(i) + \delta}{\sum_j(\deg(i) + \delta)} = \frac{\deg(i) + \delta}{2k - 1 + k\delta} \quad (*)$$

Tracer des graphes G_n pour différentes valeurs de δ. Comment peut-on décrire l'influence de δ sur G_n ?

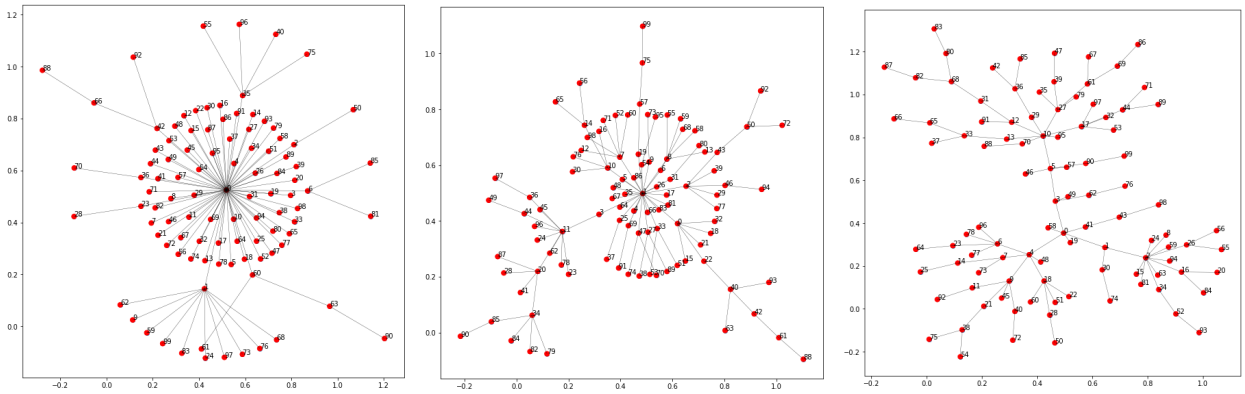
Réponse :

$$(\mathbb{P}' - \mathbb{P})(v_{k+1} = i) = \frac{(2k-1)(x+\delta) - x(2k-1+k\delta)}{(2k-1+k\delta) \times (2k-1)}$$
$$= \frac{\delta((2-x)k-1)}{(2k-1+k\delta) \times (2k-1)}$$

Oú on note x le degé du point i et k le nombre total des points existants. Sachant que $x \geq 1$ et $x \in \mathbb{N}$, quand $k \geq 2$ $\delta((2-x)k-1)$ ne peut pas être nul.

1. Si $\delta = 0$, on retrouve le cas précèdent.
2. Si $-1 < \delta < 0$ et que $x = 1$, alors $\delta((2-x)k-1) < 0$, la probabilité que le point i soit attaché par un nouveau diminue. Si $x \geq 2$, la probabilité augemente. Le "a rich-gets-richer phenomenon" est renforcé.
3. Sinon, par analogue, on a dilué l'effet de "rich-gets-richer". Le monde devient plus juste

Ici, on présente 3 trois cas typique avec $\delta \in \{-0.9, 0, 10\}$: (Fig 5)



$\delta = -0.9$ $\delta = 0$ $\delta = 10$

Figure 5

Plus δ est grand, plus uniformément les points sont répartis.

3 Clustering et détection de communautés : utilisation de la décomposition spectrale

3.1 Question 3.1

Implémenter l'algorithme sur le graphe par attachement préférentiel pour différentes valeurs de k.

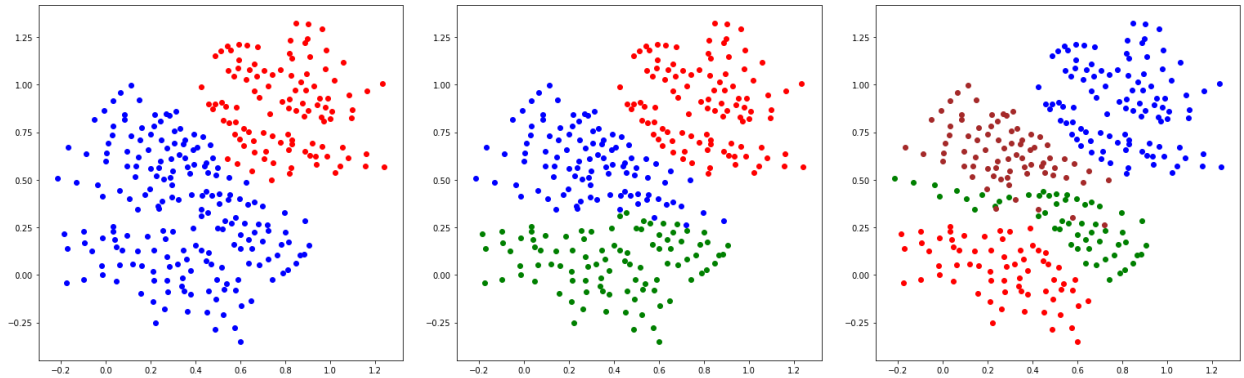
Réponse : Nous avons implementé deux méthodes mentionnée dans l'article (U.V. Luxburg, *A Tutorial on Spectral Clustering*, 2007, p.5-6).

1. Unnormalized spectral clustering (script : `TreePy->Cluster.py->Unnormalized`)
2. Normalized spectral clustering according to Ng.Jordan and Weiss (2002) (script : `TreePy->Cluster.py->Normalized_sym`)

