

TANIA MENDES DIAS
KEVIN CHEN

PAULA BURBANO
BASTIEN GUILLEMARE

Projet PageRank

RAPPORT

Janvier 2022 - Mai 2022

Encadrant :
Esteban Bautista Ruiz



Table des matières

1	Introduction	2
1.a	Contexte	2
1.b	Objectifs	3
2	Analyse du problème	4
2.a	Modélisation mathématique d'un réseau web	4
2.b	Modélisation du problème par une chaîne de Markov	5
2.c	Définition du PageRank	7
3	Mise en place du projet numériquement	9
3.a	Algorithme de calcul du PageRank	9
3.b	Comparaison entre les deux algorithmes	12
4	Applications	13
4.a	Scraping de Wikipédia	13
4.b	Catégorisation des sites	14
4.c	Moteur de recherche	18
5	Conclusion	20
6	Annexes	21
6.a	Exemples et erreurs rencontrées	21
6.b	Installation de l'outil wikisearch	22
6.c	Algorithmes	23
6.d	Bibliographie	24
6.e	Organisation du travail	24
6.f	Impact de notre travail	24
6.g	Bilan personnel	25
6.h	Remerciements	27

1 Introduction

1.a Contexte

Avec l'expansion des réseaux sociaux, les communications se simplifient mais la quantité de données ne cesse d'accroître, ce qui complexifie sa gestion. Ces données peuvent être représentées par de très grands graphes dans lesquelles nous pouvons stocker tous types d'informations. En effet, ils sont à l'origine de nombreuses applications de la vie quotidienne. On y retrouve la modélisation d'interactions sociales avec les appels téléphoniques ou encore les réseaux sociaux.

Un graphe est un ensemble de noeuds reliés par des arêtes. Pour réaliser une modélisation à partir d'un graphe, nous avons besoin d'établir des liens entre les différents noeuds et étudier l'importance de ceux-ci.

Dans ce cas de figure, on peut faire intervenir le PageRank. Il s'agit d'un indicateur qui permet de mesurer l'importance qu'ont les noeuds dans un graphe. Si l'on modélise le réseau Instagram par un graphe dont les noeuds correspondent aux utilisateurs, alors le PageRank s'agira de mesurer l'importance qu'ont les utilisateurs en fonction du nombre d'abonnés qu'ils ont et de leurs liens avec les autres communautés. Plus un utilisateur a d'abonnés, plus son PageRank sera élevé et plus il sera important dans le réseau.

Alors, nous comprenons que le PageRank associe à chaque noeud une valeur qui est proportionnelle à son importance. Faire le calcul du PageRank est un problème difficile à résoudre étant donné qu'il revient à inverser un système linéaire d'équations, ce qui est à la fois très coûteux en espace mémoire mais aussi en temps avec une forte complexité. Notamment en reprenant l'exemple d'un très grand réseau comme Instagram, cela reviendrait à inverser une matrice qui est proportionnelle à 2 milliards de noeuds. Dans cette optique de réduire le temps de calcul ou les ressources utilisées, nous nous sommes tournés vers des méthodes d'approximation du calcul du PageRank.

L'objectif de ce projet est donc d'appréhender et d'implémenter des méthodes d'approximations de calcul du PageRank appliquées à de très grands graphes.

1.b Objectifs

L'objectif principal de notre projet est d'implémenter les différents algorithmes qui approximent le calcul du PageRank dans un cas pratique : le réseau Wikipédia. Nous avons choisi Wikipédia car c'est une encyclopédie très utilisée de nos jours et qui compte plus de 6 milliards d'articles, ce qui rend l'accès à l'information rapide. Cependant, nous aurions pu décider d'implémenter nos algorithmes pour n'importe quel autre réseau.

Dans ce projet, nous allons traiter deux cas d'applications de l'algorithme du PageRank :

- **Moteur de recherche** : C'est un cas d'utilisation du PageRank très important car les moteurs de recherche font parti de notre quotidien sur Internet, des applications, des bases de données, etc. En effet, à partir du PageRank, nous obtenons un classement des différentes pages en lien avec la recherche, ce qui nous permettra de faire des suggestions de pages les plus pertinentes.
- **Catégorisation des sites** : C'est une deuxième application du PageRank. En effet, il se peut qu'il y ait un grand nombre de pages à gérer. Pour cela, il est d'une grande aide de pouvoir organiser de façon automatique toute l'information car la traiter manuellement serait très long voire impossible.

Supposons que l'on recherche un livre dans une bibliothèque. Nous voyons bien qu'il est plus simple de le trouver rapidement si les livres sont organisés par sujets ou par genres. Nous pouvons même naviguer entre les livres d'une même catégorie. Ceci est le principe de la catégorisation des sites.

L'objectif final de notre projet est de pouvoir réaliser les applications décrites précédemment directement en ligne de commandes. Cela est utile car nous travaillons principalement avec le terminal et effectuer une recherche en ouvrant un navigateur pour vérifier une information peut vite devenir pénible. Ainsi, nous allons pouvoir profiter d'un outil simple et rapide qui se repose sur des langages optimisés pour la gestion de grandes bases de données afin d'extraire les informations souhaitées.

Dans un premier temps, nous allons analyser le problème pour en tirer un modèle mathématique. Puis, nous allons nous intéresser à la mise en place des algorithmes d'approximation. Enfin, nous allons mettre en oeuvre les deux applications du PageRank décrite précédemment dans le but de créer un outil utile pour la vie quotidienne.

2 Analyse du problème

Nous allons consacrer cette partie à l'analyse mathématique du problème. Nous présentons la modélisation mathématique d'un réseau web et celle du problème par une chaîne de Markov. Puis, nous définirons formellement la notion de PageRank pour la suite du projet.

2.a Modélisation mathématique d'un réseau web

Nous allons modéliser un réseau web par un très grand graphe orienté. Chaque sommet du graphe correspond à une page web et chaque arc est un hyperlien d'un site vers un autre.

De manière plus formelle, les sommets du graphe peuvent être représentés par des entiers de 1 à N , où N est le nombre total de pages web.

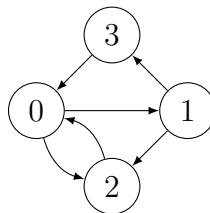
Pour chaque couple de sommets $\{i, j\} = \{1, \dots, N\}$, le graphe contient un arc de i vers j si la page i contient un hyperlien vers la page j , dans ce cas, on écrira $i \rightarrow j$.

De plus, dans notre modèle, un site peut avoir **au plus** un hyperlien vers un même site. C'est à dire que nous ne comptons qu'une fois les hyperliens multiples qui peuvent éventuellement exister dans un site.

Si dans une de nos pages web que l'on note i , il n'y a aucun hyperlien sortant, on décide d'ajouter par défaut un arc de i vers i , c'est-à-dire un arc sur lui-même. Cela aura une utilité plus tard pour nous aider à programmer l'algorithme du PageRank.

La notion de degré sortant est importante dans notre modélisation car elle correspond au nombre d'hyperliens vers lesquels la page i renvoie. On notera le degré sortant d'un sommet i notée $d^+(i)$, il correspond au nombre d'hyperliens partant du sommet i . Par analogie, le degré entrant d'un sommet i est noté $d^-(i)$ et correspond au nombre d'hyperliens qui mène vers le sommet i .

Exemple d'une représentation d'un graphe du web avec 4 sites :



On peut par exemple voir dans ce graphe que le site à l'indice 0 contient des hyperliens emmenant vers le site d'indice 1 et 2. Ainsi le degré sortant du sommet 0 est 2, ce que l'on note : $d^+(0) = 2$.

Par le même raisonnement on a : $d^+(1) = 1$, $d^+(2) = 1$ et $d^+(3) = 1$.

