

ETUDE ET VISUALISATION D'ALGORITHMES DE CLASSEMENT DE PAGES WEB



Thèse de Bachelor présentée par

Sébastien OLLQUIST

pour l'obtention du titre Bachelor of Science HES-SO en

**Ingénierie des technologies de l'information avec orientation en
Logiciel et Systèmes Complexes**

Septembre 2019

Professeur HES responsable

Orestis Malaspinas

Remerciements

Alors que je m'apprête à déposer ce mémoire sur les serveurs de Hepia, le rendant public pour les années à venir, je suis à même de constater que sa réalisation n'aurait pu être effectuée comme telle sans l'aide de quelques personnes que je ne vais surtout pas manquer de remercier.

Je souhaiterais commencer par remercier mon professeur encadrant Mr. Orestis Malaspina pour son aide précieuse tout au long de ce travail. Il a su me fournir des conseils justes relatifs à mes points faibles que j'ai pu par la suite corriger, du moins je l'espère.

J'aimerais également remercier mes camarades de bureau Kevin, Gabriel, Alexandre pour leur soutien constant durant ces trois dernières années. Ils ont su me pousser pour améliorer mes performances et mon sérieux dans mon travail. Merci les gars.

Un merci maintenant adressé aux deux personnes qui ont relu ce mémoire, sans qui de nombreuses fautes de syntaxe seraient sans doute toujours présentes.

Merci à toi Basile, tout juste bachelier et plein de bonne volonté. Tu as toujours été là pour moi. Je comprends que c'était un travail difficile de relire cette thèse pour me corriger, mais tu as persévééré et ton aide m'a été bien plus que précieuse. Merci mon poulet !

Quant à toi, Sophie, ma chère tante plus connue sous le surnom de "Didi", merci pour tes conseils, ta patience, ta persévérance et ta rigueur d'esprit qui m'ont beaucoup aidé dans la rédaction de cette thèse, mais pas que !

Enfin, je remercie ma mère pour m'avoir toujours guidé dans mes choix, parfois simples, parfois difficiles. Tu m'as sans relâche remis dans le droit chemin lorsque je commençais à chavirer et tes conseils, parfois si évidents, m'ont été d'un grand secours. Merci pour tout Maman.

Table des matières

Remerciements	3
Enoncé du sujet	9
Résumé	11
Table des figures	13
Liste des acronymes	15
Introduction	17
I Les algorithmes du Web	19
1 La naissance du Web	21
1.1 Création du World Wide Web	21
1.2 Qu'est-ce que le Web ?	22
1.3 La naissance du référencement	22
1.3.1 Qu'est-ce que le référencement ?	23
1.3.2 Search Engine Optimization	23
1.3.3 Le besoin d'être vu	23
2 L'algorithme PageRank	25
2.1 Définitions	25
2.2 Prérequis	26
2.2.1 Algèbre	26
2.2.2 Probabilités	26
2.3 Fonctionnement	27
2.3.1 Initialisation	27
2.3.2 Algorithme détaillé	28
2.3.3 Exceptions	29

TABLE DES MATIÈRES

2.4	Exemple d'application	29
2.4.1	Algorithme théorique	29
2.4.2	Algorithme implémenté	31
2.4.3	Explications et résultats	32
2.5	Robustesse des paramètres de l'algorithme	33
2.5.1	Variation du damping factor	34
2.5.2	Variation du nombre d'itérations	35
2.5.3	Variation du nombre de liens	36
3	Alternatives au PageRank	41
3.1	CheiRank : Plus de communication	42
3.1.1	Fonctionnement	42
3.1.2	Tests ciblés sur un réseau simple	43
3.1.3	Tests plus approfondis	43
3.2	HITS : Plus de critères	46
3.2.1	Fonctionnement	46
3.2.2	Comparaison des résultats	47
3.2.3	Avantages et inconvénients du HITS	48
3.3	Une solution au problème du spam ?	49
3.3.1	Qu'est-ce que le TrustRank ?	49
3.3.2	Comment fonctionne-t-il ?	50
3.3.3	Intégration dans le Web	50
II Manipulation des données		51
4	Familiarisation avec l'outil Neo4j	53
4.1	Qu'est-ce que Neo4j ?	53
4.2	Connexion à Neo4j	54
4.3	Protocole Bolt	54
4.4	Prise en compte du fichier de configurations	55
4.5	Cypher	55
4.6	Importation de données externes	57
4.6.1	Récupération de données externes	57
4.6.2	Génération de fichiers CSV	57
4.6.3	Importation des données dans Neo4j	58
4.6.4	Vérification de l'import des données	59
4.7	Contraintes à prendre en compte	59
5	Pour aller plus loin	61
5.1	Extraction d'un sous graphe	61

5.1.1	Comment extraire une partie d'un graphe ?	61
5.1.2	Approfondissement de certains tests	63
5.1.3	Visualisation d'un graphe sur une page HTML	67
5.2	Visualisation plus précise de l'évolution des PageRanks	69
5.2.1	Une seule itération	69
5.2.2	Plusieurs itérations	69
5.2.3	Résultats et discussions	71
Conclusion		73
Annexes		77
Annexe A Installation et configurations de Neo4j		77
Annexe B Installations d'Anaconda et IPython		79
Annexe C Graphe formé par les voisins du noeud principal		81
Bibliographie		83

ÉTUDE ET VISUALISATION D'ALGORITHMES DE CLASSEMENT DE PAGES WEB

ORIENTATION : LOGICIELS ET SYSTEMES COMPLEXES

Descriptif :

Le but de ce projet est l'étude des algorithmes de classement de page sur internet. Dans un premier temps il s'agit d'implémenter certains de plusieurs algorithmes de classement (PageRank, HITS, ...) et étudier leurs performances (vitesse de convergence, stabilité, ...) sur de petits graphes générés aléatoirement, ainsi que de visualiser les itérations des différents algorithmes. Puis, ces algorithmes seront appliqués sur de beaucoup plus grands graphes.

Travail demandé :

- Apprentissage théorique des algorithmes de ranking.
- Recherche des différentes technologies pour travailler sur les graphes (manipulation et visualisation).
- Implémentation de prototypes d'algorithmes de ranking sur de petits graphes et études de performances.
- Étude des possibilités de visualisation.
- Collecte de données pour des graphes beaucoup plus grands.
- Extension de l'étude des graphes aux grands graphes.

Candidat :

OLLQUIST SEBASTIEN

Filière d'études : ITI

Professeur responsable :

MALASPINAS ORESTIS

En collaboration avec :

Travail de bachelor soumis à une convention
de stage en entreprise : nonTravail de bachelor soumis à un contrat de
confidentialité : non

RÉSUMÉ

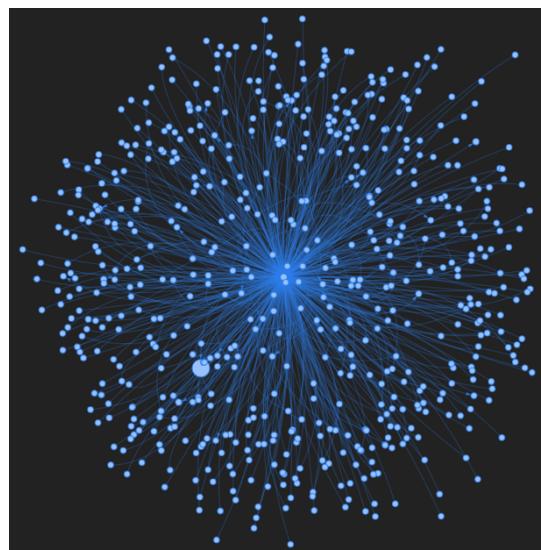
La croissance et l'évolution de l'Internet et du monde de la communication lors de ces deux dernières décennies est flagrante. Ces mondes se sont foncièrement complexifiés et l'usage du Web fait aujourd'hui partie intégrante de notre quotidien. Nous sommes tous - dans une certaine mesure - entourés par des algorithmes informatiques qui nous permettent, entre autres, de créer notre fil d'actualités personnalisé sur les réseaux sociaux ou encore de nous donner un diagnostic plus précis quant à notre état de santé.

Le Web, véritable labyrinthe, engendre de nombreux problèmes liés au guidage des internautes à travers ces milliards de sites internet.

Les moteurs de recherche s'avèrent dès lors être la solution idéale. On dénote une importante catégorie d'algorithmes liés à ce domaine-là, les algorithmes de classement de pages Web. Ce sont eux qui vont permettre à l'internaute de faciliter sa recherche de contenu en fournissant d'une part des résultats plus ciblés correspondant davantage aux critères de recherche et d'autre part, des résultats plus communs qui ont essentiellement tendance à être visités plus de fois que d'autres.

La majeure partie de cette thèse porte sur l'étude en profondeur de ces différents algorithmes en particulier l'algorithme de recherche de Google (l'algorithme PageRank) afin de mieux comprendre leur fonctionnement et leurs particularités. Ultérieurement, nous établirons une liste des avantages et des inconvénients de chacun.

L'autre partie de ce travail porte plutôt sur le côté pratique. Nous aurons la possibilité de comprendre davantage le fonctionnement de l'algorithme de Google en effectuant diverses manipulations sur un ensemble de données externes réelles du Web, auparavant récupéré et stocké dans une base de données. Elle fera également partie intégrante de notre champ d'étude.



Candidat :

OLLQUIST SÉBASTIEN

Filière d'études : ITI

Professeur responsable :

MALASPINAS ORESTIS

Travail de bachelor soumis à une convention de stage
en entreprise : non

Travail soumis à un contrat de confidentialité : non

Table des figures

2.1	Exemple de graphe	29
2.2	Tests des PR	33
2.3	Graphe de test	34
2.4	Variation du damping factor	35
2.5	Variation du nombre d'itérations	36
2.6	Variation du nombre de liens vers c ; il ne référence que certains liens du réseau	37
2.7	Variation du nombre de liens vers c ; il référence tous les noeuds du réseau	37
3.1	Statistiques de comparaison entre le PR et le CR	45
3.2	Premier test sur l'algorithme HITS	48
4.1	Fonctionnement du protocole Bolt, Source : NeoTechnology Inc.	54
4.2	Premier graphe simple	56
4.3	Vérification de la bonne importation des données	60
5.1	20 liens sortants des sites les mieux référencés	62
5.2	20 liens entrants vers le site le mieux référencé	63
5.3	Liens sortants du noeud 1'963	64
5.4	Liens sortants du noeud 0	64
5.5	Liens entrants dans le noeud 1'963	66
5.6	Liens entrants dans le noeud 0	66
5.7	Visualisation HTML de tous les noeuds entrant dans le noeud 6	68
5.8	Vérification de la bonne exécution de l'algorithme	69
5.9	Evolution des PR sur un grand graphe	71
B.1	Modification du profil bash	79
B.2	Fenêtre de démarrage de Jupyter	80
B.3	Fenêtre de démarrage de Jupyter	80
C.1	Un graphe en forme de coquillage	81

Liste des acronymes

APOC	Awesome Procedures On Cypher
ARPANET	Advanced Research Projects Agency Network
ASCII	American Standard Code for Info Interchange
CR	CheiRank
CSV	Comma Separated Values
HITS	Hyperlink-Induced Topic Search
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
PR	PageRank
SEO	Search Engine Optimization
SQL	Structured Query Language
TR	TrustRank
URL	Uniform Ressource Locator

Introduction

DEPUIS de nombreuses années, l'Internet et plus particulièrement le Web se sont métamorphosés. On recense toujours plus de sites, de données et de machines accessibles aux internautes. Lors de la création du Web, le problème majeur était la possibilité pour un internaute de retrouver son chemin parmi des millions voire des milliards de sites internet. La création d'un moteur de recherche, pour guider les internautes à travers ce grand labyrinthe était alors inévitable. La firme Google possède depuis plus de vingt ans le quasi monopole du secteur des moteurs de recherche ce qui fait d'elle, l'archétype de la réussite moderne.

Aujourd'hui, les algorithmes informatiques ont envahi notre vie quotidienne, on les retrouve en effet dans bon nombre de nos habitudes. Pourtant, malgré leur omniprésence et leur importance dans notre vie, ils restent pour la plupart relativement obscurs au grand public. Ce sont eux qui permettent la construction de notre fil d'actualités sur les réseaux sociaux, l'accélération des transactions boursières ou encore l'orientation des internautes vers des produits spécifiques suite à leur achat sur internet. Toutes ces activités sont aujourd'hui partie intégrante de notre quotidien.

On note une autre catégorie d'algorithmes tous aussi importants, visant à classer des sites internet en leur donnant une certaine quote de popularité. Les moteurs de recherche se servent pour la plupart de ces algorithmes de classement. Ils ont comme principal objectif de donner des scores à chacune des pages. Certains de ces scores se basent sur des critères concrets. Google possède son propre algorithme de ranking : le PageRank (abrévié PR). C'est cet algorithme, régissant le classement des pages Web sur internet, qui sera le principal objet de notre étude.

Il existe également des alternatives au PR, utilisées notamment par le reste des moteurs de recherche présents sur le Web. Ce sont entre autres, le HITS pour Yahoo ou le CR, plus développé que le PR. Ces alternatives se distinguent du PR en possédant notamment un certain nombre de critères supplémentaires.

Depuis mes débuts dans l'informatique, je ne cesse de m'intéresser au monde des sciences et de la technologie. L'une de mes principales sources de motivation est le simple fait de pouvoir créer librement des programmes qui me sont utiles en mettant à profit de nouvelles technologies. J'ai tout de suite saisi l'opportunité de réaliser le travail de mon choix en associant mathématiques et informatique. Le sujet du PR m'est de suite venu à l'esprit. Je souhaitais en apprendre davantage sur l'une des faces cachées de Google et ce travail était une opportunité à ne pas manquer.

L'objectif principal de ce mémoire est de comprendre dans un premier temps comment fonctionnent ces algorithmes de classement et sur quels critères ils se basent. Dans un second temps, de les comparer afin de déterminer, in fine, lequel est le plus fiable.

Le premier chapitre a pour objectif d'initier le lecteur au monde du Web. Nous retraccerons d'abord l'historique du Web, de ses débuts fulgurants à son influence mondiale actuelle. Nous chercherons ensuite à en donner une définition plus précise. Puis nous introduirons la notion de référencement.

Le deuxième chapitre a pour but de comprendre plus en profondeur la finalité du PR et son fonctionnement au sein de l'Internet. Nous favoriserons une approche plus mathématique en détaillant quelques points de vue théoriques. Puis nous décrirons étape par étape comment se comporte l'algorithme. Nous donnerons ensuite un exemple d'application et nous ferons varier les différents paramètres qu'utilise l'algorithme afin de noter l'influence qu'ils peuvent avoir sur l'agissement de celui-ci.

Dans le chapitre 3, nous proposerons quelques alternatives au PR en faisant une liste exhaustive des avantages et des inconvénients qu'ils peuvent avoir. Nous proposerons également une implémentation de chacune d'entre elles en Python.

Nous nous familiariserons avec l'outil Neo4j au cours du chapitre 4 pour créer, manipuler et afficher des graphes (parfois complexes). Nous créerons d'abord des graphes simples manuellement puis nous importerons des données externes réelles beaucoup plus imposantes afin d'y appliquer l'algorithme PR.

Pour finir, nous appliquerons dans le chapitre 5 l'algorithme PR à un graphe plus conséquent et nous effectuerons une série de tests accompagnée d'une visualisation minimale pour mieux saisir la situation.

Première partie

Les algorithmes du Web

« *Qui maîtrise l'Internet,
maîtrise le Monde.* »

— Christophe Bonnefont

Chapitre 1

La naissance du Web

AVANT de rentrer dans les particularités du Web, il semble nécessaire de retracer son historique pour mieux décrire ses principaux composants et mieux évaluer la quantité de données qu'il contient.

1.1 Crédit du World Wide Web

Vers la fin des années 1950 aux Etats-Unis, les laboratoires Bell créent leur premier modem permettant d'envoyer des données informatiques sur une ligne téléphonique. L'idée d'un premier réseau interconnecté d'ordinateurs est alors évoquée.

En 1967 a lieu la première conférence ARPANET, point de départ de l'Internet actuel. Alors, seuls 4 ordinateurs dans le monde servent de serveurs¹.

C'est au cours des années 1980 qu'apparaît le Web tel que nous le connaissons aujourd'hui. Les premiers ordinateurs apparaissent à destination du "grand public" et la révolution informatique est lancée.

En 1984, le CERN (Centre Européen de Recherche Nucléaire) rejoint Internet. Sept ans plus tard, il crée le "World Wide Web". Le nombre d'ordinateurs connectés est alors d'environ 617'000.

Conséquence de cette explosion, la nécessaire organisation du Web voit l'émergence des moteurs de recherche. Créée en 1998, Google est aujourd'hui une référence dans le domaine.

1. Cette donnée et les suivantes recensées dans ce chapitre sont tirées du site internet de l'Université de Rennes. Voir : https://www.sites.univ-rennes2.fr/urfist/internet_chiffres

Fait	En 1998	En 2013	En 2018
Nombre de pages Web indexées	26 M	60'000 Mds	130'000 Mds
Nombre de requêtes par mois	10'000	3,3 Mds	168 Mds
Nombre d'employés	3	44'777	98'771

Source : RankTracker Blog, 2018

TABLE 1.1: L'évolution flagrante de Google

Ci-dessus, quelques chiffres sur l'évolution de Google.