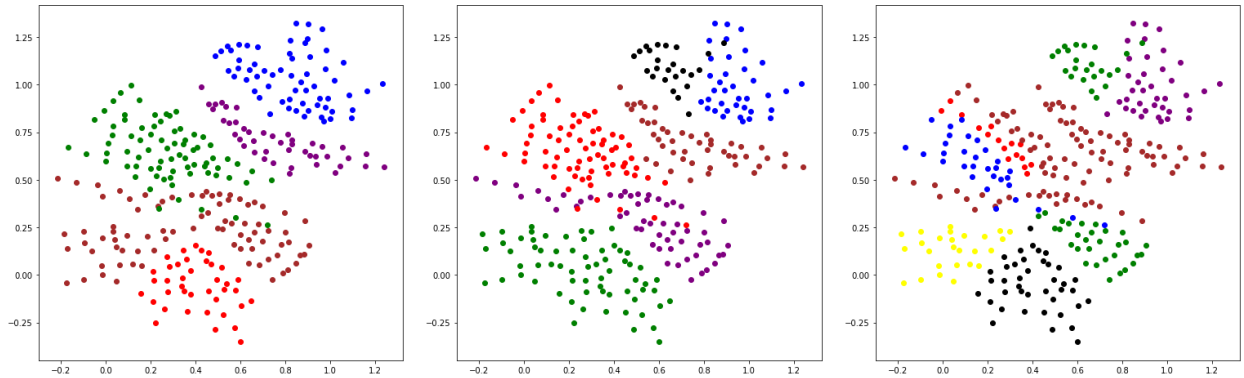
Le graphe suivant (Fig 6) montre le résultat par Normalized_sym. **On défeni une fonction distance qui se manifeste comme un critère de clustering :**

$$\text{distance} = \sum_{j=1}^{k} \sum_{i \in \mathcal{P}_j} ||(x_i, y_i) - (\bar{x}_j, \bar{y}_j)||_2$$

Où $(\bar{x}_j, \bar{y}_j)$ est la coordonnée du centre de la $j^{ième}$ partition $\mathcal{P}_j$(cluster). Si cette "distance" est grande, le clustering peut encore s'améliorer.



$k = 2$, distance=76.3 | $k = 3$, distance=71.7 | $k = 4$, distance=65.94



$k = 5$, distance=86.0 | $k = 6$, distance=98.5 | $k = 7$, distance=105.87

Figure 6 – Spectral clustering pour le graphe par attachement préférentiel. n=300

Visuellement, avec cette réalisation du graphe (n=300) par attachement préférentiel, nous retrouve 4 (ou 3 ?) sous parties, ce qui est confirmé par l'algorithme.

3.2 Question 3.2

Vous pouvez simuler quelques graphes obtenus selon le Stochastique Bloc Modèle, les tracer et essayer de retrouver les différentes classes avec l'algorithme Spectral Clustering. (Essayer avec $n \approx 200$, et $2 \leq K \leq 5$ classes.)

Réponse :
Nous avons d'abord construit des matrices adjacente par la définition du *Stochastique bloc modèle*. Avec nombre des sommets $n = 200$ et nombre des blocs $k \in \{3, 4, 5\}$. Finalement, pour la construction de la matrice Q, nous avons utilisé une forme symétrique et simple :

$$Q = \begin{bmatrix} 1 & \varepsilon & \dots & \varepsilon \\ \varepsilon & 1 & & \vdots \\ \vdots & & \ddots & \varepsilon \\ \varepsilon & \dots & \varepsilon & 1 \end{bmatrix}$$

Alors, pour un bloc, tous les points sont connctés aux les autres qui sont de même bloc. Mais, avec une faible probalbilité ε qu'ils soient connctés aux points qui sont de l'autre groupe.

Puis, on a effectué l'optimisation qui est décrite dans la section 2 pour que le graphe soit plus compréhensible.

Par le spectral clustering, nous avons obtenu des résultats suivants (Fig 8, étant donnée trop d'arêtes, on trace 5% des liens entres les sommets.) :

1. L'optimisation du graphe détecte non seulement l'existence des groupes, c'est-à-dire, de bien séparer les groupes pré-définis ; mais aussi les sous parties qui sont générés par l'interaction entre les blocs.
2. *Spetral Clustering* peut facilement retrouver les blocs.

D'ailleurs, **on a fait une optimisation du graphe pour le modèle *SBM* en 3D. Deux projection 2D, une avec la décomposition PCA et une autre sans PCA.**

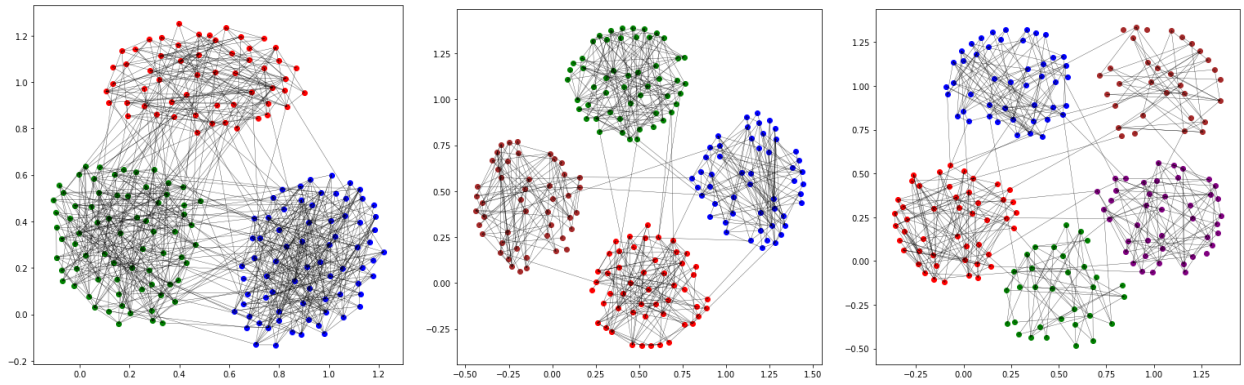
Le chevauchent entre deux groups nous pose visuellement diffuculté de distinguer les blocs originaux. PCA nous a aidé à séparer les trois blocs, ce qui confirme notre supposition dans la question 2.2. (Pour le faire, nous avons utilisé le package `sklearn.decomposition.PCA`)

Ensuit, on a repris l'exemple (matrice adjacente de taille 600×600) sur moodle pour tester le spectral clustering (Fig9).

