Licence Mathématiques & Applications

Projet d'Analyse numérique : 2017/2018

Rapport de projet : L'algorithme PageRank : Théorie & Implémentation en Scilab

---

Auteurs :
El Mehdi Bouchouat
Ayoub Abraich

Enseignant :
M. Vincent Torri

17 juin 2018

Table des matières

Introduction

Les moteurs de recherche sont d'énormes facteurs de pouvoir sur le Web, guidant les gens vers l'information et prestations de service. Google est le moteur de recherche le plus performant de ces dernières années, ses résultats de recherche très complets et précis. Quand Google était une recherche précoce projet à Stanford, plusieurs articles ont été écrits décrivant les algorithmes sous-jacents. L'algorithme dominant a été appelé PageRank et est toujours la clé pour fournir classements précis pour les résultats de recherche.

Une fonctionnalité primordiale des moteurs de recherche de pages web consiste en un tri des résultats associés à une requête par ordre d'importance ou de pertinence. Nous présentons un modèle permettant de définir une quantification de cette notion (Pagerank) a priori floue et des éléments de formalisation pour la résolution numérique du problème. On commence par une première approche naturelle non satisfaisante dans certains cas. Un raffinement de l'algorithme est donc introduit pour améliorer les résultats.

Chapitre 1

Modélisation mathématique du problème :

1.1 Contexte

On choisit les graphes orientés pour modéliser les pages comme des sommets et les arcs comme des liens (si une page j contient un lien hypertexte vers une page i) orientés qui pointent entres elles.
On introduira des objets pour quantifier la notion de pertinence et d'importance .
On présente ici un rappel sur des notions de la théorie des graphes :

1.2 Rappel sur la théorie des Graphes

1. Graphes orientés :
   - S : ensemble fini des sommets
   - $card(S) := n$
   - A : ensemble des couples ordonnées de sommets (s_i, s_j) dans S
   - (s_i, s_j) : arc représenté graphiquement $s_i \rightarrow s_j$
   - Vocabulaire :
   s_i : prédecesseur de s_j
   s_j : successeur de s_i
   Boucle : arc reliant un sommet à lui même

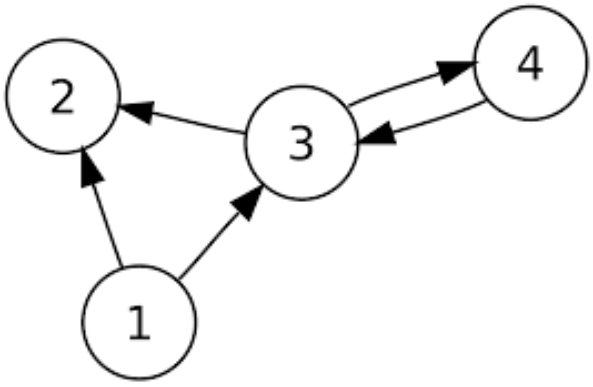

FIGURE 1.1 – Exemple de Graphe orienté

Soit le graphe $G = (S; A)$ d'ordre n. On suppose que les sommets de S sont numérotés de 1 à n.

2. Représentation des Graphes orientés :
   — Par listes d'adjancence :
   - On représente le graphe (1.1) comme une liste de liste :

$$G = [[1, 2, 3], [2], [3, 2, 4], [4, 3]]$$

De manière génerale :
$G := [[s_i, Succ(s_i)]]$ pour $i \to j$ avec la convention $[s_i, Succ(s_i)] = [s_i]$ si $Succ(s_i) = \emptyset$
- Complexité [GRA] :
• Taille de mémoire necessiare : la somme des longueurs des listes d'adjacence est égale au nombre d'arcs de A ,Par conséquent, la liste d'adjacence d'un graphe ayant n sommets et m arcs ou arêtes nécessite de l'ordre de O(n + m) emplacements mémoires.
• Opérations sur les listes d'adjacence :
il n'existe pas de moyen plus rapide que de parcourir la liste d'adjacence de T[si] jusqu'à trouver sj pour tester l'existence d'un arc (si ; sj) ou d'une arête fsi ; sjg avec une représentation par liste d'adjacence.
En revanche, le calcul du degré d'un sommet, ou l'accès à tous les successeurs d'un sommet, est très efficace : il suffit de parcourir la liste d'adjacence associée au sommet. D'une façon plus générale, le parcours de l'ensemble des arcs/arêtes nécessite le parcours de toutes les listes d'adjacence, et prendra un temps de l'ordre de p, où p est le nombre d'arcs/arêtes.
En revanche, le calcul des prédécesseurs d'un sommet est mal aisé avec cette représentation, et nécessite le parcours de toutes les listes d'adjacences de T. Une solution dans le cas où l'on a besoin de connaître les prédécesseurs d'un

sommet est de maintenir, en plus de la liste d'adjacence des successeurs, la liste d'adjacence des prédécesseurs.

