Rapport du projet STL

Sujet : Scikit network - bibliothèque Python pour les grands graphes

Encadrant : Maximilien Danisch

UE : PSTL

réalisé par

Wenzhuo ZHAO - 3971004

Zhaojie LU - 3700250

Chengyu YANG - 3703302

Zhen HOU - 3970968

22 mai 2021

À M. Maximilien Danisch.

Qu'il repose en paix.

Table des matières

Projet STL: Scikit network - bibliothèque Python pour les grands graphes

Wenzhuo ZHAO, Zhaojie LU, Chengyu YANG, Zhen HOU

22 mai 2021

Résumé

Dans la recherche des grands graphes dans le domaine d'informatique, l'algorithme de Page Rank est un indicateur pour mesurer la popularité de pages web sur le réseau du web. L'objectif de notre projet est de comprendre et comparer plusieurs versions existantes de cet algorithme classique. En effectuant la recherche sur cet algorithme, nous proposons une implémentation efficace de la version "Push based Page Rank" en faisant une contribution au Scikit-network[1] qui est une librairie Python pour l'analyse de graphes de grande taille développé à Télécom Paris.

1 Question de recherche

Le Page Rank[2] est un système de classement des pages web que les fondateurs de Google, Larry Page et Sergey Brin, ont développé à l'université de Stanford. Il est important de comprendre que le PageRank est une question de liens. Plus le PageRank d'un lien est élevé, plus il a de l'autorité. L'algorithme Page Rank

— est utilisé par Google Search pour classer les sites.

— compte le nombre et la qualité des liens vers une page.

— prend le fait qu'il existe des relations entre des sites importants.

C'est un algorithme qui analyse grossièrement l'importance des pages web avec le nombre et la qualité des hyper-liens entre les pages web comme principaux facteurs. Cet algorithme

de base n'est pas suffisant pour classer les sites web dans le monde réel car le réseau est à très grande échelle. Nous devons donc trouver et étudier certains algorithmes de moindre complexité pour répondre à nos besoins. Dans ce rapport, nous allons présenter

— l'algorithme Power Iteration qui est un algorithme itératif

— l'algorithme Push qui fait des calculs en parallélisme

2 Concepts fondamentaux

Dans cette partie, nous allons introduire des conceptions fondamentales pour vous préparer la compréhension de notre projet.

2.1 Graphes et Réseaux

Un graphe orienté $G = (V, E)$ est un couple d'ensembles.

— V est un ensemble de sommets (ou nœuds).

— $E \subseteq (V \times V)$ est un ensemble d'arêtes orientées.

Exemple 1. *Voici un exemple simple :*

$V = \{0, 1, 2, 3, 4, 5\}$

$E = \{(0, 1), (0, 2), (3, 4), (4, 5), (5, 0), (1, 4)\}$

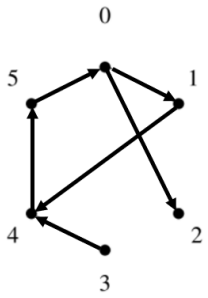


FIGURE 1 – Le graphe associé

Les liens entre nos réseaux réels sont illustrés par la structure du graphe en terme informatique. Considérons ce graphe comme un réseau composé par 6 sites web, ce schéma représente que l'utilisateur peut accéder au site web 1 en cliquant sur l'hyper-lien dans le site web 0. La

modélisation de communication du monde réel est appelée "réseaux", "réseaux complexes" ou "graphe du monde réel" en termes informatiques.

3 Notation

Dans ce rapport, nous considérons un graphe orienté G, dont l'ensemble de nœuds V et l'ensemble d'arêtes E comme introduits ci-dessus. Notons n le nombre de nœuds dans G, $d^{in}(v)$ le degré entrant du nœud v (respectivement, $d^{out}(v)$ le degré sortant du nœud v), $neighbors(v)$ l'ensemble de voisins du nœud v et e le vecteur colonne rempli par 1. Pour paramétrer la fonction, l'utilisateur peut définir ϵ la tolérance(L'algorithme s'arrête lorsque la différence de résultats entre deux itérations est inférieure à cette valeur(seuil).) de valeur de Page Rank et α la probabilité qu'un utilisateur saute à une autre page web de manière uniformément aléatoire. Pour simplifier des expressions de termes informatique, on traduit le mot "lire" en anglais à *read* et l'expression "mettre à jour" en anglais à *write*.