Aujourd'hui, Internet est un élément structurant et incontournable de notre société. Le nombre d'ordinateurs a explosé, on en recense plus de deux milliards à travers le monde. Corrélairement, le nombre de sites internet avoisine les cinq milliards.

1.2 Qu'est-ce que le Web ?

Il est très commun de définir le Web comme "la toile d'araignée mondiale" qui donne accès à l'Internet tout entier. C'est autrement dit un système qui permet aux internautes de se connecter à une page Internet via un navigateur.

Le Web est avant tout caractérisé par deux protocoles très connus, HTML et HTTP qui permettent respectivement de créer des pages internet et de naviguer entre elles.

Le HTML est un langage de balisage très facile à apprendre permettant d'écrire des documents structurés et mis en forme, pouvant contenir du texte, des formulaires et des ressources multimédia telles que des images ou des vidéos.

HTTP est quant à lui un protocole de communication client-serveur. Dans le monde du Web, il permet à un internaute d'accéder à un serveur contenant des données via une page internet.

1.3 La naissance du référencement

Avec l'essor des sites internet, nous sommes tous les jours confrontés au problème de retrouver le site que l'on souhaite. La création d'un système permettant de référencer des sites internet est dès lors inévitable.

1.3.1 Qu'est-ce que le référencement ?

Le référencement Web est un procédé de classement de sites internet basé sur des mots clés, la fréquentation quotidienne du site, sa popularité et sa publicité. Tous ces critères contribuent à sa note et à son positionnement.

Le référencement intervient lors de l'utilisation d'un moteur de recherche. Un site web avec un bon référencement figurera parmi les premiers sites affichés dans la recherche. Il aura ainsi une beaucoup plus grande chance d'être visité. Ce processus assure le fait qu'un internaute puisse plus facilement tomber sur les pages qu'il souhaite consulter.

1.3.2 Search Engine Optimization

Il existe aujourd'hui des systèmes permettant d'optimiser la recherche d'un internaute appelés SEO².

Ces systèmes d'optimisation sont apparus avec l'avènement de l'informatique et les débuts de l'internet "grand public". La création de Google en 1998 a fortement accéléré l'émergence des SEO, notamment grâce à l'algorithme PageRank (voir chapitre 2).

L'objectif principal des SEO est d'améliorer le positionnement d'une page Web dans les résultats de recherche. Si un site se trouve sur la première page des résultats de recherche, il sera considéré comme "bon" et son nombre de visites se verra considérablement augmenter.

1.3.3 Le besoin d'être vu

La prolifération du nombre d'internautes³ crée un nouveau défi aux entreprises et aux multinationales : le besoin d'être vues et reconnues.

La société actuelle et son économie sont aujourd'hui indissociables de l'évolution du Web. Si ces entreprises et multinationales veulent conserver leur position, elles sont indubitablement amenées à se livrer à la concurrence référentielle. Le référencement est donc d'une importance majeure.

2. SEO signifie : Search Engine Optimization. Les informations ci-dessus sont disponibles à l'adresse suivante : <https://www.primelis.com/blog/comment-est-ne-le-seo/>

3. 16 millions d'internautes sont recensés en 1995; 2,7 milliards en 2013 et plus de 4,4 milliards en 2018. Source : <https://www.journaldunet.com/ebusiness/le-net/1071539-nombre-d-internautes-dans-le-monde/>

Chapitre 2

L'algorithme PageRank

VERS la fin des années 1990, un dénommé Larry Page (cofondateur de Google) prenant conscience de l'expansion massive du Web décide de créer un algorithme permettant aux internautes d'optimiser leur recherche au sein de l'internet pour mieux répondre à leur besoin, l'algorithme PageRank (abrégé PR dans ce document). Son principe est simple : déterminer la popularité d'une page web dans l'internet en lui attribuant un nombre. Plus ce nombre est élevé, plus haut est le classement du site. [Rob16]

Ce chapitre va tenter d'aborder les bases à la fois conceptuelles, théoriques et axiomatiques de la méthode de référencement de Google ainsi que de fournir un ensemble de tests visant à instruire le lecteur sur l'importance du rôle qu'elle joue dans le monde du Web.

2.1 Définitions

Selon Google,

"PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites."

Cela veut dire qu'un site internet a un rang élevé si d'une part les nombreux autres sites qui le réfèrent ont eux aussi un rang élevé et d'autre part s'il ne référence qu'un petit nombre de sites. Autrement dit, le site doit s'imposer comme seule référence dans son domaine.

D'un point de vue plus mathématique, la définition stipule que le PR correspond à un indicateur sous la forme d'une distribution de probabilité représentant la possibilité qu'a un utilisateur lambda appelé "surfer" d'arriver sur une page en cliquant sur un lien de manière aléatoire. [Wil06]

2.2 Prérequis

L'étude de cet algorithme nécessite des connaissances dans plusieurs domaines des mathématiques et de l'informatique théorique notamment les probabilités, l'algèbre linéaire et la théorie des graphes. L'idée de ce chapitre est de lister les différentes manières d'aborder le problème avant de rentrer plus en détail dans les spécificités de l'algorithme.

Avant toute chose, les vecteurs seront dans ce mémoire représentés par une lettre minuscule en gras, les matrices par une lettre majuscule en gras et les scalaires par une simple lettre minuscule.

2.2.1 Algèbre

On considère dans un premier temps un vecteur \mathbf{v} comportant les rangs initiaux. Chaque importance est uniformément distribuée parmi les noeuds du graphe. Pour un graphe à n noeuds, le rang initial vaut $1/n$.

À la première itération, on multiplie \mathbf{v} par la matrice de transition¹ \mathbf{P} du graphe afin d'obtenir un vecteur comportant la nouvelle valeur temporaire des PR. On reproduit le processus en multipliant le résultat par cette matrice, et ainsi de suite jusqu'à avoir des résultats qui convergent².

Le système converge ainsi, si le nombre d'itérations k est agrandi jusqu'à de très grandes valeurs :

$$\mathbf{v}^* = \lim_{k \rightarrow \infty} \mathbf{P}^k \cdot \mathbf{v}$$

D'un point de vue purement algébrique, cette même équation peut être représentée par un système de n équations à n inconnues, n étant le nombre de noeuds. Le vecteur \mathbf{v}^* choisi sera notre vecteur de PR.

2.2.2 Probabilités

Les probabilités vont s'avérer utiles dans la mesure où l'on souhaite mesurer la popularité d'une page Web. On peut aisément déterminer cette importance grâce à la probabilité qu'un surfer sur l'internet clique de manière aléatoire sur un lien pour naviguer vers un autre site Web. Cette action là est représentée par un **processus de Markov**. Le surfer de Google est un bon exemple d'un tel processus dans lequel un système bouge d'état en état basé sur une probabilité qui dénote la possibilité de partir de chaque état pour se rendre vers chaque autre état [Mat12].

-
1. Une matrice de transition représente en théorie des probabilités un tableau récapitulant toutes les probabilités de passer d'un état à un autre.
 2. Ces résultats vont au bout d'un certain nombre d'itérations tendre vers une certaine valeur

Plus sur les chaînes de Markov Une chaîne de Markov se base sur notre matrice de transition précédemment construite. La matrice est aussi appelée **matrice stochastique**. Il s'agit d'une suite de variables aléatoires ($X_n, n \in \mathbf{N}$) qui permet de modéliser l'évolution dynamique d'un système aléatoire où X_n représente l'état du système à l'instant n .

Une des particularités d'une chaîne de Markov, dite propriété de Markov, est que **son évolution future ne dépend du passé qu'au travers de sa valeur actuelle**.

$$\mathbf{P} = \begin{pmatrix} P_{11} & P_{12} & \cdots & P_{1j} \\ P_{21} & P_{22} & \cdots & P_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ P_{i1} & P_{i2} & \cdots & P_{ij} \end{pmatrix}$$

Dans cette matrice dite "stochastique", chaque élément P_{ij} représente la probabilité de passer de l'état i à l'état j . Si l'on prend la transposée d'une matrice stochastique, on remarque que chaque colonne représente un vecteur stochastique c'est-à-dire que la somme de tous ses éléments vaut 1.

Une chaîne de Markov dans le cas de l'algorithme PR est dite **ergodique** c'est-à-dire que l'on peut se rendre vers n'importe quel noeud du graphe depuis un noeud de départ. Nous verrons au cours de ce chapitre qu'il existe quelques cas de figure où des noeuds sont absorbants et ne contiennent donc aucun lien vers un autre noeud. Dans ce cas, pour rendre la matrice ergodique, il faut mathématiquement parlant, remplacer les éléments du vecteur concerné par une suite de n événements équiprobables.[Du15]

2.3 Fonctionnement

Lorsque l'article "The PageRank citation ranking : Bringing order to the Web" [PB98] de Larry Page est révélé, il y a confusion car la définition qui y figure ne correspond pas avec la formule inscrite.

Le but de cette partie du travail est d'une part de comprendre l'algorithme en détail et d'autre part de déterminer les différences entre ces deux définitions afin de donner les bons arguments pour se concentrer que sur l'une des deux.

2.3.1 Initialisation

Le Web recense un ensemble de sites internets symbolisés par des noeuds. Il est représenté par un graphe dirigé $G = (V, E)$ [Lem] où V représente l'ensemble des pages du graphe et $e \in E$ un lien qui relie une page i à une page j .

Au départ, le PR est équitablement réparti sur tous les noeuds du réseau. Dans un graphe comportant 5 noeuds, le PR de départ sera de 0.2 pour chacun des noeuds. Globalement, pour un réseau à n noeuds :

$$PR(n_1) = PR(n_2) = \dots = PR(n_n) = \frac{1}{n} \quad (2.1)$$

2.3.2 Algorithme détaillé

Version originale La formule originale du PR s'écrit de la manière suivante :

$$PR(p_j) = (1 - d) + d \sum_{p_i \in M(p_j)} \frac{PR(p_i)}{L(p_i)} \quad (2.2)$$

Dans l'équation ci-dessus, on définit chaque élément comme suit :

- $PR(p_j)$ est le PR d'une page j
- p_j représente la page actuelle
- Les p_i sont les pages liées à la page p_j
- $M(p_j)$ représente l'ensemble des noeuds entrants dans une page p_j
- $L(p_i)$ vaut le nombre de liens sortants de la page p_i
- d est appelé **Damping Factor**. En règle générale, la théorie du PR part du principe qu'un surfer imaginaire va cliquer sur des liens pour naviguer sur le web et s'arrêter à un certain moment. La probabilité que le surfer continue s'appelle le "Damping Factor". Des études ont montré que le facteur idéal est 0.85. [BP10]

Au bout d'un certain nombre d'itérations, le PR tend vers une certaine valeur. Le but est de **déterminer après combien d'itérations la valeur converge**. C'est cette valeur-là qui différencie principalement nos deux algorithmes.

Version modifiée Dans les papiers originaux de Larry Page, la définition donnée n'est pas en accord avec la formule inscrite. En effet, lorsque l'on fait tendre le nombre d'itérations vers l'infini, l'équation 2.2 nous donne un résultat de rang très proche de n alors que la définition nous énonce clairement que la somme des PR vaut 1. Pour retrouver la valeur de 1, il nous suffit de diviser le rang par n .

On peut donc établir la formule correspondant à la définition décrite dans le papier :

$$PR(p_j) = \frac{1 - d}{n} + d \sum_{p_i \in M(p_j)} \frac{PR(p_i)}{L(p_i)} \quad (2.3)$$

Dans cette équation, n représente le nombre de noeuds du réseau. Plus le nombre de noeuds augmente, plus le rang de ceux-ci diminue. La somme que l'on calcule à la fin est égale à 1 étant donné que l'on se sert d'une **distribution de probabilités**.

De manière générale, ce qui distingue ces 2 formules n'est pas l'ordre du ranking car il est dans de nombreux cas presque le même, mais plutôt la valeur du PR elle-même. Afin d'éviter toute confusion par la suite, c'est la formule originale que nous utiliserons.

2.3.3 Exceptions

Il se peut dans certains cas qu'un noeud ne possède aucun lien sortant ce qui crée un **deadlock** pour notre surfer. Le noeud est alors appelé **absorbant**. Dans ce cas, l'algorithme PR stipule que le surfer se rendra sur un des noeuds du graphe de manière totalement aléatoire puis recommencera le processus de navigation dans le graphe, du moins jusqu'au prochain deadlock. Ce processus est appelé **téléportation**. Egalement, si le damping factor est réduit, le surfer aura plus de chances d'arrêter de cliquer et le processus de téléportation prendra de nouveau effet. Dans un cas normal, lors de la téléportation, un noeud totalement aléatoire aura 15% de chances d'être visité.

2.4 Exemple d'application

2.4.1 Algorithme théorique

Considérons le graphe suivant :

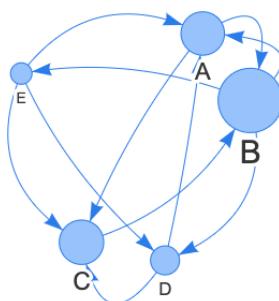


FIGURE 2.1: Exemple de graphe

On peut à partir de ce graphe déduire les deux matrices suivantes :

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}, \quad \mathbf{P} = \begin{pmatrix} 0 & 1/2 & 1/3 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 & 1 \\ 0 & 1/2 & 0 & 0 & 0 \\ 1/3 & 0 & 1/3 & 0 & 0 \\ 1/3 & 0 & 1/3 & 1/2 & 0 \end{pmatrix}$$

La matrice **A** est appelée **matrice d'adjacence** et nous sert à déterminer quels sont les liens hypertexte de notre réseau. **P** quant à elle est une **matrice stochastique** représentant pour chaque noeud les probabilités de se rendre dans un autre noeud. Elle correspond en théorie des probabilités à la matrice de transition d'une chaîne de Markov.

Si une ligne entière de cette matrice est vide, on a à faire à un deadlock et elle n'est donc pas stochastique car $\sum PR \neq 1$. Comme nous l'avons cité précédemment, pour rendre cette matrice stochastique, on va remplacer la colonne de l'état absorbant par un vecteur \mathbf{e}^T ³ comportant des valeurs équiprobables. On note au final :

$$\mathbf{e}^T = \begin{pmatrix} 1/n \\ 1/n \\ \vdots \\ 1/n \end{pmatrix} \quad (2.4)$$

Physiquement, cela représente le **repositionnement aléatoire** du surfer dans le graphe lors de l'encontre d'un deadlock.

Version modifiée On va dans un premier temps se référer à l'équation 2.3, il ressort au bout de la première itération les calculs suivants :

$$\begin{aligned} PR(A) &= 0.03 + 0.85(0.0\bar{6} + 0.1 + 0.0\bar{6}) = 0.228\bar{3} \\ PR(B) &= 0.03 + 0.85(0.1 + 0.2) = 0.285 \\ PR(C) &= 0.03 + 0.85(0.1 + 0.1 + 0.0\bar{6}) = 0.25\bar{6} \\ PR(D) &= 0.03 + 0.85(0.0\bar{6} + 0.0\bar{6}) = 0.14\bar{3} \\ PR(E) &= 0.03 + 0.85(0.0\bar{6}) = 0.08\bar{6} \end{aligned}$$

Le noeud *B* est au bout de la première itération le mieux référencé. Si l'on reproduit ce processus un certain nombre de fois, les valeurs commenceront à converger vers leur valeur finale. L'équation 2.1 est vérifiée car la somme des PR vaut bien 1.

3. Le vecteur *e* est transposé afin de travailler sur des colonnes plutôt que sur des lignes.

Version originale Si toutefois on utilise l'algorithme original, la somme totale des PR converge vers 5 en fonction de l'augmentation du nombre d'itérations. Les calculs obtenus sont ainsi différents :

$$\begin{aligned} PR(A) &= 0.15 + 0.85(0.0\bar{6} + 0.1 + 0.0\bar{6}) = 0.348\bar{3} \\ PR(B) &= 0.15 + 0.85(0.1 + 0.2) = 0.405 \\ PR(C) &= 0.15 + 0.85(0.1 + 0.1 + 0.0\bar{6}) = 0.37\bar{6} \\ PR(D) &= 0.15 + 0.85(0.0\bar{6} + 0.0\bar{6}) = 0.26\bar{3} \\ PR(E) &= 0.15 + 0.85(0.0\bar{6}) = 0.2\bar{6} \end{aligned}$$

Comment peut-on toutefois prouver la convergence de telles valeurs ? On peut dire qu'une fonction converge lorsque sa valeur commence à être très proche de sa précédente valeur. Ici, la convergence dépend principalement du damping factor. En effet, si la probabilité qu'un surfer continue de parcourir le graphe en utilisant des liens est basse, alors l'aléatoire aura beaucoup d'effet et les valeurs convergeront plus lentement. À l'inverse si celle-ci est haute, l'aléatoire aura moins d'effet et les valeurs convergeront plus rapidement. Là est tout l'intérêt de l'algorithme PR.