— Représentation par matrice d'adjacence :

- La représentation par matrice d'adjacence de G consiste en une matrice booléenne A de taille n telle que : $a_{i,j} = 1$ si $j \to i$ et $a_{i,j} = 0$ sinon .
Exemple : on prend le meme graphe (1.1) :

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

- La compléxité :[GRA]

• Taille mémoire nécessaire :

La matrice d'adjacence d'un graphe ayant n sommets nécessite de l'ordre de $O(n^2)$ emplacements mémoire. Si le nombre d'arcs est très inférieur à n^2, cette représentation est donc loin d'être optimale.

• Opérations sur les matrices d'adjacence :

Le test de l'existence d'un arc ou d'une arête avec une représentation par matrice d'adjacence est immédiat (il suffit de tester directement la case correspondante de la matrice). En revanche, connaître le degré d'un sommet nécessite le parcours de toute une ligne (ou toute une colonne) de la matrice. D'une façon plus générale, le parcours de l'ensemble des arcs/arêtes nécessite la consultation de la totalité de la matrice, et prendra un temps de l'ordre de n^2. Si le nombre d'arcs est très inférieur à n^2, cette représentation est donc loin d'être optimale.

1.3 Retour au problème :

1. Méthode d'implémentation d'un graphe orienté en Scilab :
   On choisit le type liste de liste pour modéliser le graphe orienté pour raison de optimalité/complexité. En plus , une liste est ordonnée ce qui la différencie d'un ensemble .

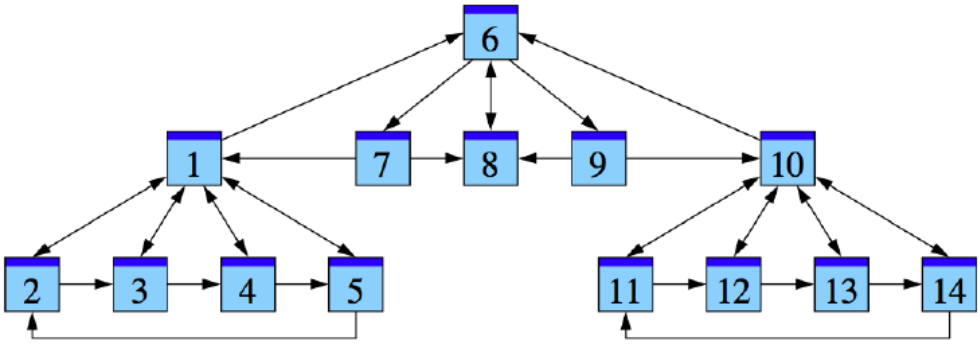

FIGURE 1.2 – Le Graphe orienté à implémenter

- En Scilab :

```
G=list([1,2,3,4,5,6],[2,1,3],[3,1,4],[4,1,5],[5,1,2],[6,7,8,9],
[7,1,8],[8,6],[9,8,10],[10,6,11,12,13,14],[11,10,12],[12,10,13],
[13,10,14],[14,10,11])
```

2. Implémentation de la fonction graphtomat(G) :
   Syntaxe : prenant en argument un graphe G du type liste de liste et renvoyant la matrice adjacente de G

-Pseudocode :

Data: Le Graphe G de type liste de liste
Result: La matrice d'adjacence A associée à G
`/* On initialise A à la matrice nulle */`
n=taille(G);
A=Zeroes(n) ;
for *liste in G* **do**
  `/* extraction de la tete/queue de la liste */`
  tete=liste[0] ;
  queue=liste[1 : :] ;
  for *i in queue* **do**
    if *queue non vide* **then**
      A[i,tete]=1 ;`/* sinon A[i,tete]=0 : A est initialisée à la matrice nulle */`
    end
  end
  ;
end
return A

Algorithm 1: Fonction graphtomat(G)