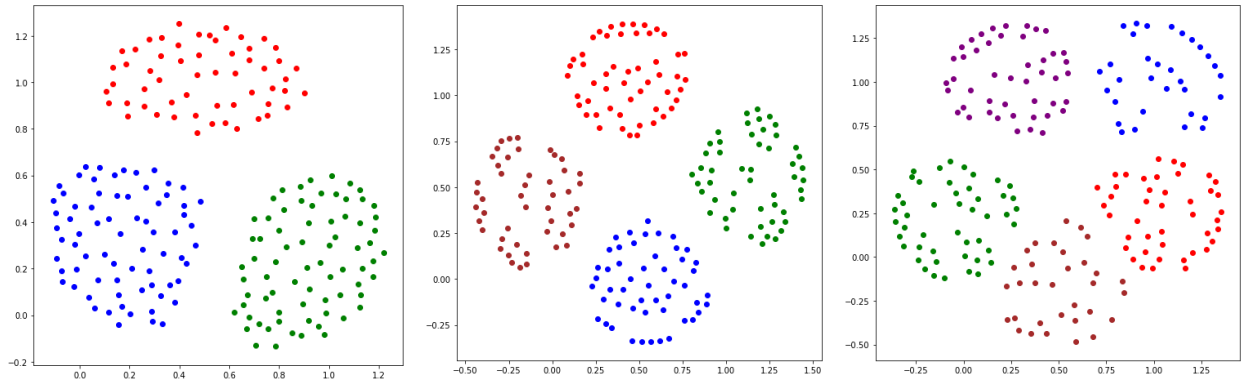
Après l'optimisation du graphe, nous avons deux blocs bien séparés mais pas de form ronde.

On a espéré que le resultat serait meilleur par une optimisation en 3D. Donc, on a cherché à ajouter plus d'informations en rajoutant une autre dimension et l'optimisant en 3D. Mais nous n'avons pas de meilleur résultat : **d'un côté, l'optimisation en 3D a besoin plus de temp, d'autre côté, l'optimisation en 3D n'est plus fiable car la taille du pas pour la méthode de gradient diminue jusqu'à 10^{-21} qui dépasse la précision du type `float`.**

Dans notre cas, il nous semble que la méthode *normalisée avec L symétrique* est plus performante.

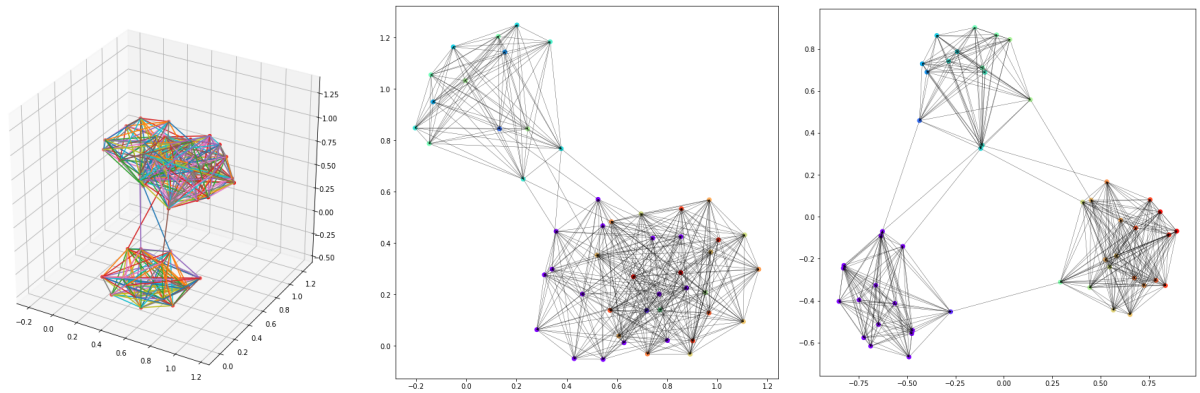


Les groups originaux, k=3 | Les groups originaux, k=4 | Les groups originaux, k=5



Le clustering détecté | Le clustering détecté | Le clustering détecté

Figure 7 – Spectral clustering (normolized method) pour le graphe par SBM. n=200



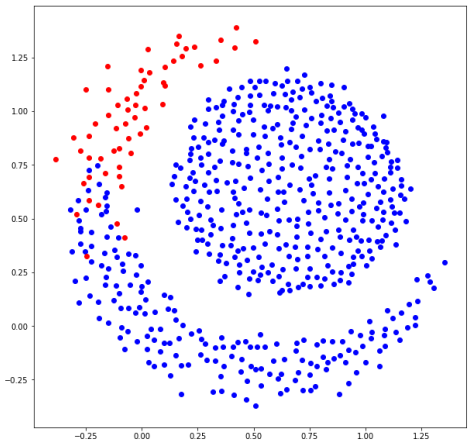
Représentation en 3D | Projection sans PCA | Projection après PCA

Figure 8 – Spectral clustering (normolized method) pour le graphe par SBM. n=200

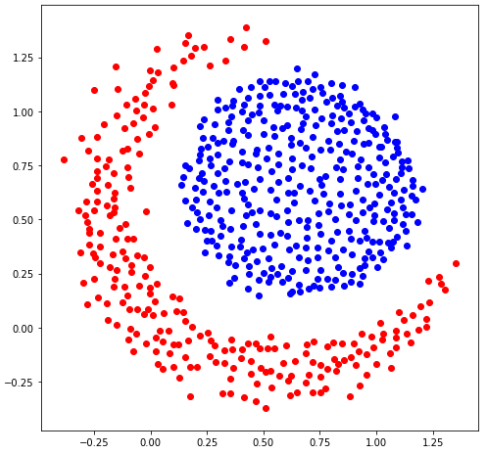
3.3 Question 3.3

Ne pas hésiter à comparer Spectral Clustering avec le clustering obtenu en effectuant simplement k-means sur les graphes dessinés dans la partie précédente.

D'abord, K-means dépend essentiellement de la position physique (la position sur le graphe). Il n'est pas sensible aux interactions dans le graphe. Ce phénomène est manifeste par les deux premiers



Unnormalized Spectral Clustering



Normalized Spectral Clustering by L_{sym}

Figure 9

colonnes des images : même avant l'optimisation, le spectral clustering est bien semblable aux blocs originaux, car le *Spectral* utilise seulement la matrice d'adjacence comme le critère de distance. Pourtant, K-means utilise seulement les positions physiques.

On peut le constater également sur les images à droite. Les points 72, 44, 79, 89 est repartis au bloc rouge bien qu'il n'y ait aucune liaison entre ces points et le bloc rouge. De mme, les deux branches bleues séparés par k-means ne doivent pas appartenir au même bloc.

