Rapport Projet CPA

BAH Thierno
3408625

lien vers le code sources : https://github.com/Asparodia/CPA_Projet.git

Rapport TME 3 CPA

Handling a large graph

Exercice 2:
J'ai fait une fonction simple qui à chaque arêtes et chaque noeuds du fichier incrémente des compteurs. Je stock les noeuds déjà croisé pour éviter de les raconter plus tard.

Exercice 3:

Pour nettoyer les fichiers j'ai utiliser les commandes bash suivante:
#tail -n +5 com-amazon.ungraph.txt >> ~/S2/CPA/tme3/amazon.ungraph.txt
#sort -g amazon.ungraph.txt|uniq > amazon_ungraphClean.txt
#sort -g amazon_ungraphClean.txt > amazon_ungraphClean2.txt

Pour trier le fichier selon la première colonne et enlever tout les doublons de type A B, B A ou A A.
Il y a aussi la fonction *clean(fileName)* qui fait le même résultat mais garde en mémoire quelques éléments du graphe.

A partir de là j'ai commencé à travailler sur les fichiers clean mis sur le vrac car nous ne pouvions pas télécharger des fichiers aussi volumineux avec le quotas mis à disposition par la ppti.

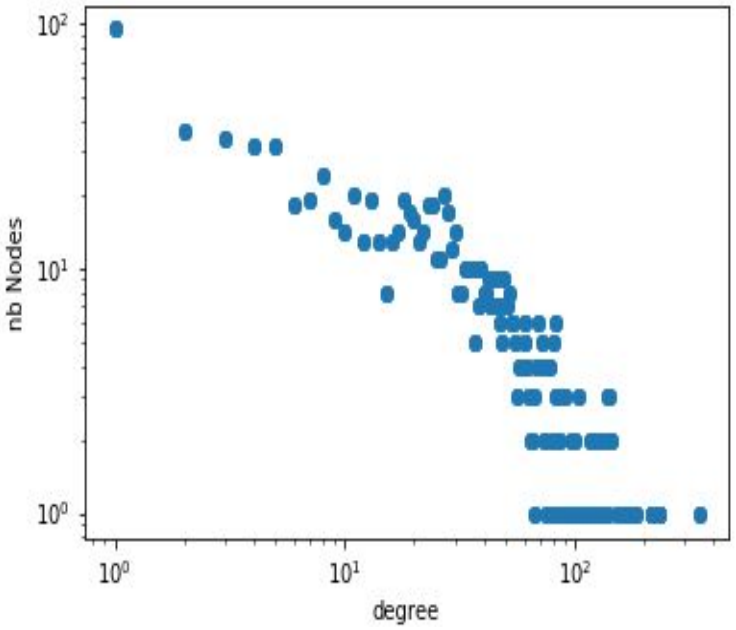
Exercice 4:
degree(fileName) parcours le fichier et stock dans un dictionnaire de clé l'identifiant du noeud et son degré en tant que valeur.

Exercice 5:
En utilisant la fonction *degree(fileName)* j'ai pus avoir accès au degré de chaque noeud dans un second parcours du fichier pour calculer la somme "Quantity" .

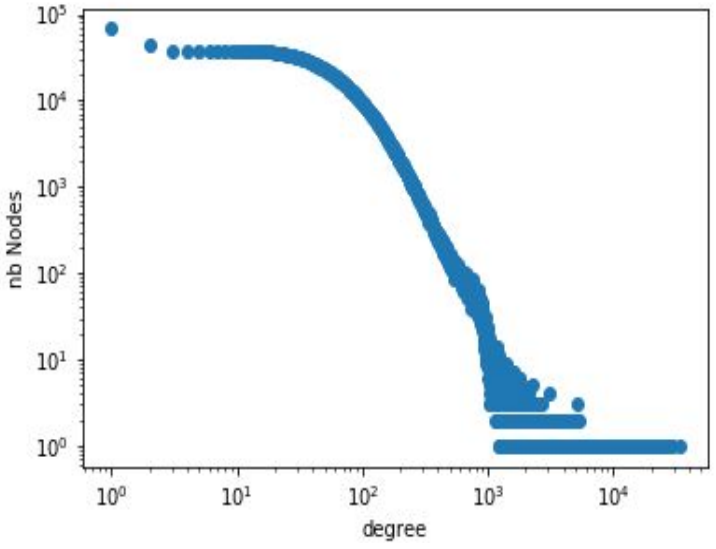
Mesures de temps pour "Quantity" sur les graphes :
email-Eu-core-clean.txt : 88109182 pour 0.024881839752197266 secondes
com-amazon.ungraph-clean.txt : 103415531 pour 1.5718021392822266 secondes
com-lj.ungraph-clean.txt : 789000450609 pour 58.566508054733276 secondes
com-orkut.ungraph-clean.txt : 22292678512329 pour 200.53568148612976 secondes
com-friendster.ungraph.txt : 379856554324947 pour 4390.462798095187 secondes