2.4.2 Algorithme implémenté

Pour mieux se rendre compte du fonctionnement de l'algorithme PR, nous allons ici proposer une implémentation de celui-ci en Python, langage choisi pour sa facilité de compréhension et d'utilisation. Il faut dans un premier temps définir la structure de données la plus adaptée au problème.

En Python, il existe ce qui s'appelle des dictionnaires comportant un certain nombre de couples clé-valeur. Dans notre cas, chaque clé est un noeud et chaque valeur un dictionnaire comportant les liens vers lesquels le noeud amène. Le PR actuel de la page est aussi rajouté. L'initialisation se déroule en plusieurs étapes :

1. Dans un premier temps, on compte le nombre total de noeuds qui nous sera utile dans la suite de l'algorithme. Il suffit de parcourir le dictionnaire et de compter le nombre de clés :

```
n = len(list(simple_web.keys()))
```

2. Ensuite on initialise le PR à 0 pour chaque noeud, puis une fois que l'on sait combien de noeuds le graphe possède au total, la valeur change à $1/n$.
3. Pour finir, on fixe le damping factor d à la valeur 0.85.

L'algorithme peut ensuite être implémenté facilement :

Listing 1 Implémentation de l'algorithme PageRank original

```

1 # PageRank algorithm
2 NB_ITERATIONS = 10
3 for i in range(NB_ITERATIONS):
4     for k,v in simple_web.items():
5         pr = 0
6         # Run through all links that refer selected page
7         for ws,params in simple_web.items():
8             if k in params['links']:
9                 pr += params['pr'][i]/len(params['links'])
10            # New page rank calculation
11            v['pr'].append((1-d) + d*pr)

```

Afin d'approximer une valeur de PR la plus plausible, il faut effectuer un certain nombre d'itérations. Celui-ci varie en fonction de l'algorithme que l'on utilise. Il est plus grand lors de l'utilisation de l'algorithme original car n est plus grand que 1. Les valeurs prennent alors plus de temps à converger.

À chaque itération, on parcourt notre dictionnaire. On parcourt ensuite chaque lien entrant de chaque noeud et on somme les valeurs du PR de chacun. On finit par enregistrer la valeur finale du PR pour chacun des noeuds du graphe.

La somme effectuée à chaque itération parcourt tous les noeuds entrants p_i dans le noeud actuel p_j :

$$\sum_{p_i \in M(p_j)} \frac{PR(p_i)}{L(p_i)}$$

2.4.3 Explications et résultats

Nous avons effectué des tests de comparaison entre les deux algorithmes en fixant un total de 50 itérations et nous avons obtenu des résultats intéressants nous montrant une évolution des valeurs des rangs de chaque site. Ils sont affichés sur la page suivante.

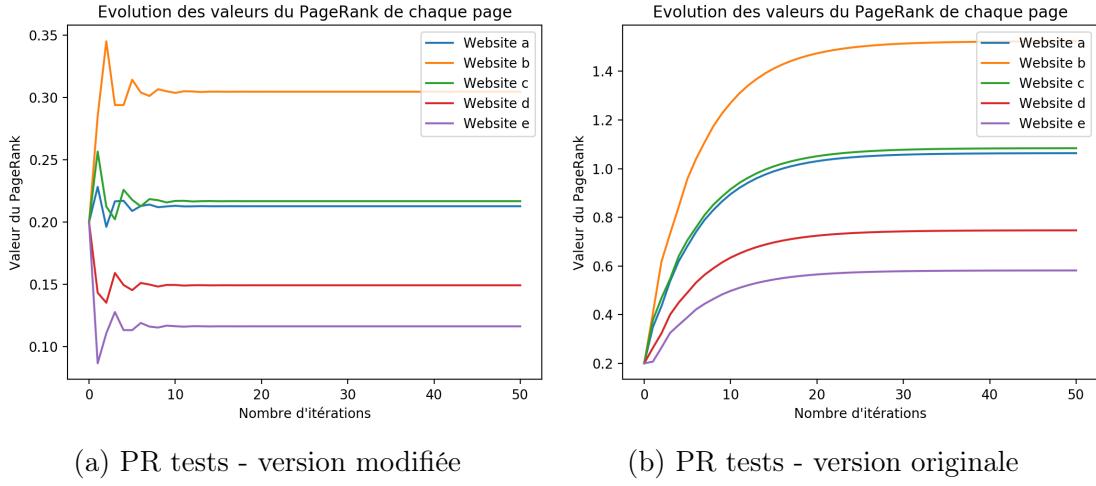


FIGURE 2.2: Tests des PR

On distingue tout de suite la différence entre les rangs. *b* est en effet le site référencé par le plus grand nombre de sites les mieux notés et ne possède que quelques liens sortants vers les sites les moins bien notés. Au contraire, *e* est le site le moins bien référencé car il n'est référencé que par un seul site et il en référence lui-même beaucoup trop.

La dissemblance fondamentale recensée entre ces deux algorithmes ne se trouve pas dans le classement des sites - il le même dans cet exemple - mais dans la valeur du PR elle-même. Les courbes sur ces deux graphiques ne varient pas de la même manière, celles de l'algorithme original sont en constante croissance et convergent donc plus lentement. Pour l'algorithme d'origine, une cinquantaine d'itérations est requise pour obtenir à terme un score représentatif.

Nous allons maintenant nous concentrer sur un ensemble de tests plus approfondis afin d'en apprendre plus sur le fonctionnement de cet algorithme et éventuellement en arriver à des valeurs convergeantes.

2.5 Robustesse des paramètres de l'algorithme

L'idée derrière cette dernière section est de créer un ensemble de tests en faisant varier des paramètres afin de se rendre compte du changement de valeur de PR. Les paramètres que nous allons faire varier sont le damping factor, le nombre d'itérations de l'algorithme et le nombre de liens.

Pour réaliser ces tests, nous allons utiliser un graphe légèrement plus complexe que le précédent, possédant 10 noeuds et 29 relations :

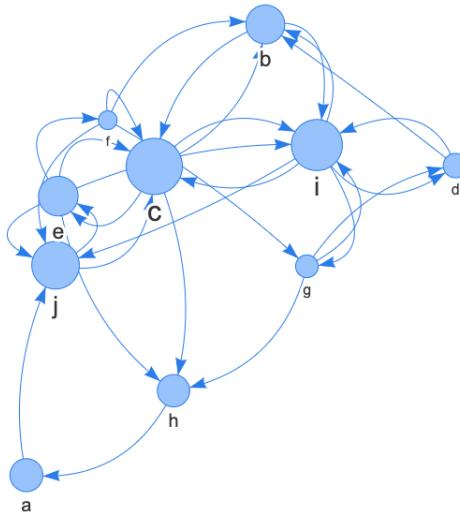
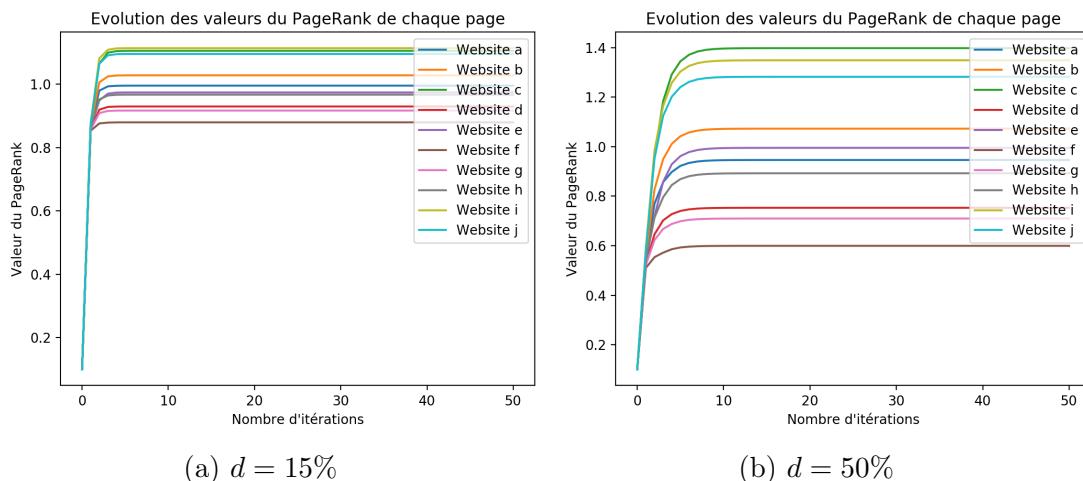


FIGURE 2.3: Graphe de test

2.5.1 Variation du damping factor

Rappelons d'abord le rôle du damping factor. Il introduit en général le concept de visite d'un lien aléatoire dans un graphe. Fixé en général à $d = 0.85$, il indique - lors de la visite de chaque noeud - le pourcentage de chance d'un surfer de continuer à naviguer dans le réseau. **Plus le facteur est élevé, moins il a de chance de s'arrêter.**

Nous avons pour ces tests choisi comme différentes valeurs de d : 0.15, 0.5, 0.85 et 0.99. Les résultats que nous avons obtenus sont affichés sur les 4 images suivantes :



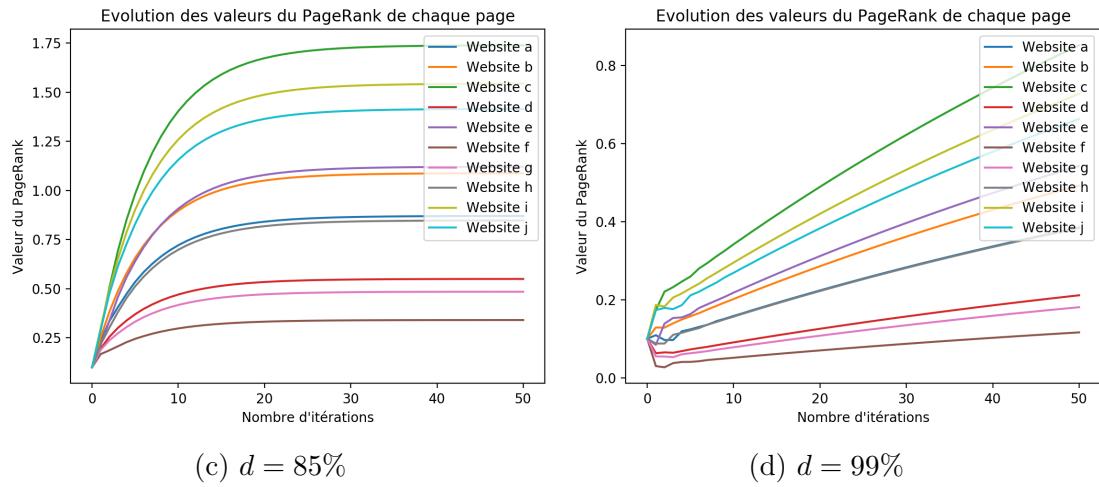


FIGURE 2.4: Variation du damping factor

Ces quatre images nous permettent de bien nous rendre compte de l'importance de ce facteur dans la formule du PR. Une hypothèse à émettre serait **la rapidité de convergence lors de la diminution de la valeur du damping factor**. En effet, plus le facteur augmente, plus le nombre d'itérations doit être grand pour que les résultats convergent.

Lorsque $d = 50\%$, il faut environ cinq itérations pour qu'il y ait convergence, alors que pour le facteur normal à 85% , une trentaine d'itérations fera le nécessaire. On comprend tout de suite que si un surfer n'a que 50% de chance de cliquer sur un lien, le graphe n'est pas d'une grande utilité et l'aléatoire reprendra le dessus.

2.5.2 Variation du nombre d'itérations

Dans les précédents tests, on a remarqué que pour un damping factor à 99% , une cinquantaine d'itérations n'était pas suffisante pour avoir une convergence des PR. C'est pourquoi il nous faut également prendre en considération le nombre d'itérations comme élément ayant une influence sur les rangs de pages Web.

Nous avons ainsi décidé de reproduire ces tests avec 100 puis avec 500 itérations, dont les résultats sont affichés en figure 2.5 sur la page suivante.

Les résultats commencent à être concluants à partir de 500 itérations, bien que la convergence ne soit pas encore assez flagrante pour les sites avec un rang élevé. Quoi qu'il en soit, le damping factor idéal étant fixé à 85% nous permet de n'effectuer qu'une trentaine d'itérations pour avoir un résultat significatif.

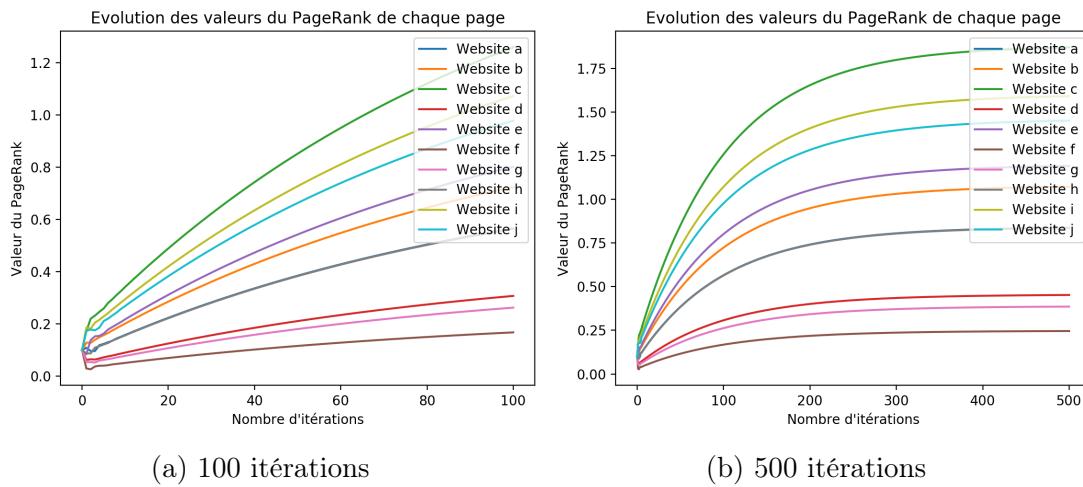


FIGURE 2.5: Variation du nombre d’itérations

2.5.3 Variation du nombre de liens

Avant de rentrer dans les détails, commençons par nous rappeler des précédents rangs (avec un damping factor de 85%) :

Noeud	Rang	Noeud	Rang
a	0.870160	f	0.340512
b	1.087720	g	0.484676
c	1.738704	h	0.847287
d	0.549640	i	1.543135
e	1.120713	j	1.414791

TABLE 2.1: Tableau des PageRanks

L'idée derrière cette dernière série de tests est d'observer les changements dans les rangs lors de suppression ou d'ajout de liens aléatoires.

Nous avons commencé par créer quatre cas de figure différents. Dans les deux premiers, nous avons successivement supprimé tous les noeuds référençant c (le noeud ayant le PR le plus élevé) et ajouté un lien vers c pour les noeuds qui ne le référaient pas encore. Dans les deux derniers cas, nous avons reproduit le même processus en ayant fait en sorte que c référence tous les autres noeuds. La différence sera minime mais tout de même présente.

Les quatre images de la page suivante permettent de nous rendre compte des différentes situations que nous avons créées :

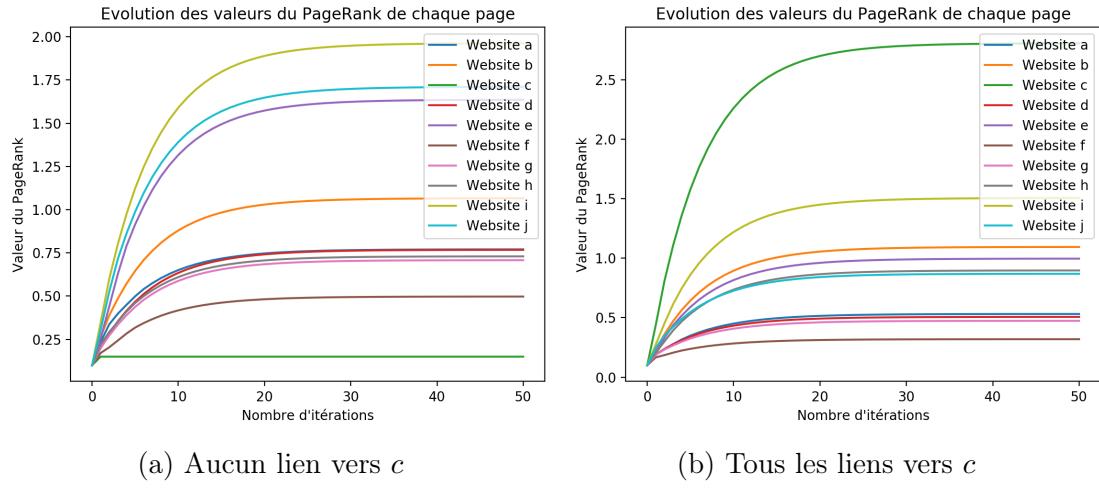


FIGURE 2.6: Variation du nombre de liens vers c ; il ne référence que certains liens du réseau

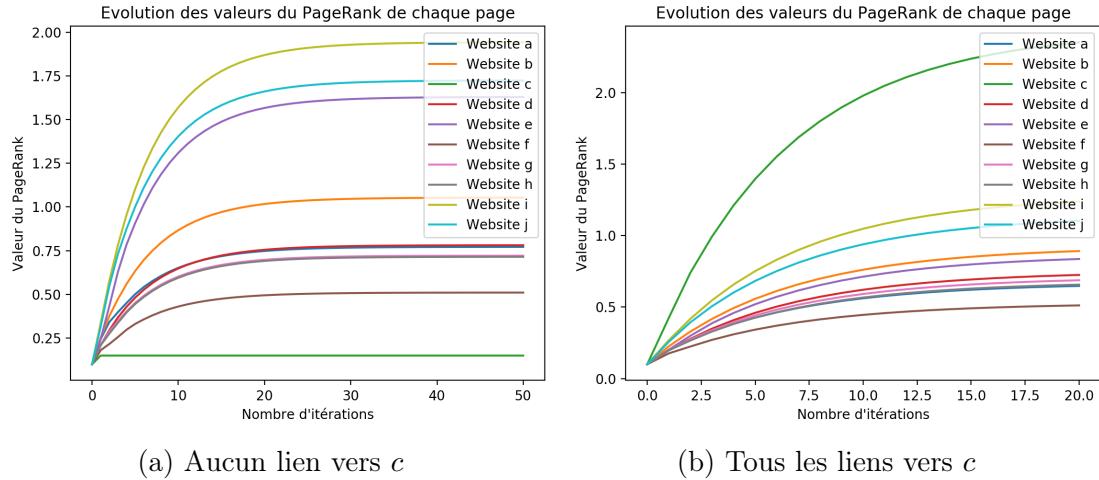


FIGURE 2.7: Variation du nombre de liens vers c ; il référence tous les noeuds du réseau

Le référencement est important car dans ces cas-là, le PR peut varier de 0.15 à 2.8.