Une deuxième remarque : intuitivement, la distance qu'on a définit au début de cette section tend décoitre avec k. En utilisant K-menas, on peut imaginer s'il y a autant de clustering que le nombre des points. Chercher un k approprié est souvent un challenge lors l'application de k-means. Et la difficulté émerge particulièrement quand la dimension du système devient grande. Alors tout point devient un clustering en soi-même, la distance est nulle. Un "meilleur" k pour k-means serait le "Coude" (`http://www.scikit-yb.org/en/latest/api/cluster/elbow.html`) (où la pente de la courbe change le plus rapide, ici k=3 pour K-means) de la courbe distance.

Pourtant, **il est intéressant de trouver un minimum pour la fonction distance de *spectral clustering.*** Il peut donc détecter automatiquement le meilleur k pour le clustering pour éviter le problème comme **Overfitting**. Encore, on remarque que le spectral clustering est moins "stable" que K-means. Cela serait dû au fait qu'on a seulement pris k les plus représentatifs composants pour clustering, où il y a une perte d'information qui amplifie l'effet aléatoire de k-menas dans l'algo *Spectral Clustering*.

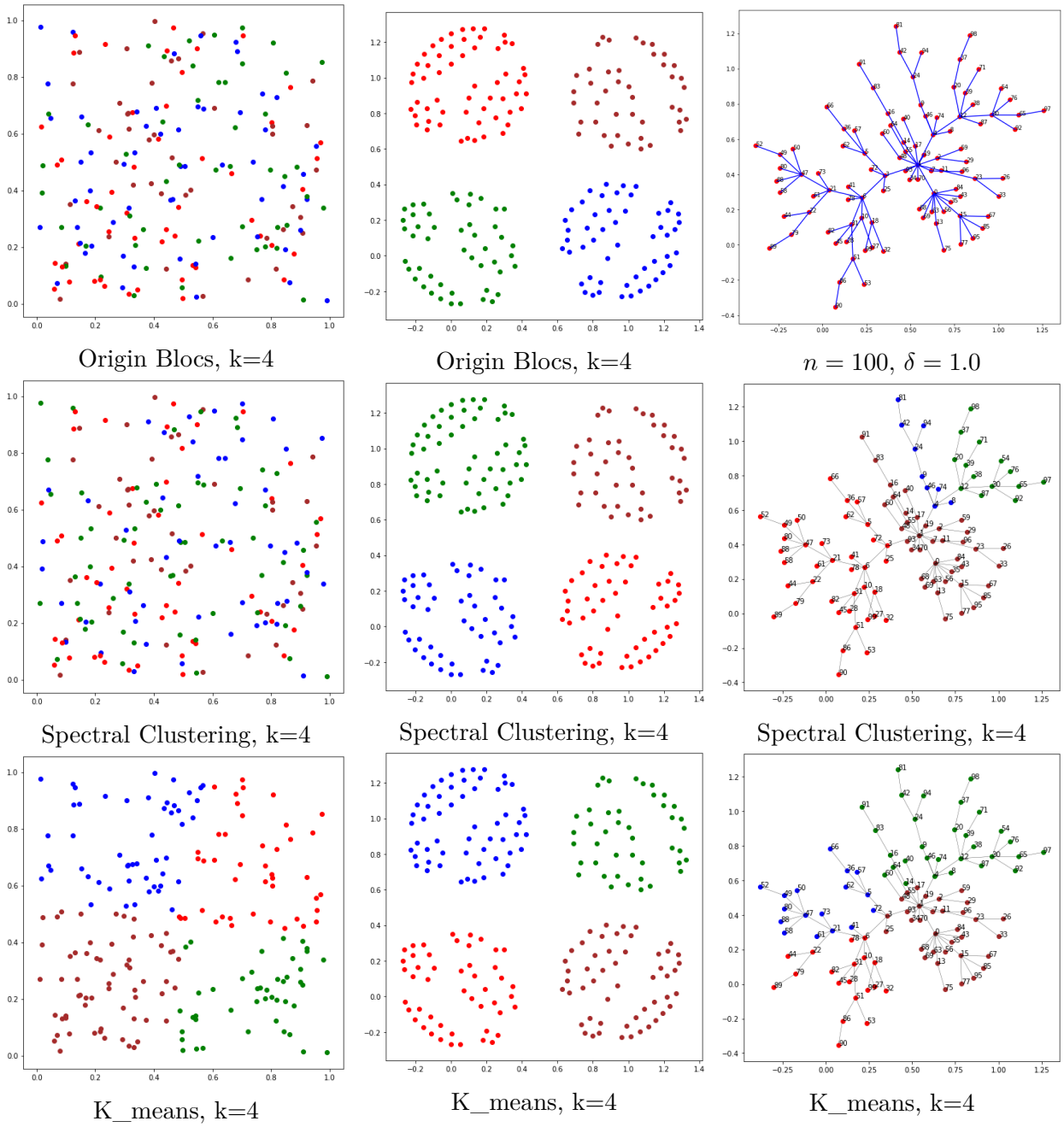
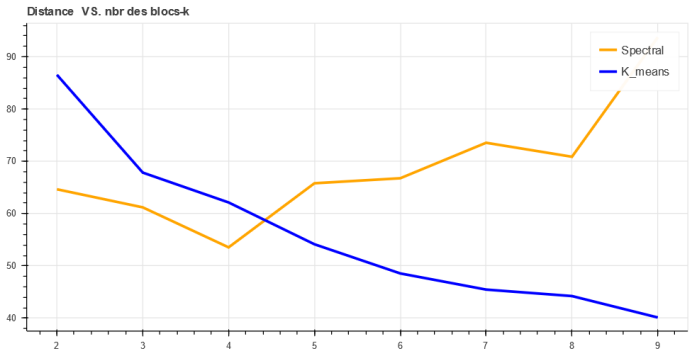


Figure 10 – Comparaison entre K-means et Spectral Clustering

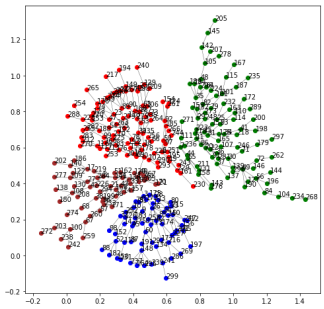
4 Popularité dans un graphe : une approche par valeurs propres (PageRank)

