Projet :
Calcul efficace du PageRank

Table des matières

Table des figures

Liste des Codes

1. Introduction

1.1. But

Le but de ce projet est d'implémenter l'algorithme permettant de classer des sites internet à la manière de Google.
Dans ce projet on cherche à mettre en œuvre l'algorithme de PageRank en langage **Ada**, qui consiste, à partir d'une liste de sites internet appartenant à un même réseau, et de connexions entre ces derniers (relations de référencement, un site référence zéro ou plus autres sites) , à donner leurs « poids » respectifs, i.e. leur popularité au sein de ce réseau.

Cette implémentation sera faite de deux manières différentes :
— Une implémentation naïve avec une matrice pleine de taille $N \times N$.
— Une implémentation moins coûteuse en ressources à l'aide d'une matrices creuse.
Pour cela on codera le programme principal une seule fois, en utilisant des fonctions spécifiques à chaque implémentation.

1.2. Auteurs

Nous sommes un binome d'élèves de l'école d'ingénieur ENSEEIHT à Toulouse (École nationale supérieure d'électrotechnique, d'électronique, d'informatique, d'hydraulique et des télécommunications) en première année, en filière Sciences du Numérique (SN) du **groupe KL-02** :

- ❏ Philippe NEGREL-JERZY
- ❏ Sébastien PONT

1.3. Rappel du sujet

Voici un résumé du sujet. Pour plus de détail, se référer au sujet.
Un fichier représentant le réseau nous est donné. Voici un exemple :

```
6
0 1
0 2
2 0
2 1
2 4
3 4
3 5
4 3
4 5
5 3
```

Code 1 – Fichier réseau exemple

La ligne 1 nous indique le nombre de sites (internet) composant le réseau (ici 6).
Chaque autre ligne est composé de deux nombres :
— le 1^{er} est le site référenceur (une de ses pages internet contient un lien vers le site référencé)
— le 2^{nd} est le site référencé
Par exemple la $2^{ème}$ ligne nous dit que le site `0` fait référence aux sites `1` et `2`.
Pour savoir quel est le site le plus populaire, il faut alors calculer le nombre de référencements. On peut répertorier tous ces liens dans une matrice en mettant en ligne les référenceurs et en colonne les référencés :

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

On retrouve bien à la $1^{ère}$ ligne que le site `0` fait référence aux sites `1` et `2`.
Après quelques calculs (c.f. sujet) on retrouve une matrice G_α de paramètre $\alpha \in [0, 1]$ qui permet de nuancer la popularité d'un site, et d'avoir des résultats plus proches de la réalité. Pour calculer les poids des sites π, il suffit alors de faire :

$$\pi_{k+1}^T = \pi_k^T \cdot G$$

avec π le vecteur ligne contenant les poids des sites.

Après M itérations, le vecteur π_M est une bonne représentation de la popularité des sites dans la réalité.

2. Architecture du programme

2.1. Architecture

Avant de commencer voici la strucure globale de notre projet :

— le fichier `pagerank.adb` est notre code principal

— le module `helpers` contient toutes les fonctions tierses ne faisant pas partie intégrante de Google, pouvant s'externaliser afin de décharger le fichier `pagerank.adb`.

— de même le module `exceptions` définit toutes les exceptions du projet

— les modules `google`, `google_naive` et `google_creuse` implémentant les différantes versions du code.

L'avantage d'externaliser les modules `exceptions` et `helpers` est que l'on peut se servir de leur contenu à la fois dans le module `google` et dans le fichier `pagerank.adb`.