Les degrés entrants sont alors :

$$d^-(0) = 1; \quad d^-(1) = 1; \quad d^-(2) = 2 \quad d^-(3) = 1$$

On introduit aussi la matrice d'adjacence $A = (a_{i,j})_{1 \leq i \leq N, 1 \leq j \leq N}$ de notre graphe du web définie par :

$$(a_{i,j}) = \mathbb{1}_{i \rightarrow j} = \begin{cases} 1 & \text{si } i \rightarrow j \\ 0 & \text{sinon} \end{cases} \quad \forall i, j = 1, \dots, N$$

Dans l'exemple 2.a précédent, la matrice d'adjacence correspond à cette matrice :

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Cette matrice nous permet de savoir quels sommets sont connectés entre eux dans notre graphe. Ainsi, en se servant de la première ligne nous pouvons voir que les sommets 1 et 2 sont des voisins du sommet 0.

Nous savons que dans le web, il y a des pages plus importantes que d'autres. En effet, plus une page est visitée, plus elle sera importante. Si nous effectuons un simple graphe comme dans l'exemple 2.a nous ne prenons pas en compte cet aspect. C'est là qu'intervient le PageRank. En effet, il permet de calculer l'importance des pages, ce qui sera un indicateur pour la recherche de sujets particuliers.

Pour procéder au calcul du PageRank, nous allons imaginer que nous sommes un surfeur sur le web qui saute de page en page de façon aléatoire. Ces sauts de page en page se font selon un hyperlien qui est présent sur la page sur laquelle il se trouve. Cela de manière **uniforme** c'est-à-dire que si le site où le surfeur se trouve contient 5 hyperliens, les 5 hyperliens auront la même probabilité d'être choisis. Il est important de préciser que dans notre modèle, un site peut avoir **au maximum** un hyperlien vers un autre site.

Cette représentation nous permet de modéliser notre problème par une notion mathématique très connue qui est la notion des **chaînes de Markov** qui nous permet de calculer le Page Rank des sites de notre réseau web.

2.b Modélisation du problème par une chaîne de Markov

Notre modèle du surfeur web nous permet de représenter le calcul du PageRank par une chaîne de Markov. Une chaîne de Markov est caractérisée par le fait que l'état futur ne dépend uniquement que de l'état présent de la chaîne. En effet, si nous assimilons les états aux sites, nous remarquons que l'état futur de notre surfeur web ne dépend uniquement que de l'état dans lequel il se trouve actuellement et ne fait donc pas intervenir les états par lesquels il est passé précédemment.

L'algorithme du PageRank calcule un $\pi(i) \in [0, 1]$ pour tous les i sites web qui correspond au PageRank du $i^{\text{ème}}$ site web.

Ainsi, nous définissons un vecteur Π qui contient le PageRank de tous les sites :

$$\Pi = [\pi(1), \dots, \pi(N)], N \text{ étant le nombre total de sites.}$$

De plus, on supposera que les PageRank sont normalisés, ainsi :

$$\sum_{i=1}^N \pi(i) = 1$$

Pour pouvoir calculer Π , nous allons modéliser Π comme une **loi stationnaire** d'une chaîne de Markov.

Ainsi, dans un premier temps, nous allons établir quelques hypothèses qui nous permettront de modéliser le calcul du PageRank par une chaîne de Markov, c'est-à-dire, par le calcul de la **loi stationnaire** Π .

Si nous considérons le modèle du surfeur aléatoire, nous partons d'un état X_0 qui correspond au site où le surfeur se trouve initialement. Puis, après n sauts, nous nous retrouvons à l'état X_n .

Ainsi, on définit $(X_n)_{n \in \mathbb{N}}$ une chaîne de Markov. En effet, le site sur lequel le surfeur se trouve à l'état $n + 1$ ne dépend que du site sur lequel il se trouvait à l'état n .

Pour passer d'un état à un autre nous définissons la matrice de transition P telle que :

$$P = (p_{i,j})_{1 \leq i \leq N, 1 \leq j \leq N}$$

Il convient de préciser que P correspond à la matrice d'adjacence du graphe orienté du réseau web définie précédemment (exemple 2.a) où chaque élément de la ligne i est divisé par le nombre de degrés sortants de notre sommet i .

De plus, la somme sur les coefficients de chaque ligne de P est égale à 1. Chaque coefficient de P est donc défini de la façon suivante :

$$(p_{i,j}) = \frac{1}{d^+(i)} \mathbb{1}_{i \rightarrow j} = \begin{cases} \frac{1}{d^+(i)} & \text{si } i \rightarrow j \\ 0 & \text{sinon} \end{cases} \quad \forall i, j \in \{1, \dots, N\}$$

Avec cette matrice de transition nous pouvons calculer la loi stationnaire Π où chaque $\pi(i)$ correspondra à la valeur du PageRank du site i , Π satisfaisant l'équation :

$$\Pi = \Pi P \tag{1}$$

Cependant, pour que le PageRank existe et soit unique, et qu'au bout d'un n sauts, Π converge en loi, nous vérifions que le modèle sur lequel nous allons implémenter le PageRank possède les propriétés suivantes :

- **homogénéité** : La valeur du Page Rank à l'état $n + 1$ ne dépend que de la valeur du PageRank à l'état n . Il n'y a pas de dépendance des états antérieurs c'est-à-dire des états $\{1, \dots, n - 1\}$. Le modèle du surfuer aléatoire est sans mémoire.
- **irréductibilité** : Le graphe du web modélisé précédemment est fortement connexe, c'est à dire que tous les sommets communiquent entre eux. Donc, la chaîne est bien fermée et irréductible.
- **récurrence** : L'espace des états est fini. Notre chaîne étant fermée et irréductible, elle est donc récurrente.

La chaîne étant homogène, irréductible et récurrente la loi stationnaire Π existe et est **unique**. Cependant, pour avoir une convergence en loi de Π il faut une dernière hypothèse :

- **apériodicité** : Nous implémentons l'algorithme du PageRank sur le même graphe. En effet, on considère qu'il n'y a pas de sites rajoutés ni supprimés. La chaîne étant irréductible, tous les états ont la même période et sont apériodiques. La chaîne est donc apériodique.

Ainsi, la chaîne de notre modèle étant irréductible et apériodique, il y a une convergence en loi de Π . C'est-à-dire qu'après un nombre n de sauts, il y a la convergence vers un unique PageRank pour tous les sites.

2.c Définition du PageRank

Cependant dans la vie réelle, nous n'avons pas toujours des graphes qui vérifient nos critères permettant d'assurer la convergence en loi de la chaîne de Markov. C'est donc pour cela qu'on ajoute un amortissement qui va nous conduire à la définition formelle de PageRank.

En effet, le facteur d'amortissement $\alpha \in [0, 1]$ va permettre d'assurer que peu importe le réseau web utilisé, les hypothèses d'irréductibilité et d'apériodicité sont bien vérifiées. Ce facteur α ajoute une probabilité supplémentaire dans la matrice de transition. Elle va correspondre au fait qu'il se peut que le surfeur aléatoire décide de choisir un site complètement aléatoire parmi tous ceux qui sont à sa disposition au lieu de choisir un des hyperliens du site où il se trouve. Le choix du prochain site du surfeur sera distribué de la manière suivante :

- une fraction $\alpha < 1$ qui correspond au choix d'un hyperlien du site où il se trouve de manière aléatoire et uniforme
- la fraction $1 - \alpha$ qui correspond à un choix uniforme et aléatoire du prochain site parmi tout le réseau qu'on possède

Nous obtenons donc une nouvelle matrice de transition telle que :

$$M = (m_{i,j})_{1 \leq i \leq N, 1 \leq j \leq N}$$

de la chaîne de Markov de notre surfeur qui est donnée par :

$$(m_{i,j}) = \alpha \frac{1}{d^+(i)} \mathbb{1}_{i \rightarrow j} + (1 - \alpha) \frac{1}{N}, \quad \forall i, j \in \{1, \dots, N\}$$

On a donc :

$$M = \alpha P + (1 - \alpha) \frac{1}{N} E \tag{2}$$

où $E \in \mathcal{M}_{\mathbb{N}, \mathbb{N}}$ dont tous les coefficients sont égaux à 1.