4.1 Question 4.1

En évaluant les scores PageRank sur les quels graphes de petite taille, discuter du rôle de ε (pour donner une idée, Brin & Page proposaient initialement de prendre $\varepsilon = 0{,}15$). Discuter de l'implémentassions de l'algorithme pour des graphes de grande taille.Quels algorithmes de recherche de



Distance en fonction de k



le "meillieur" k pour spectral Clustering

Figure 11 – La courbe de la loi puissance de la distribution du dégrée. Visuellement, on confirme le choix du k=4 par l'algo

valeurs/vecteurs propres peut-on proposer qui soient utilisables dans ce cas (on pourra commencer par étudier la méthode de la puissance) ?

Réponse :
La rôle de ϵ est mesurer la pourcentage de "trous noirs" dans le graphe. "Trous noirs" sont des sommets qui ne touchent que lui-même. Dans le sens réel, ce sont des sites qui absorbent des links des autres sites mais **ils ne veulent pas mener les clients aux autres sites**. Dans ce cas, les clients va considérer visiter une autre site aléatoirement. Selon notre expérience, si le ϵ tends vers 0, les "trous noirs" va rassembler la majorité de la probabilité. Par contre, si le ϵ tends vers 1, la distribution de la visiteur fréquence est identique. C'est raisonnable pour la situation réel. Si tous les sites sont "trous noirs", ils sont des îles isolées de telle sorte que la fréquence de leur visiter est identique. Si tous les sites connectent bien avec les autres sites, ils vont entrer dans un **état stable** comme la matrice adjacente normalisée.

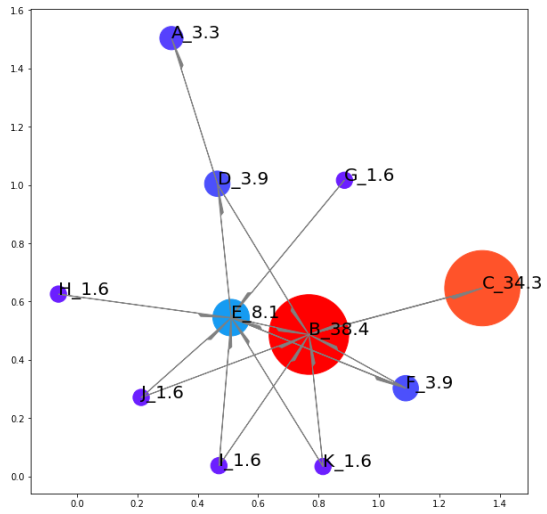
Pour calculer le grand graphe, le premier vecteur propre liée à la valeur propre 1 est le point fixé de la transformation linéaire P_ϵ, alors on parti de n'importe quelle point X initiale, et continue à multiplier la matrice P_ϵ à gauche jusqu'à la convergence. Si le processus de matrice multiplication converge lentement, on peut utiliser la puissance du matrice dans la complexité du temps $O(N^3) * \log_2(\text{nombre} - \text{itération} - \text{originale})$. De plus, si on ne supporte pas la taille énorme, on peut résoudre l'équation $(P_\epsilon - I)X = 0$ par **la gauss élimination** et construire une solution symbolique non-zéro. Après, supposé que la somme du vecteur égale 1, on peut fixer cette vecteur. Pour éviter que la ligne zéro apparaît dans un endroit non-préalable, on choisit le Abs-plus grand chiffre dans un ligne comme **le pivot**.

Le algorithme est comme suivant avec la complexité $\mathcal{O}(N^3)$:

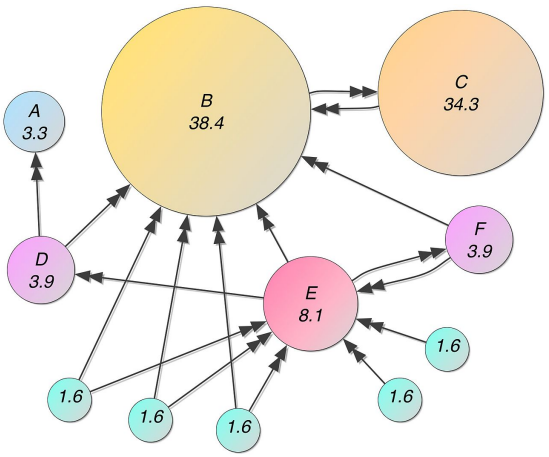
```
def Resolve(A):
    l = len(A)
    M = A.copy()
    Order = np.array([list(range(l))])
    M = np.concatenate((M, Order), axis=0)
    ans = np.zeros(l)
    for i in range(l):
```

```
        for j in range(l):
            if abs(M[i][j]) > abs(M[i][i]):
                M[:,[i, j]] = M[:,[j, i]]
        if abs(M[i,i]) < 1e-8:
            tmp = np.array([-M[x][l-1] for x in range(l-1)])
            for k in range(l-1):
                ans[int(M[l][k])]=tmp[k]
            ans[int(M[l][l-1])]=1
            ans = ans/np.sum(ans)
            return ans
        M[i]/=M[i,i]
        for j in range(l):
            if j != i:
                M[j]-=(M[j,i]/M[i,i])*M[i]
    return ans
```

Présentation de notre algorithme : (Figure 12)



La visualisation de PageRank dans notre modèle



Graphe orienté sur wikipedia et son PageRank

Figure 12 – PageRank de notre version VS PageRank à Wikipedia

Le couleur et radius représente le score. **Le couleur plus rouge et le radius plus grand représentent un score plus élevé.** Pour le même graphe orienté, notre matrice de score est exactement **la même que Wikipedia** ! On affiche le score sous la forme pourcentage pour chaque sommet.

4.2 Question 4.2

Implémenter PageRank pour un grand graphe à attachement préférentiel, est-ce que cela semble un bon critère pour la popularité d'un site ?