On considère maintenant deux autres cas de figure différents :

1. Le premier cas consiste à changer un lien aléatoire par test :

Noeuds	PageRanks		
	original	$(g, d) \rightarrow (g, a)$	$(i, g) \rightarrow (i, e)$
a	0.870	1.010	0.861
b	1.088	1.012	1.067
c	1.739	1.750	1.796
d	0.550	0.397	0.469
e	1.121	1.172	1.403
f	0.341	0.349	0.389
g	0.485	0.471	0.233
h	0.847	0.855	0.836
i	1.543	1.453	1.489
j	1.415	1.529	1.456

TABLE 2.2: Variations des liens du réseau

Nous avons décidé de prendre deux exemples différents, l'un travaillant sur des noeuds avec un PR faible et l'autre sur des noeuds avec un PR plus élevé. On remarque avec ces changements qu'ils ont peu d'influence sur les rangs des sites déjà bien notés comme le noeud c qui conserve sa première place. Si l'on prend en considération les noeuds concernés par les changements (d et a pour le premier test et g et e pour le second), les variations sont bien plus évidentes.

Dans le premier cas, on remarque plus de changements dans les noeuds i et j qui sont indirectement liés aux noeuds dont les PR changent. Dans le second cas, les noeuds e et g voient leurs scores changer énormément car le noeud i dont les liens changent possède un PR plus élevé et a donc plus d'influence.

2. Notre seconde série de tests consiste à rajouter ou à supprimer des liens aléatoires du réseau. On va appliquer cela à des rangs faibles, puis élevés. L'idée de ce dernier test est de noter l'influence que peut avoir un bon PR sur d'autres noeuds :

Noeuds	PageRanks				
	original	(+) $f \rightarrow i$	(+) $j \rightarrow a$	(-) $d \rightarrow i$	(-) $i \rightarrow d$
a	0.870	0.865	1.292	0.872	0.905
b	1.088	1.082	1.006	1.277	1.051
c	1.739	1.724	1.536	1.783	1.821
d	0.550	0.557	0.521	0.517	0.302
e	1.121	1.113	0.964	1.120	1.177
f	0.341	0.339	0.314	0.340	0.350
g	0.485	0.479	0.458	0.459	0.535
h	0.847	0.841	0.770	0.849	0.889
i	1.543	1.595	1.419	1.392	1.463
j	1.415	1.403	1.719	1.390	1.505

TABLE 2.3: Ajout et suppression de liens dans le réseau

Ajout de liens De la même manière que dans le premier test, les rangs ne varient pas beaucoup si les noeuds concernés ne sont pas reliés directement au noeud principal. Si l'on rajoute un lien vers i venant d'un noeud n'ayant pas un rang élevé, les différences sont minimes. En revanche, si un noeud avec un score élevé référence un noeud plus faible, les deux rangs augmentent. Dans notre cas, j possède cette fois le meilleur rang.

Suppression de liens d étant un noeud n'ayant pas beaucoup d'influence sur les autres noeuds du réseau étant donné son score relativement faible, on a que très peu de changements dans les rangs vis-à-vis du premier cas.

Le second est en revanche beaucoup plus intéressant étant donné le score élevé du noeud i . Ayant perdu son lien vers d qui le référençait, le noeud i voit son score baisser également.

Conclusion Nous avons maintenant une idée plus claire du fonctionnement l'algorithme PR. Un noeud ayant un score **faible** aura une **influence minime sur le reste du réseau** tandis qu'un noeud avec un score **élevé** participera grandement au **changement de rangs des autres noeuds**. Voici le principe fondamental de cette théorie.

Le prochain chapitre concerne l'étude d'autres algorithmes ainsi leurs différences principales permettant de les distinguer de l'algorithme PR.

Chapitre 3

Alternatives au PageRank

LE MONDE du Web a pris ces dernières années une importance grandissante. Pour gérer cet immense réseau correctement, les algorithmes de classement de pages Web ont fait leurs preuves. Ils possèdent tous quelques avantages et bien entendu des inconvénients. Le PR de Google est un algorithme très pratique, efficace quant au classement de pages Web et relativement facile à comprendre et implémenter. Il présente toutefois quelques défauts.

L'idée de ce chapitre est de citer quelques alternatives au PR en présentant les avantages de celles-ci propices à leur utilisation :

- Tout d'abord, l'algorithme PR est fortement basé sur les liens entrants dans un noeud mais pas sur ceux qui en sortent. Une alternative pour contourner ce problème a été mise en place dans l'algorithme **CheiRank** (abrégé CR) [And], premier étudié dans ce chapitre.
- Ensuite, le PR ne se basant que sur un seul type d'importance, il faut trouver un moyen de prendre en compte plus de critères. L'algorithme le plus connu est le **HITS** [Les12] développé par Jon Kleinberg. Il est fondé sur deux entités primordiales (hubs et autorités) qui décrivent les noeuds d'un graphe et ne se basent que sur les liens présents entre ceux-ci. C'est le précurseur de l'algorithme PR qui fera l'objet de la deuxième section de ce chapitre.
- Pour finir, nous décrirons un algorithme moins connu visant à résoudre un problème très fréquent dans le monde du Web de nos jours. Il s'agit des changements de score en raison du spam. Des personnes appelées "spammers" vont créer des pages vides et des liens allant de ces pages vers leur site afin qu'il soit mieux référencé. Une solution à ce problème mise au point par des chercheurs de l'université de Stanford s'appelle l'algorithme **TrustRank** (abrégé TR) [Ton]. Chaque page sera dotée d'un "degré de confiance" sous la forme d'un score de 0 à 10. Ce sera l'objet principal de la dernière section de ce chapitre.

3.1 CheiRank : Plus de communication

L'algorithme CR est un concept présenté par trois chercheurs russes visant à instaurer un modèle d'analyse de popularité d'une page web basé à la fois sur les liens entrants et les liens sortants de cette même page.

Selon eux, le modèle du PR qui ne tient compte que de l'aspect quantitatif et qualitatif des liens pointant sur une page n'est pas suffisamment représentatif. Le PR évaluant principalement la popularité d'un article se doit donc d'être complété par un autre algorithme prenant plus en compte la **communicativité des noeuds**.

3.1.1 Fonctionnement

Fondamentalement, la définition du CR peut se résumer simplement comme étant l'inverse de celle du PR. L'algorithme se focalise donc plus sur les liens sortants d'un noeud.

On peut exprimer l'équation du PR sous une autre forme :

$$\mathbf{G} = \alpha \mathbf{S} + (1 - \alpha)ee^T/n \quad (3.1)$$

\mathbf{G} est la matrice des PR, α est le damping factor, \mathbf{S} est la matrice de transitions stochastique, n reste toujours le nombre de liens, e est le vecteur PR du noeud actuel et e^T sa transposée. Si \mathbf{S} est notre matrice de transitions, l'idée est de trouver un vecteur propre \mathbf{x} de valeur propre 1 pour la matrice \mathbf{S} de manière à ce que l'on ait : $\mathbf{Sx} = \lambda x$, $x \neq 0$.

L'équation du CR sera la même avec comme seule différence le contenu des matrices (les liens étant tous inversés) :

$$\mathbf{G}^* = \alpha \mathbf{S}^* + (1 - \alpha)ee^T/n \quad (3.2)$$

La matrice \mathbf{S}^* n'est autre que la transposée de la matrice \mathbf{S} .

Pour pouvoir implémenter cet algorithme, nous allons fortement nous baser sur celui du PR. En revanche, afin de pouvoir appliquer l'algorithme de manière plus simple et pour qu'il soit plus performant, on va changer notre structure de données et utiliser des matrices de la librairie **numpy** au lieu d'un dictionnaire.¹

La fonction `CheiRank()` que l'on a construite prend en paramètres la matrice d'adjacence sur laquelle se baser, le damping factor ainsi que le nombre d'itérations. Un vecteur sert à représenter le rang de chaque noeud et la matrice d'adjacence - permettant de déterminer les voisins de chaque noeud - nous est utile pour calculer le nouveau rang à chaque itération.

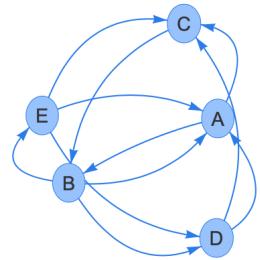
1. En ayant effectué un test sur un réseau à 5 noeuds avec 50 itérations, numpy s'est avéré 600 fois plus rapide.

Etant donné que l'on travaille sur les noeuds sortants, il nous suffit d'inverser tous les liens dans le graphe. Mathématiquement parlant, cela résulte en la transposée de notre matrice d'adjacence.

3.1.2 Tests ciblés sur un réseau simple

Nous avons effectué des tests dans un premier temps sur le graphe que nous avions créé manuellement dans le chapitre précédent puis sur un graphe plus complexe créé de manière aléatoire.

Après transposition, le graphe affiché nous donne la matrice suivante :



$$\mathbf{S}^* = \mathbf{S}^T = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Ci-dessous est affiché le tableau représentant les tests effectués pour les deux algorithmes PR et CR :

Noeuds	20 itérations		50 itérations	
	PageRank	CheiRank	PageRank	CheiRank
a	1.031	0.999	1.064	1.032
b	1.474	1.490	1.523	1.539
c	1.051	0.779	1.084	0.804
d	0.724	0.651	0.746	0.670
e	0.566	0.926	0.582	0.955

TABLE 3.1: Comparaisons des algorithmes PageRank et CheiRank

3.1.3 Tests plus approfondis

Pour tester un algorithme et éventuellement effectuer des comparaisons avec un autre, il faut effectuer un grand nombre de tests. Dans ce cas-là, un graphe de cinq noeuds n'est pas suffisant pour se donner une bonne idée des différences notables entre le CR et le PR. C'est pourquoi nous avons décidé de réaliser une série de tests supplémentaires en générant des graphes aléatoires plus grands et en établissant quelques statistiques liées à nos résultats afin de pouvoir établir une conclusion plus rigoureuse.

Génération d'une matrice aléatoire La seule condition que l'on doit instaurer lors de la création de notre graphe est qu'aucun noeud ne possède un lien sur lui-même. Mathématiquement parlant, cela veut dire qu'il ne faut pas qu'il y ait de 1 sur la diagonale descendante de \mathbf{S} .

Pour ce faire, on réalise plusieurs étapes :

1. On crée d'abord une matrice carrée M de taille $n \times n$ comportant des 0 et des 1 :

```
a = np.random.randint(2, size=(n, n))
```

2. Ensuite on déclare une matrice d'identité \mathcal{I} d'ordre n à laquelle on va permute tous ses éléments. Ainsi, on obtiendra une matrice diagonale dont tous les éléments ne se trouvant pas sur la diagonale descendante sont nuls.

```
inverse = 1-np.eye(n, dtype=np.int)
```

3. Pour finir, il faut effectuer un ET logique bit à bit afin que les éléments de la diagonale descendante restent nuls. On reproduit l'opération pour obtenir nos 2 matrices :

```
p_mat = np.bitwise_and(a, inverse)
c_mat = np.bitwise_and(np.transpose(a), inverse)
```

Statistiques Il existe en statistique plusieurs paramètres qui nous permettent de mieux étudier des résultats.

- La valeur **moyenne** notée \bar{x} est le calcul le plus évident à effectuer lors de l'obtention d'un grand nombre de résultats. Il suffit d'ajouter tous les résultats obtenus puis de les diviser par le nombre d'additions effectuées. Symboliquement,

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (3.3)$$

- La **variance** notée $VAR(X)$ ou σ^2 mesure l'écart de valeurs d'une distribution de probabilités. Elle représente mathématiquement la moyenne des carrés des écarts à la moyenne, autrement dit :

$$VAR(X) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \quad (3.4)$$

Si on a une distribution constante, la variance est nulle mais n'est en aucun cas négative.

- L'**écart-type** noté σ correspond au taux d'erreur recensé le long d'une distribution de probabilités. Il se définit plus souvent comme la moyenne quadratique des écarts par rapport à la moyenne et correspond mathématiquement à la racine carrée de la variance. On écrit :

$$\sigma = \sqrt{VAR(X)} \quad (3.5)$$

Tout comme la variance, l'écart type n'est jamais négatif mais peut être nul si toutes les valeurs sur lesquelles il se base sont constantes.

Ces trois paramètres fondamentaux en statistique vont nous être très utile pour distinguer les principales différences entre nos deux algorithmes.

Discussion des résultats Nous avons ici légèrement modifié le code et rajouté en partie les calculs de statistiques (sans la moyenne car nous savons que : $\bar{x}_{PR} = 1$). Nous avons fait en sorte d'afficher un graphique à chaque génération de réseau afin de pouvoir clairement distinguer la variance et l'écart-type pour chacun. Voici deux exemples d'affichage avec des matrices de taille différente :

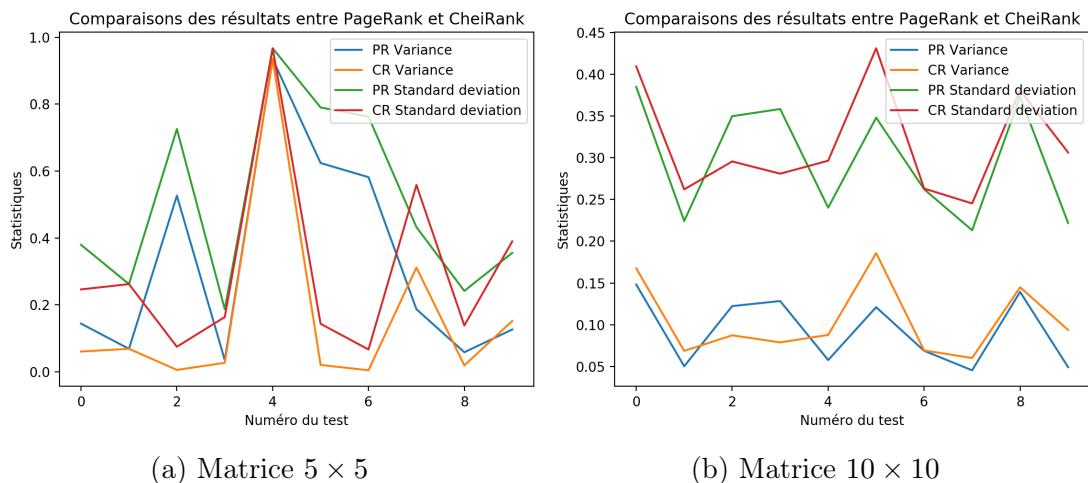


FIGURE 3.1: Statistiques de comparaison entre le PR et le CR

Plusieurs observations sont à relever dans les deux graphes ci-dessus :

1. D'abord, on remarque que plus le graphe est grand, plus les différences entre PR et CR sont moindres. Cela est dû au fait que les réseaux plus grands minimisent la différence lorsque tous les liens sont inversés. De ce fait, les scores varient moins et la variance et l'écart-type sont donc plus faibles.

2. Ensuite, il est intéressant de reconstruire les matrices du graphe 4 de la figure 3.1a, celui possédant des statistiques quasiment identiques :

$$\mathbf{S} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \quad \mathbf{S}^* = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Les troisièmes et quatrièmes noeuds du réseau \mathbf{S} que l'on appellera c et d possèdent les meilleurs rangs. En effet, c est référencé par tous les autres noeuds du réseau puis d étant référencé par c , son score est influencé. Le premier noeud a référence tous les autres sites du réseau et possède donc un score relativement faible.