PageRank Personnalisé :

Il existe une autre version du PageRank appelé PageRank Personnalisé. Comme son nom l'indique, le PageRank personnalisé calcule le PageRank en fonction d'une personnalisation donnée. En effet, on va considérer que certains sommets de notre graphe sont plus importants que d'autres. Dans notre cas, c'est l'utilisateur qui donne la personnalisation.

Ainsi on modifie notre matrice de transition définie précédemment (2), en remplaçant le terme E par un vecteur y . Ce vecteur correspondra aux sites choisis par l'utilisateur en tant que personnalisation. Ainsi, notre vecteur y vaudra 1 pour les sites choisis par l'utilisateur et 0 pour les autres sites.

On obtient donc une nouvelle définition de notre matrice de transition pour l'algorithme de PageRank Personnalisé qui est celle-ci :

$$M = \alpha P + (1 - \alpha) \frac{1}{N} y$$

On peut aussi remarquer que l'algorithme de PageRank que l'on connaît comme "classique" est un cas particulier du PageRank Personnalisé où tous les sommets ont été choisis par l'utilisateur comme importants.

3 Mise en place du projet numériquement

D'après l'équation 1, nous pouvons nous ramener aux équations suivantes :

$$\begin{aligned}\Pi &= \alpha \Pi P + (1 - \alpha) \frac{y}{|y|} \\ \Leftrightarrow \Pi &= (1 - \alpha P)^{-1} (1 - \alpha) \frac{y}{|y|}\end{aligned}$$

Ainsi, calculer le PageRank directement viendrait à inverser une matrice de taille proportionnelle au réseau, ce qui serait extrêmement coûteux. Nous allons donc approximer numériquement le PageRank via deux algorithmes.

3.a Algorithme de calcul du PageRank

Nous allons implémenter deux algorithmes pour le calcul du PageRank : la méthode Power Iteration et la méthode Forward Push.

- **Power Iteration**

L'objectif de cette méthode est d'obtenir une valeur qui converge vers la valeur du PageRank. Cela en partant d'une valeur initiale du PageRank, nous itérons pour obtenir les valeurs suivantes. En partant des hypothèses précédentes sur le fait que notre chaîne de Markov $(X_n)_{n \in \mathbb{N}}$ est homogène, irréductible et récurrente, au bout de n itérations, le PageRank converge vers des valeurs uniques pour chaque site.

En effet, nous avons d'abord un premier théorème nommé le **théorème ergodique** qui assure l'existence et l'unicité la **loi stationnaire** Π . Cette **loi stationnaire** est bien un vecteur qui correspond au PageRank de tous nos sites.

En ajoutant l'hypothèse de l'apériodicité, nous avons le **théorème de la convergence en loi** qui nous permet de dire que la distribution stationnaire Π de $(X_n)_{n \in \mathbb{N}}$ est aussi sa distribution limite, soit :

$$\pi(i) = \lim_{n \rightarrow +\infty} \mathbb{P}(X_n = i) \quad \forall i \in \{1, \dots, N\} \quad (3)$$

Algorithme :

Nous notons π_0 la distribution initiale de la chaîne de Markov avec

$$\pi_0(i) = \mathbb{P}(X_0 = i), \quad \forall i \in \{1, \dots, N\}$$

De même pour chaque $n \geq 1$, on note π_n , la distribution de la chaîne de Markov après le $n^{\text{ème}}$ saut.

Pour chaque $n \in \mathbb{N}$, on a :

$$\mathbb{P}(X_{n+1} = i) = \sum_{j=1}^N \mathbb{P}(X_n = j) \mathbb{P}(X_{n+1} = i | X_n = j), \quad \forall i \in \{1, \dots, N\} \quad (4)$$

En réécrivant avec π on a :

$$\pi_{n+1}(i) = \sum_{j=1}^N \pi_n(j) (p_{j,i}) \quad (5)$$

Pour chaque $n \in \mathbb{N}$, nous assimilons la distribution π_n au vecteur ligne :

$$\pi_n = [\pi_n(1), \pi_n(2), \dots, \pi_n(N)] \quad (6)$$

nous pouvons donc réécrire le système 5 ci-dessus sous forme matricielle :

$$\pi_{n+1}(i) = \pi_n P, \quad \forall n \in \mathbb{N} \quad (7)$$

Avec le **théorème de convergence en loi** énoncé précédemment dans l'équation 3, nous obtenons une valeur approchée de Π de manière récursive en appliquant soit l'équation avec la somme [5] soit l'équation matricielle [7].

Cette algorithm marche pour tous graphes qui respectent les critères d'irréductibilité et d'apériodicité, cependant il se peut que ces hypothèses ne soient pas respectées, c'est pour cela qu'on modifie cet algorithme en ajoutant la condition d'un amortisseur présenté précédemment.

On obtient donc la récurrence suivante, pour chaque $n \in \mathbb{N}$:

$$\pi_{n+1}(i) = \sum_{j=1}^N \pi_n(j)(m_{j,i}) = \sum_{j=1}^N \pi_n(j) \left(\alpha \frac{1}{d^+(j)} \mathbb{1}_{j \rightarrow i} + (1-\alpha) \frac{1}{N} \right), \forall i \in \{1, \dots, N\} \quad (8)$$

Sous forme matricielle, on a :

$$\pi_{n+1} = \pi_n M = \alpha \pi_n P + (1-\alpha) \frac{1}{N} e \quad (9)$$

avec $M = \alpha P + (1-\alpha) \frac{1}{N} E$ où $E \in \mathcal{M}_{\mathbb{N}, \mathbb{N}}$ dont tous les coefficients sont égaux à 1 et le vecteur ligne de dimension N dont tous les coefficient sont égaux à 1.

Dans le cas du PageRank personnalisée, nous avons cette formule :

$$\pi_{n+1} = \alpha \pi P + (1-\alpha) \frac{y}{|y|} \quad (10)$$

Ce sont donc ces deux suites récurrentes sur lesquelles nous nous basons dans nos algorithmes **power Iteration**. Ceci avec deux versions : une en partant de la somme et l'autre de manière matricielle. Ces deux versions ont comme condition d'arrêt que la différence entre les valeurs des PageRank entre deux itérations successives soit plus petite qu'un ϵ défini au préalable. Les codes sont disponibles sur le [GitHub](#).

Le pseudo-code de cette méthode est présent en annexe : 1.

• Forward Push

Cette deuxième méthode de calcul du PageRank est basée sur la diffusion d'un résidu que l'on notera r . Considérons le graphe suivant représentant un réseau R .