- En Scilab :

```
// Fonction graph_to_mat(G) //
function A=graph_to_mat(G)
    n=length(G)
    A=zeros(n)
    for j=1:n
        lj=G(j); // liste de G
        p=length(lj);
        tete=lj(1); // extraction tete et queue
        queue=lj(2:p);
        for i=1:p-1
            A(queue(i),tete)=1;
        end
    end
endfunction
// Test sur le Graphe G //
A=graph_to_mat(G)
A  =
   0.    1.    1.    1.    1.    0.    1.    0.    0.    0.    0.    0.    0.    0.
   1.    0.    0.    0.    1.    0.    0.    0.    0.    0.    0.    0.    0.    0.
   1.    1.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
   1.    0.    1.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
   1.    0.    0.    1.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
   1.    0.    0.    0.    0.    0.    0.    1.    0.    1.    0.    0.    0.    0.
   0.    0.    0.    0.    0.    1.    0.    0.    0.    0.    0.    0.    0.    0.
```

```
25   0.   0.   0.   0.   0.   1.   1.   0.   1.   0.   0.   0.   0.   0.
26   0.   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.
27   0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   1.   1.   1.   1.
28   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   0.   1.
29   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   1.   0.   0.   0.
30   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   1.   0.   0.
31   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   1.   0.
```

3. Implémentation de la fonction mat_to_graph(A) :
   Syntaxe : prenant en argument A la matrice adjacente et renvoyant le graphe associé G .
   -Pseudo-code :

Data: La matrice d'adjacence A associée à G
Result: Le Graphe G
`/* Initialisation */`
n=taille(A);
G=[];
for *j :1..n* **do**
  `/* Lj : la liste des i pour lesquels j pointe */`
  Lj=[];
  for *i :1..n* **do**
    if *A[i,j]>0* **then**
      Lj.append(i) ;
    end
  end
  G.append([j]+Lj);
end
return G

Algorithm 2: Fonction mattograph(A)

- En Scilab :

```
// fonction mat_to_graph(A) //

function G=mat_to_graph(A)
    n=size(A,"r"); // taille de A
    G=list();
    for j=1:n
        Lj=list();
        for i=1:n
            if A(i,j)>0 then
                Lj($+1)=i;
            end
        end
        G($+1)=list(j,Lj)
    end
```

```
endfunction
// Test :
--> A=[0 0 0 0;1 0 1 0;1 0 0 1;0 0 1 0]
 A  =

   0.    0.    0.    0.
   1.    0.    1.    0.
   1.    0.    0.    1.
   0.    0.    1.    0.


--> mat_to_graph(A)
 ans  =list([1,2,3],[2],[3,2,1],[4,3])
```

4. Implémentation de la fonction union_graph(G1,G2) :
   Syntaxe : prenant en argument prenant en argument deux graphes G1 et G2 et renvoyant le graphe $G1 \cup G2$.
   -Pseudo-code :

Data: Deux graphes G1 et G2 de type liste de liste
Result: Le Graphe $G1 \cup G2$
`/* Initialisation */`
G=[];
for *liste1 in G1* **do**
  `/* Parcours des graphes puis extraction de tetes/queues */`
  for *liste2 in G2* **do**
    liste1=[t1,q1];
    liste2=[t2,q2];
    `/* Appel de fonction de concaténation disjointe de deux listes */`
    q=union_disjoint(q1,q2);
    if *t1==t2* **then**
      G.append([t1]+q)
    end
  end
end
return G

Algorithm 3: Fonction mattograph(A)

- En Scilab :

```
// union_graph(G1,G2) //

function G=union_graph(G1,G2)
    G=list();
    n=length(G1);
```

```
    p=length(G2);
    for i=1:n
        for j=1:p  // Parcours des graphes et extraction de tete-queue
            l1=G1(i);
            l2=G2(j);
            t1=l1(1); // tete G1
            t2=l2(1); // tete G2
            q1=l1(2:length(l1)); // queue G1
            q2=l2(2:length(l2)); // queue G2
            q=union(q1,q2); // union disjoint
            if t1 == t2 then
                G($+1)=list(t1,q);
            end
        end
    end
endfunction
// Test :
--> union_graph(list([1,2,3,5,6],[2,2,3,6]),list([1,2,3,4,5,6],[2,6]))
 ans  =


       ans(1)


              ans(1)(1)

   1.


              ans(1)(2)

   2.    3.    4.    5.    6.


       ans(2)


              ans(2)(1)

   2.


              ans(2)(2)

   2.    3.    6.
```