Si l'on inverse tous les liens du réseau, le noeud a est cette fois référencé par tous les autres liens et possède donc le même score que c dans le graphe original. Le deuxième noeud b étant référencé par a voit son score augmenter rapidement.

Les noeuds qui possédaient un très bon PR dans le graphe original se voient avoir le pire dans le graphe inversé.

3.2 HITS : Plus de critères

L'algorithme HITS a été inventé plus ou moins en même temps que l'algorithme PR par Jon Kleinberg en ayant pour but de classer des sites internet selon deux critères fondamentaux : les **autorités** et les **hubs**. Fondamentalement,

- Une autorité est une page référencée par un grand nombre de hubs. Ce référencement est basé sur des informations et des critères fiables.
- Un hub est une page qui référence un grand nombre de pages autoritaires.

Chaque page possède donc deux scores : son autorité, c'est-à-dire l'estimation de la valeur du contenu de la page et sa valeur de hub, autrement dit l'estimation de la valeur de ses liens vers chaque autre page.

3.2.1 Fonctionnement

L'algorithme démarre avec deux vecteurs. Chaque vecteur contient les valeurs respectives des autorités et hubs de chaque page. Au départ, chacune d'entre elle est initialisée à 1.

Tout comme le PR, l'algorithme HITS est itératif c'est-à-dire que les scores de chaque page changent à chaque parcours du graphe. Ils finissent à la longue par converger vers une certaine valeur.

Pour donner une définition plus mathématique de ce que représentent les autorités et les hubs, on peut symboliser le tout par deux équations fondamentales :

$$\begin{aligned} a_i &= \sum_{j \in \mathcal{M}(\text{in})} h_j, \\ h_i &= \sum_{j \in \mathcal{M}(\text{out})} a_j \end{aligned} \tag{3.6}$$

Les autorités vont sommer la valeur "hub" h_j de chacun des liens entrants dans le lien actuel tandis que les hubs vont sommer la valeur "autoritaire" a_j de tous les liens sortants du lien actuel. Dans les équations ci-dessus, $\mathcal{M}(\text{in})$ et $\mathcal{M}(\text{out})$ sont respectivement l'ensemble des liens entrants et sortants du noeud actuel.

Voici à quoi ressemble le pseudo-code de l'algorithme HITS :

Algorithm 1 Pseudo code de l'algorithme HITS

```

1: procedure HITS_ALGO
2:   All weights initialized to 1
3:   loop until convergence :
4:     for every hub  $i \in \mathcal{H}$ 
5:        $h_i = \sum_{j \in \mathcal{M}(\text{out})} a_j$ 
6:       for every authority  $i \in \mathcal{A}$ 
7:          $a_i = \sum_{j \in \mathcal{M}(\text{in})} h_j$ 
8:       Normalize

```

Pour calculer les nouvelles autorités et hubs on effectue un simple produit scalaire entre le vecteur comprenant nos dernières valeurs et les éléments de la matrice (parcours ligne par ligne pour les hubs et colonne par colonne pour les autorités). Une fois les valeurs calculées, il suffit de normaliser celles-ci c'est-à-dire diviser ces valeurs par la somme totale des éléments de chaque nouveau vecteur.

Au bout d'un certain nombre d'itérations, les valeurs convergent éventuellement vers une valeur entre 0 et 1.

3.2.2 Comparaison des résultats

Nous avons comme pour les autres algorithmes effectué un test avec comme graphe celui affiché en figure 2.1. Ce test nous permet de mieux comparer l'utilité des deux algorithmes les plus connus du Web.

Les résultats obtenus peuvent être observés en figure 3.2 sur la page suivante.

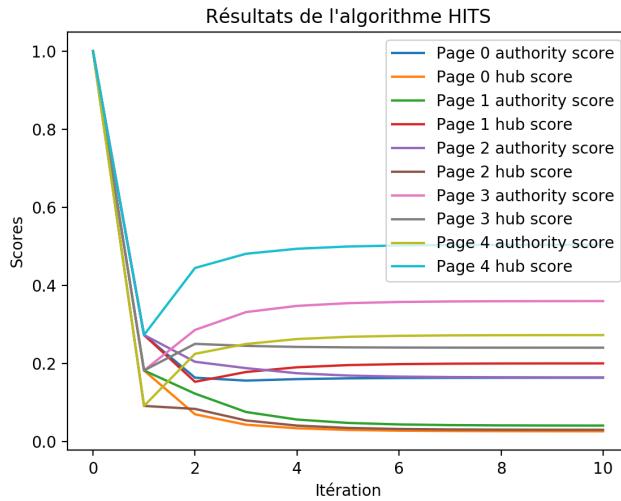


FIGURE 3.2: Premier test sur l'algorithme HITS

On remarque que le noeud e est le plus autoritaire, c'est en effet celui qui référence le plus grand nombre de hubs. Le noeud d est en revanche celui possédant le meilleur score de hub car il est référencé en majeure partie par e qui est très autoritaire.

Les résultats obtenus sont assez différents que ceux du PR car d et e s'avèrent être les noeuds les plus importants du réseau. Or, lors des résultats obtenus avec l'algorithme PR, c'était le noeud b qui était le plus important.

3.2.3 Avantages et inconvénients du HITS

HITS possède quelques avantages notables qui le distinguent bien du PR mais présente également quelques inconvénients, notamment sur des questions de performance. L'idée de cette section est de lister ces points puis de les comparer avec les plus et moins de l'algorithme PR.

Points positifs

1. La force du HITS est la possibilité de classer les pages en se basant sur le sujet de la requête et cela donne forcément des autorités et des hubs plus pertinents, ce qui n'est pas le cas du PR. L'algorithme se base sur deux critères de recherche.
2. Un autre avantage majeur du HITS est qu'il est facile à implémenter et à comprendre. Le concept est plus abordable que celui du PR.

Points négatifs

1. L'un des problèmes du HITS est la possibilité toujours existante de tomber sur une page autoritaire ou hub qui n'est pas la page que l'on souhaite rechercher. Le PR ne présente pas ce problème étant donné que les recherches ne sont pas faites selon un certain critère.
2. L'autre inconvénient recensé est la complexité de l'algorithme. Elle est en effet de l'ordre de $\mathcal{O}(n^2)$ tandis que celle du PR est de $\mathcal{O}(n)$ ce qui est indéniable.

Au final le but n'étant pas de déterminer quel est l'algorithme le plus pertinent, ceux précédemment étudiés sont tous deux utilisables et possèdent chacun leurs avantages et leurs inconvénients dans leur propre domaine.

3.3 Une solution au problème du spam ?

Nous avons choisi de ne pas implémenter le dernier algorithme étudié dans ce chapitre pour des raisons de temps et de complexité. Nous avons toutefois recensé quelques notions importantes visant à compléter les trois autres algorithmes précédemment étudiés.

3.3.1 Qu'est-ce que le TrustRank ?

Le problème majeur avec le référencement de sites sur internet aujourd'hui est qu'hormis les calculs effectués pour obtenir un rang en se basant sur les liens entrants, sortants et le référencement des noeuds voisins, il manque un critère d'une importance primordiale : **la notion de confiance**.

L'algorithme TR a été créé pour parvenir à résoudre un tel problème survenant typiquement lorsque des individus appelés "spammers" créent une collection de "pages de boost" dans le but d'améliorer leur rang et ainsi diminuer le score d'autres pages précédemment de très bonne qualité. [GGMP04]

Il vise à donner en plus d'un rang classique, une note de confiance se basant sur les éléments suivants :

- Le nombre de pages du site (le nombre total de noeuds du réseau)
- L'audience mesurée quotidiennement (la fréquentation individuelle de chacun de ces noeuds)
- Les indications fournies par le nom de domaine (les fréquentations peuvent varier en fonction du nom de domaine du site)
- L'intervention humaine (les changements régulièrement apportés au site)
- Quelques autres critères encore mal connus

3.3.2 Comment fonctionne-t-il ?

Il est important d'effectuer au préalable une liste exhaustive de sites de qualité afin que l'algorithme TR puisse les analyser et ainsi déterminer la qualité des pages de l'Internet tout entier. Cette liste est appelée "liste de référence". Elle va en général contenir un grand nombre de sites de qualités tels que ceux appartenant au gouvernement ou des sites éducatifs.

L'algorithme TR fournit une note supplémentaire à chaque site sous la forme d'un nombre entre 0 et 10. Si le nombre est proche de 0, il se peut que le site soit un spam et donc pas un site de confiance. En revanche, plus le nombre se rapproche de 10, plus sa référence est valide et le site de confiance.

Il faut voir cet algorithme comme une pyramide de niveaux où le niveau 1 est celui réservé aux sites les plus fidèles. Tous les sites référencés par ceux de niveau 1 seront appelés sites de niveau 2, et auront un degré de confiance toujours correct mais cependant inférieur aux sites de niveau 1. Ce processus se reproduit sur plusieurs paliers. Les sites se trouvant au plus bas de la pyramide ne seront certainement pas des sources sûres.

3.3.3 Intégration dans le Web

La pertinence des résultats dans les moteurs de recherche représente un facteur non négligeable par autrui. Avec le temps, de nouveaux critères sont découverts et le fonctionnement des moteurs de recherche est ainsi en constante amélioration.

Le TR fait partie d'un des nombreux critères de recherche que Google utilise pour classer ses résultats et comprendre son fonctionnement nous permet d'avoir une meilleure idée des types de site sur lesquels on va tomber.

Deuxième partie

Manipulation des données

« *La théorie, c'est quand on sait tout et que rien ne fonctionne.
La pratique, c'est quand tout fonctionne et que personne ne sait pourquoi.* »

– Albert Einstein

Chapitre 4

Familiarisation avec l'outil Neo4j

MINTENANT que nous avons établi tous les points théoriques importants dans le cadre de ce travail, la question est de trouver un outil capable de traiter correctement des graphes et de les visualiser facilement. L'outil qui nous a paru le plus approprié s'appelle Neo4j.

Notre objectif est de présenter dans un premier temps toutes les spécificités que propose l'outil Neo4j de manière à mieux comprendre pourquoi il nous est utile dans le cadre de ce travail puis dans un second temps, importer des données externes réelles que nous utiliserons plus tard afin de mieux comprendre d'une part à quoi ressemble vraiment le Web puis d'autre part les effets de l'algorithme PR. Il sera également question dans ce chapitre d'établir une liste exhaustive de contraintes à respecter si l'on souhaite importer des données de la manière la plus simple et efficace possible.

4.1 Qu'est-ce que Neo4j ?

Neo4j est un logiciel open source permettant de stocker et de requêter des données en tant que noeuds et relations. Chaque élément d'une base de données Neo4j est stocké sous la forme d'un noeud, d'un lien ou d'un attribut. Chaque noeud ou lien peut avoir autant d'attributs que l'on souhaite. C'est à l'aide du langage **Cypher** que l'on va pouvoir créer, manipuler et représenter des graphes complets. Un paragraphe est prévu à cet effet plus tard dans cette thèse.

Il existe deux possibilités pour déployer l'environnement Neo4j :

1. Neo4j **Desktop**
2. Neo4j **Browser**

La version de bureau permet de gérer directement tous ses projets et est plus complète et plus accessible mais nous utiliserons toutefois la version browser qui pour un projet comme le notre nous permet d'en faire autant et plus rapidement.

4.2 Connexion à Neo4j

Le schéma habituel à suivre pour se connecter à Neo4j comporte les opérations suivantes :

1. Le client demande l'URL de connexion Bolt (voir section 4.3) par le port HTTP 7474 au serveur Neo4j sous la forme d'une requête GET.
2. Le client tente cette fois une connexion au serveur Neo4j par le biais d'une connexion Bolt en passant par le port WS 7687 sans information d'authentification. À ce moment-là, si les informations d'identification sont trouvées, la connexion est autorisée. S'il ne trouve au contraire aucun identifiant dans l'entrepôt de stockage local du browser, le serveur répond par une demande d'authentification.
3. Une fois la connexion réussie et le problème des identifiants illucidé, le client communique avec le serveur par des websockets sur le port Bolt 7687.

4.3 Protocole Bolt

Neo4j Browser se base sur une connexion TCP entre le serveur de base de données et lui-même. Le protocole utilisé s'appelle le **protocole Bolt**. Ce protocole étant basé sur l'envoi d'instructions, il permet au client d'envoyer des messages composés de requêtes avec des paramètres et attend la réponse du serveur généralement composée d'un accusé de réception sous forme de message d'information et d'une série de résultats optionnels. Il faut cependant que l'envoi de websockets soit autorisé sur le réseau local auquel cas un message d'erreur sera affiché dans le browser. Voici un exemple classique de fonctionnement de ce protocole :

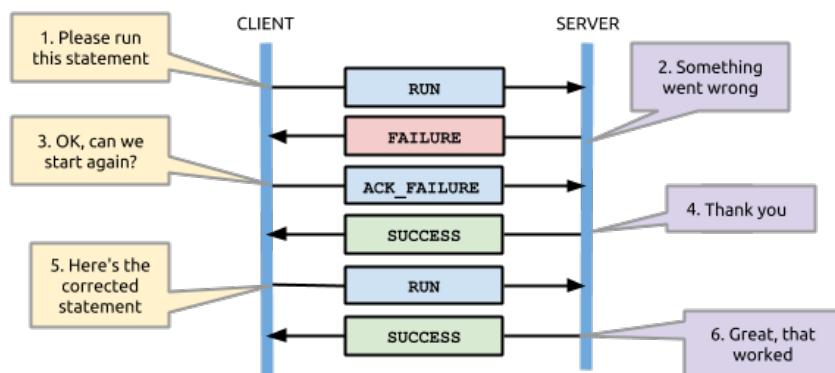


FIGURE 4.1: Fonctionnement du protocole Bolt, Source : NeoTechnology Inc.

4.4 Prise en compte du fichier de configurations

L'installation de Neo4j vient avec un fichier de configurations `neo4j.conf` représentant la source principale des configurations et réglages de Neo4j. La plupart de ces configurations s'appliquent sur l'outil lui-même mais il en existe d'autres qui permettent de rajouter des "plugins" donnant entre autres la possibilité d'utiliser des algorithmes de traitement de graphe, dont celui du PR.

Dans le fichier original, un grand nombre de lignes sont écrites en commençant par le symbole `#`. Cela représente un commentaire. Pour appliquer un quelconque changement, on peut soit enlever ce symbole des lignes que l'on souhaite utiliser soit en rajouter de nouvelles.

Décommenter la ligne suivante nous permettra d'utiliser tous les algorithmes nécessaires au traitement de graphe :

```
dbms.security.procedures.unrestricted=
    apoc.trigger.*,algo.* ,apoc.*
```

Ce fichier de configurations permet également de modifier la base de données utilisée actuellement. Pour notre ensemble de tests, nous créerons la base : `graph.db`.

4.5 Cypher

Cypher est un langage déclaratif très puissant permettant d'effectuer des requêtes orientées graphe. Il donne la possibilité contrairement à certains autres langages de se concentrer sur ce que l'on veut plutôt que la manière dont on le veut. C'est un langage simple à apprendre et comprendre, en constante évolution [FGG⁺¹⁷] et s'avère être très performant. En raison du fait qu'il soit construit de manière à être lisible par autrui, cela en fait un langage beaucoup plus accessible.

C'est également un langage fortement basé sur le SQL. L'action que doit produire le programmeur est la même c'est-à-dire écrire une requête pour retourner le résultat souhaité. Une requête Cypher va travailler sur un graphe et avoir un tableau de données - affiché à l'aide d'un ASCII art - comme valeur de retour. L'ensemble de fonctions dont se sert le langage est fortement inspiré du SQL. Les fonctions MATCH et WHERE sont les plus courantes. Voici une structure possible de requête :

Listing 2 Exemple de requête Cypher

```
1 MATCH (x:Airline)-[:DESTINATION]-(dest:'Geneva') RETURN dest;
```

Le résultat retourné par cette requête sera toutes les compagnies aériennes qui volent vers ou de Genève.