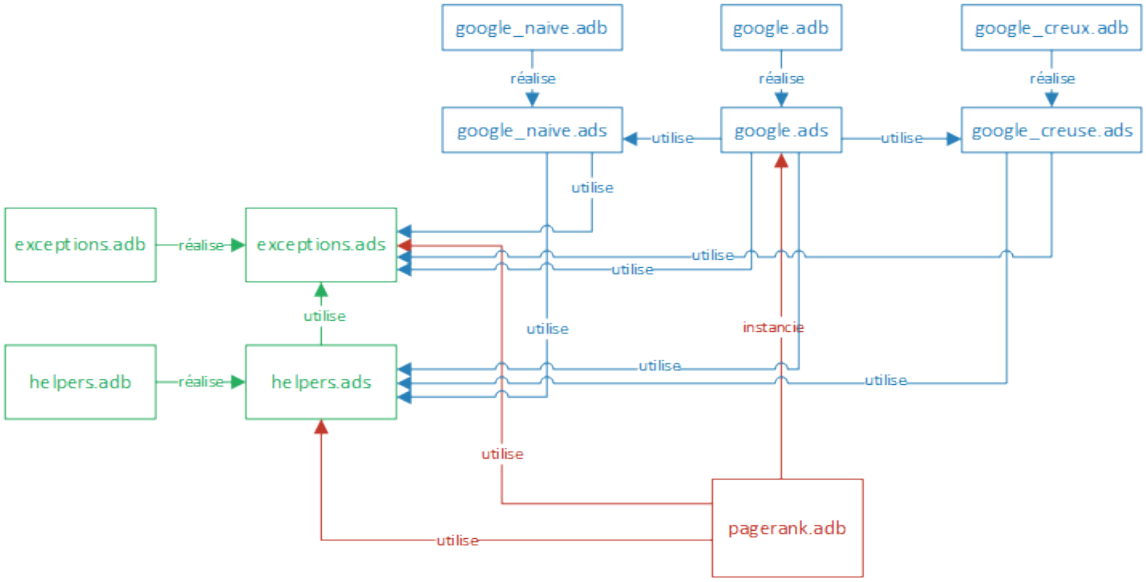

FIGURE 1 – Architecture du projet

2.2. Structure des fichiers

Pour être plus clair, nous avons voulu séparer chaque module dans un dossier différent. Pour cela il faut créer un fichier `<nom-module>.gpr` dans le dossier définissant le module :

```
1  project Nom_Module is
2      for Source_Dirs use (".");
3      for Object_Dir use "../obj";
4  end Nom_Module;
```

Code 2 – Fichier générique *.gpr définissant un nouveau module

Pour pouvoir se servir d'un module `A` dans un module `B` il faut importer le module `A` dans le fichier `module-b.gpr` :

```
1  with "<chemin-relatif-du-module>/<nom-du-module>";
2  ...
```

Code 3 – Import de module

La Figure 2 représente la structure des fichiers de notre projet. On peut remaquer qu'il y a un dossier `obj`. Lors de la compilation, le compilateur génère une multitude de fichiers... Pour éviter qu'ils ne polluent nos dossiers, on a dit à chaque module de générer ses fichiers de compilation dans un seul dossier (ligne 3 du Code 2)