5. Implémentation de la fonction nbr_page(G,j) :
   Syntaxe : prenant en argument un graphe G et j et renvoyant lj :nombre de pages

vers lesquelles pointe j.
-Pseudo-code :

Data: G : graphe de type liste de liste , j : entier
Result: lj : entier , nombre de pages vers lesquelles pointe j
`/* Initialisation */`
lj=0;
for *liste in G* **do**
  `/* Extraction de tete/queue */`
  liste=[t,q];
  if *t==j* **then**
    lj=taille(q)
  end
end
return lj

Algorithm 4: Fonction nbr_page(G,j)

- En Scilab :

```
// nbr_page(G,j) //
function lj=nbr_page(G,j)
    lj=0;
    n=length(G);
    for i=1:n
        liste=G(i); // parcours de G
        t=liste(1); // tete
        q=liste(2:length(liste)); // queue
        if t==j then
            lj=length(q);
        end
    end
endfunction
--> lj=nbr_page(G,10)
 lj  =
    5.
```

6. Implémentation de la fonction liste_nbr_page(G) :
   Syntaxe : prenant en argument un graphe G et renvoyant la liste des lj.
   -Pseudo-code :

Data: G : graphe de type liste de liste
Result: l :liste des lj
/* Initialisation */
l=[];
n=taille(G);
for *j :1..n* **do**
  /* inserer les lj en appelant la fonction nbr_page */
  l.append(nbr_page(G,j))
end
return l

Algorithm 5: Fonction liste_nbr_page(G)

- En Scilab :

```
1     // liste_nbr_page(G) //
2
3  function l=liste_nbr_page(G)
4      l=list();
5      n=length(G);
6      for j=1:n
7          l($+1)=nbr_page(G,j);
8      end
9
10 endfunction
```

Chapitre 2

Quantification de la notion de pertinence

On donne une définition naturelle de la pertinence de chaque page du graphe G obtenu à partir d'une requête. Pour chaque page i, on appelle μ_i la pertinence de la page i où $\mu = (\mu_i)$ est solution de :

$$\mu_i = \sum_{j \to i} \frac{1}{l_j} \cdot \mu_j$$

1. La pertinence de la page i est proportionnelle au nombre de liens qui pointent vers elle. Autrement dit : Plus on parle de vous, plus vous êtes quelqu'un d'intéressant et qui mérite d'être bien classé. Mais , l'intérêt des sites qui parlent de vous est également pris en compte : plus les gens qui parlent de vous sont intéressants, plus vous êtes considéré comme quelqu'un d'intéressant.
   Cet intérêt porte le nom de pagerank (PR). [PR]
2. La matrice stochastique A :

   $$A\mu = \mu \Leftrightarrow \forall i \quad \sum_j a_{i,j}\mu_j = \mu_i \Leftrightarrow a_{i,j} = \begin{cases} \frac{1}{l_j} & si \quad j \to i \\ 0 & sinon \end{cases}$$

   Donc on a :

   $$A = (a_{i,j})_{1 \leqslant i,j \leqslant n} \quad tq \quad a_{i,j} = \frac{1}{l_j} \quad si \quad j \to i \quad , 0 \quad sinon.$$

   On suppose que à priori on a : $l_j \geqslant 1 \quad \forall j \Rightarrow a_{i,j} \leq 1 \quad \forall \quad i, j$

   De plus :

$$\sum_i a_{i,j} = \sum_{i \ tq \ j \to i} a_{i,j} \ + \sum_{i \ tq \ j \not\to i} a_{i,j} = \sum_{i \ tq \ j \to i} \frac{1}{l_j} = \frac{1}{l_j} \cdot Card(\{i \ tq \ j \to i\}) = \frac{l_j}{l_j} = 1$$

La matrice A est une matrice stochastique (les $a_{i,j}$ sont des probabilitées !)

