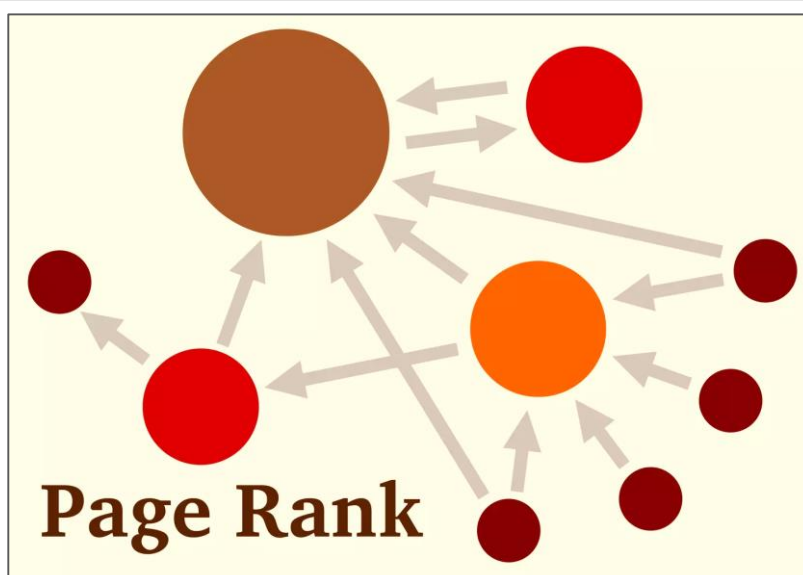


1ème Année Master Informatique, Semestre 2

Module : Méthodes de ranking

Sujet 1 : *Utilisation du pagerank précédent pour initialiser un nouveau calcul-graphe Erdos*



Rapport réalisé par :

SARAH OUHOCINE (AMIS)

DJILLALI BOUTOUILI (AMIS)

ABDELOUAHAB TAFAT (AMIS)

KHALED MEBHAH (AMIS)

ABDELGHANI SEDJAL (AMIS)

Projet proposé par :

M. JM FOURNEAU

SOMMAIRE

1. Introduction	
2. Implémentation	
2.1 Données	
2.2 Nouvel Algorithme (Programme)	
2.2.1 Structure de données	
2.2.2 Convergence	
2.2.3 PageRank	
2.2.4 Erdos	
2.3 Exécution	
2.3.1 Inputs	
2.3.2 Outputs	
3. Analyse Des Résultats	
3.1 Métriques d'analyse	
3.2 Résultats numériques et Graphes	
3.2.1 Graphe du Web 1	
3.2.2 Graphe du Web 2	
3.2.3 Graphe du Web 3	
3.2.4 Graphe du Web 4	
3.2.5 Graphe du Web 5	
3.2.6 Graphe du Web 6	
3.3 Interprétation des résultats	
4. Conclusion	
5. Bibliographie	
6. Annexe	

1. Introduction

PageRank est un algorithme d'analyse des liens, utilisé par Google pour le classement des pages web après une requête de recherche sur Google.

Nommé après le co-fondateur du Google *Larry Page*, cet algorithme estime l'importance d'une page web en se basant sur le nombre et la qualité des liens vers cette dernière, démarrant de l'hypothèse : plus la page reçoit des liens, plus son importance augmente (Démocratie du Web).