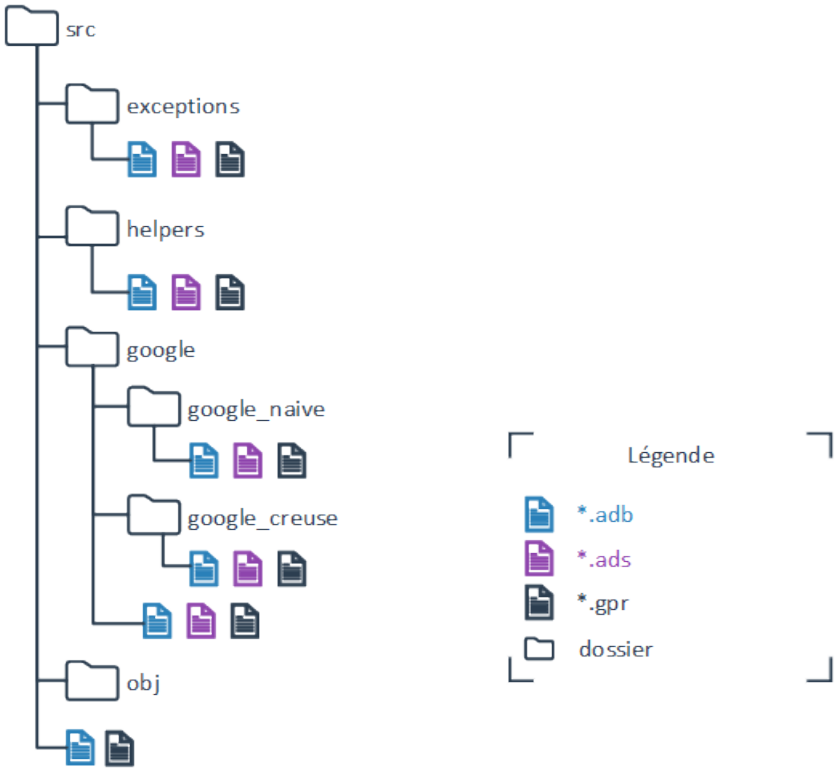

FIGURE 2 – Structure des fichiers

2.3. Héritage

Même si on ne l'a pas vu dans le cours sur la programation impérative et que ce n'est pas le but de l'exercice, comme le projet s'y prête beaucoup, on a décidé d'avoir une approche orientée objet. Cela n'était pas possible en ada, mais depuis 1995 (Ada95), le langage a été adapté pour la POO (Programmation Orientiée Objet). On a donc créé une classe mère `Google`, et deux classes filles `Google_Naive` et `Google_Creuse`. La classe mère définit toutes les fonctions et procédures, sans qu'elles ne soient implémentées (leur corps n'est composé que de la ligne `null;`). Les classes filles *override* ces fonctions et procédures. En Ada voici comment se code un *objet* :

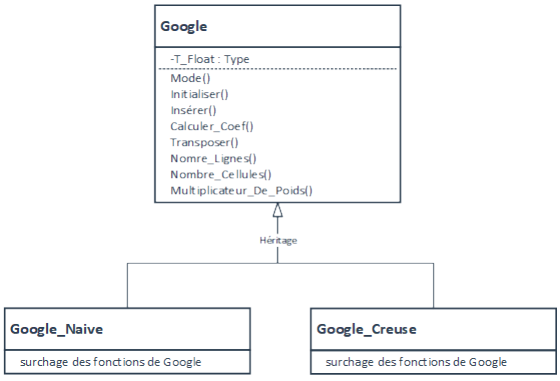

FIGURE 3 – Héritage de la classe Google

```
1   type Classe_Mere is tagged null record;
2
3   procedure Procedure_1 (Self : in out Classe_Mere);
4   procedure Procedure_2 (Self : in out Classe_Mere; Un_Argument : in Integer);
5
6   type Classe_Fille is new Classe_Mere with record
7      Parametre : Float;
8   end record;
9
10  procedure Procedure_2 (Self : in out Classe_Fille);
```

Code 4 – POO en Ada - Définition

Avec son implémentation :

```
procedure Procedure_1 (Self : in out Classe_Mere) is
begin
   null;
end Procedure_1;
procedure Procedure_2 (Self : in out Classe_Mere; Un_Argument : in Integer)

begin
   null;
end Procedure_2;
procedure Procedure_2 (Self : in out Classe_Fille; Un_Argument : in Float)

begin
   null;
end Procedure_2;

...

declare
   A : Classe_Mere;
   B : Classe_Fille;
begin
   A.Procedure_1;
   A.Procedure_2(4);

   B.Procedure_1;
   B.Procedure_2(0.59);
end;
```

Code 5 – POO en Ada - Implémentation

3. Code

3.1. Style de programmation

Nous avons choisi de coder avec une programmation défensive (avec des erreurs levées) pour plusieurs raisons :
— On est sûr que toutes les erreurs sont prises en compte, tandis qu'avec une programmation offensive, nous ne sommes pas sûrs d'avoir prévu tous les cas possibles avec les Pre/Post conditions.
— Il y a une gestion générale des erreurs, donc si une erreur Ada est levée, elle sera aussi gérée
— Les autres langages de programmation ont plus souvent une approche défensive
Voici comment nous avons procédé :

❏ Un module `excpetions` définissant toutes les erreurs du projet :

```
package Exceptions is
   ERREUR_Une_Erreur : Exception;
   ERREUR_Argument_Manquant : Exception;
   ERREUR_Mauvais_Parametre_Iteration : Exception;
   ERREUR_Mauvais_Parametre_Alpha : Exception;
   ERREUR_Mauvais_Parametre_Fichier : Exception;
   Erreur_Mauvais_Parametre_Taille_Reseau : Exception;
   Erreur_Mauvais_Parametre_Taille_Memoire : Exception;
   Erreur_Mauvais_Parametre_Taille_Hachage : Exception;
   ERREUR_Fichier_Manquant : Exception;
   ERREUR_Lire_Args : Exception;
   ERREUR_Lecture_Taille : Exception;
   ERREUR_Capacite_Max_Depassee : Exception;
end Exceptions;

```

Code 6 – Module Exceptions

Ce module est importé dans tous les autres modules du projet

❏ Lorsqu'une erreur est détectée, nous la levons avec une message explicatif

```
...
raise Nom_Erreur with "Message a transmettre a l'utilisateur.";
...