Exercice 6:

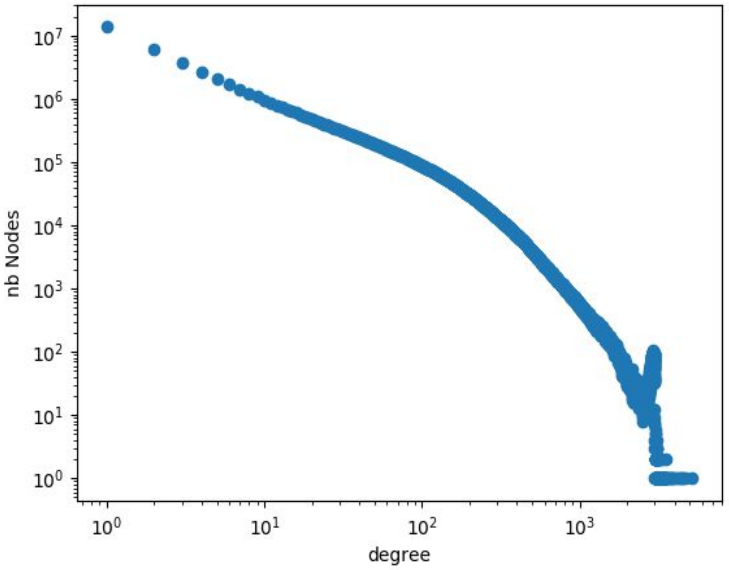
Pour chaque graphe nous obtenons les distributions suivante



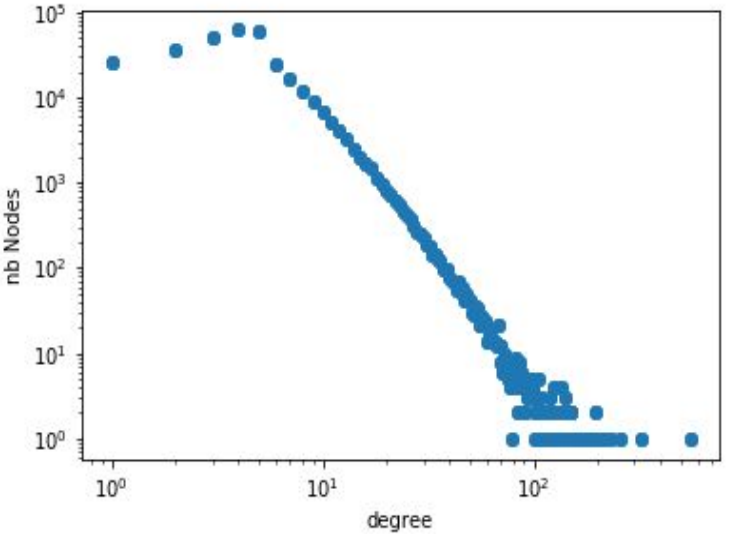
Distribution email-EU-core



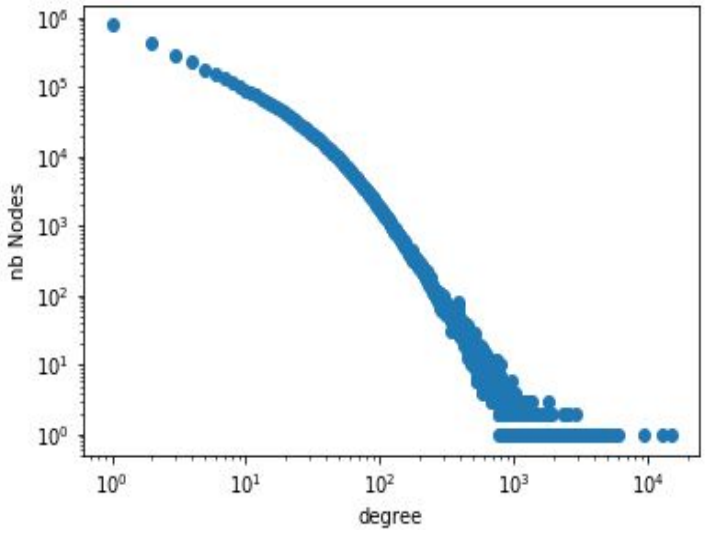
Distribution orkut



Distribution friendster



Distribution amazon



Distribution lj

les échelles des abscisses et des ordonnées sont logarithmiques pour avoir une bonne visibilité de l'ensemble qui est trop "étendue" .

Exercice 7 :

Charger un graphe en mémoire:

Pour charger ma liste d'arêtes je pars d'un fichier clean et pour chaque ligne (qui représente une arête entre deux noeuds) je la stock dans une liste.

