PageRank avec Spark

Léa Briand Charles Cony

Février 2017

1 Introduction

Dans un premier temps, on peut définir le *PageRank* comme la popularité d'une page Web. Celle-ci peut être estimée à partir de l'analyse des liens entre la page analysée et toutes les autres pages du réseau. En effet, les pages Web font référence à d'autres pages Web et ainsi de suite. Le moteur de recherche Google utilise notamment cet indicateur pour classer les résultats proposés à l'issue d'une recherche. Par extension, *PageRank* désigne également l'algorithme permettant de calculer cet indicateur qui fût inventé par Larry Page.

Les demandes de recherche survenant à une fréquence très élevée et le *World Wide Web* comptant plusieurs milliards de pages, recalculer la valeur du *PageRank* pour chaque page à chaque demande devient rapidement très coûteux en opérations. Pour généraliser, une page est appelée nœud et les pages auxquelles elle fait référence sont ses voisins.

2 Théorie

2.1 Intuition

Intuitivement, plus un nœud est le voisin de plusieurs nœuds plus on souhaite que ce nœud soit considéré comme populaire. Par exemple, sur la figure 1, le nœud n°1 sera considéré plus populaire que le nœud n°2 car 5 nœuds (rouges) ont contribué à le populariser contre 2 pour le nœud n°2.

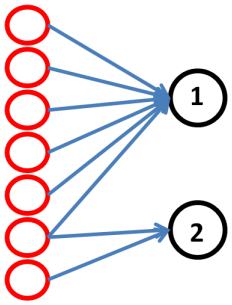

FIGURE 1 – Exemple 1

Cependant, le nombre de voisins d'un nœuds ne suffit pas à discriminer la popularité des nœuds dans l'exemple 2 où les nœuds n°1 et n°2 sont chacun voisins de 2 nœuds rouges.

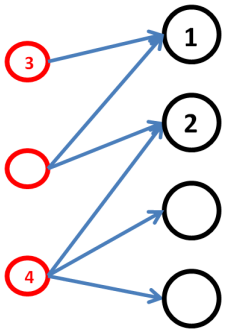

FIGURE 2 – Exemple 2

Cependant, le nœud n°4 contribue dans une moindre mesure à populariser le nœud n°2 car il possède 3 voisins par rapport au nœud n°3 qui contribue uniquement à populariser le nœud n°1. Dans ce cas, on souhaite que le nœud n°1 soit considéré plus populaire.

2.2 Algorithme

Les rangs des nœuds peut être calculé de manière itérative par l'algorithme proposé par Chen, Gal et Suel [2] et repris par Bar-Yossef et Mashiach dans l'article suggéré [1]. On connaît pour chaque nœud $p \in \{1, ..., N\}$ son nombre de voisins $d(p)$ aussi appelé *out-degree*, c'est-à-dire le nombre de nœuds auxquels il fait référence. Par suite, pour chaque nœud p, on connaît également les nœuds dont il est un voisin. On note $q \to p$ l'ensemble de ces nœuds. A l'initialisation, tous les nœuds sont de même rang $r^0(p) = \frac{1}{N}$ avec N le nombre total de nœuds. Puis, à chaque itération, les rangs deviennent :

$$r^{i+1}(p) = \sum_{q \to p} \frac{r^i(q)}{d(q)}$$

Cette méthode revient à chercher un vecteur r vérifiant :

$$r = L.r$$

où $L_{p,q} = \frac{1}{d(q)}$ si p est un voisin de q et zéro sinon. Cependant, cela suppose que tous les nœuds ont au moins un voisin ($d(q) \neq 0$) ce qui n'est pas évident sur les bases de données réelles car la relation de voisinage décrite ici n'est pas réflexive. On pourrait imaginer une page Web sans hyperlien qui constituerait un nœud sans voisin mais à laquelle plusieurs pages feraient référence. C'est pourquoi on préfère itérer à l'aide de la formule suivante :

$$r^{i+1}(p) = \frac{1-\alpha}{N} + \alpha \sum_{q \to p} \frac{r^i(q)}{d(q)}$$

Un facteur d'amortissement α bien choisi permet d'éviter qu'un groupe de nœuds voisins entre eux mais sans lien de voisinage avec les autres nœuds n'ignore complètement ces derniers.