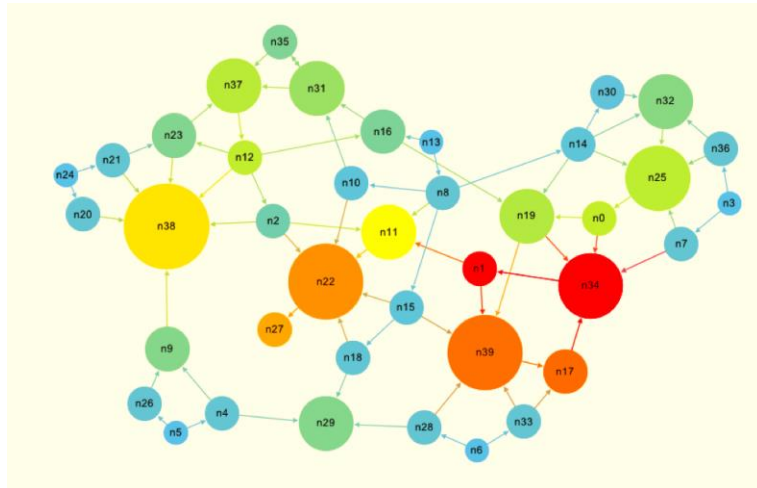


Figure 1- Exemple d'un graphe du web avec des liens utilisés par PageRank pour le classement.

Les résultats de PageRank sont calculés itérativement, au début (temps t), toutes les pages n ont le même score $1/n$, dans la prochaine itération (temps $t+1$), le score de la i -ème page est calculé comme suit :

$$p_{i,t+1} = \frac{1-d}{n} + d \sum_{j=1}^n A_{ij} P_{j,t}$$

Avec :

- d : Le facteur d'amortissement ou le dumping factor.
- n : Le nombre des pages à classer.
- A_{ij} : La probabilité de transition de la page i vers la page j , en temps donné t (valeurs données par la matrice de transition).

Évolution des graphes du web :

Il est connu que le graphe du web évolue avec le temps (création de nouvelles pages et des nouveaux liens), or que PageRank est calculé chaque mois, il faut prendre en considération donc l'évolution du graphe du web en calculant le score (pertinence) des nouvelles pages.

Pour rajouter les probabilités de transition pour ces nouvelles pages (sommets / nœuds), ces dernières doivent être incluses dans la matrice de transition. Le problème qui se pose pour de tels nouveaux sommets dans le graphe du web est l'initialisation des probabilités de ces derniers.

Une des solutions proposées c'est l'utilisation d'un modèle de graphe aléatoire pour la génération des nouveaux sommets et l'affectation des nouvelles probabilités de transitions.

Modèle de graphe aléatoire :

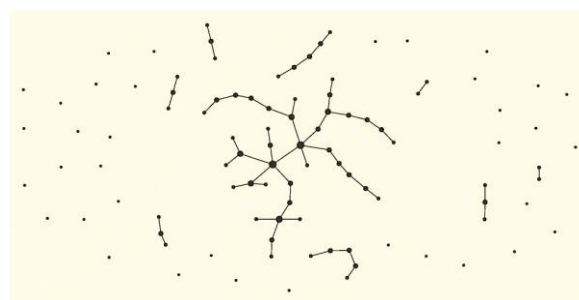
Un modèle de graphe aléatoire est une collection \mathbf{G} de graphes et une loi de probabilité \mathbf{P} sur cette collection \mathbf{G} .

Le modèle de graphe aléatoire le plus simple est le modèle Erdos, introduit dans les années 1950 par Erdős and Rényi.

Erdos :

Il s'agit de la collection $\mathbf{G}(\mathbf{n}, \mathbf{M})$ de tous les graphes simples non dirigés d'ordre n et de taille M , munie de la loi uniforme \mathbf{P} sur cette collection. Ainsi, la collection $\mathbf{G}(\mathbf{n}, \mathbf{M})$ contient (nM) graphes différents.

Nous allons utiliser le modèle Erdos pour la génération des nouveaux nœuds et arcs aléatoirement dans le graph, muni des probabilités.



***Figure2-** Graphe aléatoire généré par Erdos avec la loi de probabilité binomiale ($p=0.01$)*

Dans notre solution nous allons utiliser ($p=0.0$) pour les nouveaux sommets, comme demandé.

2. Implémentation

Nous allons discuter dans cette section l'implémentation de l'algorithme PageRank avec la génération des nouveaux nœuds en utilisant le modèle Erdos.

2.1 Données

Nous allons utiliser les données (Graphes du Web) présentes dans le site de l'ENT.

