Projet CPA

Code Source : https://github.com/nassimerrahoui/largeGraph

TME 3 - Handling a large graph

I. Propriétés des structures de données utilisées pour les graphes

- Liste d'arêtes : Simple représentation, pas adapté pour les grands graphes
- Matrice adjacence : Adapté au graph très dense (Beaucoup d'arrêtes)
- Liste adjacence : Adapté au graphe peu dense (Peu d'arrêtes)

II. Comparaison des structures de données

Pour un **graphe creux** (un graphe avec peu d'arêtes), une liste d'adjacence est beaucoup plus efficace en espace qu'une matrice d'adjacence stockée dans un tableau) : la place requise par une liste d'adjacence est proportionnelle au nombre d'arêtes et de sommets du graphe, tandis que, pour une matrice d'adjacence stockées en tableau, l'espace est proportionnel au carré du nombre de sommets.

Pour des **graphes non connexes**, la liste d'arêtes n'est pas optimale car on va omettre une partie du graphe. Aussi, si notre graphe est dense, alors le temps d'accès à une arête sera au pire des cas de la taille de liste. Cependant pour calculer le nombre de triangle, c'est optimal si on représente chaque composante connexe par une liste d'arêtes.

Prenons deux graphes dont la densité d'arêtes diffère fortement :

Le graphe LiveJournal social network comporte :

- 3997962 sommets
- 34681189 arêtes

Alors que, le graphe Orkut social network comporte :

- 3072441 sommets
- 117185083 arêtes

Le graphe Orkut social network est beaucoup plus dense que le graphe LiveJournal social network. Alors la matrice d'adjacence serait beaucoup plus adaptée pour Orkut par exemple.

Question 2

Graph	Nombre d'arêtes	Nombre de sommets
Email	25571	1005
Amazon	925872	548552
LivreJournal	34681189	4036538

Question 3

Notre méthode de nettoyage de donnée suit les étapes suivantes :

- Pour chaque Edge, trier les points pour que le plus petit sommet se trouve en début de ligne dans le fichier
- Ecrire en parallèle chaque résultat dans un fichier sorted_data
- Ecrire dans un fichier cleaned_data chaque Edge en évitant de créer des doublons

Exemple pour le graph Email :

Graph	Sorted Data	Cleaned Data
Email	Nombre d'arêtes : 25571	Nombre d'arêtes : 24929

Question 4

La première partie du code sert à savoir quel est le sommet le plus grand noté i. Une fois calculé, nous allons initialiser une matrice de dimension (i x 2), qui va nous servir à stocker le nombre de voisins de chaque sommet.

Exemple : [[0,2],[1,5],[2,8],[3,2],[4,1],[5,2]] Le degré du sommet 0 est 2, de 1 est 5, de 2 est 8, etc.

Une fois remplie nous allons stocker son contenu dans un fichier degree_data.

Question 5

Graph	Quantité	Temps exécution (secondes)
Email	366958786	0.064 s
Amazon	103415531	2.850 s
LivreJournal	789000450609	93.730 s

Question 6

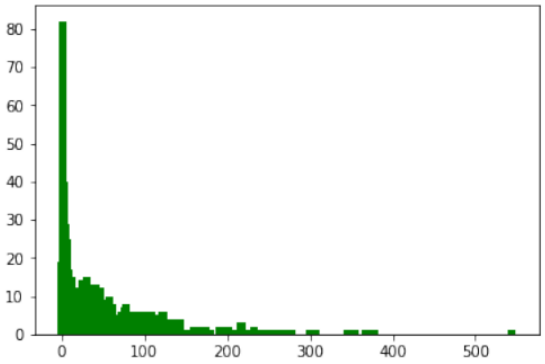


Figure 1 - Email

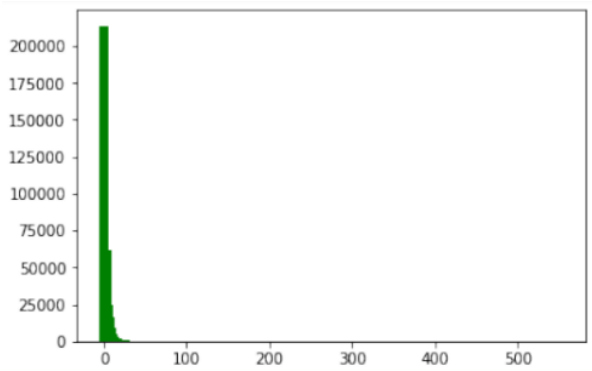


Figure 2 - Amazon

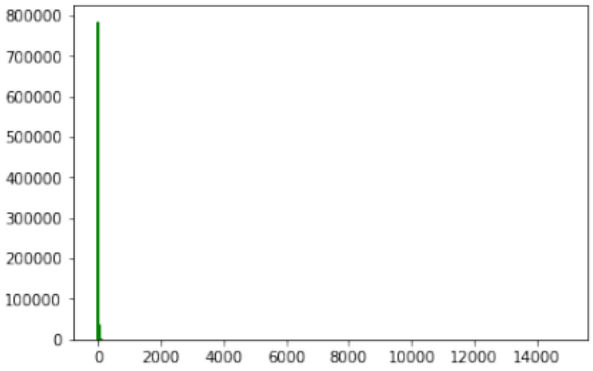


Figure 3 - LivreJournal

Abscisse : Degrés. Ordonnée : Nombre d'occurrences de ces degrés.