3. Implémentation de la fonction graph_to_stoch(G) :
   Syntaxe : prenant en argument un graphe et renvoyant la matrice stochastique A associée.
   -Pseudo-code :

Data: G : graphe de type liste de liste
Result: A :matrice stochastique associée à G
`/* Initialisation */`
A=Zeroes;
n=taille(G);
`/* M la matrice adjacente de G */`
M=graphtomat(G);
for *j :1..n* **do**
  lj=nbr_page(G,j) `/* appel de nbr_page */`
  for *i :1..n* **do**
    if *M[i,j]>0* **then**
      A[i,j]=1/lj `/* j→ i */`
    end
  end
end

Algorithm 6: Fonction graph_to_stoch(G)

- En Scilab :

```
// graph_to_stoch(G) //

function A=graph_to_stoch(G)
    n=length(G);
    A=zeros(n);
    M=graph_to_mat(G);
    for j=1:n
        lj=nbr_page(G,j);
        for i=1:n
            if M(i,j)>0 then
                A(i,j)=1/lj;

            end
        end
    end
endfunction
 --> A=graph_to_stoch(G)
A  =
0.    0.5   0.5   0.5   0.5   0.    0.5   0.    0.    0.    0.    0.    0.
      0.
```

```
20 0.2  0.  0.  0.  0.5  0.  0.  0.  0.  0.  0.  0.  0.
       0.
21 0.2  0.5  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
       0.
22 0.2  0.  0.5  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
       0.
23 0.2  0.  0.  0.5  0.  0.  0.  0.  0.  0.  0.  0.  0.
       0.
24 0.2  0.  0.  0.  0.  0.  0.  1.  0.  0.2  0.  0.  0.
       0.
25 0.  0.  0.  0.  0.  0.33  0.  0.  0.  0.  0.  0.  0.
       0.
26 0.  0.  0.  0.  0.  0.33  0.5  0.  0.5  0.  0.  0.  0.
       0.
27 0.  0.  0.  0.  0.  0.33  0.  0.  0.  0.  0.  0.  0.
       0.
28 0.  0.  0.  0.  0.  0.  0.  0.  0.5  0.  0.5  0.5
     0.5  0.5
29 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.2  0.  0.  0.
       0.5
30 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.2  0.5  0.  0.
       0.
31 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.2  0.  0.5  0.
       0.
32 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.2  0.  0.  0.5
     0.
```

Chapitre 3

Marche aléatoire sur un graphe : une première méthode itérative

Cette approche de la pertinence des pages dans un graphe mène à la question : existe-t-il une solution de $A\mu = \mu$ (1) et est-elle unique, en un sens à spécifier ?
D'autre part, il est à noter que le résultat d'une requête peut contenir un très grand nombre n de pages et le système (1) est de dimension n^2 . Il est donc nécessaire d'utiliser un algorithme performant pour calculer le vecteur de pertinence de ces pages en un temps raisonnable, voire court.

1. Soit A une matrice stochastqiue :
   Comme $\sum_i a_{i,j} = 1$, on obtient : $A^t u = u \quad u = (1, \cdot\cdot, 1)^t$
   Donc , $1 \in Sp(A^t) = Sp(A)$
2. On suppose que la multiplicité de la valeur propre 1 est $m_1 = 1$
   Or on sait que $1 \leq dim E_1 \leq m_1 = 1$, donc le sous espace caractéristique E_1 est une droite : $E_1 = \mathbb{R}u$ avec u vecteur stochastique car $Au = u$. Si u, v deux vecteurs stochastiques tels que : $Au = u \quad et \quad Av = v$ on a : $\exists \lambda \in \mathbb{R} : u = \lambda v$ ce qui implique que $\lambda = 1$, d'ou l'unicité !
3. On sait que $\rho(A) := max_{\lambda \in Sp(A)} |\lambda| \quad$ donc $\rho(A^t) = \rho(A) \leq max_j \{\sum_i a_{i,j}\} = 1$, d'autre part $\rho(A) \geq 1$

Afin de comprendre les conditions sur le graphe de pointage pour qu'il existe un unique vecteur stochastique solution de (1), nous introduisons la notion de matrice irréductible et le théorème de Perron-Frobénius.

Définition 3.1. Matrice positive : Une matrice positive $A \in M_n(\mathbb{R}^+)$ est dite irréductible si $(A + I)^{n-1}$ est strictement positive, i.e. à coefficients tous strictement positifs.

Théorème 1 (Perron-Frobenius)**.** Soit $A \in M_n(\mathbb{R}^+)$ une matrice stochastique irréductible. Alors, le rayon spectral $\rho = 1$ de A est une valeur propre simple de A et le sous-espace propre associé est une droite vectorielle engendrée par un vecteur strictement positif. De plus, la suite $(A^k)_{k\geq 0}$ converge.