Les tests sont exécutés sur les Graphes du Web (dataset) suivants :

Nom du graphe du web	Nombre de sommets	Nombre d'arcs	Taille en Mo
1 - wb-cs-stanford	9914	36854	0.6
2 - Stanford	281903	2312497	37
3 - Stanford_BerkeleyV2	68344	7583376	120
4 - in-2004v2	1382908	16917053	277
5 - wikipedia20051105V2	1634989	19753078	309
6 - wb-edu	9845725	57156537	1060

2.2 Nouvel Algorithme (Programme)

Le programme utilise le pagerank précédent (vu en cours) pour initialiser un nouveau calcul-graphe Erdos. Par ailleurs, la représentation de la matrice de transitions comme vue en TP consomme beaucoup d'espace mémoire bien que cette dernière est une matrice creuse. Ce qui empêche l'algorithme de s'exécuter sur les grands graphes du web (notamment wb-edu). Nous avons eu besoin donc de ré-implémenter la représentation de la matrice d'une façon à optimiser plus l'espace mémoire, en prenant avantage du fait que cette dernière soit aussi une matrice creuse.

2.2.1 Structure de données

Nous avons utilisé la représentation **CSR** (Compressed Sparse Row), qui est une représentation de matrices très utile dans le cas des matrices creuses de très grande taille. En effet, Pour minimiser la mémoire occupée par la matrice, seuls les coefficients non-nuls sont stockés.

Ceci permet également d'améliorer notablement les performances du produit vecteur-matrice, en passant d'une complexité de $O(N^2)$ à $O(\text{nombre de coefficients non-nuls})$.

CSR utilise 3 vecteurs (à 1 seule dimension) pour représenter une matrice :

- ✓ Deux vecteurs d'entiers (`row_index` et `col_index`), souvent classés de sorte que `row` est croissant et `col` est « croissant par morceaux » (ou « croissant par ligne »).
- ✓ Un vecteur (`V` ou `val`) contenant les valeurs non-nulles de la matrice.

Plus précisément :

→ **val** : contient les valeurs non-nuls de la matrice, lues de gauche à droite, ligne par ligne de haut en bas. Sa taille est égale au nombre des éléments non nuls.

→ **col** : de la même taille que `val`, `col[i]` correspond à la colonne de `val[i]`. Sa taille est égale au nombre des éléments non nuls

→ **row** : contient les position des élément de `val` qui correspondent à une nouvelle ligne, `row[i]`, sa taille est fixée à `n+1` (`n`=nombre de lignes de la matrice).

Pour la matrice suivante :

$$\begin{pmatrix} 3 & 0 & 0 & 2 & 1 \\ 0 & 0 & 5 & 8 & 0 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 10 & 4 & 0 \end{pmatrix},$$

le stockage CSR est alors :

`val = [3, 2, 1, 5, 8, 1, 2, 9, 10, 4]`
`col = [0, 3, 4, 2, 3, 1, 2, 2, 2, 3]`
`row = [0, 3, 5, 7, 8, 10].`

↓

Indice	0	1	2	3	4	5	6	7	8	9
row	0	3	5	7	8	10				
col	0	3	4	2	3	1	2	2	2	3
val	3	2	1	5	8	1	2	9	10	4

Le format CSR est spécialisé dans les opérations d'algèbres linéaires. Son nom vient du fait que le tableau `row` est *compressé*.

Cette représentation permet un accès rapide à la mémoire lors de la multiplication des vecteurs par des grandes matrices (par exemple, Algorithme PageRank avec Erdos).

PS : L'indexation des éléments doit commencer par 0 pour le bon fonctionnement du CSR.

> Dans le code nous avons :

```
float *val = calloc(e, sizeof(float));
int *col_ind = calloc(e, sizeof(int));
int *row_ptr = calloc(n+1, sizeof(int));
```

Représentant respectivement les 3 vecteurs, tels que : **e** est le nombre d'éléments non nuls et **n** est nombre de lignes de la matrice.