Cypher va nous permettre de créer le réseau que l'on souhaite avec les paramètres de notre choix. Dans un premier temps en guise d'exemple, on ne fixe qu'un seul paramètre à un noeud, son nom (sans compter son identifiant). En Cypher, les noeuds et liens se créent de la manière suivante :

Listing 3 Requête pour créer un noeud et un lien

```
1 CREATE (a:Website {id:0, name:"A"}); // Nodes
2 CREATE (a)-[:LINK]->(b); // Links
```

Lorsque l'on reproduit ce code pour chaque noeud et lien que l'on souhaite avoir, on peut exécuter une requête qui nous permet d'afficher notre graphe complet :

Listing 4 Requête pour afficher le contenu d'un graphe

```
1 MATCH (n) OPTIONAL MATCH (n)-[r]->(m) RETURN n,r,m;
```

Ce code nous produit la sortie suivante :

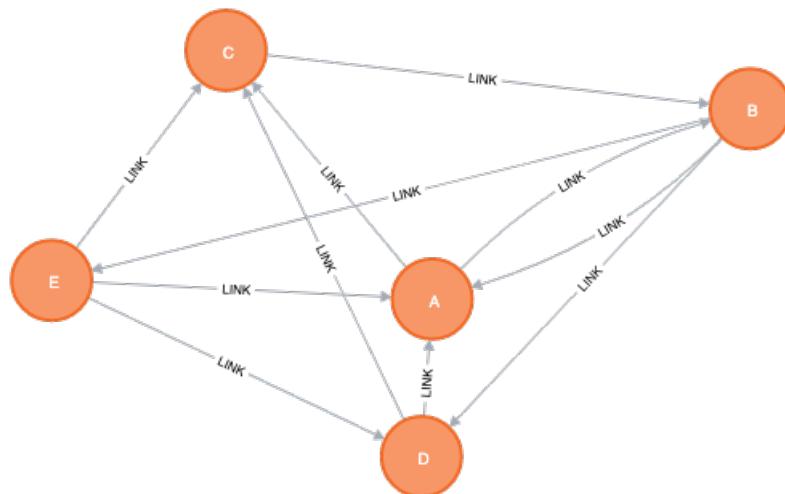


FIGURE 4.2: Premier graphe simple

4.6 Importation de données externes

L'objectif de cette section est d'importer des données externes réelles afin d'obtenir des réseaux plus complexes qui seront plus intéressants à étudier. Cette importation se déroule en plusieurs étapes qui feront l'objet des parties suivantes.

4.6.1 Récupération de données externes

La première étape est d'aller chercher sur internet un ensemble de données qui pourrait être utilisé dans le cadre de notre projet. L'université de Stanford propose sur son site plusieurs ensembles de données à importer dont quelques uns peuvent faire partie de notre choix idéal [LK14].

Le site propose plusieurs ensembles différents mais celui qui nous intéresse ne doit posséder qu'un seul type de noeud et de relation entre eux. Le graphe que nous avons choisi correspond à celui de l'université de Notre-Dame aux Etats-Unis. Il comporte au total 325'729 noeuds et 1'497'134 relations.

Les données sont mises à disposition sous la forme d'un fichier texte compressé une liste de toutes les relations entre les noeuds.

4.6.2 Génération de fichiers CSV

Il n'existe pas beaucoup de moyens d'importer des données externes dans Neo4j. Une possibilité serait d'en importer depuis un fichier JSON mais il faut passer par une librairie de procédures supplémentaires appelée : APOC. Elle contient une fonction `apoc.load_json` qui permet de récupérer des données depuis une URL et de les transformer en un ensemble de type "map" depuis lequel toutes les données pourront être lues et traduites par Cypher afin d'en créer un graphe. Ce n'est toutefois pas la solution la plus simple.

Il existe heureusement une fonction `LOAD CSV` qui nous permet de lire des données d'un fichier CSV et de directement les transcrire sous forme de graphe Neo4j. La seule chose à faire au préalable est de créer ces fichiers à partir des données que l'on a récupérées.

Dans le dossier d'imports de Neo4j, on effectue les étapes suivantes :

1. On ouvre le fichier texte et on remplace les tabulations par des virgules.
2. On modifie également l'entête de ce fichier de manière à avoir des noms pertinents.
3. On tape la commande suivante dans le terminal qui va nous créer un fichier comportant tous les noeuds du réseau :

```
$ seq 0 325729 >> NotreDame-nodes.csv
```

Une fois ces étapes effectuées, on peut désormais se concentrer sur l'import de nos données dans Neo4j à l'aide de Cypher.

4.6.3 Importation des données dans Neo4j

Pour commencer, on part du principe que notre base de données est vide. L'import des données se fait à partir de ce moment-là en plusieurs étapes :

1. La première chose à faire est de créer un index sur nos noeuds, celui-ci pouvant par la suite augmenter les performances de Neo4j :

Listing 5 Création d'un index

```
1 CREATE INDEX ON :Website(id)
```

2. Ensuite on peut importer le fichier contenant tous les sites :

Listing 6 Importation des sites internet (noeuds)

```
1 LOAD CSV WITH HEADERS FROM 'file:///NotreDame-nodes.csv' AS
  ↪ websites
2 CREATE (w:Website {
 3   id:toInteger(websites.id)
4 })
```

3. On continue ensuite avec l'importation du second fichier, celui des liens :

Listing 7 Importation des liens (relations)

```
1 USING PERIODIC COMMIT 10000
2 LOAD CSV WITH HEADERS FROM 'file:///NotreDame-links.csv' AS
  ↪ links
3 MATCH (w1:Website {id:toInteger(links.FromNodeId)}),
  (w2:Website {id:toInteger(links.ToNodeId)})
4 CREATE (w1)-[:LINK]->(w2)
```

4. On peut continuer en appliquant l'algorithme PR au réseau :

Listing 8 Application de l'algorithme PR sur tous les noeuds

```
1 CALL algo.pageRank( 'Website' , 'LINK' , {  
2   write:true, writeProperty:'pagerank'  
3 })
```

Le mot clé `writeProperty` va nous permettre plus tard d'afficher les noeuds avec une taille variant selon leur rang.

5. Pour finir, on peut si l'on souhaite afficher les cent meilleurs rangs calculés (à noter que le mot clé `DISTINCT` est utilisé pour assurer que chaque noeud n'est affiché qu'une seule fois) :

Listing 9 Affichage des 100 meilleurs noeuds

```
1 MATCH (w:Website) RETURN DISTINCT(w.id), w.pagerank  
2 ORDER BY w.pagerank DESC LIMIT 100
```

4.6.4 Vérification de l'import des données

Une fois toutes ces étapes effectuées, il nous reste à vérifier si notre graphe est complètement construit et que toutes les données ont bien été importées correctement.

Pour ce faire, un des moyens serait d'exécuter une requête dans le terminal Cypher qui aurait pour but de nous donner les dix sites possédant le meilleur PR.

Le terminal de la page suivante nous affiche bien sous forme ASCII les dix meilleurs sites Web comportant leur identifiant ainsi que leur score équivalent.¹

4.7 Contraintes à prendre en compte

Lorsque l'on effectue l'importation d'un grand nombre de données - pouvant aller jusqu'à environ 10 millions d'entrées - il nous faut prendre en compte certaines contraintes qui peuvent éventuellement avoir un impact négatif sur les performances de Neo4j :

1. On dénote sur l'image le pouvoir du référencement. En effet, le meilleur score n'appartient pas au site principal (noeud 0) mais au noeud 1'963 car il est référencé par ce noeud.

```

neo4j> match (n) return n order by n.pageRank desc limit 10;
+-----+
| n
+-----+
| (:Website {pageRank: 673.7651922889811, id: 1963})
| (:Website {pageRank: 670.8524142288182, id: 0})
| (:Website {pageRank: 569.0339101284744, id: 10336})
| (:Website {pageRank: 441.2020507968845, id: 212843})
| (:Website {pageRank: 395.4874412756647, id: 124802})
| (:Website {pageRank: 330.0489089203215, id: 12129})
| (:Website {pageRank: 328.55325501649656, id: 191267})
| (:Website {pageRank: 327.420171397987, id: 32830})
| (:Website {pageRank: 294.64849868305026, id: 83606})
| (:Website {pageRank: 282.36983163696715, id: 1973})
+-----+
    
```

FIGURE 4.3: Vérification de la bonne importation des données

1. En important une grande quantité de données comme dans notre cas, il est essentiel de rajouter un : USING PERIODIC COMMIT *n* au début de notre requête. Elle permet en effet d'ajouter les données à la base de manière temporaire pendant l'importation, toutes les *n* itérations. Cela permet d'augmenter les performances de Neo4j et peut sur un très grand nombre de données s'avérer être un énorme gain de temps.
2. Ensuite, il faut modifier les configurations mémoire de Neo4j dans le fichier neo4j.conf. On veut pouvoir être sûr que la mémoire ne sera pas un problème lorsque l'on veut importer beaucoup de données. Pour ce faire, il faut modifier les lignes suivantes :

```

dbms.memory.heap.max_size=4g
dbms.memory.heap.initial_size=4g
dbms.memory.pagecache.size=4g
    
```

Par défaut, les tailles sont fixées à 10[Mo]. Changer la limite à 4[Go] nous assure qu'il n'y aura aucun problème lors de l'importation.

3. Pour finir, il est bien d'utiliser le shell personnalisé de Cypher au lieu d'écrire les commandes dans le browser. On obtient généralement les résultats souhaités plus rapidement. Ce shell est accessible en tapant la commande : \$./bin/cypher-shell depuis le dossier "neo4j-community-3.5.5". Il suffit juste de rentrer les informations d'authentification puis l'effet est le même que depuis le browser.

Même si Neo4j est un outil très puissant, il est nécessaire de prendre en compte ces trois contraintes afin de ne pas avoir à constater une diminution flagrante des performances.

Chapitre 5

Pour aller plus loin

Nous venons de voir comment fonctionne l'outil Neo4j ainsi que ce à quoi ressemblent les données importées. L'objet de ce dernier chapitre, qui est une extension du précédent, est de nous familiariser d'avantage avec cet outil. Nous y verrons entre autres comment extraire une partie du graphe afin de résoudre le problème de temps de calcul pendant la visualisation puis analyser en détail l'évolution des scores des pages les mieux classées.

5.1 Extraction d'un sous graphe

Etant donné que nous travaillons avec un très grand nombre de données, il paraît évident de pouvoir trouver une solution pour ne travailler qu'avec une seule partie du graphe, par exemple si l'on veut afficher uniquement les noeuds possédant le plus gros score ou dans le cas d'une utilisation sur une page Web, laisser la possibilité à l'utilisateur de choisir le noeud dont il souhaite voir apparaître les relations.

5.1.1 Comment extraire une partie d'un graphe ?

Il va falloir pour répondre à cette question écrire une requête Cypher qui va sélectionner uniquement les noeuds que l'on souhaite. Dans un premier cas, nous pouvons nous concentrer sur l'affichage des noeuds possédant le meilleur rang ainsi que tous leurs liens sortants ou entrants.

Pour parvenir à la situation où l'on se retrouve avec les plus gros noeuds du réseau, il faut les trier par PR. La requête Cypher sera la suivante (où x est le nombre de noeuds à afficher) :

Listing 10 Affichage des plus gros noeuds du réseau

```

1      MATCH r=(n:Website)-[:LINK]->(p:Website)
2      RETURN r
3      ORDER BY n.pageRank DESC LIMIT x

```

Cette requête va précisément faire les actions suivantes :

1. Elle commence dans un premier temps par récupérer le noeud possédant le meilleur score
2. Ensuite, elle affiche tous les **liens sortants** de ce noeud.
3. Elle continue le même processus avec le second noeud jusqu'à ce que le nombre total de noeuds sur l'écran soit égal à x .¹

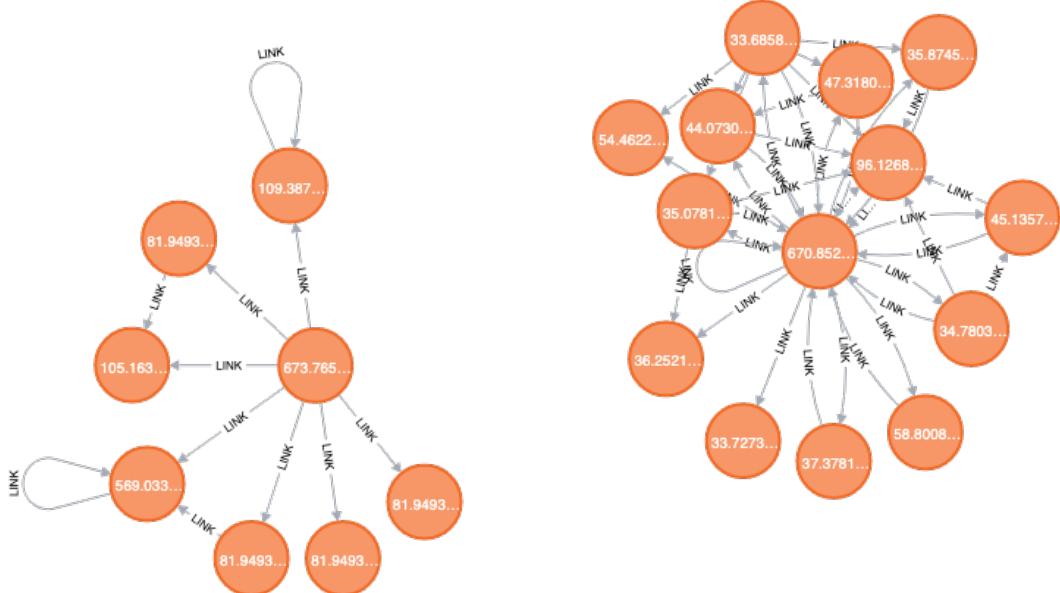


FIGURE 5.1: 20 liens sortants des sites les mieux référencés

Dans l'image ci-dessus, on recense uniquement deux sites avec tous leurs liens sortants (avec $x = 20$). On peut aisément noter que le site possédant le meilleur rang possède également moins de liens sortants.

Si l'on souhaite cette fois voir les liens entrants dans ces sites, on peut inverser le sens de la flèche :

1. Il est important de noter que dans ce cas précis, x doit correspondre au nombre de noeuds ayant des liens sortants.

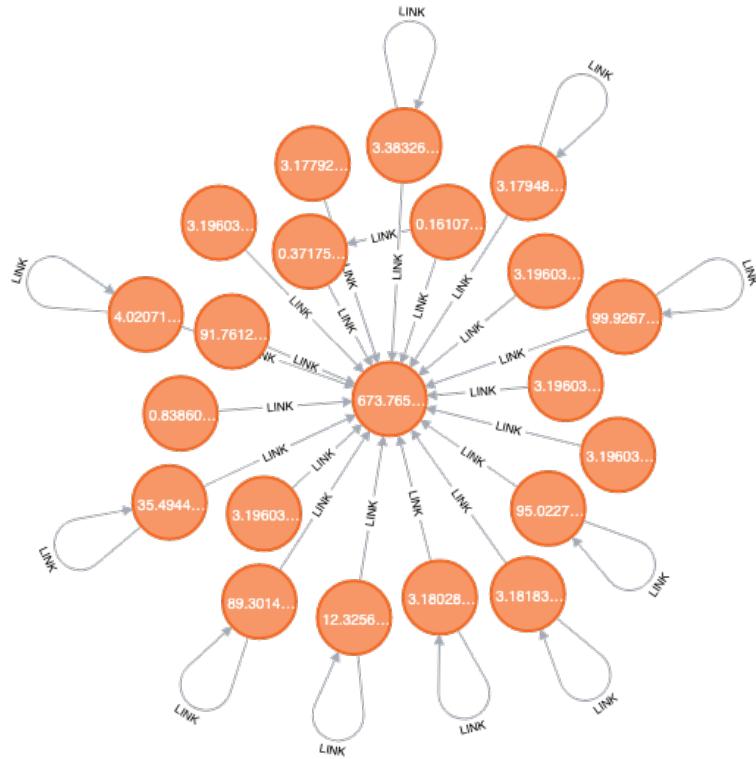


FIGURE 5.2: 20 liens entrants vers le site le mieux référencé

Cette fois, uniquement le meilleur site est affiché car le nombre de liens entrants est largement supérieur au nombre de liens sortants.

On peut de plus changer la limite du nombre de sites affichés, pour s'apercevoir que le premier site possède environ une cinquantaine de liens entrants au total.

5.1.2 Approfondissement de certains tests

Le but de cette partie est d'approfondir certains tests afin d'essayer de mieux comprendre la structure du site et l'influence des liens sur le rang de chacun. Les réseaux obtenus seront plus grands et plus complets que les précédents.

Récupération de liens sortants Le premier point intéressant dans Neo4j est la possibilité d'afficher tous les liens **sortants** d'un noeud que l'on aura choisi au préalable. Ce premier exemple sera d'afficher les liens associés aux deux sites possédant les meilleurs PR. Ces noeuds sont le 1'963 et le 0, le second étant la page principale du site.