Enfin, soit $\epsilon > 0$, on considère que $r^i(p)$ a convergé lorsque :

$$\left|\frac{r^i(p) - r^{i-1}(p)}{r^{i-1}(p)}\right| < \epsilon$$

c'est-à-dire lorsque le taux d'évolution du rang du nœud par rapport à la dernière itération est suffisamment petit. Le membre de gauche est la perte issue du nœud p au terme de i itérations.

3 Implémentation

3.1 Mise en place de l'algorithme

Afin de valider le fonctionnement de notre algorithme, on crée dans le notebook Python une base mimant des relations de nœuds sortants que l'on peut facilement maîtriser. Une fois l'algorithme bien en place et testé sur cette première base, on choisit une base de données du site Stanford Large Network Dataset Collection référençant quelques 2 millions de liens présents sur 280 000 pages Web.

Pour choisir le paramètre d'amortissement α et le critère de convergence ϵ, on récupère les valeurs conseillées dans les études citées précédemment :

— $\alpha = 0.85$

— $\epsilon = 10^{-3}$

Par ailleurs, on a un problème si l'on a des nœuds qui n'ont aucun nœud sortant. Afin de contourner cet obstacle, Chen *et al.* [2] conseillent de supprimer de la base toutes les lignes où des nœuds qui reçoivent des liens mais qui n'en envoient aucun.

Tout d'abord, on met en place notre algorithme sur un petit réseau à 4 nœuds dont une représentation est le graphe 3.

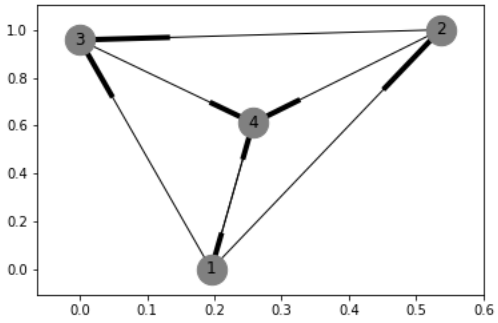

FIGURE 3 – Représentation du graphe de petite taille

On applique l'algorithme *PageRank* à ce petit réseau, en sauvegardant l'évolution du rang pour chaque nœud (voir tableau 1). Le nœud n°4 est celui qui se trouve au milieu du graphe : il est "cité" par tous les autres nœuds du réseau (c'est un voisin de tous les nœuds). Ainsi, il gagne de l'importance au cours des itérations, puisqu'il tire profit des rangs des nœuds le citant. Par ailleurs, le nœud n°1 lui cite tout le monde (à part lui-même), mais a pourtant un rang plus faible, ce qui est logique selon l'algorithme *PageRank*. Néanmoins, on voit que son rang est tout de même important. Cela peut s'expliquer par le fait que le nœud n°4 a uniquement un nœud sortant, qui est dirigé vers le n°1. Le nœud n°1 récupère alors toute l'importance du nœud n°4, celui-ci n'ayant qu'un seul nœud sortant.

TABLE 1 – Évolution des rangs par nœud et par itération

Nœud/Itération	0	1	2	3	4	5
Nœud n°1	0.250	0.250	0.401	0.324	0.317	0.344
Nœud n°2	0.250	0.108	0.108	0.151	0.129	0.127
Nœud n°3	0.250	0.215	0.154	0.197	0.193	0.182
Nœud n°4	0.250	0.427	0.337	0.328	0.361	0.346

3.2 Mesures de performance

On veut pouvoir évaluer l'évolution du nombre de nœuds qui convergent : on retient donc en mémoire le pourcentage de nœuds que l'on considère comme stables à chaque itération (voir figure 4). Afin d'avoir plusieurs critères de performance que l'on peut étudier, on introduit aussi la moyenne des taux de variation des rangs de chaque page, ainsi que leur valeur maximale, pour chaque itération (voir figure 5).