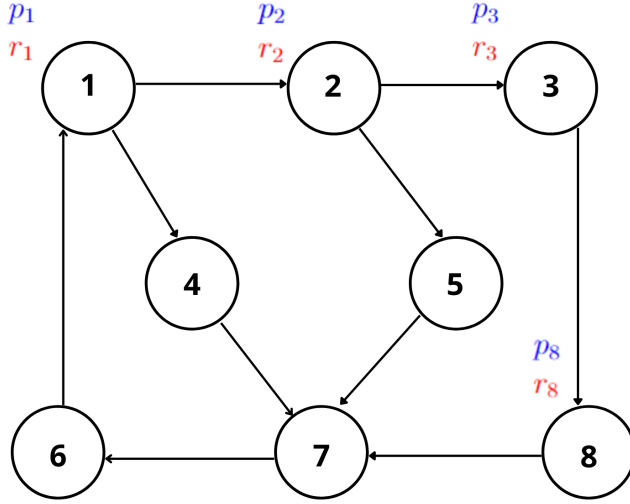


FIGURE 1 – Graphe du réseau R

Nous allons prendre l'exemple du graphe précédent pour expliquer cette méthode. Nous considérons ainsi n sommets. Dans notre exemple, les sommets i vont de 1 à $n = 8$. Nous associons deux informations à chaque sommet i : p_i et r_i . p_i correspond à l'approximation du PageRank du sommet i calculé à un certain instant et r_i correspond à son résidu, étant une quantité liée à la différence entre p_i et $\Pi_\alpha(y)$ qui correspond à la solution exacte. En effet, dans ce graphe, chaque noeud va diffuser de l'information à ses voisins, plus spécifiquement l'erreur r . Ainsi, à chaque itération, il existe une formule qui permet d'estimer une quantité proportionnelle à l'erreur entre p_i et $\Pi_\alpha(y)$ qu'on appelle r_i . Lorsque le résidu r sera inférieur à un certain epsilon ϵ , nous considérons que nous avons un PageRank calculé suffisamment proche du PageRank théorique. Cette méthode est basé sur l'invariant de boucle suivant :

$$\Pi_\alpha(y) = p^{(t)} + \frac{1}{(1 - \alpha)} \Pi_\alpha(r^{(t)}) \quad (11)$$

Cet invariant dit que s'il existe une règle de mise à jour pour p et r tout en satisfaisant l'invariant, alors si nous conduisons r vers zéro, l'estimation p devrait converger vers le PageRank exact par linéarité. Pour introduire ces règles de mise à jour nous définissons les variables suivantes :

- $p^{(t)}$: Approximation p de $\Pi_\alpha(y)$ à l'instant t .
- $r^{(t)}$: Erreur lié à p et $\Pi_\alpha(y)$ à l'instant t .

A chaque itération, les deux équations suivantes vont vérifier l'invariant de boucle :

$$p^{(t+1)} = p^{(t)} + r_u^{(t)} \delta_u \quad (12)$$

$$r^{(t+1)} = r^{(t)} - r_u^{(t)} \delta_u + \alpha P^T r_u^{(t)} \delta_u \quad (13)$$

Le second membre correspondant à la diffusion, α et P , matrice de transition du graphe, définis précédemment et δ correspondant à un vecteur colonne constitué de zéros sauf en u où il contient 1. Ainsi, à chaque itération nous mettons à jour p et r associé à un seul sommet.

Si on considère notre exemple 1, partons de p_1 . Nous aurions pu choisir un autre sommet de départ. À la première étape, p_1 va diffuser r_1 à p_2 et à p_4 . Puis, une mise à jour s'effectue pour p_2, r_2, p_4, r_4 et p_1 correspondant à ses informations à l'instant $t + 1$ ainsi r_1 est égale à 0 mais il y aura les résidus des sommets adjacents qui vont s'ajouter à r_1 en continuant l'algorithme. En effet, nous recommençons en choisissant un autre sommet aléatoirement jusqu'à trouver le résidu r pour lequel notre méthode converge.

Il est facile de voir que l'initialisation suivante $p^{(0)} = 0$ et $r^{(0)} = (1 - \alpha)y$ satisfait l'invariant. Ainsi, nous utilisons cette initialisation pour la méthode.

Le pseudo-code de cette méthode est accessible dans l'annexe 2.

3.b Comparaison entre les deux algorithmes

Nous modifions nos deux algorithmes précédents pour comparer la vitesse de convergence de Power Iteration et Forward Push tout en faisant varier les valeurs de α et de γ qui correspond à l'erreur entre les π_n et π_{n-1} .

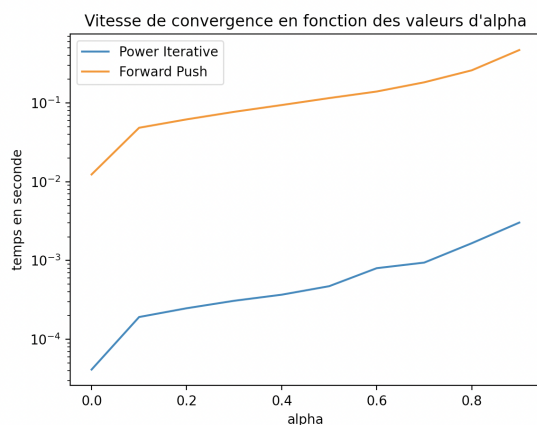


FIGURE 2 – Comparaison de la vitesse de convergence avec $\gamma = 1 \times 10^{-10}$

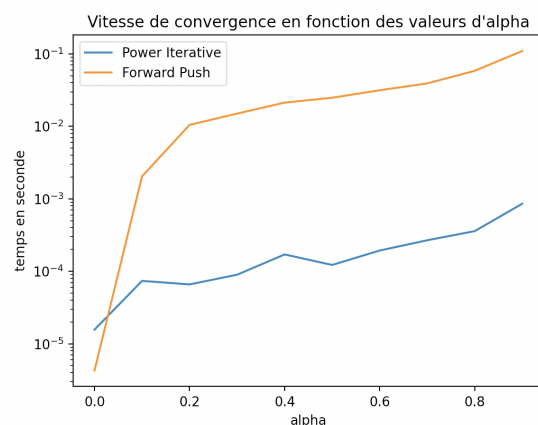


FIGURE 3 – Comparaison de la vitesse de convergence avec $\gamma = 1 \times 10^{-3}$

Nous voyons d'après ces deux graphiques que plus la valeur de α augmente plus le temps de convergence aussi. De plus, on remarque que plus l'erreur γ est grande plus le temps de convergence en fonction des valeurs de α est rapide. Au niveau des deux algorithmes, nous pouvons constater que dans notre contexte de moteur de recherche, l'algorithme Forward Push est plus lent à converger que l'algorithme Power Iteration. Cependant, nous voyons que dans le graphique de droite (Fig.2) pour une valeur α très petite Forward Push converge plus rapidement que Power Iteration.

Nous allons donc prioriser l'algorithme Power Iteration pour notre moteur de recherche. Cependant dans d'autres contextes où nous n'avons pas assez de mémoire dans le système, la méthode du Forward Push est plus adaptée vu qu'elle ne nécessite pas que tout le graphe soit chargé en mémoire.

4 Applications

Dans cette partie, nous allons présenter et expliquer le cheminement pour réaliser nos deux applications au calcul du PageRank. La première étant la catégorisation des sites du réseau et la seconde étant le moteur de recherche.

4.a Scraping de Wikipédia

La première partie de notre projet consiste à effectuer le scraping de Wikipédia. Cela permet de récolter l'ensemble des pages présentes sur Wikipédia. Notre réseau est représenté par un graphe dont les pages constituent les sommets de celui-ci. Une arête entre deux sommets correspond à un hyperlien qui relie deux pages. L'action de "scrapper", dans notre cas, correspond donc à récupérer l'ensemble des arêtes et des sommets composant notre graphe. En d'autres termes, nous récupérerons toutes les pages wikipédia que nous stockons dans un fichier afin d'avoir "une base de données" sur laquelle nous pouvons effectuer le moteur de recherche et la catégorisation des sites.

Méthode :

Dans un premier temps, il nous fallait récupérer les pages Wikipédia ainsi que celles vers lesquelles elles mènent. Pour cela, nous avons utilisé l'API *wikipediaapi* de Python qui permet d'aller chercher, à partir d'un titre de page, des données de pages Wikipédia comme les hyperliens internes.

Nous avons alors créer un code qui permet de créer le graphe représentant le réseau de Wikipédia. Ce code stocke dans un fichier les pages qu'il visite associé à son identifiant (ID) afin d'avoir notre base de données dans un fichier. Il va aussi au fur et à mesure écrire dans un fichier les relations entre les différents sommets (pages) qui existent.