```

Code 7 – Levée d'une exception

❏ Toutes les erreurs sont propagées au programme principal `pagerank.adb` qui les affiche :

```
function Pagerank return Integer is
...
begin
...
exception
   when E : others =>
      Ada.Text_IO.Put_Line (Exception_Message (E));
      return 1;
end Pagerank;

```

Code 8 – Gestion des exceptions

3.2. Conventions

❏ **Nommage des types.** En cours nous avons toujours nommé les types `T_...`. Nous avons choisi de ne pas suivre cette convention car lorsque l'on on déclare un entier par exemple, on écrit : `I : Integer;`, et non pas `Integer : T_Integer;`. Nous avons voulu continuer dans cette logique.

3.3. Programme principal

Notre programme principal se veut le plus léger possible (c'est notamment pour ça que nous avons fait un module `helpers`), tout en implémentant le code.

❏ Définition de la fonction `pagerank` :

```
---------------- CONSTANTES ----------------
Nb_Digit     : constant Integer := 3;
--------------------------------------------

type T_Float is digits Nb_Digit;

package Matrice_Google is new Google (T_Float);
use Matrice_Google;

A : Args;

```

Code 9 – Définition de `pagerank`

Nous n'avons malheureusement pas réussi à faire passer la constante `Nb_Digit` en variable potentiellement modifiable en définissant un paramètre de plus dans la commandline (`./pagerank -P test.net`). Elle est donc écrite *en dur* dans le programme.

`Args` est un type défini dans le module `helpers` importé au début du fichier.

❏ **Astuce de la fonction.** Jusqu'ici tous nos programmes principaux étaient des procédures. Nous avons choisi d'en faire une fonction. De cette manière nous pouvons exécuter `return 0;` à tout moment pour stopper la fonction. Cela est utile car nous avons rajouté la possibilité de passer le paramètre `-h` à la commandline qui affiche l'aide. Si l'aide est affichée, nous ne voulons pas que le programme s'exécute, mais nous ne trouvions pas *joli* d'avoir tout le code principal englobé dans un `if`. Voici donc l'*astuce* :

```
function Pagerank return integer is
...

   A : Args;
begin

   -- Lecture des arguments
   A := Lire_Arugments;

   -- Si on a affiche l'aide, on ne fait rien d'autre
   if A.Aide_Demande then
      return 0;
   end if;

   -- Suite du code
   ...

   -- Fin
   return 0;

```

```
21 exception
22    when E : others =>
23       Ada.Text_IO.Put_Line (Exception_Message (E));
24       return 1;
25 end Pagerank;
26
```

Code 10 – Astuce de la fonction principale

❏ **Contenu du code principal.**

```
1  ...
2  declare
3     G : Google'Class := Initialiser(A);
4     P : Poids(1..A.Taille_Reseau);
5  begin
6     -- Initialiser la matrice Google
7     -- Initialiser le tableau des poids
8     -- Lire le fichier reseau
9     ...
10    while not End_Of_File (Fichier_Net) loop
11       Get (Fichier_Net, Referenceur);
12       Get (Fichier_Net, Destinataire);
13       C := C + Octets(Referenceur) + Octets(Destinataire) + 2;
14       G.Inserer(Referenceur, Destinataire, Un);
15       Tampon.Ajouter(Referenceur, Un);
16       Log_P(C, Taille_Fichier);
17    end loop;
18    ...
19    -- Calculer les coefficients de la matrice Google
20    -- Calculer les poids par iteration
21    -- Trier les poids
22    -- Exporter les poids dans les fichiers
23
24 end;
25
```