Mesures pour la liste d'arête :

email-Eu-core-clean.txt : 0.01457236289978027 secondes
com-amazon.ungraph-clean.txt : 1.353329181671143 secondes
com-lj.ungraph-clean.txt : 1248.564356843643468 secondes
com-orkut.ungraph-clean.txt : ram pleine pc bloqué
com-friendster.ungraph.txt :

Pour charger la matrice je trouve le noeud d'indice maximum et je fait une matrice de taille (maximum*maximum) pour couvrir tous les indices possibles, ensuite je parcours mon fichier clean et pour chaque arête (i,j) je mets les cases d'indices [i][j] et [j][i] à 1. On considère qu'une arête est bidirectionnel.

Mesures pour la matrice d'adjacence :

email-Eu-core-clean.txt : 0.09201383590698242 secondes
com-amazon.ungraph-clean.txt : MemoryError
com-lj.ungraph-clean.txt :
com-orkut.ungraph-clean.txt :
com-friendster.ungraph.txt :

Pour charger la liste d'adjacence je parcours mon fichier et j'ai un dictionnaire qui a comme clé les noeuds et comme valeur la liste des noeuds avec qui il partage une arête (ses voisins).

Mesures pour la liste d'adjacence :

email-Eu-core-clean.txt : 0.04520750045776367 secondes
com-amazon.ungraph-clean.txt : 1.8801968097686768 secondes
com-lj.ungraph-clean.txt : 70.97990083694458 secondes
com-orkut.ungraph-clean.txt : MemoryError
com-friendster.ungraph.txt :

Exercice 8 :

J'ai implémenter l'algorithme BFS vu en cours, il me retourne tout les noeuds accessibles depuis un point de départ.
Pour trouver la plus grande composante connecté je lance BFS de façon à couvrir tous les noeuds de mon graphe et je retourne la plus grande composante.

Mesures pour la plus grande composante connecter :

email-Eu-core-clean.txt : 986
com-amazon.ungraph-clean.txt : 334863
com-lj.ungraph-clean.txt : 3997962
com-orkut.ungraph-clean.txt :ram pleine pc bloquer
com-friendster.ungraph.txt :

Pour la borne inférieure du diamètre j'ai un BFS2 qui donne la hauteur de chaque points de la composante connecter par rapport au point à partir duquel on lance le BFS2.
Grâce à ce BFS2 je peux trouver un des points les plus éloignés du point à partir du quel j'ai lancer BFS2 et ensuite je peux relancer BFS2 a partir de ce point et ainsi de suite jusqu'à avoir la hauteur maximum qui a été atteinte entre deux points et cela est le diamètre.

Mesures pour la borne inférieure du diamètre :

email-Eu-core-clean.txt : 7
com-amazon.ungraph-clean.txt : 47
com-lj.ungraph-clean.txt : ram pleine pc bloquer
com-orkut.ungraph-clean.txt :
com-friendster.ungraph.txt :

Exercice 9 :

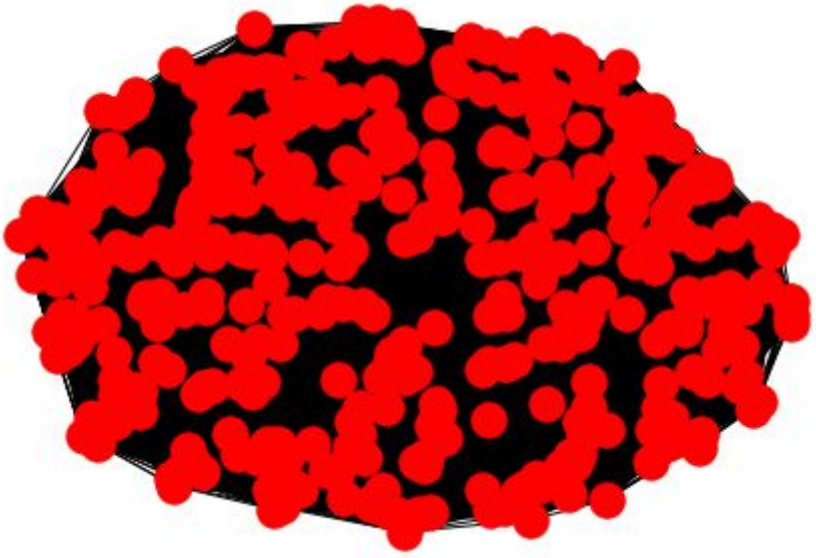
J'ai implémenter l'algorithme du cours, ca marche pour les très petit graphe mais malheureusement je n'ai pas réussi à l'optimiser ou à stocker les données de manière assez intelligente pour me permettre de lancer sur les graphes réel.