On remarque qu'il y a beaucoup plus de degrés entre 0 et 30 qu'entre 30 et plus, on peut affirmer que les sommets ne sont pas connectés en majorité à plus de 30 autres sommets.

Graph	Borne inférieure diamètre du graphe
Email	985
Amazon	334862
LivreJournal	3997961

Question 9

On itère sur chaque sommet noté v, et on boucle sur ses voisins avec u et on boucle sur l'intersection des voisins de u et de v avec k. On obtient le triplet (u, v, k) c'est notre triangle et on incrémente le nombre de triangles.

Notre algorithme compte chaque triangle 3 fois, cependant nous aurions pu résoudre ce problème en testant si on a déjà calculé le triangle ou en utilisant la méthode présentée en cours du degré minimum, au préalable il aurait fallu trier la liste des voisins par degré minimum.

On réussit donc à compter le nombre de triangle en divisant par 3 le nombre calculé avec l'algorithme, idem pour le nombre de triangle par sommet.

Graph	Temps d’exécution (secondes)	Nombre Triangles
Email	0.637 s	105461
Amazon	9.685 s	667129
LivreJournal	1693.723 s	177820130

Community Detection

Exercice 1 – Simple Benchmark :

Les graphes de 4 clusters avec différents paramètres.
Chaque paire de nœuds dans le même cluster est connectée avec une probabilité p.
Chaque paire de nœuds dans de clusters differents est connectée avec une probabilité q<=p.

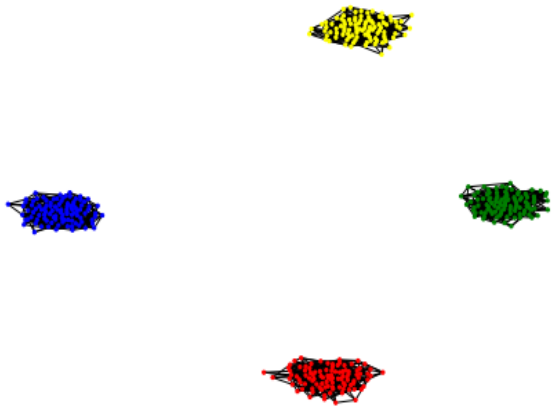


Fig 1 : Représentation du graphe avec les paramètres p = 0.1 et q = 0

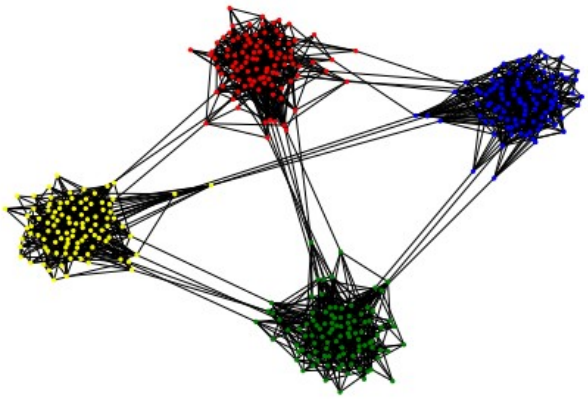


Fig 2 : Représentation du graphe avec les paramètres p = 0.1 et q = 0.005

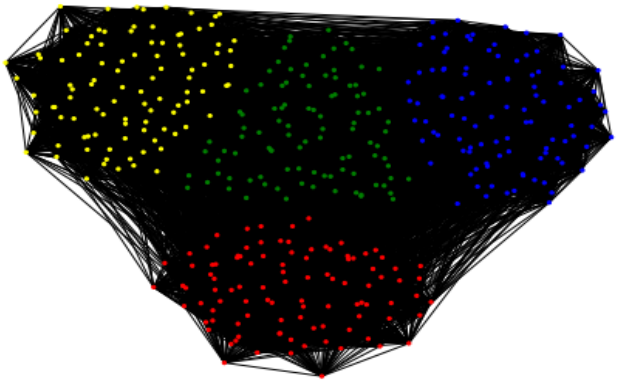


Fig 3 : Représentation du graphe avec les paramètres p = 0.5 et q = 0.05

- Pour p = 0.1 et q tend vers 0 : le rapport tend vers l'infini, il n'y a pas de liens entres nos différents clusters.