> Utilisation de CSR avec PageRank : Nous avons utilisé la structure creuse CSR pour la représentation de la matrice de transitions entre les nœuds, cette structure représente un cas idéal pour l'optimisation de la mémoire.

En effet, nous récupérons les données à partir des 6 graphes du web et nous insérons ces derniers dans les 3 vecteurs avec une méthode d'accès spéciale (prédéfinie). Cette méthode remplit les 3 vecteurs et les initialise pour les utiliser ensuite dans la méthode itérative de PageRank (la boucle de PageRank / l'algorithme des puissances).

```
row_ptr[0] = 0;

int fromnode, tonode;
int cur_row = 0;
int i = 0;
int j = 0;
// Elements for row
int elrow = 0;
// Cumulative numbers of elements
int curel = 0;

FILE *wptr;
wptr = fopen("iter.txt", "w");
while(!feof(fp)) {

    fscanf(fp, "%d%d", &fromnode, &tonode);

    // printf("From: %d To: %d\n", fromnode, tonode);

    if (fromnode > cur_row) { // change the row
        curel = curel + elrow;
        int k = 0;
        for (k = cur_row + 1; k <= fromnode; k++) {
            row_ptr[k] = curel;
        }
        elrow = 0;
        cur_row = fromnode;
    }

    val[i] = 1.0;
```

```

col_ind[i] = tonode;
elrow++;
i++;
int nods = 0;
int save = 0;
while (nods <= nbr) {

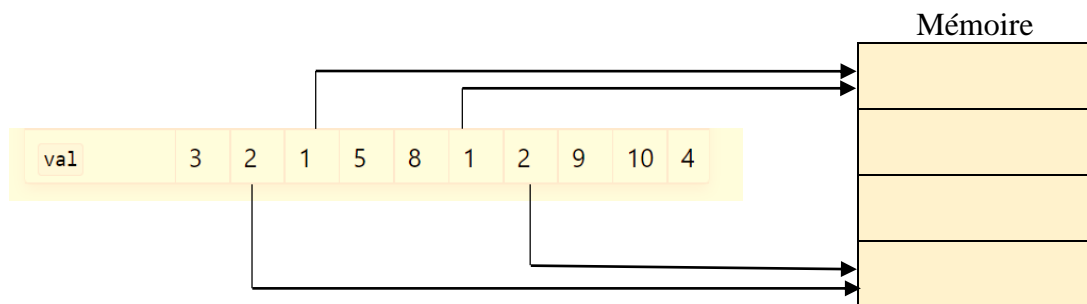
    save = elrow;
    nods = nods + nbr/10;
} }

```

> Comme vu précédemment les 3 vecteurs sont déclarés comme **des vecteurs de pointeurs**, ce qui est possible en C, cette déclaration permet de pousser l'optimisation encore un pas en avant en réduisant l'utilisation de l'espace mémoire. En effet les valeurs (non-nulles) redondantes vont être stockées qu'une seule fois, avec des pointeurs se pointant vers la même adresse mémoire.

Cette dernière optimisation sera très utile car en examinant les graphes du web nous avons constaté qu'il y a plusieurs valeurs de probabilités de transitions redondantes (notamment avec Erdos).

Nous utilisons ces matrices dans la méthode itérative de PageRank.



2.2.2 Convergence

Qui dit convergence de PageRank, dit epsilon. Epsilon représente une des conditions d'arrêt de l'algorithme des puissances. En effet, si 2 exécutions consécutives de PageRank (2 itérations) donnent 2 vecteurs presque identiques (avec une différence égale à Epsilon), l'algorithme s'arrête et les résultats sont retournés. Le choix d'Epsilon doit être fait avec précaution, si il est trop petit l'algorithme risque de ne jamais converger, si il est trop grand le résultat risque d'être très loin de l'optimum.

Après plusieurs exécutions et analyses des valeurs d'Epsilon nous avons choisi la valeur 10^{-6} .