Rapport TME 4 CPA

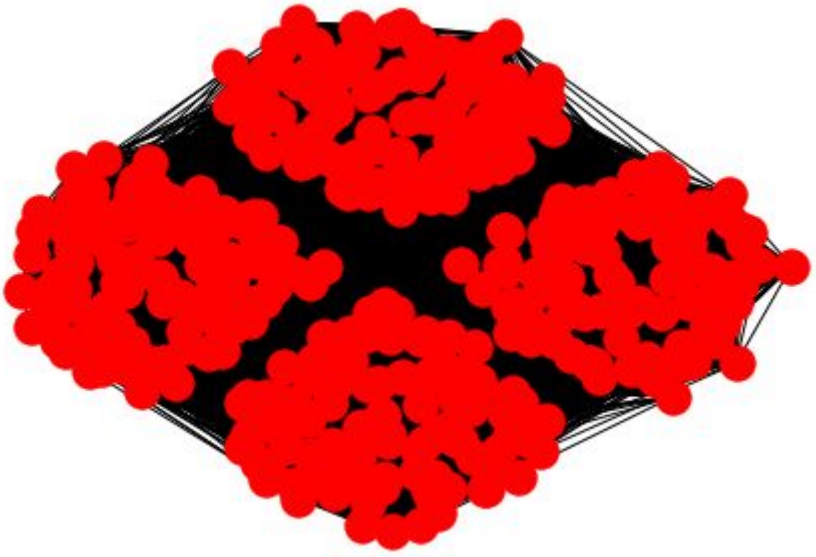
Community detection

Exercice 1 : Simple Benchmarck

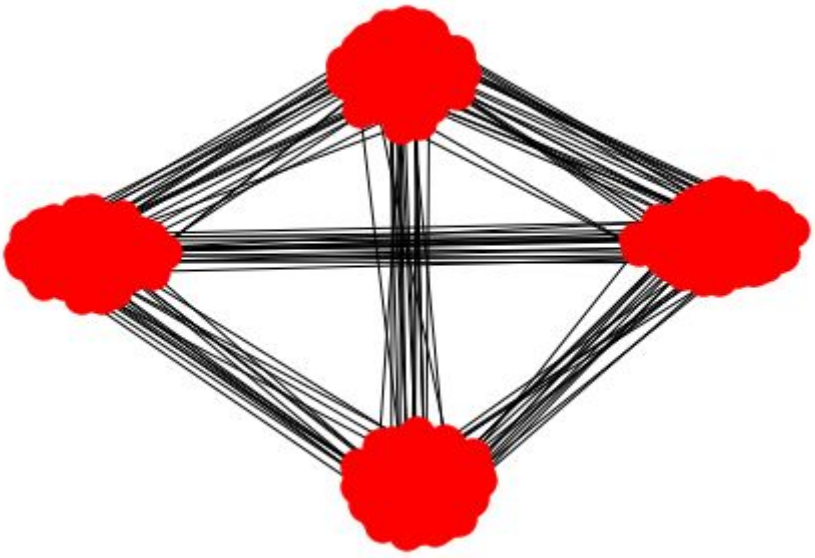
Plot pour p = 0.8 et q = 0.7 :



Plot pour p = 0.8 et q = 0.1 :



Plot pour p = 0.8 et q = 0.002 :

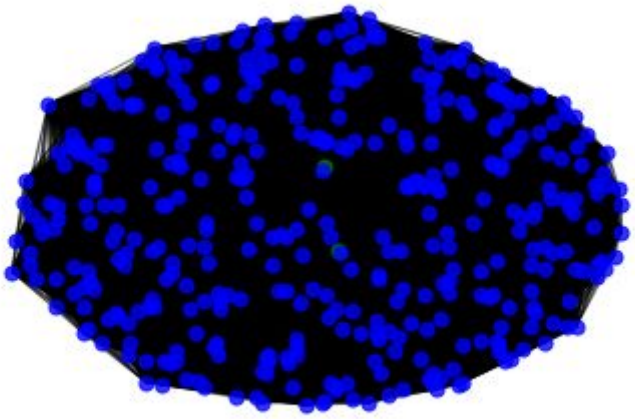


On peut remarquer que plus on autorise des liens entre différents clusters (c'est à dire on augmente la valeur de q) moins on voit la séparation entre les clusters. Le premier plot autorise quasiment autant de connexion entre les différents clusters que de connexion au sein du même cluster et cela donne juste un énorme cluster, plus on diminue q et plus on voit naître les clusters.