4 Théorie

4.1 Power Iteration

L'algorithme Page Rank a plusieurs versions dont la version la plus simple s'appelle "Power Iteration" qui est sous forme de nombreuses itérations de traitement de tous les nœuds à la fois.

4.1.1 Marche aléatoire

L'algorithme PageRank est basé sur «Marche aléatoire» («Random Walk» en anglais).

- 1. Un marcheur part d'un nœud u.
- 2. Selon la probabilité α, il détermine le déplacement ou non vers un nœud choisi uniformément au hasard.
- 3. S'il ne se déplace pas alors :
  - il suit l'un des liens sortants choisi uniformément au hasard.
  - si le nœud n'a pas de lien sortant, il se déplace vers un autre nœud aléatoire.

— 4. Aller à l'étape 2

D'après cette procédure, nous pouvons obtenir la valeur Page Rank d'un nœud v qui est approximativement égale à la probabilité que le marcheur aléatoire visite le nœud v après une suite d'itérations.

4.1.2 Matrice de transition d'un graphe

Étant donné un graphe orienté G à nnœuds et m arêtes, sa matrice de transition T est définie comme suivant :

— T est une matrice de taille $n \times n$ avec m valeurs non nulles.

— Pour chaque arête (u, v) dans G, $T_v u = \frac{1}{d^{out}(u)}$.

Si G n'a pas d'impasse (nœuds avec $d^{out} = 0$), alors le vecteur PageRank P est donné par l'équation suivante :

$$P_{t+1} = (1 - \alpha) \times T \times P_t + \alpha \times I$$

Où P_1 est le vecteur tel que chaque entrée vaut $\frac{1}{n}$, $0.1 \leq \alpha \leq 0.2$ sous une convention, t est le nombre d'itérations exécutées et I un vecteur immuable qui est rempli par $\frac{1}{n}$.

Exemple 2. *Nous pouvons obtenir une matrice de transition P_1 et le vecteur I du figure 1 comme au-dessous avec $\alpha = 0.15$.*

$$P_1 = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{pmatrix}, T = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix},$$

Donc

$$P_2 = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{40} \\ \frac{37}{120} \\ \frac{1}{6} \end{pmatrix}$$

Après un nombre infini d'itérations, le résultat va tendre vers un vecteur P ci-dessous.

$$P = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ 0 \\ \frac{1}{3} \\ \frac{1}{6} \end{pmatrix}$$

Ce vecteur représente l'importance de chaque nœud(site web) dans le graphe(réseau) qui est le but de l'algorithme Page Rank.

4.1.3 Itération

Plus d'itérations sont exécutées, plus la précision du résultat est élevée. Dans la production, pour arrêter l'exécution du programme, une tolérance ϵ est nécessaire pour déterminer la fin de suite d'itérations. On prend le vecteur P_n comme le résultat de Page Rank où n est le nombre d'itérations exécutées tout au long de calcul.

4.1.4 Complexité

Avec un certain nombre d'itérations, cet algorithme a une complexité polynomiale en temps et une complexité $O(n^2)$ en mémoire. Ceci entraîne qu'il peut prendre un assez long temps pour traiter un grand ensemble de données et exiger une très grande mémoire qui est impossible pour un ordinateur ordinaire. De ce fait, nous allons chercher des autres versions de cet algorithme existantes ayant une meilleure complexité par rapport à l'algorithme Power

Itération.

4.2 Data-Driven Algorithmes

Au lieu de traiter tous les nœuds dans une itération, on peut considérer un algorithme qui maintient une pile de nœuds. Notons *worklist* une telle pile dans la suite de cette section. Initialement, *worklist* contient tous les nœuds du graphe. L'algorithme dépile un nœud depuis *worklist*, calcule la valeur Page Rank de ce nœud puis ajoute ses voisins sortis (*out-coming neighbors*) à la fin de *worklist*.

Dans les algorithmes du type Data-Driven[3], les nœuds sont activés par leurs voisins, autrement dit un nœud devient *actif* ou *inactif*. Dans le calcul de valeur Page Rank d'un nœud, il existe différentes manières d'accéder aux voisins, par exemple soit un nœud actif lit (*read*) (*pull*) les valeurs de ses voisins puis met à jour (*write*) sa propre valeur, soit un nœud actif met à jour sa propre valeur puis met à jour (*push*) les valeurs de ses voisins. Notons ces deux manières **pull-based** algorithme et **push-based** algorithme et nous parlerons de la théorie du push-based algorithme dans la suite.