Cette procédure se fait en partant d'une page quelconque présente dans Wikipédia et l'algorithme va récupérer ses URL sortants. Ces URL sont ensuite ajoutées à la file, que l'on dépile au fur et à mesure pour créer notre graphe et l'on poursuit ainsi jusqu'à ce que la file soit vide.

Pour chaque lien interne, on enregistre les arêtes :

ID page courante → ID page sortante

Pendant le parcours de la file, on garde en mémoire un dictionnaire des pages parcourues pour ne pas avoir à les traiter plusieurs fois. Ce dictionnaire sera utilisé pour permettre le calcul du PageRank en utilisant les différentes méthodes présentées précédemment.

Problèmes rencontrés :

Au cours de notre récupération de données, nous avons eu plusieurs problèmes auxquels nous avons dû palier. Le premier a été un blocage de la part de Wikipédia dans la récupération des titres des pages. En effet, au bout d'environ 20000 itérations, notre code s'arrêtait subitement et ne permettait plus d'effectuer le scraping, comme nous pouvons le voir à la figure 7 en annexe.

Nous avons tenté de pallier à ce problème en recherchant une base de données déjà prête mais celles que nous avons trouvées ne correspondaient pas à notre modèle.

Toutefois, nous avons pu récupérer environ 71612 pages Wikipédia. Nous nous sommes donc appuyés sur cette petite base de données afin d'effectuer le moteur de recherche et les calculs de PageRank.

Une fois notre base de données récupérée, nous avons effectué le calcul du PageRank pour chacune des pages que nous avons également stocké dans un fichier pour nous simplifier la tâche lors de l'écriture du script pour le moteur de recherche.

Nous sommes ensuite passé à l'écriture du script Bash pour la création de notre moteur de recherche. L'écriture du script s'est faite exclusivement à partir des fichiers que nous avons créés pour pouvoir manipuler les pages wikipédia. Nous avons utilisé Bash car nous souhaitions faire un moteur de recherche en ligne de commandes afin de gagner en rapidité et efficacité.

Finalement, avec cette base de données nous avons pu passer à deux applications du calcul du PageRank que nous décrivons ci dessous.

4.b Catégorisation des sites

Dans cette partie, nous allons montrer le second exemple d'implémentation du calcul du PageRank. C'est la classification des pages du réseau Wikipédia. Pour cela, nous allons nous servir de l'algorithme du PageRank Personnalisé. Nous allons commencer par expliquer ce qu'est la catégorisation des pages puis l'implémentation de l'algorithme.

Pour expliquer l'intérêt de la catégorisation, nous avons utilisé le logiciel Gephi pour construire une partie du graphe de Wikipédia centrée sur le sujet "Python" ⁴. Nous aurions pu choisir un autre domaine. Nous avons utilisé **Force Atlas 2** qui est un algorithme de spatialisation qui rassemble les noeuds qui ont des chemins courts entre eux. Nous remarquons des groupements avec beaucoup de liens qui se forment. Ainsi, si on choisit la catégorie d'un de ces noeuds elle se propagera dans le groupe par le PageRank.

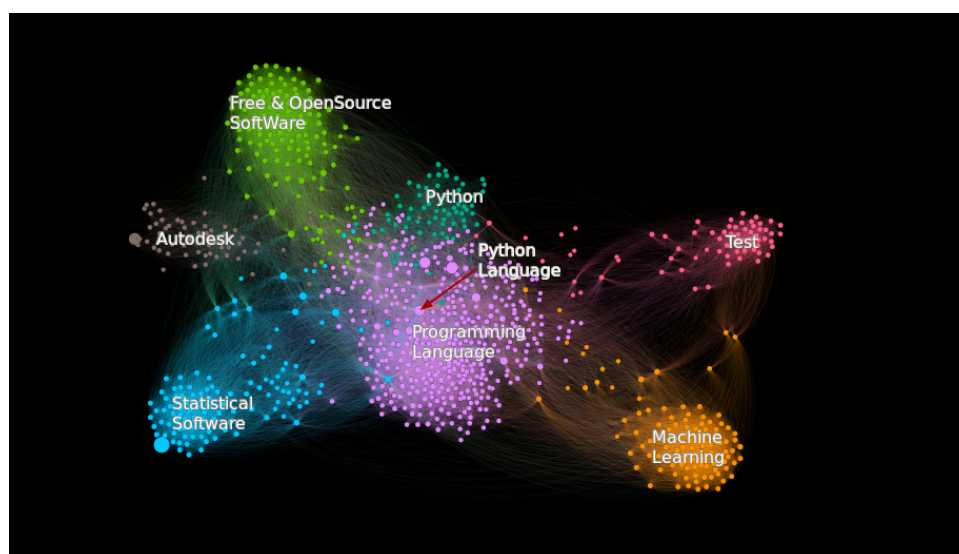


FIGURE 4 – Graphe Wikipédia du sujet "Python Programming Language"

Catégorisation des sites :

Un de nos objectifs correspond à catégoriser les pages de notre réseau. En effet, nous constatons que dans un réseau il y a des sites qui ont plus de liens entre eux que d'autres. Le but est donc de regrouper les sites entre eux en fonction des liens en commun et ainsi pouvoir les organiser. Nous souhaitons attribuer une catégorie spécifique à chaque page de notre réseau. Cela permettrait de naviguer directement dans une catégorie sans passer par tout le réseau. Pour cela nous allons modéliser notre base de données de sites en un ensemble E de sites. Lors de la catégorisation, nous allons diviser cet ensemble E en sous ensembles e_i , avec i allant de 1 à N , N étant le nombre total de catégories. Chaque sous-ensemble e_i correspond à une catégorie différente. Il est important de préciser que chaque site ne peut appartenir qu'à une seule catégorie, c'est à dire que les sous-ensembles e_i seront définis tel que l'ensemble E soit une réunion de sous-ensembles e_i deux-à-deux disjoints. C'est-à-dire :

$$E = \bigcup_{i=1}^N e_i, \quad e_i \cap e_j = \emptyset \quad \text{avec} \quad i \neq j \quad \text{tel que} \quad \{i, j\} = \{1, \dots, N\}$$

Implémentation de l'algorithme du PageRank Personnalisé :

Tout d'abord, l'utilisateur choisira les catégories qui seront utilisées pour la catégorisation des sites, c'est-à-dire les catégories dans lesquelles l'algorithme classifiera les sites. Nous avons crée par défaut une catégorie "Autres". En effet, il se peut que dans les sites choisis pour être catégorisés par l'utilisateur lors de la catégorisation manuelle, celui-ci considère qu'il y en ait certains qui n'appartiennent à aucune des catégories que lui-même a choisi au préalable. Ainsi, il pourra toujours les mettre dans une catégorie "Autres". L'algorithme va échantillonner aléatoirement les sites à catégoriser par l'utilisateur.

Pour l'implémentation, nous avons décidé de choisir 20 sites aléatoires. C'est un nombre qui permet à l'utilisateur de catégoriser les sites un par un et aussi d'avoir une catégorisation correcte. Celle-ci, serait plus précise avec un nombre plus grand de sites pour la catégorisation manuelle.

Ainsi, après cette catégorisation manuelle, nous obtenons un dictionnaire avec les 20 sites organisés en fonction des catégories, qui sont les clés du dictionnaire. En catégorisant les sites en fonction des choix de l'utilisateur, le dictionnaire dans lequel à chaque catégorie on associe un sous ensemble de sites ne dépend que du critère de l'utilisateur. Puis, nous allons utiliser ce critère pour donner une catégorisation générale de tous les sites du réseau.

Pour cela, nous calculons le PageRank Personnalisé sur tout le réseau un nombre N de fois égal au nombre total de catégories. Dans chaque calcul de PageRank Personnalisé pour chaque catégorie, le vecteur y correspond aux sites classifiés comme appartenant à cette catégorie par l'utilisateur. C'est la personnalisation. Ainsi, y change pour chaque calcul de PageRank Personnalisé, en fonction de la catégorie. Lors du calcul, nous sauvegardons toutes les valeurs des PageRank pour tous les sites pour toutes les catégories.