- Pour p = 0.1 et q = 0.0005 : le rapport est de : 200.0, il y a un nombre moyen de liens entre les clusters.

- Pour p = 0.5 et q = 0.05 : le rapport est de : 10.0, les clusters sont liées par un grand nombre de liens.

Plus p/q augmente, plus les clusters sont moins connectés, dans le pire des cas on a un rapport qui tend vers l'infini, ce qui nous donne 4 clusters sans liens communs. Dans le cas moyen, le rapport nous donne un certain nombre qui n'est pas dans le voisinage de 0 , ce qui nous donne un certain nombre moyen de liens entre les clusters. Et dans le cas ou p/q tend vers 0, les clusters sont hyper-connectés avec des liens.

Exercice 2 – Label Propagation :

Nous avons exécuté l'algorithme de propagation de label sur un graphe généré à l'exercice 1 et nous avons obtenu ce resultat, sachant que chaque couleur représente une communauté.

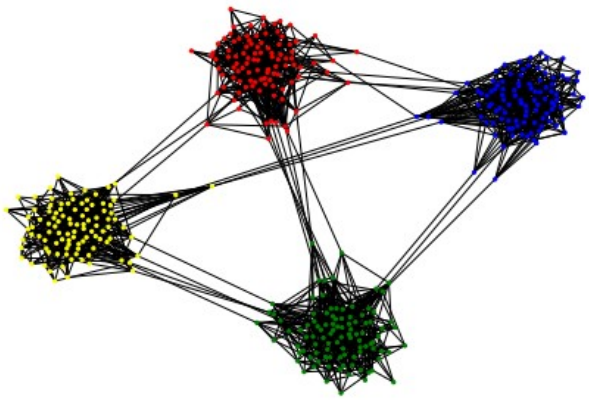
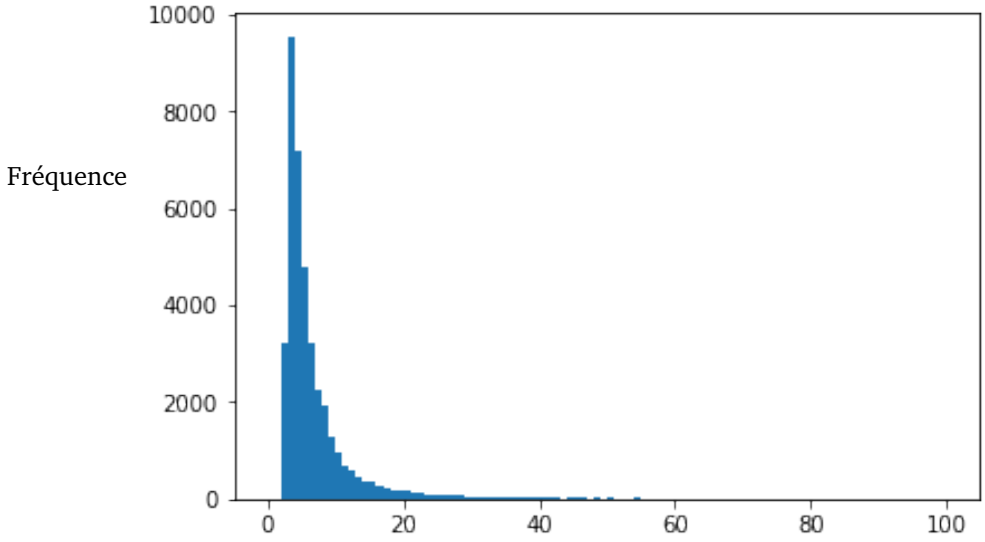


Fig 4 : Représentation du graphe obtenu après exécution du label propagation algorithm.