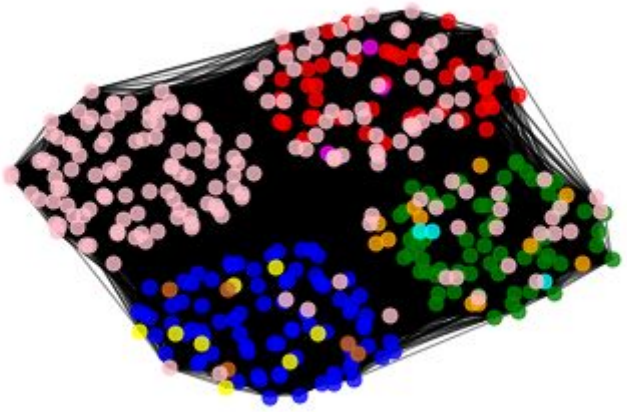
Exercice 2 : Label propagation

J'ai lancer l'algorithme de label propagation sur les graphes obtenus à la question précédente pour voir les communaute que j'obtiens:

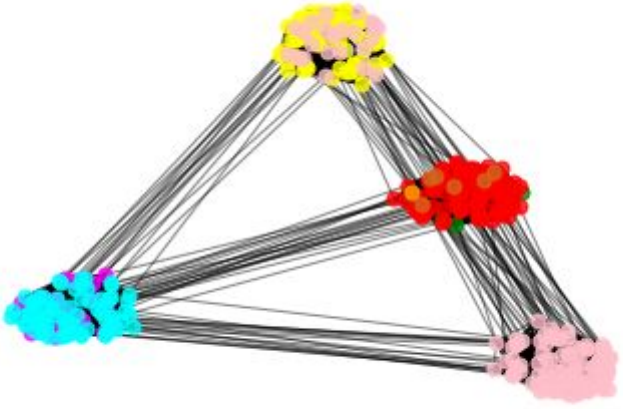
Au bout de 3 itérations l'algorithme s'arrête, j'ai trouver 2 communautés sur le graphe qui ressemble a un énorme cluster.



Au bout de 3 itérations l’algorithme s'arrête, j’ai trouver 13 communautés sur le graphe ou on commence à deviner les clusters.



Au bout de 3 itérations aussi l’algorithme s'arrête, j’ai trouver 10 communautés sur le graphe les clusters sont bien en évidence.



L’algorithme de label propagation converge plutôt rapidement vers un nombre de communauté.

Sur le graphe de Youtube au bout de la 100 eme itérations j’ai trouver 13780 communauté (que je n’ai pas pus afficher).

Rapport TME 5 CPA

PageRank

Exercice 1 : PageRank

Sur le graphe de wikipedia au bout de la 14 eme itération on observe plus aucun changement dans les scores de pagerank ou alors ce sont des changements négligeable, on commence à se stabiliser.

Les pages avec les plus petits scores sont :

Name	ID	Score
Leroy Township Michigan	12588429	8.029414407102666e-08
WEZG	141140	8.029414407102666e-08
WJCS	8612315	8.029414407102666e-08
WLJR	12899417	8.029414407102666e-08
WBFR	12899501	8.029414407102666e-08

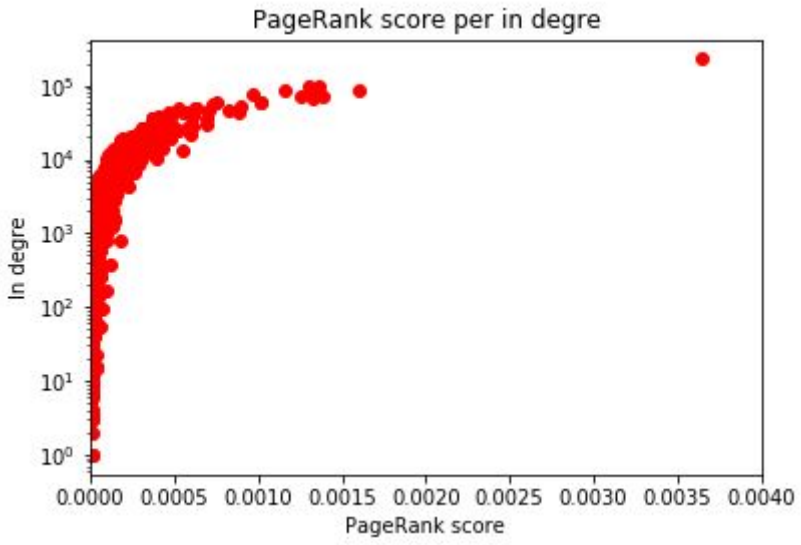
Les pages avec les plus grands scores sont :

Name	ID	Score
Germany	3434750	0.003641641771341633
United Kingdom	31717	0.001595610280130652
2006	11867	0.001387173142026925
United States	36164	0.0013607089814780002
France	5843419	0.0013295272513515564