Démonstration. Voir Page 73 [PRO] □

On peut interpréter la composante u_j d'un vecteur stochastique u comme une probabilité de se trouver sur une page j du graphe et $a_{i,j}$ comme une probabilité d'aller de la page j à la page i en suivant un des l_j liens. Ainsi, l'évènement "être sur la page j" est alors représenté par le j-ème vecteur de la base canonique de $\mathbb{R}^n$ noté e_j .
En partant de $x^{(0)} = e_j$, la composante $x_i^{(1)}$ de $x^{(1)} = Ax^{(0)}$ est donc la probabilité de se trouver sur la page i du graphe en partant de la page j en suivant la loi de probabilités définie par la matrice A. On introduit naturellement la méthode itérative :(Power Method)

$$\begin{cases} x^{(0)} \ \ vecteur\ stochastique \\ \qquad x^{(k+1)} = Ax^{(k)} \end{cases}$$

(4)

1. Notons : $B := A + I$, on remarque que B est exactement la matrice associée au graphe G avec les sommets pointent vers eux meme (boucles), donc on peut intérpreter la i-ème composante Be_j comme la probabilité de se trouver sur la page i du graphe G en partant de la page j en suivant la loi définié par A avec la probabilité de $i \to i$ =1 . Idem , $B^k e_j$ fournit, après k itérations, le vecteur des probabilités de se trouver sur chaque page du graphe avec la probabilité de $i \to i$ =1 .On remarque aussi que Be_j est le j-ème vecteur colonne de B .

```
--> B=A+eye(14,14)
 B  =
1.    0.5   0.5   0.5   0.5   0.  0.5   0.   0.    0.    0.    0.    0.    0.
0.2   1.    0.    0.    0.5   0.  0.    0.   0.    0.    0.    0.    0.    0.
0.2   0.5   1.    0.    0.    0.  0.    0.   0.    0.    0.    0.    0.    0.
0.2   0.    0.5   1.    0.    0.  0.    0.   0.    0.    0.    0.    0.    0.
0.2   0.    0.    0.5   1.    0.  0.    0.   0.    0.    0.    0.    0.    0.
0.2   0.    0.    0.    0.    1.  0.    1.   0.    0.2   0.    0.    0.    0.
0.    0.    0.    0.    0.   0.33 1.   0.   0.    0.    0.    0.    0.    0.
0.    0.    0.    0.    0.   0.33 0.5  1.   0.5   0.    0.    0.    0.    0.
0.    0.    0.    0.    0.   0.33 0.   0.   1.    0.    0.    0.    0.    0.
0.    0.    0.    0.    0.    0   0.   0.   0.5   1.    0.5   0.5   0.5   0.5
0.    0.    0.    0.    0.    0.  0.   0.   0.    0.2   1.    0.    0.    0.5
0.    0.    0.    0.    0.    0.  0.   0.   0.    0.2   0.5   1.    0.    0.
```

```
 0.    0.    0.    0.    0.    0.  0.    0.   0.    0.2   0.    0.5   1.    0.
 0.    0.    0.    0.    0.    0.  0.    0.   0.    0.2   0.    0.    0.5   1.
--> ej=[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
--> B*ej' = [1 0.2 0.2 0.2 0.2 0.2 0 0 0 0 0 0 0 0]'
--> B*B*ej' = [1.4 0.5 0.5 0.5 0.5 0.4 0.066 0.066 0.066 0 0 0 0 0]'
--> B*B*B*B*ej' > 0
```

2. On remarque que : à partir de $k = 4$, $(A + I)^k > 0$, donc A est irréductible !
   NB : A est irréductible équivaut à dire : De n'importe quel état, il y a une probabilité non nulle de passer d'un état à un autre.
   Si un graphe est associée à une matrice positive irréductible , on peut dire d'après Perron-Frobenius , que la pertinence de G μ existe , strictement positive et unique !

3. On a A matrice stochastique irréductible positive , alors d'après Perron-Frobenius la méthode (4) converge !

Une des caractéristiques de la matrice A est le relativement faible nombre de ses coefficients qui sont non nuls. Ainsi pour calculer l'itération (4) , on préfère, pour des raisons de performance de l'algorithme, utiliser directement le graphe G : on parcourt toutes les pages émettrices (contenant un lien vers au moins une autre page) et, pour chaque page j émettrice, on parcourt les lj pages vers lesquelles pointent j.