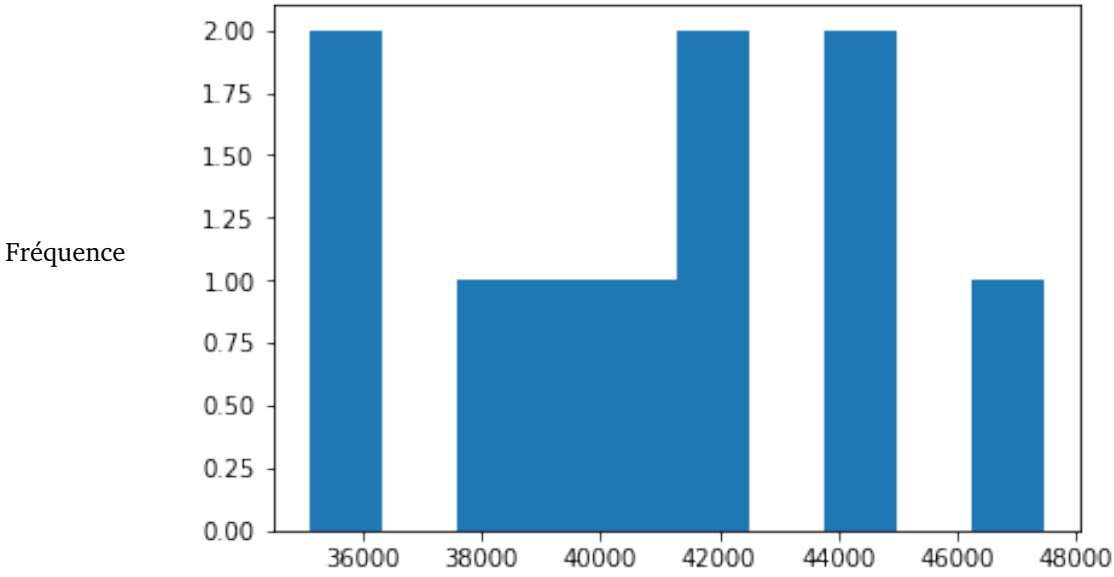
Puis nous avons executé ce dernier algo sur le graphe YouTube : (*http://snap.stanford.edu/data/com-Youtube.html*) *et* nous avons obtenu répartition de tailles de communautés, représentée par l'histogramme suivant :



Taille des communautés

On a 38542 communautés dans notre cas répartis sur des tailles variants de 0 à 55, le graphe de YouTube étant un graphe de terrain et comme expliqué en TME, le nombre de communautés qu'il a réellement est différent que celui calculé ici, et sachant que le calcul du nombre de communautés dépend de la qualité du shuffle qu'on fait à chaque fois qu'on exécute.

Il nous ai demandé d'exécuter l'algorithme 1000 fois sur le graphe de YouTube, mais on ne l'a fait que 10 fois mais on peut s'attendre à des résultats similaires si on l'exécutait 1000 fois.
On a obtenu un résultat qu'on a représenté sous forme d'un histogramme :



Nombre de communautés

On obtient donc 10 nombres de communautés avec une intervalle 38000 - 42000 qui est la plus large donc c'est celle qui comporte le nombre de communautés en moyenne.

Exercice 3 – Validation :

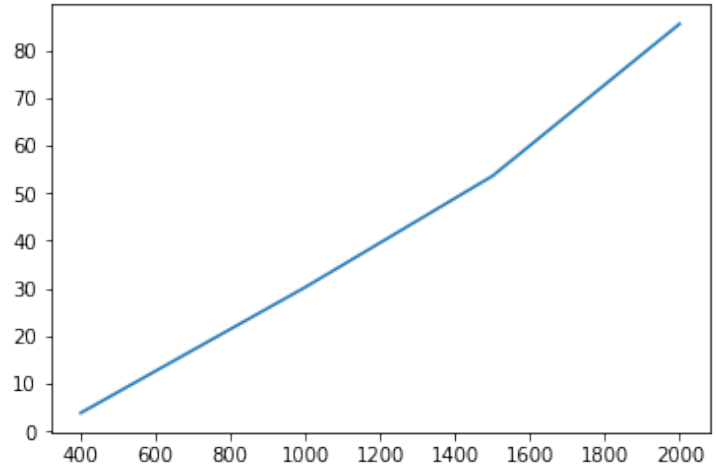
Dans cette section, nous allons comparer la rapidité d'exécution et la précision des algorithmes de Louvain et de Label Propagation.

1) Évaluation de la rapidité d'exécution des 2 algorithmes :

Nous allons tester les implémentations de l'algorithme de Louvain et de propagation de label sur différents jeux de données générées grâce à notre fonction de benchmark utilisée à l'exercice 1 avec des tailles de [400,1000,1500,2000].

Nous obtenons cette courbe-ci pour l'algorithme de Louvain, représentant le temps d'exécution en fonction de la taille du graphe :