Exercice 2 : Corrélations

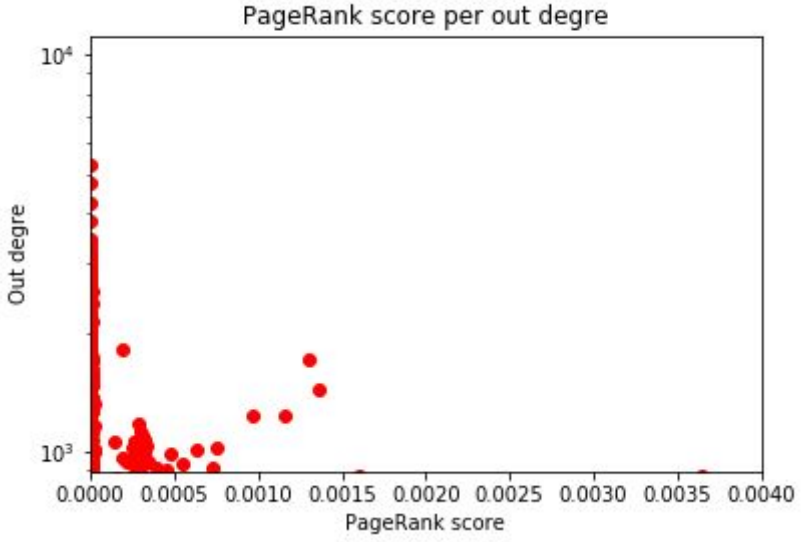
Les plots sont fait sur le graphe de wikipedia.

1 - Plot avec les degrés entrant et a = 0.15 :



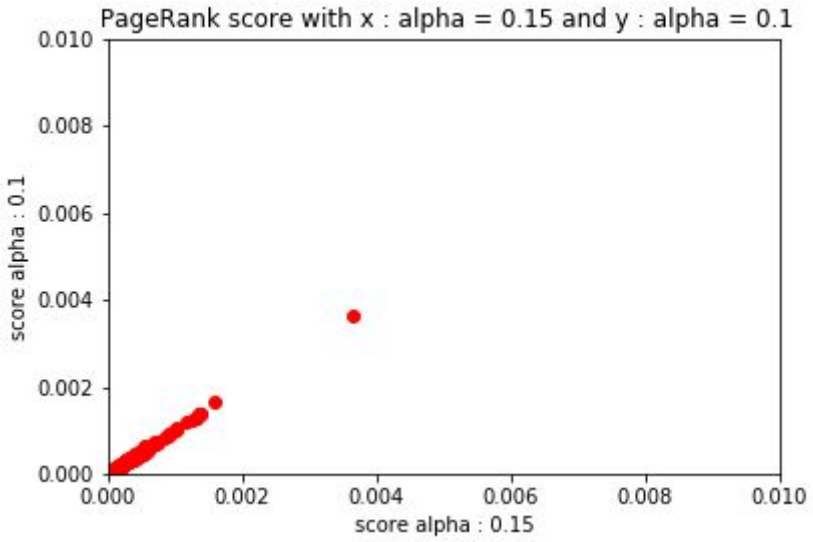
On remarque que plus le degré entrant d'une page est élevé plus son score vas être important, plus on référence une page et plus elle a de l'importance.
Pour ce graphe j'ai mis l'axe des ordonnées à l'échelle logarithmique pour avoir une meilleure visibilité car les degrés s'étendent sur un grand intervalle.

2 - Plot avec les degrés sortant et a = 0.15 :

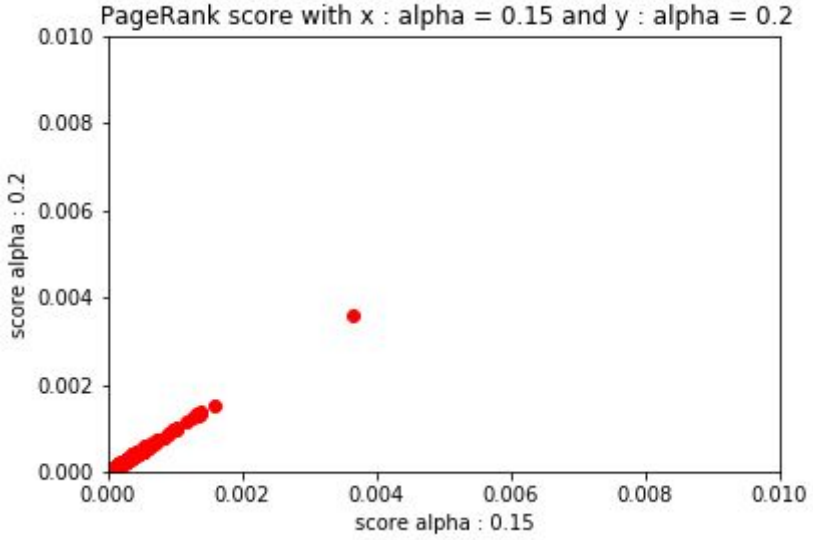


On remarque que le nombre de page qu'une autre page référence n'as pas d'impact sur son score.
Pour ce graphe j'ai mis l'axe des ordonnées à l'échelle logarithmique pour avoir une meilleure visibilité car les degrés s'étendent sur un grand intervalle.