1. L'algorithme correspondant à la méthode décrite ci-dessus permettant de calculer $x^{(k+1)} = Ax^{(k)}$ connaissant $x^{(0)}$.

   -Pseudo-code :

Data: G : graphe de type liste de liste ; xk connu
Result: x_{k+1}
/* Initialisation */
u=[];
n=taille(G);
for *i :1..n* **do**
  /* On calcule : $x_k^{(i)} \leftarrow \sum_{j \to i} \frac{1}{l_j} \cdot x_k^{(j)}$ */
  if *pred(i) non vide* **then**
    for *j in pred(i)* **do**
      u(i)+=xk(j)/lj
    end
    ;
  end
  else
    u(i)=0;
  end
end
La fonction pred(G,i) permet de calculer la liste des predecesseurs de i .

Algorithm 7: Algorithme de calcul : $x^{(k+1)} = Ax^{(k)}$

2. Implémentation en Scilab :

```
// Fonction : calcul liste des predecesseur de i : les j tq j--> i //
function liste=pred(G,i)
    n=length(G);
    liste=list();
    for k=1:n
        L=G(k);
        for p=L(2:length(L))
            if p==i then
                liste($+1)=L(1);
            end
        end
    end
endfunction


// Fonction : Algorithme d'it ration selon la m thode d crite //
function u=iteration(G,k,xk)
    n=length(G);
    u=zeros(1,n);
    for p=1:k
        for i=1:n
            pr=pred(G,i);
```

```
23              if ~(pr==list()) then
24                  for j=pr,
25                      u(i)=u(i)+xk(j)/nbr_page(G,j) ;
26                  end
27              end
28          end
29          xk=u;
30          disp(timer());
31      end
32
33  endfunction
34
35
36
37
38  // Test : //
39  --> xk
        =[1/14,1/14,1/14,1/14,1/14,1/14,1/14,1/14,1/14,1/14,1/14,1/14
40       ,1/14]
41  --> iteration(G,6,xk)
42
43     0.2808018
44   ans  =
45
46
47            column 1 to 9
48
49     4.2076997    1.740872    1.740872   1.740872
50     1.740872    4.2397725    1.3735344    2.6707831
51     1.3735344
52            column 10 to 14
53
54     4.2076997    1.740872    1.740872    1.740872    1.740872
```

Chapitre 4

Raffinement du modèle, convergence et implémentation

Pour résoudre le problème précédemment mis en lumière, on considère un modèle plus raffiné, dépendant d'un paramètre $c \in [0,1]$ (apparemment Google utilise $c = 0,85$). A chaque itération :
— avec une probabilité $1-c$, on suit un des pointages de la page de laquelle on part en appliquant la loi de probabilités donnée par la matrice A
— avec une probabilité c, on choisit une des n pages du graphe de manière équiprobable.

Ceci évite en particulier de se faire piéger par une page sans lien vers une autre page. Plus généralement, elle garantit de pouvoir explorer toutes les pages du graphe, indépendamment des questions de connexité.
Ce nouveau modèle utilise donc la fonction de transition :

$$T : \begin{cases} \mathbb{R}^n \to \mathbb{R}^n \\ x \mapsto c\varepsilon + (1-c)Ax \end{cases} \quad (5)$$

où A est la matrice stochastique associée à (1) et $\varepsilon = (\frac{1}{n}, \cdot\cdot, \frac{1}{n})$ est le vecteur stochastique correspondant à l'équiprobabilité sur toutes les pages.

1. Interprétation de c : C'est la probabilité de choix d'une des n pages de manière équiprobable

   Notons $S := \frac{c}{n} ee^T + (1-c)A \quad tq \quad e = (1, \cdot\cdot, 1)^T$: On a S : est stochastique et irréductible . Or :

$$\mu = T(\mu) \Leftrightarrow \mu = S\mu \quad car \quad e^T\mu = 1 : \mu \quad stochastique$$

D'ou d'après Perron-Frobenius , la méthode est convergente et le vecteur stochastique μ existe et il est unique !