Réponse : Selon notre expérience cela ressemble un bon critère pour la popularité d'un site. Notre **visualisation design** est : La taille et le couleur représentent le pourcentage de PageRank comme nous avons discuté dans Question 4.1. Ici, le score est affiché sur le sommet. On lance 200 sommets dans une simulation ou les sommets sont tellement proches qu'ils se cachent. En ce moment là, le **couleur rouge** va montrer un sommet avec PageRank haut. On considère que le modèle ici est bi-direction graphe. Tous les PageRanks s'adaptent bien avec l'importance de chaque sommet. Le sommet original va être marqué plus haut parce que toute le monde vote pour lui. On a testé les δ différents : (Figure 13)

4.3 Question 4.3

On se demande si on peut "tricher" pour augmenter son PageRank. Pour répondre à cette question, estimer ce qu'il se passe pour X si l'on crée artificiellement quelques sommets qui ne pointent que vers un sommet fixé.

Réponse : On a découvrir trois façons principales pour tricher.
1. Zombie link
On crée un graphe orienté simple A->B et A->C. Après on attache 4 sommets au B, le PageRank de B augmente (Figure 14).

Il y a une hypothèse – si on arrive à un trous noir, on va aller à un sommet aléatoirement. Mais dans ce cas, les 4 nouveaux sommets ne sont pas légale dans la liste de visite. Mon idée est crée une **liste de confiance**. C'est un ensemble de sommets que un trous noir peut visiter. Si on ne mets que A,B,C dans cette liste, on va observer que le changement de PageRank dépends que de ϵ. Si epsilon égale 0, la liste de confiance va totalement garder le même PageRank. Le résultat est dans le figure suivant :(Figure 15)

Si ϵ est petit, **l'influence de zombie link sur PageRank va être limité par liste de confiance.**

2. Effets des flatteurs
Étant donné un sommet qui n'a pas d'attachement, comment augmente son PageRank ? Comme les flatteurs dans la vie, si un sommet qui n'a pas d'attachement connecte chaque sommet avec un score plus haut, il va augmenter **un petit peu** son PageRank. Le test est comme suivant :(Figure 16)

Le sommet A augment son PageRank du 8.4 jusqu'a 8.7 par voter les sommets plus forts. Mais pour un sommet qui est déjà pointé, connecter le sommet plus fort n'est pas un bon choix. Cette confection va renforcer la position monopole du sommet le plus fort. (Figure 17)

3. Pouvoir de négociation
Comme nous avons vu le mot "monopole", on pense à **la théorie du jeux**. Pour les sommets forts (qui ont de grands scores), Un pôle qui vote pour des autres pôles va renforcer le pouvoir monopole des autres et perdre son score PageRank. Mais, si deux sommets forts se pointent, ils vont gagner ensemble. **Si ils négocient ensemble, ils vont devenir un groupe monopole et augmenter leurs PageRanks immédiatement.** Le processus est comme suivant : (Figure 18)

Par la collusoire, sommet B et C dépasse A et ils ont augmenté 200% leurs PageRanks.

Pour les sommets ni forts, ni très fables, la négociation interne est aussi une bonne stratégie. En tissant des liens entre les sommets moyens, ils ont augmenté leur PageRank du 2.6 au 7.1. Le processus est comme suivant : (Figure 19)