2.2.3 PageRank

Après l'initialisation d'Epsilon, ainsi que les paramètres (choisis par l'utilisateur) tels que :

- Le graphe du web (1 à 6).
- Le nombre de nouveaux sommets.
- La probabilité de transitions entre les nouveaux sommets : p (entre 0 et 1).
- Le facteur d'amortissement : Alpha (entre 0 et 1).

Nous pouvons maintenant entamer le PageRank. Le code suivant résume l'itération de PageRank.

```
int looping = 1;
int k = 0;

// initialisation d'un nouveau vecteur p
float p_new[n];

while (looping){

    for(i=0; i<n; i++){
        p_new[i] = 0.0;
    }

    int curcol = 0;
    // Pagerank modifiedie algorithme
    for(i=0; i<n; i++){
        rowel = row_ptr[i+1] - row_ptr[i];
        for (j=0; j<rowel; j++) {
            p_new[col_ind[curcol]] = p_new[col_ind[curcol]] + val[curcol] * p[i];
            curcol++;
        }
    }

    // DANGELING ELEMENTS
    for(i=0; i<n; i++){
        p_new[i] = d * p_new[i] + (1.0 - d) / n;
    }

    /*
    for (i=0; i<n; i++){
        printf("%f ", p_new[i]);
    }*/

    // CHECK EXIT CONDITION
    float error = 0.0;
    for(i=0; i<n; i++) {
        error = error + fabs(p_new[i] - p[i]);
    }
    // EPSILON
    if (error < 0.000001){
```

```

        looping = 0;
    }

    // mettre a jour p[]
    for (i=0; i<n;i++){
        p[i] = p_new[i];
    }

    // Nombre d'iterations
    k = k + 1;
}

```

> Nous initialisons un nouveau vecteur p avec float p_new[n], ce vecteur va être multiplié par la matrice de transition (algorithme des puissances) et mis à jour dans chaque itération.

```

for(i=0; i<n; i++){
    rowel = row_ptr[i+1] - row_ptr[i];
    for (j=0; j<rowel; j++) {
        p_new[col_ind[curcol]] = p_new[col_ind[curcol]] + val[curcol] * p[i];
        curcol++;
    }
}

```

La multiplication par une matrice représentée en CSR se fait avec une optimisation de calcul.

> Nous récupérons toutes les valeurs non-nulles de la matrice un par un, et nous entrons dans une nouvelle boucle qui calcule la multiplication des valeurs de p et la matrice, cette implémentation de la multiplication est prise de l'implémentation de CSR.

> La gestion des éléments suspendus (Dangling elements) se fait de la même manière que PageRank l'original.

> Nous calculons ensuite l'erreur, qui représente la différence entre les vecteurs de 2 itérations consécutives, si cette dernière est inférieure à Epsilon, l'algorithme s'arrête (convergence de PageRank). En effet, Nous prenons en compte **le nombre d'itérations** jusqu'à la convergence, qui sera utilisé comme métrique d'analyse, ainsi que **le temps d'exécution**. Ce dernier est calculé avec la fonction Clock() prédéfinie dans la standard du langage C.

2.2.4 Erdos

Le graphe web du monde réel évolue chaque jour, les valeurs de PageRank sont recalculées pour régénérer des nouvelles valeurs cohérentes avec l'évolution du graphe.

> Pour simuler ce comportement dans notre analyse, nous allons implémenter la fonction Erdos pour la génération des nouveaux sommets et arcs associés avec leurs probabilités dans nos graphes du web en s'assurant de garder la propriété du graphe stochastique.