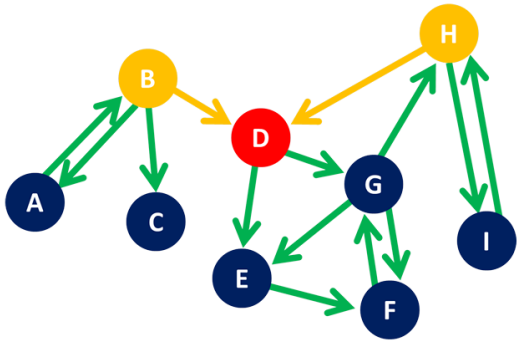


FIGURE 2 – La valeur Page Rank de D est mise à jour en lisant celles de B et H

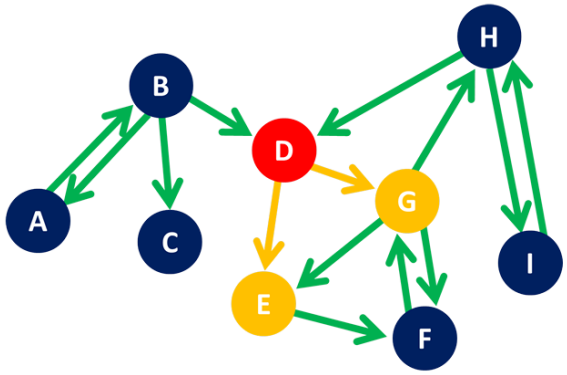


FIGURE 3 – Si la valeur Page Rank de D est mise à jour, alors il ajoute sa valeur résiduelle à E et G

4.2.1 Push

Dans l'algorithme push-based, un nœud va mettre à jour sa valeur et aussi celles de ses voisins. Il sera plus coûteux que l'algorithme Pull car il demande plus d'opérations *write*. Néanmoins, cet algorithme peut invoquer des mises à jour plus fréquentes qui peuvent faire la propagation d'information plus rapide sur le graphe donc il est plus efficace que l'algorithme pull-based.

Nous allons construire une pile *worklist* qui contient tous les nœud au début, le vecteur $x = (1 - \alpha) \cdot e$ de valeurs Page Rank et un vecteur de valeur résiduelle r. A chaque étape, la fonction dépile un nœud u à la tête de *worklist* puis met à jour sa valeur $x_u^{k+1} = x_u^k + r_u^k$ et les valeurs résiduelles de ses voisins v tel que $r_v^{old} = r_v$ et $r_v^{new} = r_v + \frac{r_v \cdot \alpha}{d[u]}$. Si nous trouvons un voisin tel que $r_v^{new} \geq \epsilon$ et $r_v^{old} < \epsilon$, nous l'ajoutons à la fin de *worklist*. Cette procédure n'arrête pas si le *worklist* n'est pas vide. A la fin, normalisons le vecteur x et on obtient le résultat.

Algorithm 1: Push based algorithm

input : graph G = (V, E), α, ϵ

output: page rank vector x

```
1  x = (1 − α)e;
2  r = 0;
3  for v ∈ V do
4  |  for w ∈ neighborsIn(v) do
5  |  |  r_v = r_v + 1/degree(w) ;
6  |  end
7  |  r_v = (1 - α )α r_v ;
8  end
9  for v ∈ V do
10 |  worklist.push(v)
11 end
12 while ! worklist.isEmpty() do
13 |  v = worklist.pop() ;
14 |  x_v = x_v + r_v ;
15 |  for w ∈ neighborsOut(v) do
16 |  |  r_w^old = r_w ;
17 |  |  r_w = r_w + r_v α / degree(v);
18 |  |  if r_w > ε & r_w^old < ε then
19 |  |  |  worklist.push(w)
20 |  |  end
21 |  end
22 |  r_v = 0;
23 end
24 x.normalize();
```

4.2.2 All Push

Dans l'algorithme mentionné ci-dessus, la pile *worklist* est initialisée par tous les nœuds du graphe (lignes 9-11). Une autre version de cet algorithme[4] implémentée[5] en C traite différemment la pile *worklist*. Prenons un nœud quelconque v et ajoutons-le à *worklist*. A partir de ce nœud, faisons le calcul de sa valeur Page Rank et des opérations *push* à ses voisins. Refaisons cette procédure sur un autre nœud jusqu'à ce que l'on ait fini de traiter tous les nœuds du graphe.