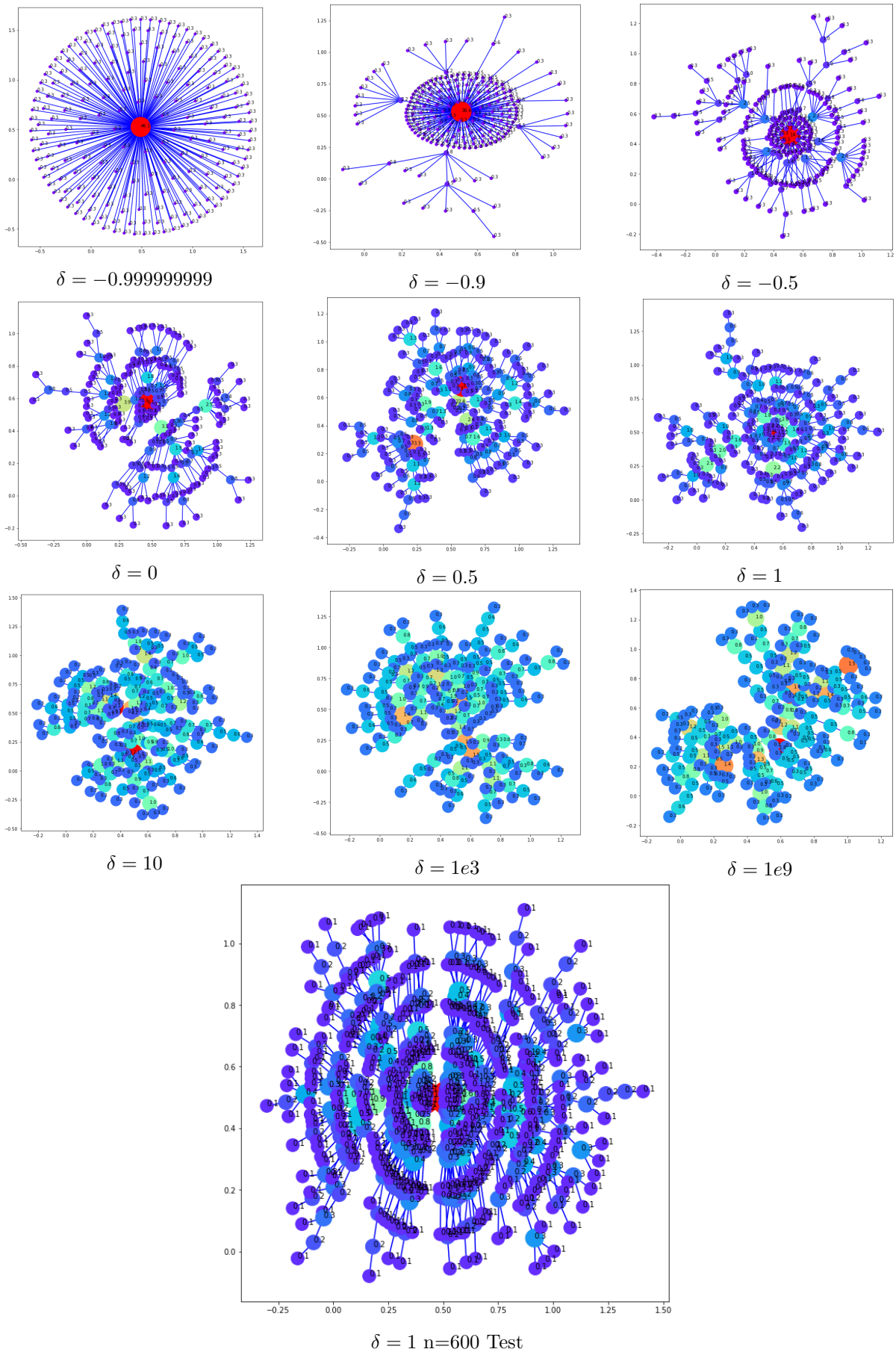
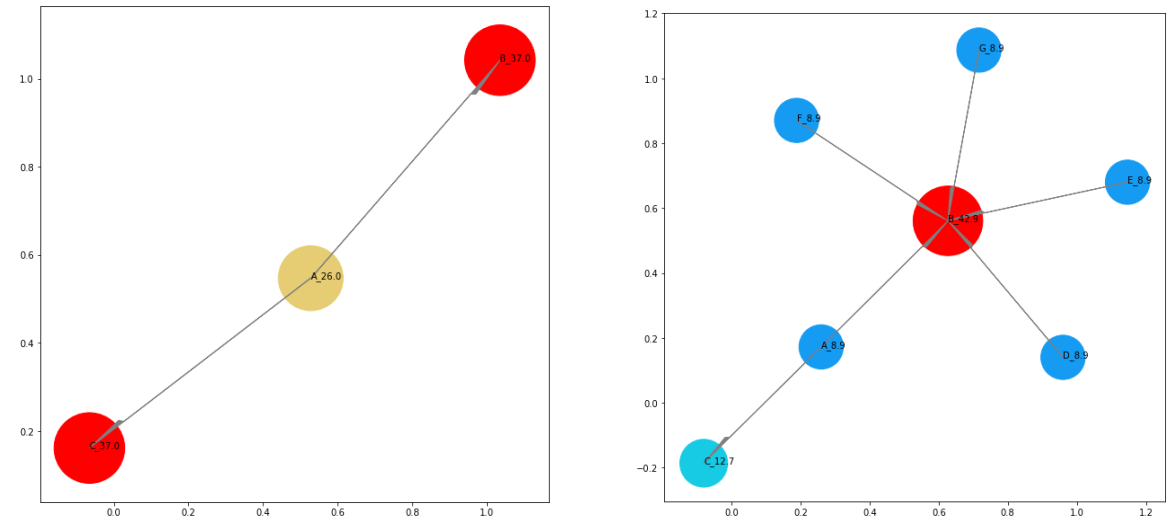
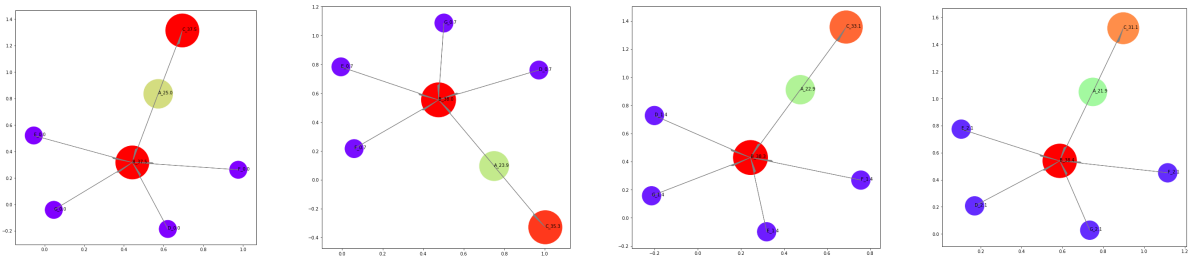


Figure 13 – La distribution de PageRank sur un grand graphe à attachement préférentiel