Les deux images sont affichées sur la page suivante :

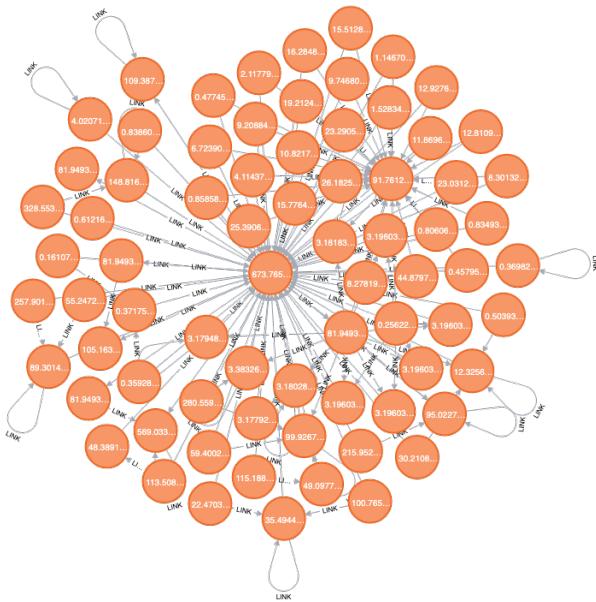


FIGURE 5.3: Liens sortants du noeud 1'963

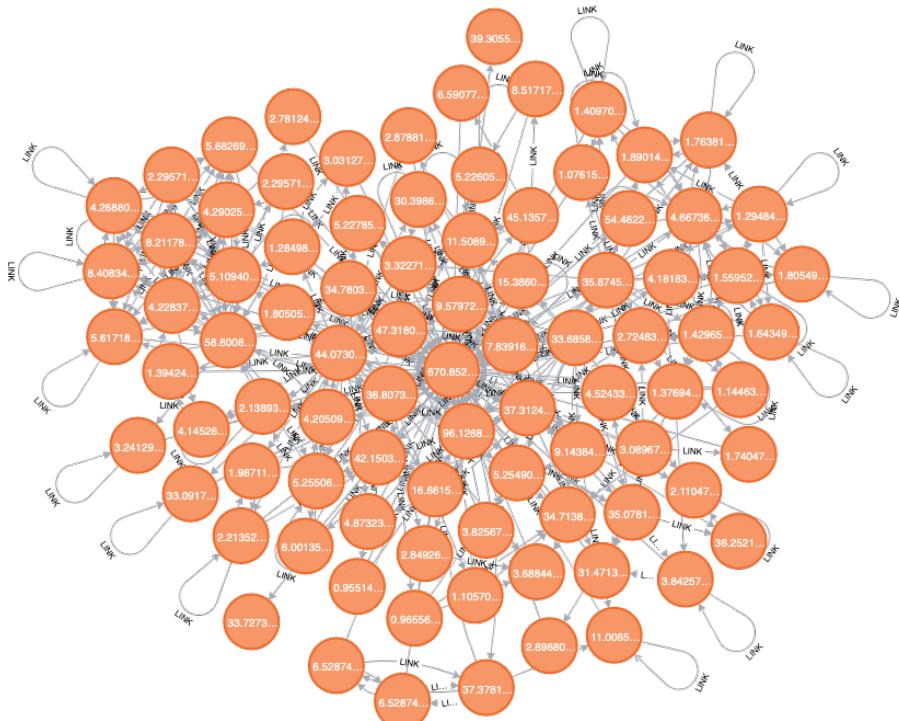


FIGURE 5.4: Liens sortants du noeud 0

On compte sur la première image 74 noeuds et 150 liens, 88 noeuds et 402 liens sur la seconde (Neo4j propose de compter le nombre de noeuds et de liens qui composent le graphe).

Environ 400 liens ne paraît pas beaucoup pour la page principale du site. En fait, dans le fichier de configurations, le nombre de voisins d'un simple noeud est fixé à 100. Neo4j fixe donc la limite à 100 liens directs depuis le noeud principal. Ceci nous explique le graphe relativement petit. En temps normal, le noeud 0 possède 7'636 voisins. Pour éveiller la curiosité des lecteurs, nous avons rajouté en annexe C une image (provenant d'un affichage sur une page HTML) de tous les noeuds voisins de la page principale du site de l'université.

On peut obtenir le noeud souhaité grâce à la requête Cypher suivante :

Listing 11 Sélection d'un noeud particulier en Cypher

```
1 MATCH (n:Website) WHERE n.id=x RETURN n
```

Récupération de liens entrants On peut considérer l'option d'afficher cette fois tous les liens **entrants** dans ces deux meilleurs noeuds. Cependant, la marche à suivre n'est pas tout à fait la même.

La requête Cypher à taper est la suivante :

Listing 12 Affichage de tous les liens entrants dans un noeud en Cypher

```
1 MATCH r=(n:Website)<- [:LINK] - (p:Website) WHERE n.id=x RETURN r
```

Elle veut dire que l'on va prendre tous les liens entrant dans le noeud n . Cette fois, nous n'avons pas besoin de toucher à un bouton supplémentaire, la requête parle d'elle-même. De nouveau, on reproduit ce processus pour les deux meilleurs noeuds, soit : $x = 1'963$ et $x = 0$. On obtient les images de la page suivante.

Globalement, le noeud 1'963 est référencé par beaucoup moins de sites que le noeud principal mais chacun de ses sites référents ont un PR plus élevé ce qui lui donne au final, un score meilleur.

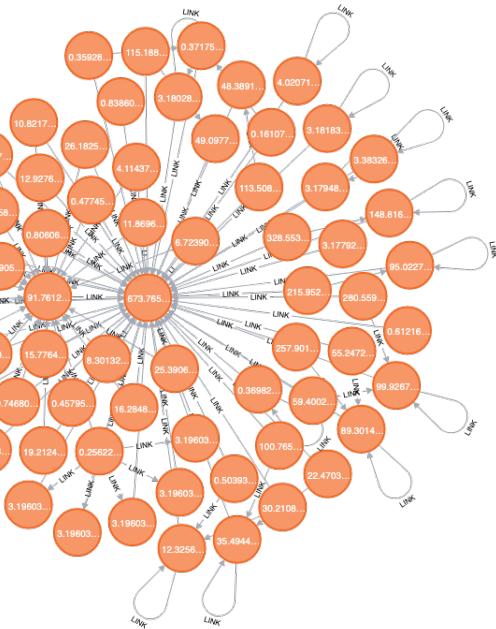


FIGURE 5.5: Liens entrants dans le noeud 1'963

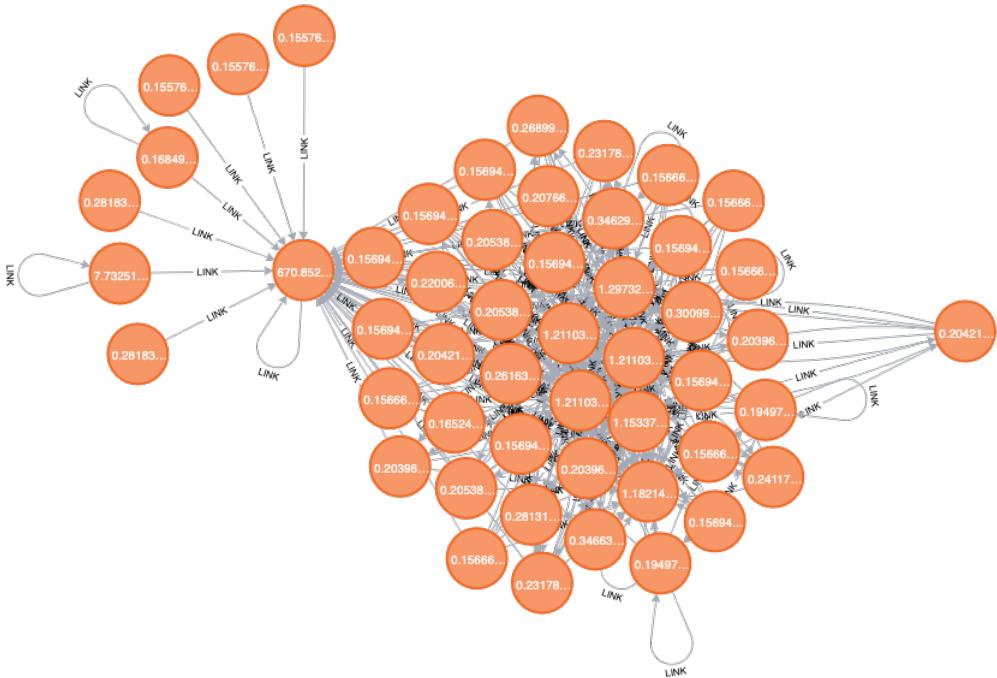


FIGURE 5.6: Liens entrants dans le noeud 0

5.1.3 Visualisation d'un graphe sur une page HTML

En Javascript, la librairie Neovis.js nous permet de créer une connexion avec notre base de données Neo4j pour récupérer les données et les afficher sur une page HTML. C'est la librairie dont nous avons besoin pour afficher au propre ces données.

Si l'on veut initialiser une connexion à la base de données, il faut passer par une variable de configurations où l'on peut rajouter les paramètres que l'on souhaite. Le listing de code 13 correspondant à cette partie se trouve ci-dessous :

Listing 13 Récupération des données de Neo4j

```
1  function draw(num) {
2      var config = {
3          container_id: "viz",
4          server_url: "bolt://localhost:7687",
5          server_user: "neo4j",
6          server_password: "super",
7          arrows: true,
8          physics: {
9              enabled: false
10         },
11         labels: {
12             "Website": {
13                 caption: "id",
14                 size: "pagerank"
15             }
16         },
17         relationships: {
18             "LINK": {
19                 caption: false,
20                 thickness: "count"
21             }
22         },
23         initial_cypher: "MATCH p=(:Website {id: " + num +
24             "})-[r]-() RETURN p"
25     }
26     var viz = new NeoVis.default(config);
27     viz.render();
28 }
```

La variable de configurations `initial_cypher` nous permet de rentrer la requête Cypher dont nous avons besoin, avec les paramètres que l'utilisateur va rentrer. Les paramètres d'affichage tout comme la taille d'un noeud ou bien la couleur et l'épaisseur de ses liens peuvent ainsi être choisis facilement. Dans cet exemple, l'utilisateur choisit le noeud - donné par la variable `num` - dont il souhaite voir apparaître les relations.

La page HTML ressemble à ceci :

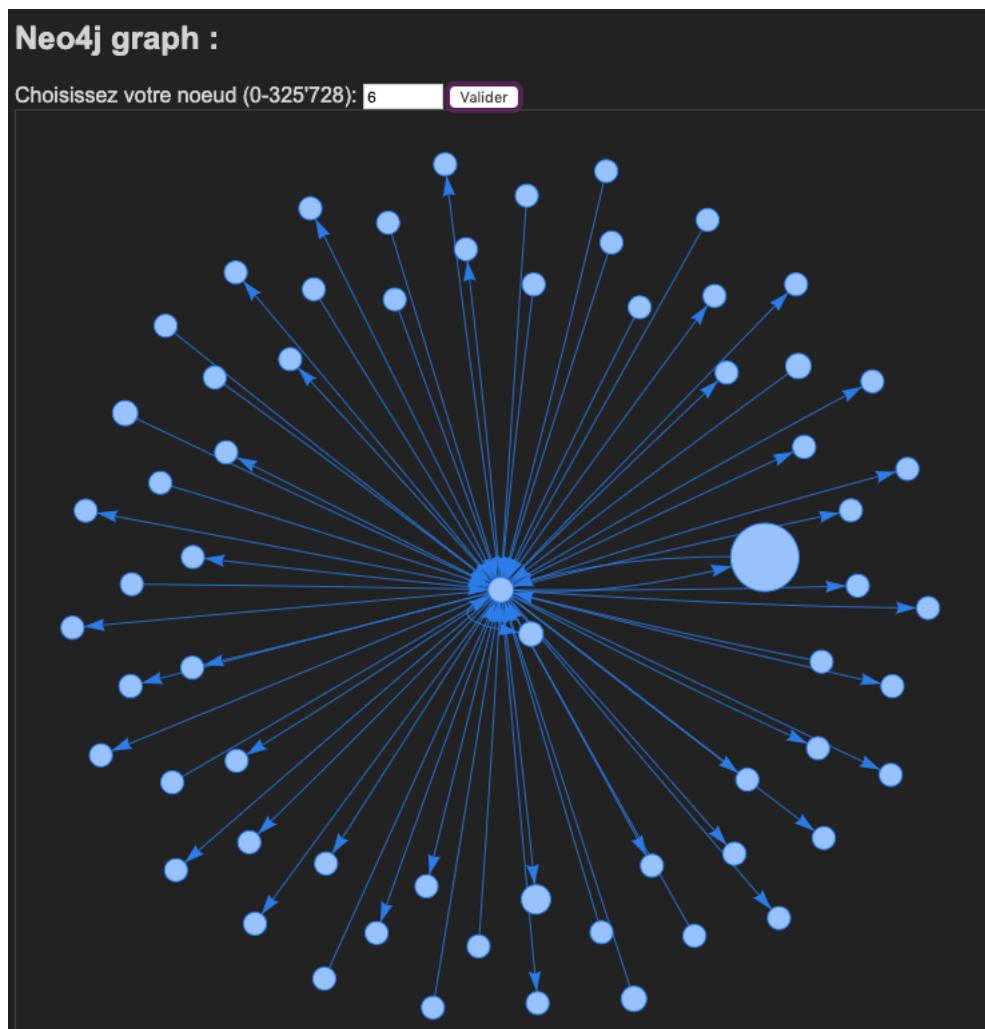


FIGURE 5.7: Visualisation HTML de tous les noeuds entrant dans le noeud 6

5.2 Visualisation plus précise de l'évolution des PageRanks

Dans cette toute dernière section, nous allons nous concentrer sur une visualisation plus itérative de l'algorithme PR appliqué à un grand graphe. Pour ce faire, on va devoir se servir de notre précédente requête Cypher en faisant augmenter progressivement le nombre d'itérations de l'algorithme. Pour ne pas qu'il y ait de confusions, le paramètre `writeProperty` sera appelé "pr" au lieu de "pagerank".

5.2.1 Une seule itération

On va exécuter la requête Cypher en fixant comme paramètre supplémentaire : `iterations : 1`. On peut vérifier que tout s'est bien passé en réaffichant les dix meilleurs sites, on obtient :

```
neo4j> match (n) return n order by n.pr desc limit 10;
+-----+
| n
+-----+
| (:Website {pr: 267.3846771240234, id: 191267, pagerank: 328.55325501649656}) |
| (:Website {pr: 216.00387878417968, id: 142732, pagerank: 280.55917869843546}) |
| (:Website {pr: 206.3487492799759, id: 0, pagerank: 670.8524142288182}) |
| (:Website {pr: 198.86772778688464, id: 124862, pagerank: 395.4874412756647}) |
| (:Website {pr: 198.14384002685546, id: 143218, pagerank: 257.901323541766}) |
| (:Website {pr: 145.49928665161133, id: 149039, pagerank: 215.95201298040337}) |
| (:Website {pr: 124.77494375705719, id: 12838, pagerank: 168.18391477176337}) |
| (:Website {pr: 124.10069780349731, id: 81878, pagerank: 166.75264958057176}) |
| (:Website {pr: 100.47072565853595, id: 1973, pagerank: 282.36983163696715}) |
| (:Website {pr: 95.26726779937745, id: 88118, pagerank: 113.50857722822111}) |
+-----+
```

FIGURE 5.8: Vérification de la bonne exécution de l'algorithme

On voit ici que le site 1'963 possédant le meilleur rang ne fait pas partie des dix premiers lors de la première itération de l'algorithme. On peut donc estimer que son rang va croître plus rapidement que le site principal d'identifiant 0.

5.2.2 Plusieurs itérations

Le meilleur moyen de reproduire cette requête de manière itérative est de créer un script Python approprié. Ce script va exécuter la requête de calcul de PR 20 fois pour atteindre le nombre d'itérations voulu. Il suffit de placer cette requête dans une boucle `for` et d'y rajouter le paramètre correspondant à l'itération actuelle.

Nous nous servons du module Python `py2neo` qui permet d'instancier une connexion avec une base de données `neo4j` et d'exécuter les requêtes que l'on souhaite en nous renvoyant les résultats dans la forme que l'on souhaite avoir. Dans notre cas, on retourne les résultats sous la forme d'une liste de dictionnaires.

La fonction que nous avons écrite est la suivante :

Listing 14 PR détaillé sur un grand graphe en Python

```

1 def add_node_properties():
2     """Calculate iterative pagerank for all biggest 10 nodes"""
3     for it in range(1, nb_ite+1):
4         # Run PageRank algorithm from Neo4j
5         s = str(it)
6         p = g.run(
7             "CALL algo.pageRank('Website', 'LINK', {write:true,
8                 ↵ writeProperty:'pr" + s + "'", iterations:" + s + "})"
9         ).data()
10        r = g.run(
11            "MATCH (n) RETURN n "
12            "ORDER BY n.pr" + s + " DESC LIMIT 10"
13        ).data()
14    return True

```

L'ensemble de données qui est retourné par cette requête n'est pas idéal pour dessiner un graphe récapitulatif. On doit donc rajouter un petit morceau de code qui va récupérer les valeurs souhaitées et créer un dictionnaire que l'on pourra ensuite utiliser pour dessiner les rangs.