Algorithm 2: La méthode Push dans All Push Based algorithm

input : graph G = (V, E), α, ϵ

output: page rank vector x

```
1  for u ∈ V do
2      initialisation x et r with 0 ;
3      r_u = 1 ;
4      worklist.push(u) ;
5      while ! worklist.isEmpty() do
6          v = worklist.pop() ;
7          x_v = x_v + αr_v ;
8          for w ∈ neighborsOut(v) do
9              r_w^old = r_w ;
10             r_w = r_w + r_v α / degree(v);
11             if r_w > ε & r_w^old < ε then
12                 worklist.push(w)
13             end
14         end
15         r_v = 0;
16     end
17 end
18 x.normalize();
```

5 Implémentation

Après avoir appris tous ces algorithmes mentionnés ci-dessus, en tenant compte du sujet de ce projet STL et à la demande de notre encadrant M.Maximilien Dansich, nous avons implémenté l'algorithme Push based Page Rank 1 en mode hybride de langage Python et Cython pour faire une contribution[6] à scikit-network[7] qui est une librairie de Python dans le domaine de réseau complexe. En lisant la suite de l'article[3] introduisant le "push based algorithme", nous avons trouvé des manières pour accélérer le calcul. Nous utilisons un ordonnancement de priorité simple pour maximiser la propagation d'information au premier temps de calcul afin d'accélérer le calcul sur un point de vue global. Avec l'utilisation de Cython, nous pouvons utiliser la fonctionnalité de calcul en concurrence pour profiter de la grande puissance de la machine PC.

5.1 Ordonnancement

L'ordonnancement des tâches, c'est-à-dire l'ordre dans lequel les tâches sont exécutées, peut être très important pour les algorithmes de graphes. Par exemple, dans un Page Rank piloté par les données, nous constatons que chaque fois que la valeur Page Rank d'un nœud v est mise à jour, le résidu total est diminué au moins de $r_v(1-\alpha)$. Cela implique que si nous traitons d'abord les nœuds ayant une valeur résiduelle plus grande, l'algorithme pourrait converger plus rapidement.

5.2 Concurrence

Nous savons que Python est un langage de type d'interprétation qui est relativement plus lent par rapport aux autres langages impératifs. Pour éviter cette insuffisance d'efficacité de calcul, nous trouvons que le langage Cython, qui est une extension de Python et une combinaisons de fonctionnalités de Python et de C++, permet de faire une boucle en concurrence de la manière assez facile au point de vue du syntaxe de programme.
Il y a plusieurs boucles *for* dans le pseudo code de l'algorithme push based Page Rank. Pour accélérer le calcul dans ces boucles, nous utilisons la concurrence de parallélisation de Cython

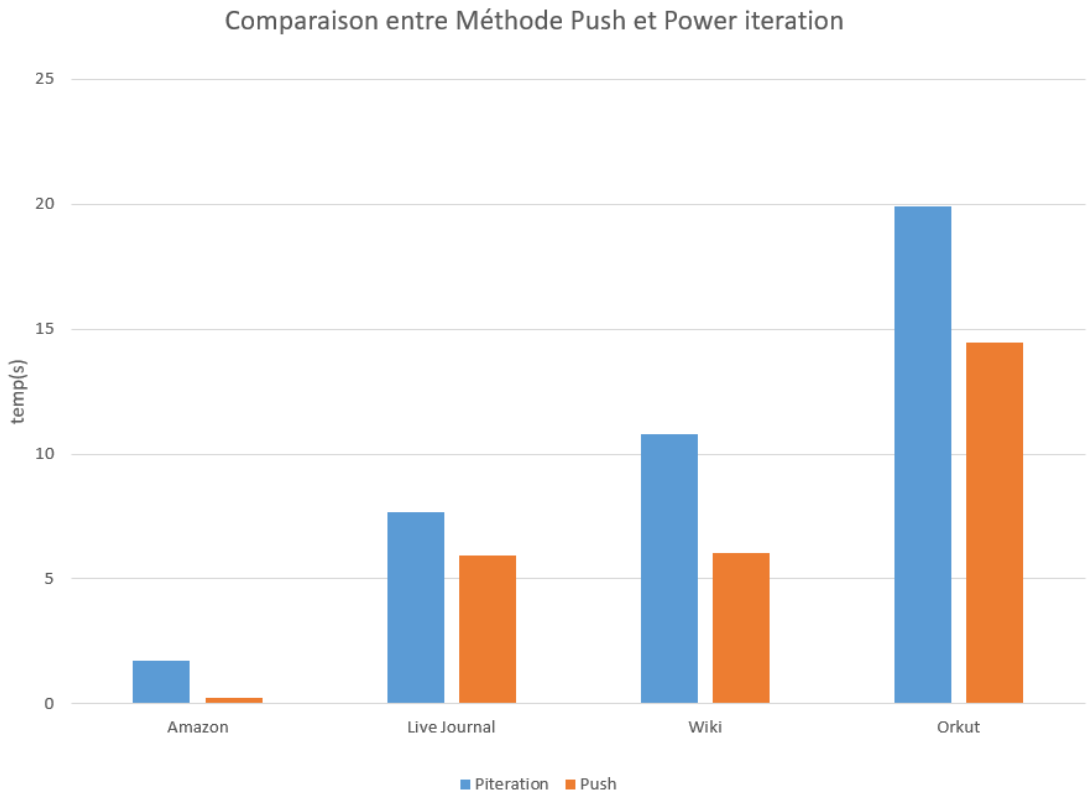
donc une meilleure utilisation de puissance de CPU de machine est bien réalisée.