Enfin, nous réalisons une comparaison des valeurs obtenues. En effet, nous comparerons tous les PageRank d'un même site, obtenus pour chaque catégorie. Puis, nous choisissons la catégorie pour laquelle la valeur du PageRank d'un site est maximale et nous associons cette catégorie au site. Nous faisons cela pour tous les sites. Au final, nous obtenons un nouveau dictionnaire qui a pour clés les pages et à chaque page est associé sa nouvelle catégorie. Pour avoir tout ça de façon plus accessible, nous importons les données dans un fichier `.csv`.

Exemple de catégorisation :

Prenons un exemple avec 10 sites à personnaliser et 4 catégories qui sont mathématiques, informatique, physique et autres et la catégorisation manuelle ci-dessous :

```
Nom de la catégorie ? :mathématiques
Nom de la catégorie ? :informatique
Nom de la catégorie ? :physique
Les différentes catégories sont
{'mathématiques': 0, 'informatique': 1, 'physique': 2, 'Autres': 3}
Rentrez l'entier associé
Pour le site: Portal:Computer programming
Dans quelle catégorie veux-tu la mettre ? :1
Pour le site: OpenAI
Dans quelle catégorie veux-tu la mettre ? :1
Pour le site: Autodesk Alias
Dans quelle catégorie veux-tu la mettre ? :1
Pour le site: VHDL
Dans quelle catégorie veux-tu la mettre ? :1
Pour le site: X-12-ARIMA
Dans quelle catégorie veux-tu la mettre ? :1
Pour le site: Mathematics
Dans quelle catégorie veux-tu la mettre ? :0
Pour le site: Chinese room
Dans quelle catégorie veux-tu la mettre ? :1
Pour le site: MedCalc
Dans quelle catégorie veux-tu la mettre ? :3
Pour le site: Pointer (computer programming)
Dans quelle catégorie veux-tu la mettre ? :1
Pour le site: Free software
Dans quelle catégorie veux-tu la mettre ? :1
```

FIGURE 5 – Initialisation

Après avoir effectué la catégorisation automatique voici la catégorie des 13 premiers sites parmi notre réseau composé de 71000 sites :

```

1 Python_(programming_language),informatique
2 """"Hello, World!"" program",informatique
3 3ds Max,physique
4 ?:,Autres
5 ABC (programming language),informatique
6 ADMB,Autres
7 ALGOL,informatique
8 ALGOL 68,informatique
9 APL (programming language),informatique
10 Abaqus,Autres
11 Academic Free License,informatique
12 Academic conference,Autres
13 Action selection,informatique
14 Activation function,informatique
15 Ada (programming language),informatique
16 Advanced Simulation Library,Autres
17 Adversarial machine learning,informatique
18 AlexNet,informatique
19 Alex Graves (computer scientist),informatique
20 Alex Martelli,physique
21 Algebra,mathématiques
22 AlphaFold,informatique
23 AlphaGo,informatique
24 AlphaZero,informatique
25 Alternative terms for free software,informatique
26 Amazon (company),informatique
27 AmigaOS 4,Autres
28 Amoeba (operating system),physique
29 Anaconda (installer),physique
30 Analyse-it,Autres
31 Andrew Ng,informatique
32 Anonymous function,Autres
33 Apache Groovy,physique

```

FIGURE 6 – Catégorisation

Nous pourrions dire que l'algorithme fonctionne bien sur le réseau. Mais, nous remarquons que pour quelques pages la catégorie associée a l'air fausse. Cela est dû au fait que le nombre de pages choisies aléatoirement pour la catégorisation manuelle est petit en comparaison avec le nombre total de pages. Cependant, pour être sûr de cela il faudrait regarder la catégorisation de toutes les pages pour voir si elle est correcte ou pas. Ici nous observons la catégorisation que sur 33 pages et nous n'avons pas de fonction d'évaluation pour la catégorisation.

La limite de cette méthode est qu'il nous faudrait une autre modélisation pour trouver les groupements de noeuds (4) pour identifier une catégorie.

4.c Moteur de recherche

Une des applications courante du PageRank dans la vie quotidienne est le principe du moteur de recherche.

Un moteur de recherche est un outil qui permet d'obtenir des informations à partir de mots-clés.

Dans le cadre de notre projet, nous avons décidé de mettre en place cet outil en tant qu'application de la vie courante du calcul du PageRank. Cet outil est utilisable exclusivement en lignes de commandes. Il s'adresse aux utilisateurs qui travaillent principalement depuis le terminal et qui souhaitent obtenir des informations rapidement.

Cette application se base sur notre base de données collectée. Wikipédia est une encyclopédie qui collecte près de 6 millions d'articles et donne donc un large choix d'informations disponibles. C'est un outil que nous utilisons au quotidien pour des recherches basiques et faciliter son accès depuis le terminal est un réel avantage pour l'accès rapide à l'information sans avoir à ouvrir un navigateur.

Toutefois, effectuer une recherche depuis ce grand réseau peut devenir très long et fastidieux si l'on a pas de repères. C'est dans ce cas de figure que nous voyons l'intérêt du PageRank. En effet, le PageRank est une solution pour apporter des suggestions en fonction de l'importance des pages Wikipédia.

À partir des algorithmes d'approximation du PageRank, nous pouvons donc associer une importance à chaque page du réseau. Ainsi, plus une page est importante, plus son PageRank est élevé.

Alors, pour chaque mot recherché, nous allons récupérer les pages qui ont ce mot dans leur titre. Puis, nous allons effectuer un tri en ordre décroissant du PageRank de ces pages. Cela nous permet d'effectuer une suggestion des pages les plus pertinentes en rapport avec le mot voulu. Ainsi, la page ayant un pageRank le plus élevé sera affichée en première suggestion de notre moteur de recherche.

Nous voyons alors que le PageRank est nécessaire pour avoir un moteur de recherche efficace et pertinent.

Utilisation de l'outil :

Pour utiliser notre outil, une installation préalable est nécessaire. Vous pouvez la retrouver dans la page [GitHub](#) ou encore en annexe à 6.b.

Finalement, vous pouvez utiliser la commande :

```
wikisearch [-p|-perso] <argument>
```

Ici, <argument> correspond au mot que vous recherchez sur wikipédia.

Une fois, cette commande lancée vous allez avoir 5 propositions de pages Wikipédia qui pourraient correspondre au mot que vous cherchez. Ces propositions sont effectuées en fonction du calcul du PageRank de celles-ci.

Pour avoir le résumé de la page s'afficher, vous devez taper le numéro associé au titre de la page que vous souhaitez consulter.

Une fois cela fait, un résumé de la page s'affiche sur votre terminal.

Si vous souhaitez consulter la page wikipédia, il vous suffit de taper la commande `i` sur votre terminal. Vous verrez ensuite une page wikipédia s'afficher sur le navigateur web de votre ordinateur.

Vous pouvez également faire une recherche personnalisée en tapant :

```
wikisearch [-p|-perso] <argument>
```

Vous pouvez accéder au petit exemple dans le [GitHub](#) ou encore [6.b](#)

5 Conclusion

Pour conclure, nous avons réussi à implémenter des algorithmes qui permettent d'approximer le calcul du PageRank pour de très grands graphes : Power Iteration et Forward Push. Ces algorithmes présentent tous deux des avantages et inconvénients, mais dans le cadre de notre projet nous avons vu que la méthode de Power Iteration est plus adaptée car nous pouvions nous permettre de charger la totalité des données pour gagner du temps en calcul. À partir de là, nous avons pu l'utiliser sur le réseau Wikipédia pour mettre en place nos différentes applications telles que le moteur de recherche et la classification des pages en différentes catégories.