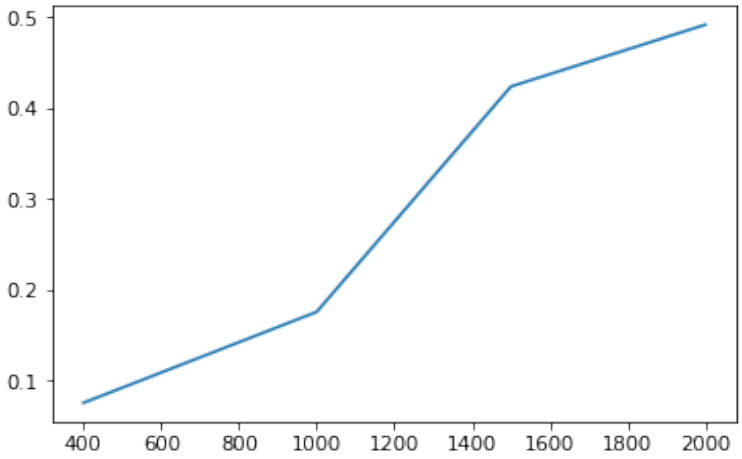
Temps d'exécution en secondes



Taille des graphes

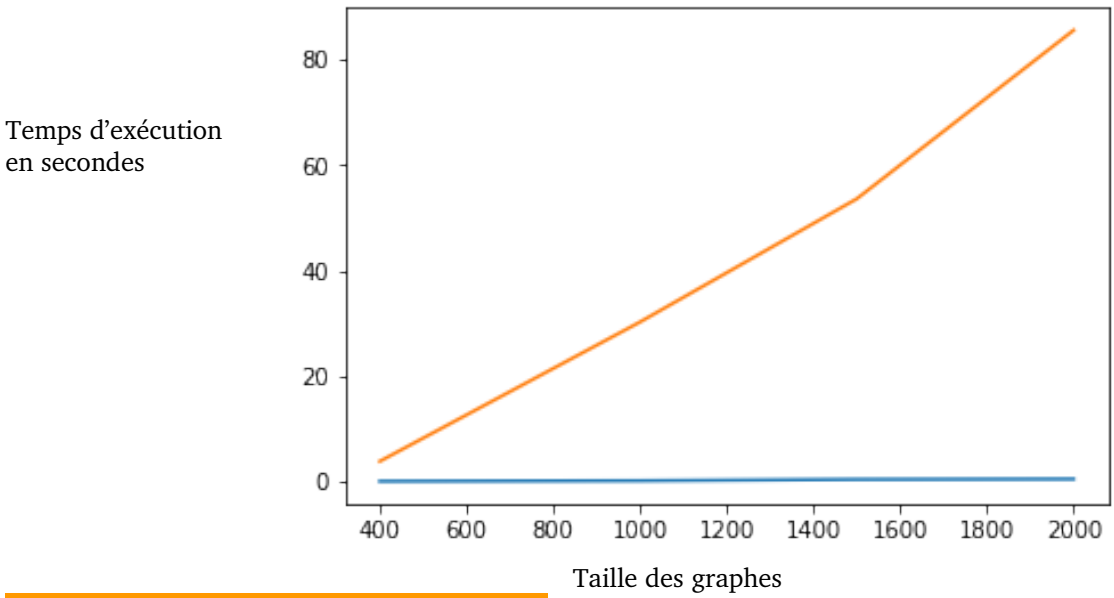
Nous obtenons cette courbe-ci pour l'algorithme de propagation de label, représentant le temps d'exécution en fonction de la taille du graphe :

Temps d'exécution en secondes



Taille des graphes

On remarque la courbe de l'algorithme de louvain croit exponentiellement alors que celle de l'algorithme de propagation de label croit en logarithmique et si on superpose les deux courbes :



Courbe d'exécution de l'algorithme de Louvain

Courbe d'exécution de l'algorithme de Propagation de Label

On voit très bien que l'algorithme de propagation de Label est beaucoup plus rapide que celui de Louvain.

2) Évaluation de la précision des 2 algorithmes :

Nous allons maintenant tester nos deux algorithmes sur des graphes à partir de notre benchmark du 1er exercice afin d'évaluer leur précision et les comparer en utilisant la métrique NMI abordée en cours.

En exécutant nos algorithmes sur le LFR benchmark en ayant des nœuds avec des chevauchement nous avons obtenu deux fichiers représentant les communautés généré de chaque algorithme.

Metrique	LFR BenchMark	Youtube
NMI (file louvain – file propag)	0.426 (Louvain) – 0.474 (P.Label)	0.018(Louvain) – 0.021 (P.Label)

Nous remarquons que l'application de la métrique NMI sur le graphe créé avec le LFR benchmark est beaucoup plus important que celui appliqué au graphe Youtube, ceci est expliqué par le fait que le graphe est un graphe de terrain et les communautés de terrains ne sont pas les mêmes que celles calculés avec les algorithmes.

Question précision, l'algorithme de propagation de Label a un score plus élevé que celui de Louvain.