6 Expérimentation

Nous avons utilisé le programme de Power Itération (réalisé par Scikit-network) et notre programme sur de grands graphes du monde réel suivant et testé le temps utilisé.

Nom du graphe	Nombre de nœuds	Nombre d'arêtes
Amazon	10^5	10^5
LiveJournal	10^6	10^7
Wiki	10^6	10^7
Orkut	10^6	10^8

Dans le schéma ci-dessous, nous pouvons visualiser que le programme de Push based Page Rank utilise moins du temps d'exécution par rapport celui de Power Itération. De ce fait, nous pouvons conclure que notre implémentation est bien plus efficace.



7 Conclusion

Pendant le travail de ce projet, nous avons bien appris des conceptions enseignées dans le cours l'UE MU4IN505 Conception pratique d'algorithmes et des autres algorithmes plus complexes. Nous avons aussi acquis des expériences sur la programmation en Python. Nous somme très enchantés de faire une contribution à la librairie Scikit-Network. En suivant le cours du tutorat de M.Adrien Demilly, nous avons aussi appris une bonne procédure d'écrire un rapport.

Nous tenons à remercier toutes les personnes qui nous ont aidés pendant ce projet. Il est malheureux d'apprendre la nouvelle de la disparation de M.Maximilien Danisch qui est notre encadrant de ce projet. Nous remercions également M.Quentin Lutz qui nous a aidés pour faire la contribution à Scikit-Network.

Nous gardons un excellent souvenir de ce projet qui constitue une expérience académique valorisante et encourageante.

Références

[1] Thomas Bonald, Nathan de Lara, Quentin Lutz et al. « Scikit-network: Graph Analysis in Python ». In : *Journal of Machine Learning Research* 21.185 (2020), p. 1-6. URL : http://jmlr.org/papers/v21/20-412.html.

[2] Larry Page, Sergey Brin, R. Motwani et al. *The PageRank Citation Ranking: Bringing Order to the Web*. 1998.

[3] Joyce Jiyoung Whang, Andrew Lenharth, Inderjit S. Dhillon et al. « Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned ». en. In : *Euro-Par 2015: Parallel Processing*. Sous la dir. de Jesper Larsson Träff, Sascha Hunold et Francesco Versaci. T. 9233. Berlin, Heidelberg : Springer Berlin Heidelberg, 2015, p. 438-450. DOI : 10.1007/978-3-662-48096-0_34. URL : http://link.springer.com/10.1007/978-3-662-48096-0_34 (visité le 29/03/2021).

[4] R. Andersen, F. Chung et K. Lang. « Local Graph Partitioning using PageRank Vectors ». In : *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. 2006, p. 475-486. DOI : 10.1109/FOCS.2006.44.

[5] M.Danisch. *Implémentation de "Page Rank All Push" en C*. https://github.com/maxdan94/push/blob/main/allpush.c.

[6] W.Zhao. *Implémentation de "Page Rank Push" en Cython de la contribution de Scikit-network*. https://github.com/sknetwork-team/scikit-network/blob/master/sknetwork/linalg/push.pyx.

[7] *Python package for the analysis of large graphs*. https://scikit-network.readthedocs.io/en/latest/.