6 Annexes

6.a Exemples et erreurs rencontrées

Voici l'erreur survenue lors du scraping de Wikipédia :

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash - PageRank + - [ ] [ ] ^ X

21250
21251
21252
Traceback (most recent call last):
  File "/usr/lib/python3/dist-packages/urllib3/response.py", line 425, in _error_catcher
    yield
  File "/usr/lib/python3/dist-packages/urllib3/response.py", line 752, in read_chunked
    self._update_chunk_length()
  File "/usr/lib/python3/dist-packages/urllib3/response.py", line 682, in _update_chunk_length
    line = self._fp.readline()
  File "/usr/lib/python3.8/socket.py", line 669, in readinto
    return self._sock.recv_into(b)
  File "/usr/lib/python3.8/ssl.py", line 1241, in recv_into
    return self.read(nbytes, buffer)
  File "/usr/lib/python3.8/ssl.py", line 1099, in read
    return self._sslobj.read(len, buffer)
socket.timeout: The read operation timed out

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/lib/python3/dist-packages/requests/models.py", line 750, in generate
    for chunk in self.raw.stream(chunk_size, decode_content=True):
  File "/usr/lib/python3/dist-packages/urllib3/response.py", line 560, in stream
    for line in self.read_chunked(amt, decode_content=decode_content):
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash - PageRank + - [ ] [ ] ^ X

urllib3.exceptions.ReadTimeoutError: HTTPConnectionPool(host='en.wikipedia.org', port=443): Read timed out.

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "main.py", line 65, in <module>
    graphel, taille=scraping('Python (programming language)')
  File "main.py", line 38, in scraping
    for p in page_obj.links:
  File "/home/tania/.local/lib/python3.8/site-packages/wikipediaapi/_init_.py", line 1096, in links
    self._fetch('links')
  File "/home/tania/.local/lib/python3.8/site-packages/wikipediaapi/_init_.py", line 1148, in _fetch
    getattr(self.wiki, call)(self)
  File "/home/tania/.local/lib/python3.8/site-packages/wikipediaapi/_init_.py", line 412, in links
    raw = self._query(
  File "/home/tania/.local/lib/python3.8/site-packages/wikipediaapi/_init_.py", line 580, in _query
    r = self._session.get(
  File "/usr/lib/python3/dist-packages/requests/sessions.py", line 546, in get
    return self.request('GET', url, **kwargs)
  File "/usr/lib/python3/dist-packages/requests/sessions.py", line 533, in request
    resp = self.send(prepare, **send_kwargs)
  File "/usr/lib/python3/dist-packages/requests/sessions.py", line 686, in send
    r.content
  File "/usr/lib/python3/dist-packages/requests/models.py", line 828, in content
    self._content = b''.join(self.iter_content(CONTENT_CHUNK_SIZE)) or b''
  File "/usr/lib/python3/dist-packages/requests/models.py", line 757, in generate
    raise ConnectionError(e)
requests.exceptions.ConnectionError: HTTPConnectionPool(host='en.wikipedia.org', port=443): Read timed out.
```

FIGURE 7 – Erreur du scraping de Wikipédia

6.b Installation de l'outil wikisearch

La partie ci-dessous permet d'expliquer comment installer l'outil que nous avons créé.

À noter : Notre outil se base sur le wikipédia anglais, les mots recherchés devront être en anglais.

Pour utiliser notre moteur de recherche, vous devez importer sur votre ordinateur de le dossier du GitHub.

Une fois cela fait, vous devez ouvrir un terminal et vous placer à l'emplacement du dossier que vous venez d'importer.

Ensuite, vous tapez la commande suivante :

```
chmod 755 wikisearch
ou
chmod +x wikisearch
```

Pour pouvoir utiliser notre outil, il faut rendre la commande globale c'est-à-dire utilisable depuis n'importe quel répertoire.

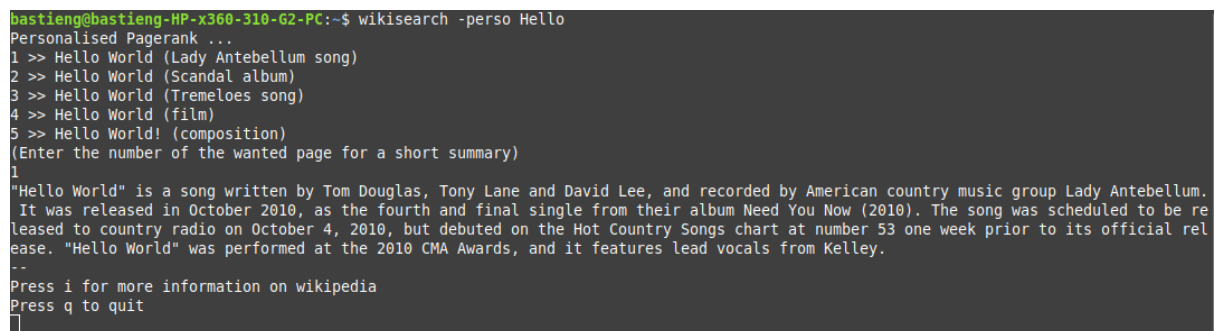
Pour cela vous devez vous taper les commandes suivantes (présentes sur le GitHub) :

```
sudo cp ./wikisearch ~/.usr/bin/wikisearch
sudo mkdir -p ~/.usr/share/wikisearch/code ~/.usr/share/wikisearch/data
sudo cp ./code/* ~/.usr/share/wikisearch/code/
sudo cp ./data/* ~/.usr/share/wikisearch/data/
```

Pour avoir plus d'informations vous pouvez installer la page man de la commande via :

```
sudo cp ./man ~/.usr/share/man/man1/wikisearch.1
```

Exemple :



```
bastieng@bastieng-HP-x360-310-G2-PC:~$ wikisearch -perso Hello
Personalised Pagerank ...
1 >> Hello World (Lady Antebellum song)
2 >> Hello World (Scandal album)
3 >> Hello World (Tremeloes song)
4 >> Hello World (film)
5 >> Hello World! (composition)
(Enter the number of the wanted page for a short summary)
1
"Hello World" is a song written by Tom Douglas, Tony Lane and David Lee, and recorded by American country music group Lady Antebellum.
It was released in October 2010, as the fourth and final single from their album Need You Now (2010). The song was scheduled to be re
leased to country radio on October 4, 2010, but debuted on the Hot Country Songs chart at number 53 one week prior to its official rel
ease. "Hello World" was performed at the 2010 CMA Awards, and it features lead vocals from Kelley.
--
Press i for more information on wikipedia
Press q to quit
█
```

FIGURE 8 – Exemple wikisearch

6.c Algorithmes

Algorithm 1 Power iteration

Data: M : matrice de transition, y : vecteur de personalisation

Result: π : vecteur contenant les valeurs du PageRank

```
 $\pi_0 \leftarrow y$   
  while  $\|\pi_n - \pi_{n-1}\| > \epsilon$  do  
     $\pi_{n-1} \leftarrow \pi_n$   
     $\pi_n \leftarrow \pi_{n-1} M$   
end  
return  $\pi$ 
```

Algorithm 2 Forward push

Data: M : matrice de transition, y : vecteur de personalisation

Result: p : vecteur contenant les valeurs du PageRank