PageRank

Exercice 1 – PageRank d'un graphe dirigé:

Pour alpha = 0,15, l'algorithme ne converge pas, mais le mieux qu'on puisse faire c'est de faire 13 itérations pour commencer à avoir un vecteur de Pagerank stable c'est à dire qu'on pourrait extraire les 5 plus grands PageRanks et les plus petit.
Et c'est ce qu'on a fait :

Les 5 pages ayant le plus grand PageRank :

1) United States
2) United Kingdom
3) Germany
4) 2007
5) 2006

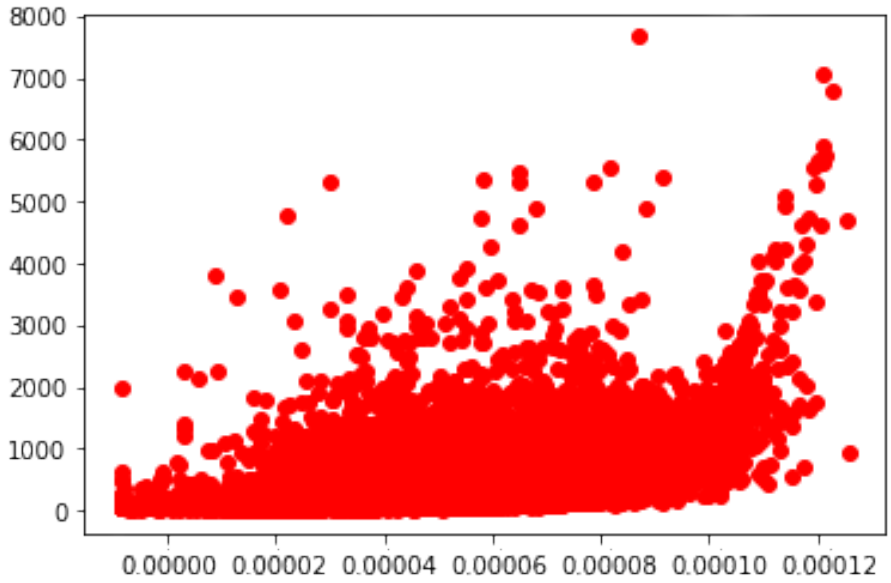
Les 5 pages ayant le plus petit PageRank :

1) Salt of aspartame-acesulfame
2) Jon Cole
3) Hotel Babylon episode list
4) Teatr Biuro Podróży
5) Reliance damages

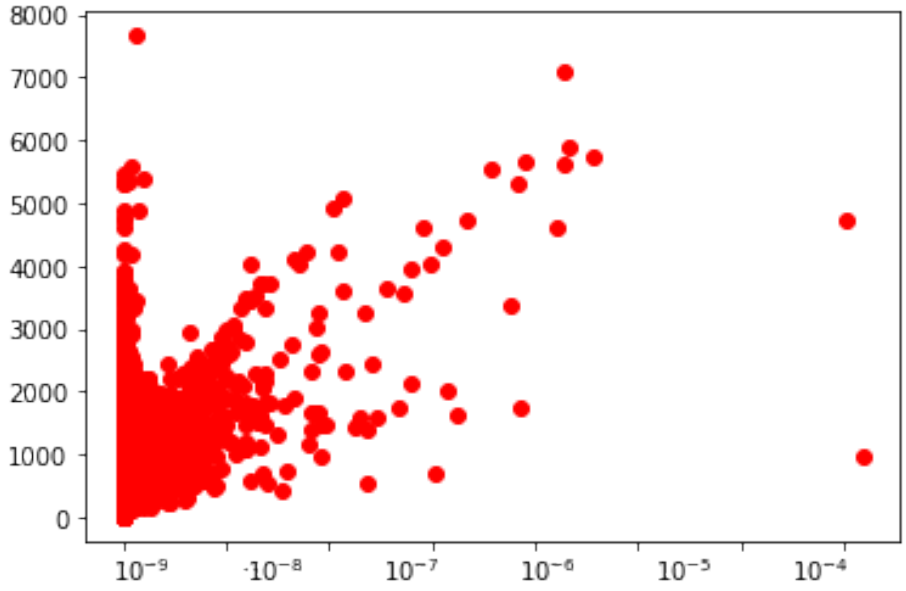
Exercice 2 – Corrélations :

Pour le 1er Scatter Plot :

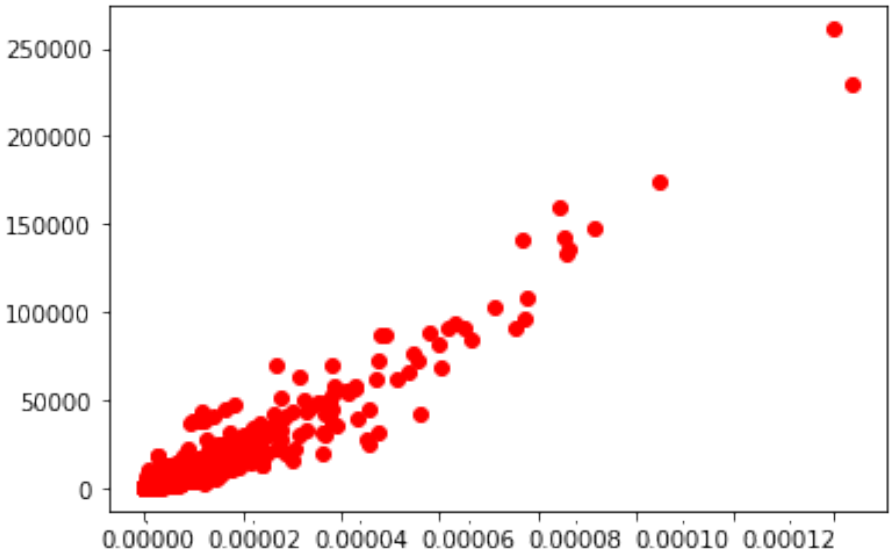
Échelle linéaire pour x = PageRank avec alpha = 0,15 et y = in-degrees