Listing 15 Affichage du détail des rangs en Python

```

1 query = g.run(
2     "MATCH (n) RETURN n "
3     "ORDER BY n.pagerank DESC LIMIT 10"
4 ).data()
5 # Create dictionary of all pageranks associated to nodes
6 id = {}
7 for i in range(10):
8     id[query[i]['n']['id']] = [query[i]['n']['pr'+str(j)] for j
9         ↵ in range(1, nb_ite+1)]
10 for k,v in id.items():
11     plt.plot([- for _ in range(1, 21)], v)

```

5.2.3 Résultats et discussions

On peut grâce à ce code obtenir le graphe souhaité qui est par ailleurs très intéressant :

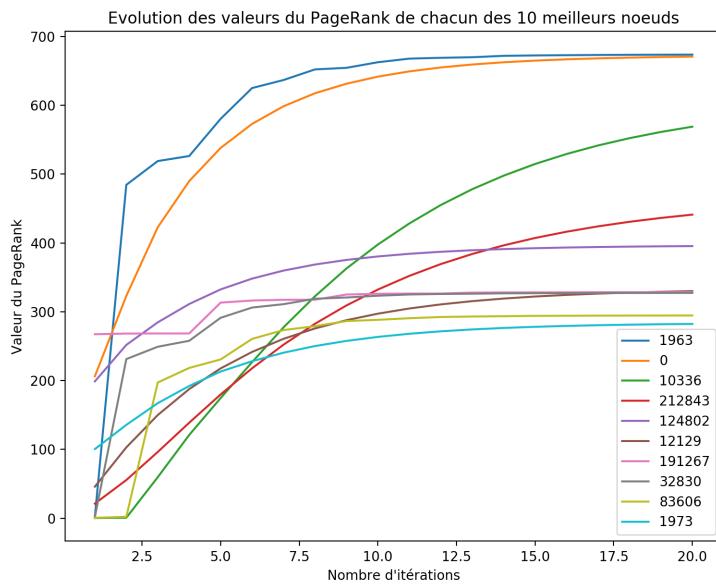


FIGURE 5.9: Evolution des PR sur un grand graphe

On remarque dans ce dessin que la page principale du site (généralement la page d'accueil) n'est pas forcément celle qui sera la mieux référencée dans le site complet. En effet, le noeud 1'963 possède le meilleur PR à partir de deux itérations seulement. Les deux noeuds restent de loin les meilleurs même si le troisième noeud n'est pas si loin derrière.

Dans tous les cas, lorsqu'un site internet bien noté en référence un autre aussi bien noté, il perd automatiquement des points de référencement. Cela amène à la conclusion suivante : **lorsque le rang de certaines pages augmente, le rang d'autres diminue !**

Conclusion

À PRÈS ces trois mois de travail, je me rend compte que mes connaissances dans le domaine des algorithmes du Web et de ses utilisateurs pionniers se sont considérablement développées. Je me rappelle encore avoir choisi ce sujet quelques semaines avant le début de ce travail et l'avoir présenté à mon professeur encadrant. J'ai tout de suite su que c'était le sujet qu'il fallait que je choisisse. Le but de cette conclusion est de revenir sur le travail déjà réalisé, mais aussi sur celui qu'il reste à faire.

Nous avons introduit dans le premier chapitre la notion du Web en rappelant de manière succincte son historique et son fonctionnement.

Nous nous sommes concentrés tout au long du deuxième chapitre sur la théorie du PR en fournissant quelques exemples concrets. Nous avons dans un premier temps défini ce qu'était un moteur de recherche et indiqué son but dans le monde du Web. Par la suite, un certain nombre de détails mathématiques ont été exposés afin de mieux comprendre les fondements de l'algorithme. Nous avons conclu le chapitre par quelques exemples afin d'affiner nos connaissances sur son mode de fonctionnement. L'étude de cet algorithme a pris un certain temps. Il a fallu d'abord saisir la signification implicite des diverses théories mathématiques expliquant son fonctionnement. Un grand nombre de tests ont ensuite été effectués afin d'en cerner toutes les particularités.

Dans le troisième chapitre, quelques concurrents de l'algorithme PR ont été exposés. En précisant leurs points théoriques fondamentaux, deux nouveaux algorithmes ont été introduits. Par la suite, nous avons implémenté ceux-ci afin de les comparer avec l'algorithme PR. Une liste exhaustive des avantages et inconvénients de chacun a été dressée. La dernière section de ce chapitre a permis l'exposition d'un dernier algorithme capable de corriger en grande partie le problème de fiabilité d'un site. Son implémentation en raison d'un manque de temps et de la complexité des trois précédents algorithmes n'a pas été possible.

Après une approche théorique, la deuxième partie de ce travail davantage pratique consistait d'abord en l'étude d'une nouvelle technologie pour stocker,

manipuler et afficher des graphes complets. L'idée était alors d'utiliser cet outil pour récupérer des données réelles du Web puis de les afficher sur une page internet. L'étude de cette nouvelle technologie nous a pris un certain temps compte tenu du grand nombre de fonctionnalités implémentées.

Nous avons terminé ce travail par l'implémentation d'un certain nombre de fonctionnalités supplémentaires. Typiquement, nous avons proposé un affichage facile et rapide d'une certaine partie du graphe offrant ainsi à l'utilisateur la possibilité de choisir la partie qu'il souhaite afficher. Nous avons conclu par une application itérative de l'algorithme PR sur les données récupérées afin d'étudier l'évolution du rang de chaque site. Là encore, cette dernière partie a été prolongée par la difficulté rencontrée d'offrir une solution plausible.

L'étude de ces algorithmes a été une très bonne opportunité de me familiariser avec le fonctionnement des moteurs de recherches et le monde du référencement sur le Web. Largement complexifié depuis la dernière décennie et toujours en forte croissance, il m'est désormais plus facile de saisir les particularités de son fonctionnement.

Malgré le travail effectué déjà conséquent, le nombre de choses à découvrir et à faire reste considérable. On pourrait par exemple, effectuer des tests sur une machine plus performante afin de pouvoir travailler avec de plus grands graphes ou encore créer une application permettant de prédire le rang d'une page en fonction des liens qu'on lui rajoute ou qu'on lui enlève. Face à l'immensité et à la complexité de l'Internet, les pistes d'exploration restent multiples et variées. Des résultats prometteurs, initiateurs d'évolution, restent encore à découvrir.

Annexes

Annexe A

Installation et configurations de Neo4j

Installation L'installation se déroule en plusieurs étapes :

1. Rendez vous sur le site de Neo4j dans la section téléchargements accessible par le lien suivant : <https://neo4j.com/download-center/>
2. Téléchargez la dernière version du Community Server (le fichier .tar mis à disposition)
3. Vérifiez en cliquant sur le lien "SHA-256" en dessous, qu'en tapant la commande `shasum -a 256 neo4j-community-3.5.5-unix.tar.gz`, le hash obtenu dans la ligne de commande soit le même que celui affiché sur le site. En principe, il n'y a pas de différence. Si toutefois il y en a une, recommencez le processus de téléchargement.
4. Assignez une variable d'environnement au dossier téléchargé, décompréssé. Pour ce faire, tapez la commande suivante : `NEO4J_HOME=neo4j-community-3.5.5`. Ensuite, rendez vous dans le dossier racine en tapant : `cd $NEO4J_HOME`.
5. A partir de ce moment-là, vous pouvez démarrer le serveur en tapant : `./bin/neo4j console`. Cette commande démarre un serveur local qui peut être accédé sur un navigateur à l'adresse suivante : `http://localhost:7474/browser`

Configurations En temps normal, Neo4j est configuré pour être utilisé à bon escient. En revanche, si vous souhaitez appliquer des algorithmes externes tel que le PR dans notre cas, il faut effectuer quelques configurations supplémentaires :

1. Dans un premier temps, téléchargez les deux archives qui vont nous permettre de récupérer les éléments permettant d'exécuter n'importe quel algorithme de traitement de graphe :

- a) apoc-3.5.0.3-all.jar
- b) graph-algorithms-algo-3.5.4.0.jar

Placez les dans le sous-dossier \$NEO4J_HOME/plugins. Des indications supplémentaires sont marquées dans le fichier README.txt situé dans le même sous-dossier.

2. Ouvrez le fichier \$NEO4J_HOME/conf/neo4j.conf, et rajoutez les lignes suivantes :

- a) dbms.security.procedures.unrestricted=apoc.trigger.*,algo.* , apoc.*
- b) apoc.export.file.enabled=true

Décommentez également la ligne : dbms.directories.plugins=plugins offrant la possibilité d'utiliser des plugins, en l'occurrence nos algorithmes.

3. Finissez par redémarrer le serveur, et vous pouvez maintenant utiliser Neo4j comme vous le souhaitez.

Annexe B

Installations d'Anaconda et IPython

Anaconda Il va être possible de trouver un historique des différentes versions d'Anaconda sur l'adresse suivante : <https://repo.continuum.io/archive/>. La version d'Anaconda à installer est la 5.0.1 car c'est la dernière compatible avec Python 3.6.

Une fois le package téléchargé, il faut suivre les instructions à l'écran et choisir le dossier : /Users/username comme emplacement d'installation. Une fois l'installation terminée, il reste à placer le package dans la corbeille.

Il se peut cependant que la version de Python utilisée par le terminal ne soit pas compatible avec celle téléchargée et installée par Anaconda. Il faut donc ouvrir le fichier /Users/username/.bash_profile et y rajouter la ligne suivante :

```
export PATH="/Users/username/anaconda3/bin:$PATH"
```

Puis, taper la ligne : source /Users/username/.bash_profile pour activer le profil. On se sert ainsi de la version de Python souhaitée :

A screenshot of a terminal window showing two commands being run. The first command is 'source ~/.bash_profile' and the second is 'python -V'. The output shows 'Python 3.6.7'.

FIGURE B.1: Modification du profil bash

Jupyter L'installation d'Anaconda amène automatiquement à celle de Jupyter. On peut ainsi démarrer un Notebook à l'aide de la commande suivante : jupyter notebook. Une fois le notebook démarré, on arrive sur une fenêtre possédant l'interface suivante :

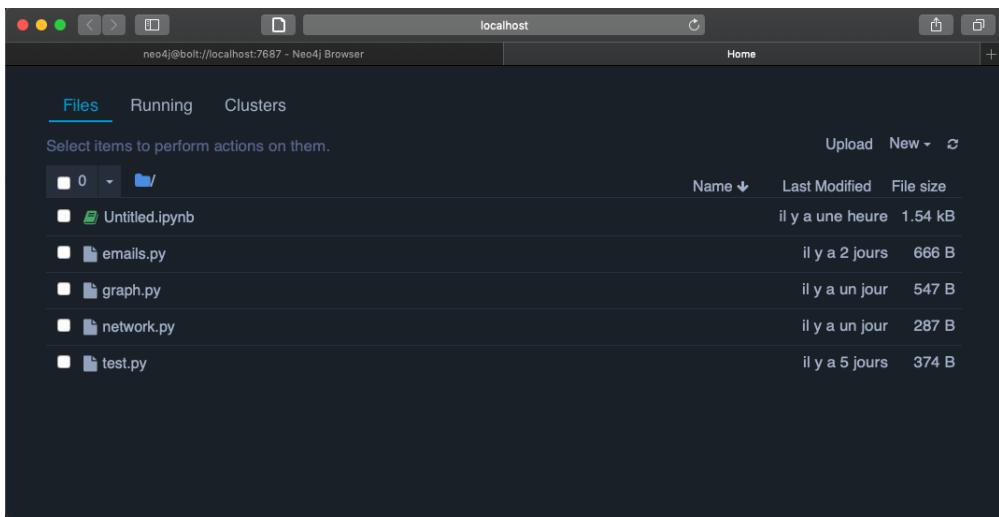


FIGURE B.2: Fenêtre de démarrage de Jupyter

On retrouve tous les fichiers python que l'on a créés. On peut également créer un nouveau Notebook à l'aide de l'onglet "New" en haut à droite.

IPython Il est déjà installé lors de l'installation de Jupyter. On peut ainsi en ouvrant un Notebook, commencer à écrire le code que l'on souhaite.

A screenshot of a Jupyter Notebook kernel window titled "localhost" showing a code cell output. The window has a top menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". A status bar at the top right indicates "Trusted | Python 3". The main area shows a code cell with the following Python code: "In [1]: for x in range(10): print(x)". The output of this code is displayed below the cell, showing the numbers 0 through 9. There is also an empty cell below the first one, indicated by "In []:".

FIGURE B.3: Fenêtre de démarrage de Jupyter

Annexe C

Graphe formé par les voisins du noeud principal

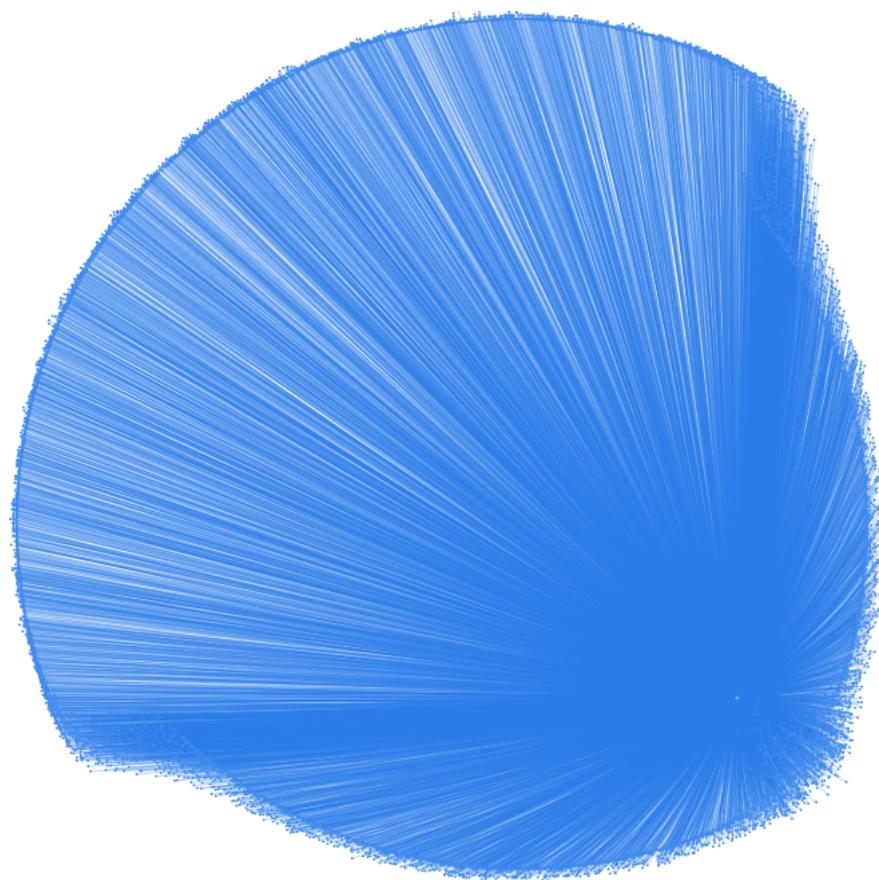


FIGURE C.1: Un graphe en forme de coquillage

Bibliographie

- And. Olivier Andrieu. Le cheirank, futur du pagerank ? <https://www.abondance.com/20100723-10417-le-cheirank-futur-du-pagerank.html>. [Online ; accessed 2-June-2019].
- BP10. Marco Bressan and Enoch Peserico. Choose the damping, choose the ranking? *Journal of Discrete Algorithms*, 2010. Voir : <https://www.sciencedirect.com/science/article/pii/S1570866709000926>.
- Du15. Phan Tran Thanh Du. Modélisations mathématiques : Google pagerank et chaîne de markov. Technical report, Ecole Nationale Supérieure de Cognitique - Institut Polytechnique de Bordeaux, Bordeaux, France, 2015. Voir : <https://duperhan.files.wordpress.com/2015/12/google-pagerank-et-chaiccc82ne-de-markov.pdf>.
- FGG⁺17. Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher : An evolving query language for property graphs. Technical report, ACM - Association for Computing Machinery, New York, USA, 2017. Voir : https://victor.marsault.xyz/resources/articles/CypherSigmod_v1.pdf.
- GGMP04. Zoltan Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating web spam with trustrank. Technical report, Stanford University, California, US, 2004. Voir : <http://ilpubs.stanford.edu:8090/770/1/2004-52.pdf>.
- Lem. Daniel Lemire. Recherche et filtrage d'informations. <http://benhur.teluq.ca/SPIP/inf6460/spip.php?article53>. [Online ; accessed 6-May-2019].
- Les12. Jure Leskovec. Link analysis : Pagerank and similar ideas, February 2012.

- LK14. Jure Leskovec and Andrej Krevl. SNAP Datasets : Stanford large network dataset collection. <http://snap.stanford.edu/data>, June 2014.
- Mat12. Fabien Mathieu. *Graphes du Web, Mesures d'importance à la PageRank*. PhD thesis, Université de Montpellier II - Sciences et Techniques du Languedoc, 2012. Voir : <https://tel.archives-ouvertes.fr/tel-00667563/document>.
- PB98. Larry Page and Sergey Brin. The pagerank citation ranking : Bringing order to the web. Technical report, Stanford University, California, US, 1998. Voir : <http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf>.
- Rob16. Eric Roberts. The google pagerank algorithm. Technical report, Stanford University, San Francisco, US, 2016. Voir : <https://web.stanford.edu/class/cs54n/handouts/24-GooglePageRankAlgorithm.pdf>.
- Ton. Tony. Trustrank : Explications. http://www.infowebmaster.fr/35_news-trustrank-explications.html.
- Wil06. Rebecca S. Wills. Google's pagerank : The math behind the search engine. Technical report, North Carolina State University, Raleigh, US, 2006. Voir : <https://pdfs.semanticscholar.org/3356/6b740d3cd0c0dde57e13b5da148bef37376f.pdf>.