```
 $p \leftarrow 0$   
 $r \leftarrow (1 - \alpha)y$   
  while  $\max(r) > \epsilon$  do  
     $p^{(t+1)} = p^{(t)} + r_u^{(t)} \delta_u$   
     $r^{(t+1)} = r^{(t)} - r_u^{(t)} + \alpha M^T r_u^{(t)} \delta_u$   
end  
return  $p$ 
```

6.d Bibliographie

[1] Pour une meilleure organisation, nous avons mis en place un GitHub dans lequel nous avons pu séparer notre travail en différentes branches en fonction des parties du projet que nous traitons, voici le lien du GitHub :

<https://github.com/taninhaa/PageRank>.

[2] Pour la modélisation mathématique du PageRank nous avons visionné une vidéo Youtube où le concept est bien expliqué : [Vidéo Youtube explicative du PageRank](#).

[3] Notre professeur encadrant nous a ensuite dirigé vers un cours :

<https://www.win.tue.nl/~ccomte/teaching/mdi230/pagerank.pdf>.

Ce cours nous a permis d'en apprendre plus sur l'algorithme du PageRank et la méthode itérative. Il permet de mieux visualiser la méthode mathématique à adopter et son explication.

6.e Organisation du travail

Nous avons mis en place un [GitHub](#) dans lequel nous avons pu mettre nos codes.

En fin de séance, nous faisons un point sur le travail réalisé et les difficultés que nous avons rencontrés.

Afin d'effectuer notre projet correctement, le travail s'est divisé par groupe de deux : Kevin et Paula se sont basés principalement sur les différentes méthodes pour implémenter le calcul du PageRank.

Tania et Bastien se sont concentrés sur la partie scraping de Wikipédia et mise en place du moteur de recherche.

Chaque semaine, nous fixons les tâches que nous devons réaliser pour la fin de la séance. Nous avons fixé une date limite pour finaliser les codes du projet au 13 avril 2022. Le [GitHub](#) est a été organisé en conséquence avec différentes branches pour que chaque groupe puisse avancer de son côté sur sa partie.

6.f Impact de notre travail

Nous avons essayé de faire en sorte que notre travail soit optimisé dans la gestion des fichiers les plus volumineux, cela permet une réponse plus rapide mais aussi un coût énergétique moindre avec l'utilisation au possible de langage plus adaptés comme l'awkien et le C.

Néanmoins, nous n'avons pas pu nous passer du langage python qui est plus coûteux mais qui rend l'accès plus aisé au scraping de pages web et la gestion des calculs sous forme matricielles avec numpy.

Notre travail n'entame pas les valeurs éthiques. Notre application promeut une suggestion personnalisée et un accès simplifié à l'information présente sur wikipédia. La base de donnée wikipédia étant libre de droit, il s'agit d'un intermédiaire.

6.g Bilan personnel

Bilan personnel Tania

Ce projet m'a permis de développer mes compétences en programmation notamment en écriture de scripts Bash et l'utilisation du Awkien que je ne connaissais pas. Je me suis également familiarisé avec la bibliothèque *wikipediaapi* de Python qui nous a été très utile pour le scraping de Wikipédia. J'ai trouvé intéressant de voir l'intérêt du calcul du PageRank pour la mise en place d'un moteur de recherche et son importance pour la classification des pages.

Sur le plan mathématiques, j'ai pu comprendre et mettre en pratique les connaissances théoriques que nous avons vu en cours sur les chaînes de Markov. J'ai pu voir l'importance de faire des hypothèses qui nous permettent d'assurer la convergence de notre modèle afin de créer un code qui fonctionne.

J'ai également développer des connaissances dans la mise en place d'un GitHub avec la séparation de celui-ci en différentes branches et la création d'un fichier README qui permet de présenter le sujet et de donner des indications sur l'installation de notre outil. Cette compétence est très utile pour les travaux en groupe et permet une meilleure organisation. D'un point de vu personnel, c'était une expérience enrichissante qui m'a permise de développer mon esprit d'équipe ainsi que mon organisation et ma rigueur.

Bilan personnel Bastien

Au cours de ce projet, j'ai appris à utiliser une application des chaînes de Markov en lien avec nos cours sur les processus stochastiques. J'ai aussi pu développer mes compétences en script bash avec l'utilisation d'executable python et C depuis le script pour traiter les données et une introduction au langage awkien.

De plus, tout au long des séances, j'ai su me familiariser de plus en plus aux commandes et la gestion de branche GitHub. Parmi ce qui m'a le plus intéressé, on retiendra :

- les recherches pour la création d'un script bash afin d'obtenir un résultat qui imite les commandes Unix, avec notamment l'accès à la commande en global et le manuel
- la manipulation de fichiers volumineux
- l'utilisation d'une API python pour faire du scraping et tirer notre propre base de donnée.

Ce projet m'a permis de mettre en lien plusieurs langage (Python, C, Awkien, Bash) afin d'obtenir une application viable et optimiser dans un souci d'optimisation pour ne pas être ralenti par la taille des bases de données. Il m'a aussi fait approfondir des acquis sur les commandes Unix et GitHub.

Bilan personnel Paula

D'une part, lors de ce projet, je me suis retrouvée à travailler sur une application réelle à l'interface entre les mathématiques et l'informatique. Ça a été l'occasion d'implémenter le cours de Processus Stochastiques et d'approfondir mes connaissances sur les différentes méthodes de calcul de la loi stationnaire Π , par exemple. De plus, lors des implémentations numériques, j'ai développée le fait de toujours chercher à minimiser le coût des algorithmes et comment le faire avec les outils à disposition, les bibliothèques de Python dans ce cas.

D'autre part, ça a été l'occasion de se confronter au travail en équipe et surtout à la répartition des tâches. En effet, en évaluant en rétrospective cela, le projet n'aurait pas pu avancer à la même vitesse si on ne s'aurait pas divisé en deux équipes. Cela a même permis à chaque équipe de maîtriser dans sa totalité sa partie du sujet. Cette division du travail a nécessité d'apprendre à bien utiliser GitHub pour ensuite mettre tout en commun. Enfin, j'ai appris que la communication est essentielle, car de cette manière les codes créés pouvaient être ou pas, appliqués avec la base de données obtenue. Ainsi, une bonne définition des structures de données pour les algorithmes de calcul au début permettait de gagner du temps et évitait le fait d'avoir des algorithmes qui ne marchent pas sur notre base de données.

Bilan personnel Kévin

Personnellement, je trouve que ce projet m'a permis d'approfondir et de développer mes compétences en programmation Python notamment avec l'utilisation de la bibliothèque numpy et la découverte de la bibliothèque sparse qui nous a été utile pour la modélisation des matrices de transition.

J'ai aussi pu enfin mettre en parallèle ce que j'apprends en cours à une application de la vie réelle avec la mise en pratique de la théorie des chaînes de Markov pour notre algorithme de PageRank.

Enfin, au niveau de l'organisation et le travail d'équipe, j'ai pu approfondir mes connaissances sur le GitHub notamment la séparation de celui-ci en plusieurs branches en fonction des différentes tâches ainsi que l'importance d'avoir un planning qui donne une idée de ce que l'on doit faire à chaque séance ainsi que les différentes deadlines.

6.h Remerciements

Nous tenons à remercier Mme Braunstein qui nous a permis d'effectuer ce projet durant notre cursus académique en lien avec les Mathématiques Appliquées et l'Informatique. Cela nous a permis de développer de nouvelles compétences informatiques, ainsi que d'approfondir des connaissances théoriques en lien direct avec nos cours.

Nous tenons à remercier particulièrement M. Bautista Ruiz qui nous a proposé ce sujet et qui nous a totalement intégrés dans la réalisation de celui-ci. Nous ne pouvons être que reconnaissants de son travail et de l'aide qu'il nous a apporté pour mener à bien le projet. Cela a permis à tous les membres du groupe de se sentir totalement investis dans le travail. En effet, son implication et sa bonne humeur ont été source de motivation pour nous et nous ont poussé à vouloir bien faire.

Il a su nous guider et nous remettre en question, ce qui nous a permis d'avancer et de découvrir de nouvelles applications et méthodes que nous ne connaissions pas.

Enfin, c'est grâce à M. Bautista Ruiz que nous avons pu poser une stratégie de travail d'équipe claire dès le début. De cette façon, nous avons réussi à suivre les différentes étapes et mener le projet jusqu'au bout.