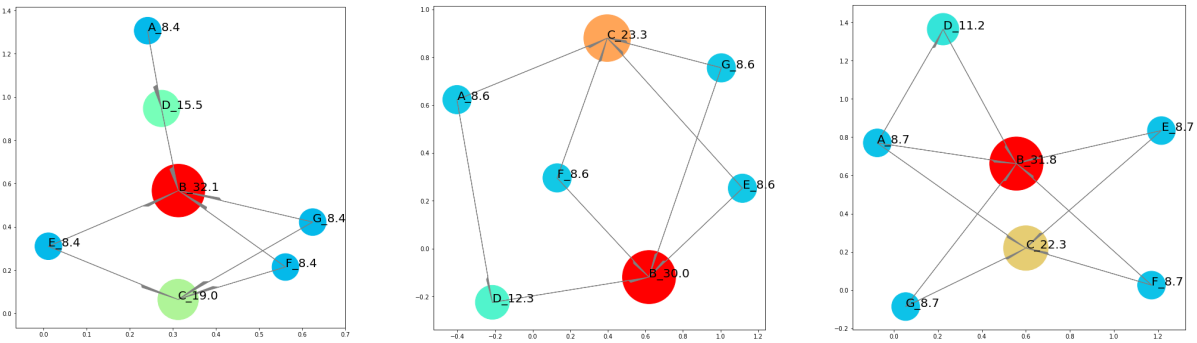
Graphe original — 4 sommet attaché à B

Figure 14 – Zombie link augmente le PageRank de sommet cible



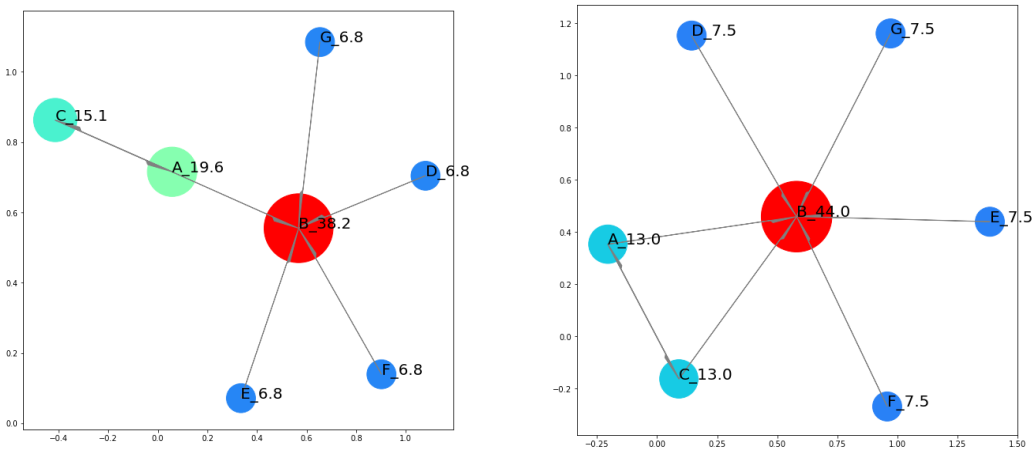
$\epsilon = 0$ — $\epsilon = 0.05$ — $\epsilon = 0.1$ — $\epsilon = 0.15$

Figure 15 – L'influence de ϵ sur la liste de confiance contre zombie link



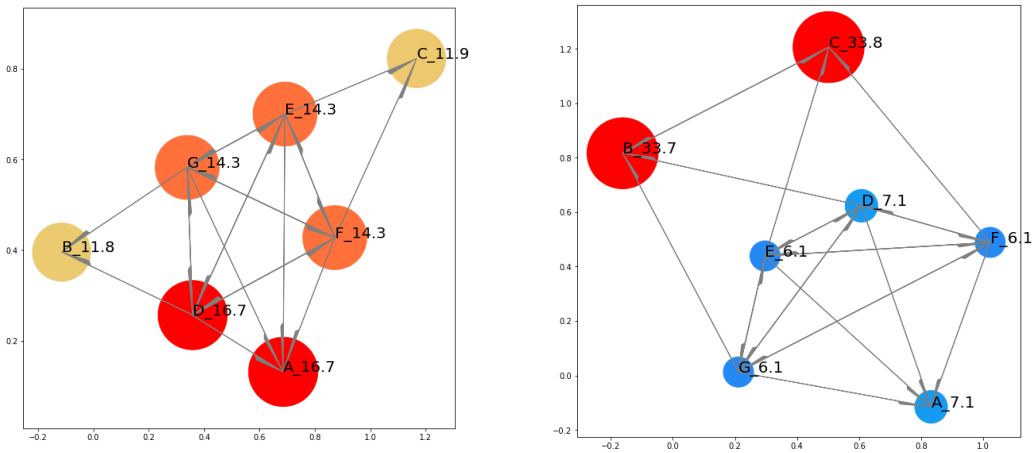
État original — Connecter un sommet plus fort — Connecter deux sommets plus forts

Figure 16 – Effet des flatteurs – une augmentation limité



État original — Connecter le sommet le plus fort

Figure 17 – Échec de voter en face d'im monopôle



État original — Collusoire

Figure 18 – La formation de la collusoire pour augmenter PageRank entre deux sommets forts

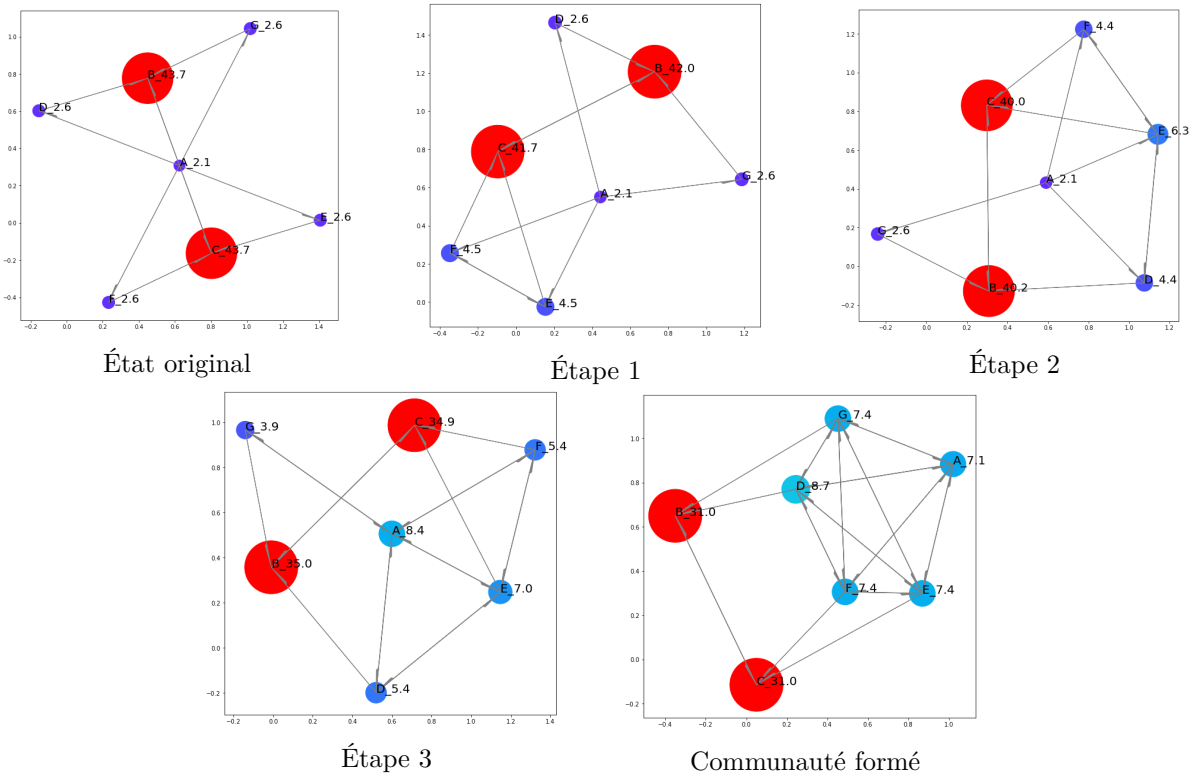


Figure 19 – La formation du communauté pour augmenter le PageRank