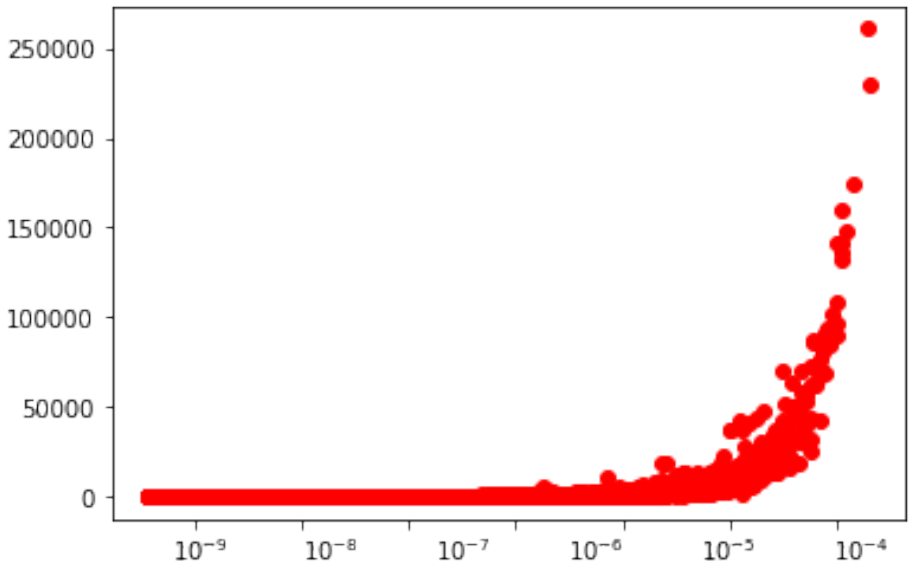
Échelle logarithmique pour x = PageRank avec alpha 0,15 et y = in-degrees



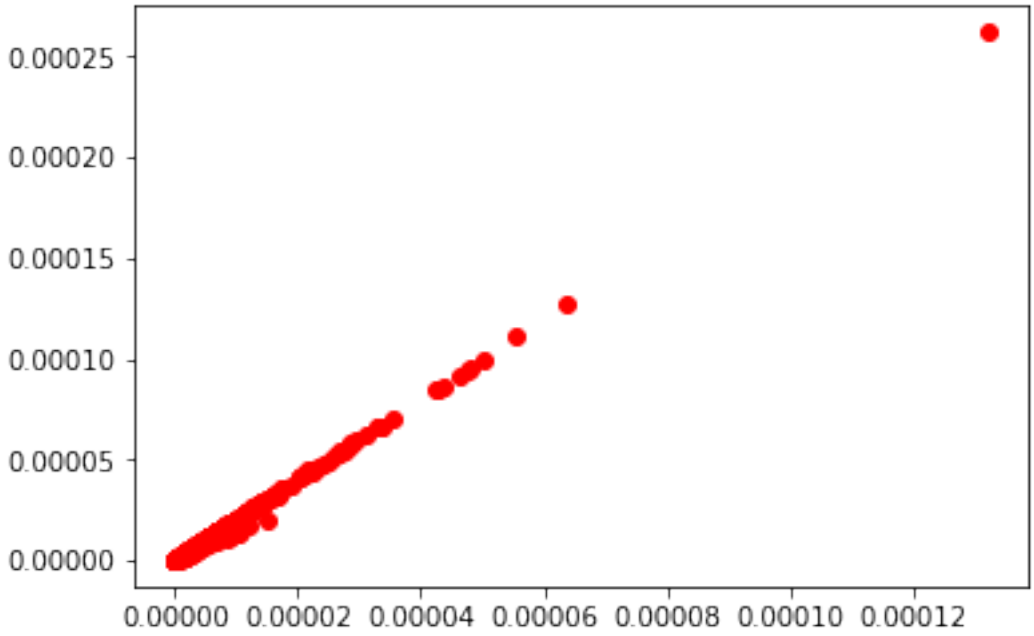
Échelle linéaire pour x = PageRank avec alpha = 0,15 et y = out-degrees