Code 11 – Contenur du programme principal

Pour voir l'intégralité du code, se référer au fichier `pagerank.adb`.
Nous n'avons pas copié tout le code du programme principal, mais nous avons laissé un bout du code pour vous montrer comment nous gérons les deux implémentations creuses et naives. Nous pouvons voir à la ligne 3 que `G` est du type `Google'Class`. En effet, la fonction `Initialiser(A)` renvoie une classe : soit `Google_Naive`, soit `Google_Creuse` en fonction des arguments `A` qui est un enregistrement, tel que `A.Est_Naif` est un `Boolean` vrai si l'implémentation dite naïve a été choisie.
Analysons maintenant la ligne 14 : la fonction `Inserer(I, J, Valeur)` dépend de la classe de `G` (`Google_Naive` /`Google_Creuse`), et son implémentation est différente :
— Un simple `Table(I, J) := Valeur;` pour la version naïve
— Un programme plus complexe pour la version creuse
De cette manière nous pouvons écrire l'intégralité du code une seule fois, et pour les parties spécifiques à la gestion des matrices G, nous avons un code différent selon le type d'implémentation.

3.4. Paramètres

La gestion des arguments (paramètres de la commandline `./pagerank [parametres]`) se fait dans le module `helpers`.

❏ **Les différents paramètres possibles.**
— `-P` : (Optionnel) Permet de choisir l'implémentation creuse de Google. Si cet argument n'est pas présent, c'est l'implémentation naïve qui est choisie.
— `-I` : (Optionnel) Doit être suivi d'un nombre entier positif. Permet de paramétrer le nombre d'itérations faites pour calculer les poids. Valeur par défaut : 150
— `-A` : (Optionnel) Doit être d'un réel. Le paramètre doit être un réel compris entre 0 et 1. Permet d'indiquer le paramètre alpha du la matrice Google.
— `-V` : (Optionnel) Active le mode verbeux. Si activé, toutes les étapes de calcul seront loguées dans la console.
— `-H` : (Optionnel) Affiche cet aide.
— `nomfichier.net` : (Obligatoire) La ligne de commande doit se terminer par le nom .net qui contient les liens entre les pages web.

❏ **Le type `Args`.** Le type `Args` est l'enregistrement suivant :

```
type T_Args is record
   Nom_Fichier    : Unbounded_String;
   A              : Float    := 0.85;
   Max_Iter       : Integer  := 150;
   Est_Naif       : Boolean  := True;
   Aide_Demande   : Boolean  := False;
   Taille_Reseau  : Positive;
end record;
```

Code 12 – Le type `Args`

- **Lecture des arguments.** La fonction `Lire_Arguments` se charge de lire les arguments un par un, et à l'aide d'un `switch` détermine si un argument est présent ou non, et éventuellement sa valeur.
- **Mode verbeux.** La fonction `Log(Message);` par exemple, n'affiche le message que si le mode verbeux à été passé en paramètre lors du lancement du programme. De même la fonction `Log_P(Progression, Maximum)` affiche une *Progress Bar*.

3.5. Google naïve

- **Un type `array`.** Comme demandé explicitement dans le sujet : « `Google_Naive` definit et manipule une matrice G sous la forme d'un tableau de reels à deux dimensions statiques ». Nous somme donc obligé de faire un tableau de dimension $N \times N$ avec N la taille du réseau. La dimension n'est pas à proprement parler *statique* car nous avons voulu que la dimension du tableau varie en fonction de N :

```
-- Google_Naive.ads
type G_N is array (Positive range <>, Positive range <>) of T_Float;
type Google_Naive(Size: Positive) is new Google with record
   Table : G_N(1..Size, 1..Size);
end record;
...
-- Google.adb
declare
   G : Google_Naive (A.Taille_Reseau);
   ...

```

Code 13 – Type de `Google_Naive`

- **Implémentation des fonctions.** La surchage des fonctions de la classe mère `Google` sont relativement simples : des boucles for imbriquées, et un accès aux données via `Table(I,J)`.
- **Mémoire.** La taille utilisée par cette matrice est rapidement très grande (N^2). Il n'est donc pas possible de traiter des fichiers réseaux trop gros avec ce type de matrice. Nous avons codé une autre implémentation de la matrice « naïve » en enregistrant son contenu au fur et à mesure dans un fichier, mais en la testant nous obtenions des fichiers de taille théorique dépassant les 300Go. (Nous n'avons pas attendu la fin de l'exécution.) Nous avons jugé inutile de continuer dans cette voie, et de laisser une erreur Stack Overflow si le réseau est trop grand.