2. Test de convergence :
   Notons $e_n := x_{n+1} - x_n$ l'erreur d'ordre n d'aprroximation de la méthode itérative.
   On sait que la méthode est convergente $\Leftrightarrow$ $e_n \underset{n\to\infty}{\to} 0$.
   Donc on pose le test : $\|e_n\|_2 \leq \varepsilon$ avec ε fixé suffisement petit

3. L'algorithme correspondant à la méthode décrite ci-dessus permettant de calculer $x^{(k+1)} = Sx^{(k)}$ connaissant $x^{(0)}$. Le seule changement par rapport à l'algorithme précedent est sur les éléments de la matrice S :
   -Pseudo-code :

Data: G : graphe de type liste de liste ; xk connu
Result: x_{k+1}
/* Initialisation */
u=[];
n=taille(G);
for *i :1..n* **do**
  /* On calcule : $x_k^{(i)} \leftarrow \frac{c}{n} + (1-c)\sum_{j\to i} \frac{1}{l_j} \cdot x_k^{(j)}$ */
  $u(i) = \frac{c}{n}$
  if *pred(i) non vide* **then**
    for *j in pred(i)* **do**
      $u(i) += (1-c)xk(j)/lj$
    end
    ;
  end
end
La fonction pred(G,i) permet de calculer la liste des predecesseurs de i .

Algorithm 8: Algorithme de calcul : $x^{(k+1)} = Ax^{(k)}$

4. Implémentation en Scilab :

```
1
2  // Fonction : Algorithme d'iteration selon la m thode decrite //
3  function u=iteration_v2(G,k,xk,c)
4      n=length(G);
5      u=zeros(1,n);
6      for p=1:k
7          for i=1:n
8              u(i)=c/n
9              pr=pred(G,i);
10             if ~(pr==list()) then
11                 for j=pr,
```

```
                        u(i)=u(i)+(1-c)*xk(j)/nbr_page(G,j) ;
                    end
                end
            end
            xk=u;
            disp(timer());
        end

endfunction



// Test : //
--> xk
    =[1/14,1/14,1/14,1/14,1/14,1/14,1/14,1/14,1/14,1/14,1/14,1/14,1/14
     ,1/14]
--> c=0,7
--> iteration_v2(G,8,xk,c)

   2.610339

   0.020804

   0.024754

   0.020468

   0.019988

   0.020997

   0.012573

   0.025668
 ans  =


         column 1 to 6

   0.1263379    0.0515713    0.0515713    0.0515713    0.0515713    0.14258

         column 7 to 12

   0.0493917    0.0933899    0.0493917    0.1263379    0.0515713
    0.0515713

         column 13 to 14

```

```
   0.0515713   0.0515713


iteration_v2(G,8,xk,0,8)

   13.034225

   0.021205

   0.020075

   0.019682

   0.019343

   0.020311

   0.02472

   0.017762
ans =


        column 1 to 6

   0.1263379   0.0515713   0.0515713   0.0515713   0.0515713   0.14258

        column 7 to 12

   0.0493917   0.0933899   0.0493917   0.1263379   0.0515713
   0.0515713

        column 13 to 14

   0.0515713   0.0515713
```

Bibliographie

[GRA] *Quelques rappels sur la théorie des graphes.* https://perso.liris.cnrs.fr/samba-ndojh.ndiaye/fichiers/App_Graphes.pdf.

[PR] *Le PageRank : qu'est-ce que c'est ?* http://www.rankspirit.com/pagerank.php.

[PRO] *Théorèmes de Perron-Frobenius.* http://dyna.maths.free.fr/docs/lecons/developpement_algebre_533.pdf.

Résumé — Les moteurs de recherche sont d'énormes facteurs de pouvoir sur le Web, guidant les gens vers l'information et prestations de service. Google est le moteur de recherche le plus performant de ces dernières années, ses résultats de recherche très complets et précis. Quand Google était une recherche précoce projet à Stanford, plusieurs articles ont été écrits décrivant les algorithmes sous-jacents. L'algorithme dominant a été appelé PageRank et est toujours la clé pour fournir classements précis pour les résultats de recherche.

Une fonctionnalité primordiale des moteurs de recherche de pages web consiste en un tri des résultats associés à une requête par ordre d'importance ou de pertinence. Nous présentons un modèle permettant de définir une quantification de cette notion (Pagerank) a priori floue et des éléments de formalisation pour la résolution numérique du problème. On commence par une première approche naturelle non satisfaisante dans certains cas. Un raffinement de l'algorithme est donc introduit pour améliorer les résultats.

Mots clés : PageRank,Scilab

UEVE
23 Boulevard de France
91000 Évry