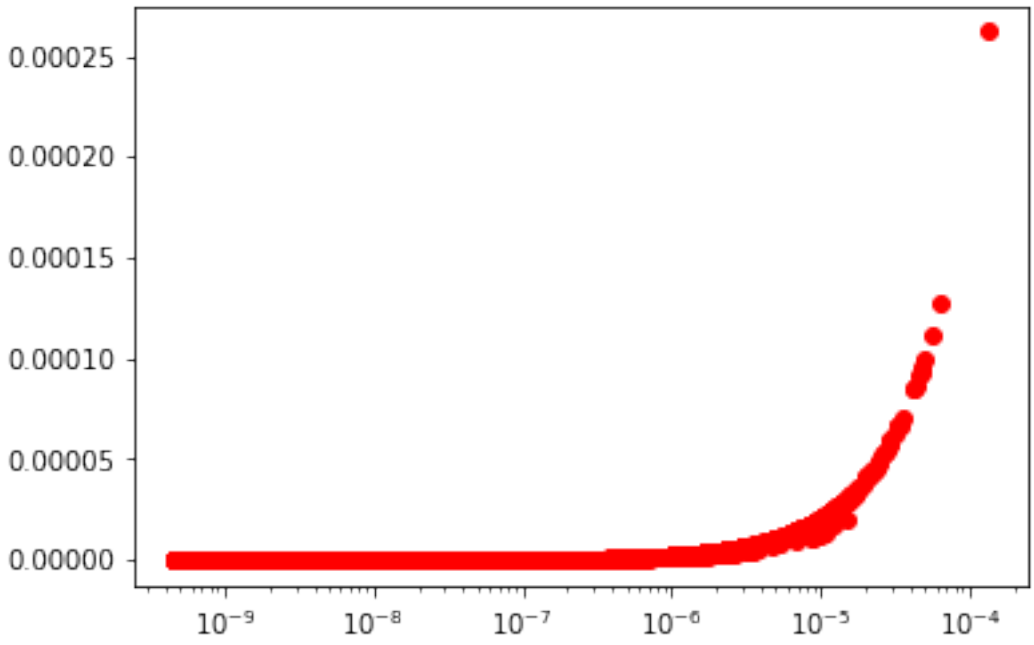
Échelle logarithmique pour x = PageRank avec alpha = 0,15 et y = out-degrees



Échelle linéaire pour x = PageRank avec alpha = 0,15 et y = PageRank avec alpha = 0,1



Échelle logarithmique pour x = PageRank avec alpha = 0,15 et y = PageRank avec alpha = 0,1



L’échelle linéaire est la plus adaptée parce qu’elle montre vraiment que l’augmentation de x est proportionnelle à l’augmentation de y à l’opposée de l’échelle logarithmique qui nous montre pas cela pour les valeurs de x les plus basses.

Donc x et y sont fortement corrélées puise que l’augmentation de x, donne l’augmentation de y.

TME 6 - Densest Subgraph

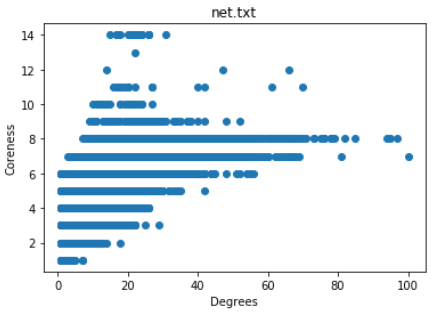
I. K-core decomposition

Graph	K-core	Average degree density	Edge density	Size of densest core ordering prefix
Email	*34*	*27*	*0,13*	*228*
Amazon	6	3,9	0,05	18
LivreJournal	360	15,4	-	-
Orkut				

II. Graph mining with k-core

Les auteurs ayant une Coreness égale à 14 semblent être en tous coréens cela peut s'expliquer par la technique « je suis co-auteur et tu deviens mon co-auteur » :

- Sa-kwang Song
- Sung-Pil Choi
- Chang-Hoo Jeong
- Yun-soo Choi
- Hong-Woo Chun
- Jinhyung Kim
- Hanmin Jung
- Do-Heon Jeong
- Myunggwon Hwan
- Won-Kyung Sung
- Hwamook Yoon
- Minho Lee
- Won-Goo Lee
- Jung Ho Um
- Dongmin Seo
- Mi-Nyeong Hwang
- Sung J. Jung
- Minhee Cho
- Sungho Shin



III. Densest subgraph

Au-dessus de 100 itérations les valeurs se stabilisent et nous renvoi aux valeurs trouvées par la K-core décomposition.

Après 100 itérations :

Graph	K-core	Highest Density Score	Average Degree Density	Edge density	Size of densest core ordering prefix
Email	*34*	*28*	*27,59*	*0,129*	*228*
Amazon	6	3,91	3,8	0,05	18
LivreJournal	360	15,54	15,39	-	-
Orkut					