3.6. Google creuse

- **Une table de hachage.** G doit maintenant être creuse, donc tous les 0 ne sont pas pris en compte. Pour cela nous avons choisi d'implémenter une table de hachage :

```
-- Google_Creuse.ads
...
type Cellule is record
   Index: Integer;
   Donnee: T_Donnee;
   Suivant: Ptr_Cellule;
end record;
type Ligne is record
   Index: Integer;
   Cellule: Ptr_Cellule;
   Suivant: Ptr_Ligne;
end record;
type Th is array (Positive range <>) of Ptr_Ligne;
type Table_Hachage(Size: Integer) is tagged record
   Table : Th(1..Size);
end record;

...
```

```
19 -- Google.adb
20 declare
21    G : Google_Creuse (A.Taille_Hachage);
22    ...
23
```

Code 14 – Type de `Google_Creuse`

Nous avons donc un `record` `Cellule` qui contient le coefficient de G. Tous les coefficients d'une ligne se rattachent l'un à l'autre (grâce à la propriété `Suivant`). Chaque ligne est `record` contenant une *grappe* de coefficients. Ces lignes elles-mêmes se suivent de la même manière. Enfin un tableau de taille relativement petite contient des lignes. La clé de hachage est l'index de la ligne. Ainsi, si la taille du tableau est 100, et la taille du réseau est 1000, alors, la première case du tableau contiendra les lignes d'index $0, 100, 200, 300, ..., 900$.

- ❏ **Implémentation des fonctions.** Il est maintenant plus compliqué d'implémenter les fonctions que précédemment. Par exemple la fonction `Inserer` doit insérer dans la bonne ligne si elle existe, et entre deux Cellules, si elles existent la valeur voule...
- ❏ **Transposée.** Notons que pour le calcul des poids : $\pi_{k+1}^T = \pi_k^T \cdot G$, c'est la colonne de G qui nous intéresse. C'est pourquoi, lors de la création de G, nous calculons réellement sa transposée.

4. Performances et intérêt de la méthode creuse

4.1. Temps d'exécution

Nous n'avons pour l'instant pas pu faire des tests rigoureux, mais voilà ce que nous avions obtenu lors de nos tests :

- ❏ `Google_Naive` avec un tableau dans la RAM :
  - — Le temps d'éxécution du fichier `worm.net` durait environ 20 secondes.
  - — Les fichiers `brainlinks.net` et `Linux26.net` ne peuvent pas être testés.
- ❏ `Google_Naive` avec enregistrement de fichiers :
  - — Le temps d'éxécution du fichier `worm.net` durait environ 2 minutes.
  - — Le fichier `Linux26.net` générait des fichiers dépassant les 300 Go, et des temps d'éxécution dépassant les 6 heures (dus à la lecture sur disque).
- ❏ `Google_Creuse`
  - — Le temps d'éxécution du fichier `brainlinks.net` était d'environ 5 minutes.

4.2. Mémoire

- ❏ L'implémentation de `Google_Naive` avec un tableau en mémoire RAM retournait une erreur Stack Overflow pour des réseaux dépassant une taille de 2895, ce qui représentait environ 23.4 Mo dans la mémoire RAM.

5. Obstacles rencontrés et conclusion

5.1. Obstacles

Nous avons rencontré un certain nombre de problèmes au cours de ce projet, dont voici un résumé succinct :

- ❏ `Google_Naive` avec un tableau dans la RAM : Stack Overflow
- ❏ `Google_Naive` avec enregistrement de fichiers : taille des fichiers. Solutions tentées :
  - — Enregistrement en ASCII
  - — Enregistrement en binaire
  - — Séparation en 1 fichier par colonne de la matrice
- ❏ Avoir un code propre. Solutions tentées :
  - — Tout dans le même fichier
  - — Plusieurs modules séparés
  - — Module `Google`, `Google_Naive` et `Google_Creuse` au même niveau
  - — Héritage à l'aide de la POO

5.2. Conclusion

Ce projet met en évidence la nécessité d'utiliser des matrices creuses pour résoudre ce problème. Ceci est d'autant plus vrai que la taille du plus grand réseau testé ne dépassé pas 30 000 sites, alors que de nos jours, il y a près de 2 milliards de sites sur internet.