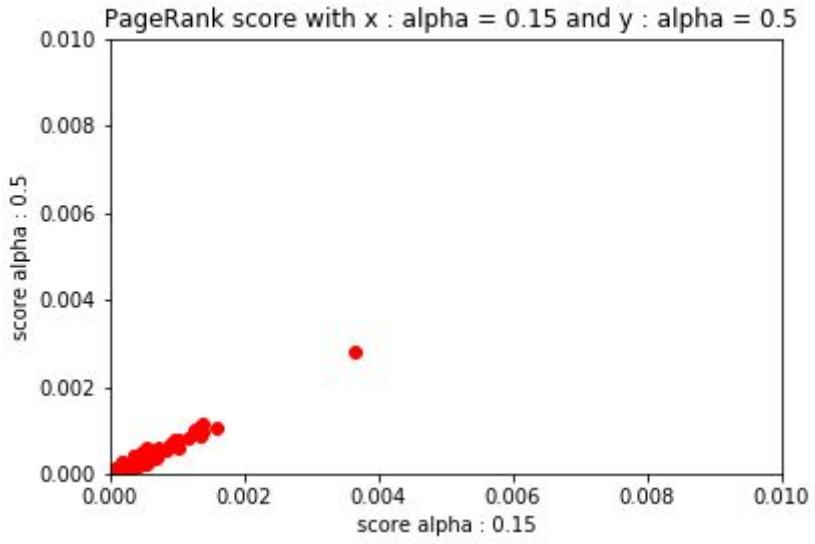
3 - Plot avec a = 0.15 et a = 0.1 :



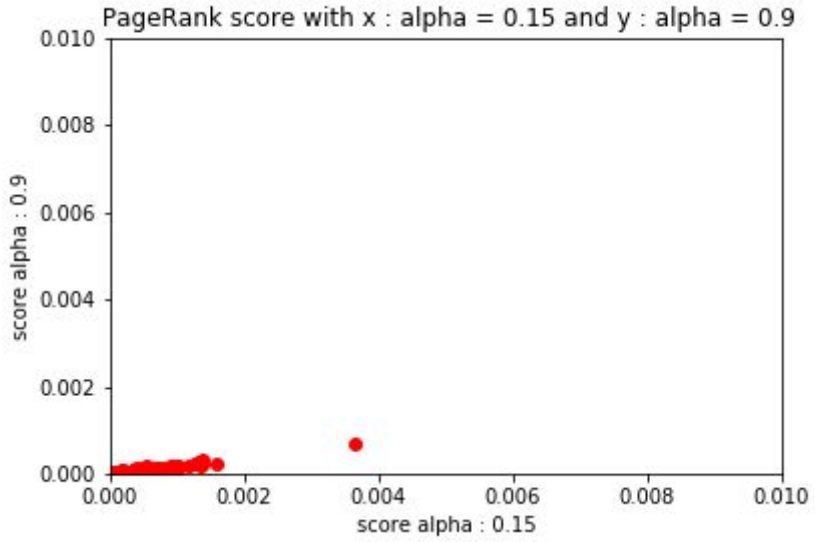
4 - Plot avec a = 0.15 et a = 0.2 :



5 - Plot avec a = 0.15 et a = 0.5 :



6 - Plot avec a = 0.15 et a = 0.9 :



Sur les plot 3 à 5 on voit que pour deux valeurs de ‘a’ différentes les PageRank obtenus reste proportionnel, les courbes sont linéaire. Sur le plot 6 on peut aussi remarquer que plus on augmente la valeur de notre “coefficient de téléportation” moins les scores obtenus sont élevés, ils commencent tous à être les mêmes, comme si on avait distribué les scores de manière aléatoire.

Rapport TME 6 CPA

Densest subgraph

Exercice 1 : k-core decomposition

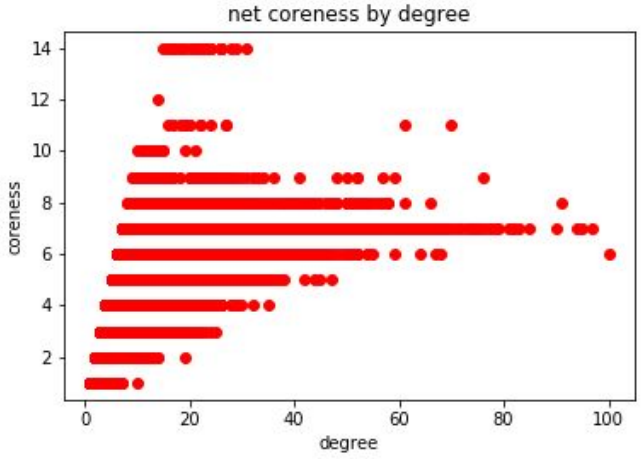
Pour le graphe email-Eu-core j'obtiens les valeurs suivante après mon core décomposition :
Maximum kcore : 34
Average degree density : 27.691244239631338
Edge density : 0.12820020481310804
Densest core ordering prefix: 217