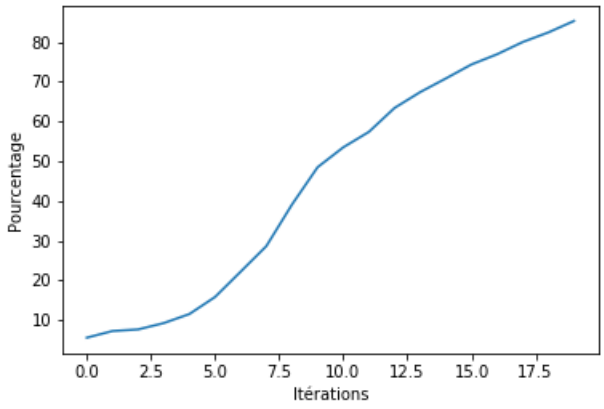

FIGURE 4 – Part des nœuds dont le rang a convergé en fonction du nombre d'itérations

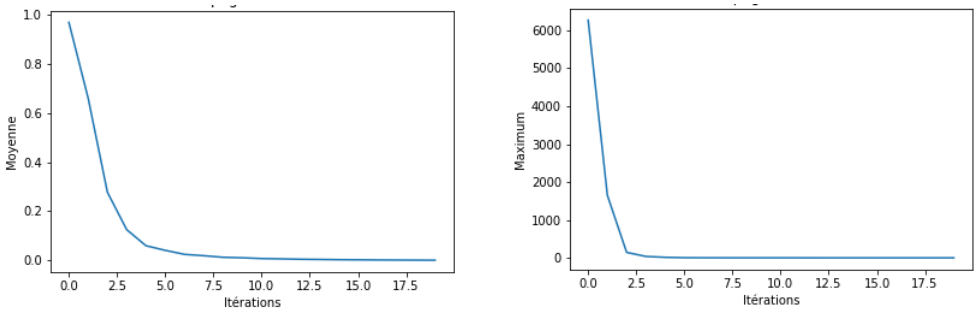

FIGURE 5 – Moyenne et maximum des pertes par itération

On constate que l'algorithme converge rapidement sur les données réelles : plus de 80% des rangs ont convergé au terme de 18 itérations. De plus, cette convergence parait relativement uniforme car les pertes maximales et moyennes sur l'ensemble des nœuds décroît très rapidement. Afin d'étudier les rangs finaux des sites obtenus via l'algorithme, on représente la distribution des rangs de tous les noeuds. On remarque que la queue de distribution est plutôt fine : beaucoup de sites ont un rang faible. Les sites avec un rang élevé sont au final plutôt rares, avec très peu de sites avec un rang supérieur à 4.10^{-6}.

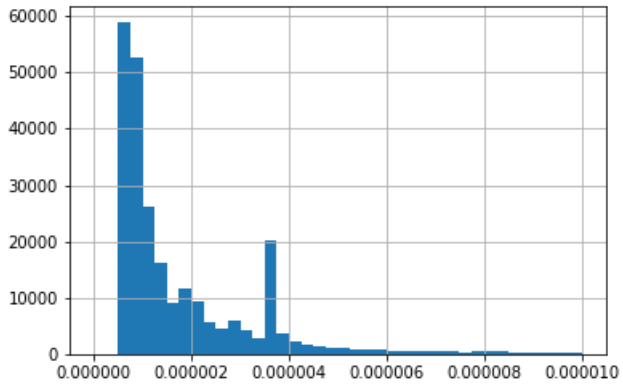

FIGURE 6 – Distribution des rangs après 20 itérations

4 Conclusion

L'algorithme *Pagerank* atteint un état stable après un nombre relativement faible d'itérations. Même pour une base conséquente, il est donc plutôt efficace. Par nature des données, définissant une architecture complexe de pages web, il aurait été beaucoup plus fastidieux d'effectuer tous ces calculs sans l'aide de Spark.

Références

[1] Z. BAR-YOSSEF et L.T. MASHIACH. « Local Approximation of PageRank and Reverse PageRank ». In : *CIKM* (oct. 2008).

[2] Q. Gan Y. CHEN et T. SUEL. « Local methods for estimating PageRank values ». In : *CIKM* (2004), 381–389.