Pour le graphe amazon j'obtiens les valeurs suivante après mon core décomposition :
Maximum kcore : 6
Average degree density : 3.9444444444444446
Edge density : 0.23202614379084968
Densest core ordering prefix: 18

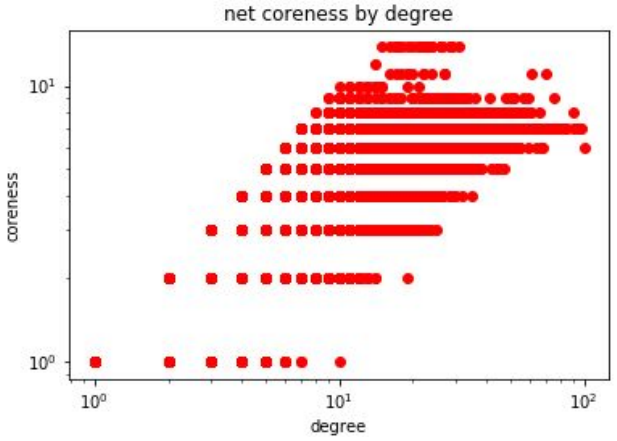
Pour le graphe lj j'obtiens les valeurs suivante après mon core décomposition :
Maximum kcore : 360
Average degree density : 191.4805194805195
Edge density : 0.49864718614718617
Densest core ordering prefix: 385

Exercice 2 : Graph mining with k-core

On observe qu'il y a une clique d'auteur qui ont un k core de 14 et qui n'ont pas forcément un degré très élevé et que tout ses auteurs sont de la même nationalité. Ce sont des auteurs qui doivent beaucoup se citer entre eux.
Maximum kcore : 14
Average degree density : 10.074074074074074
Edge density : 0.38746438746438744
Densest core ordering prefix: 27
la clique avec le plus grand kcore (14) :
Sa-kwang, Sung-Pil, Chang-Hoo, Yun-soo, Hong-Woo, Jinhyung, Hanmin, Do-Heon, Myunggwon, Won-Kyung, Hwamook, Minho, Won-Goo, Jung, Dongmin, Mi-Nyeong, Sung, Minhee, Sungho, Seungwoo, Heekwan, Jinhee, Taehong, Mikyoung ,Ha-neul ,Seungkyun, Yun-ji.



Quand on mets les échelles logarithmiques on se rapproche des graphes vu en cours, on voit qu'il y a une limite qui n'est pas depasser.



Exercice 3 : Densest subgraph

On remarque que plus on augmente le nombre d'itération plus on doit se rapprocher des valeurs obtenues par le label propagation sur les mêmes graphes.

Pour le graphe email-Eu-core j'obtiens les valeurs suivantes pour *t* itérations:

t	Average degree density	Edge density	Densest core ordering prefix
10	21.22051282051282	0.1093840867036743	195
100	22.569060773480665	0.125383670963781	181
1000	22.648936170212767	0.1211173057230629	188

Pour le graphe amazon j'obtiens les valeurs suivantes pour *t* itérations:

t	Average degree density	Edge density	Densest core ordering prefix
10	2.4818615257048093	0.0002572677024675	9648
100	3.768421052631579	0.040089585666293	95
1000	3.5625	0.0375	96

Pour le graphe amazon j'obtiens les valeurs suivantes pour *t* itérations:

t	Average degree density	Edge density	Densest core ordering prefix
10	168.89300582847628	0.0703427762717518	2402
100	184.519364448858	0.1834188513408131	1007
1000	184.50445103857567	0.18267767429561949	1011

Pour le graphe orkut j'obtiens les valeurs suivantes pour *t* itérations:

t	Average degree density	Edge density	Densest core ordering prefix
10	178.81546683432882	0.006246173914850	28629
100	189.165930599369	0.00745951853777	25360
1000			

Exercice 4 : Graph not fitting in main memory

Pour cet exercice j'ai fait un algo qui parcours plusieurs fois le fichier sans le stocker juste en incrémenter des indices dans un tableau

Conclusion

Python et ses structures de données sont parfait pour manipuler de petit graphe mais on atteint très vite les limites sur des graphes de l'ordre de la centaine de mo au giga.
Ce n'est pas le langage le plus adapter pour travailler avec de très grands graphes.