> Nous avons intégré l'implémentation d'Erdos avec l'implémentation de PageRank dans la partie de chargement des données. En effet, nous générons un nouveau nombre de sommets (**nbr** choisi à l'exécution par l'utilisateur), ces sommets vont être reliés aléatoirement au graphe du web (choisi lors de l'exécution), avec des probabilités (**prob** choisies aussi lors de l'exécution).

```
while (nods <= nbr) {  
    save = elrow;  
    nods = nods + nbr/10;  
}
```

> On fixe la stochastisation (pour s'assurer de garder la propriété du graphe stochastique dans le nouveau graphe généré avec Erdos)

```
int out_link[n];  
for(i=0; i<n; i++){  
    out_link[i] = 0;  
}
```

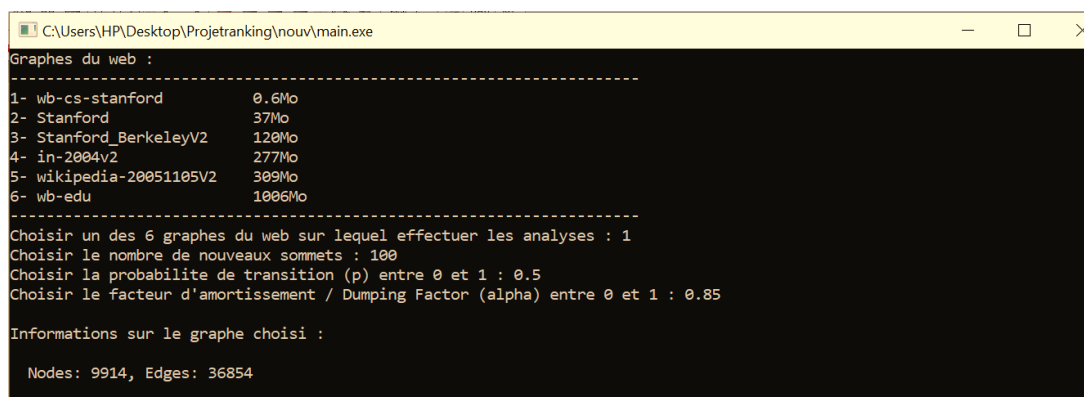
2.3 Exécution

Lorsque on compile et on exécute le programme on obtient :

2.3.1 Inputs

Dans la console, l'utilisateur définit :

- Le graphe du web (de 1 à 6).
- Le nombre de nouveaux sommets.
- La probabilité de transitions p (entre 0 et 1).
- Le facteur d'amortissement Alpha (entre 0 et 1).

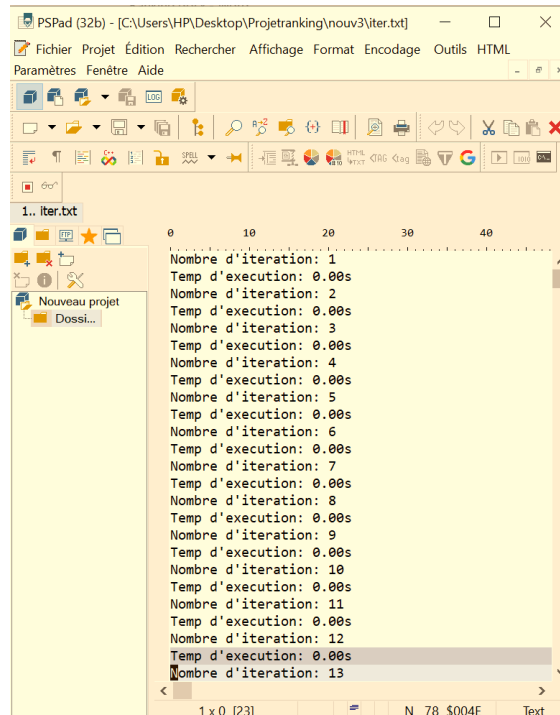


```
C:\Users\HP\Desktop\Projetranking\nouv\main.exe  
Graphes du web :  
-----  
1- wb-cs-stanford      0.6Mo  
2- Stanford            37Mo  
3- Stanford_BerkeleyV2 120Mo  
4- in-2004v2           277Mo  
5- wikipedia-20051105V2 309Mo  
6- wb-edu              1006Mo  
-----  
Choisir un des 6 graphes du web sur lequel effectuer les analyses : 1  
Choisir le nombre de nouveaux sommets : 100  
Choisir la probabillite de transition (p) entre 0 et 1 : 0.5  
Choisir le facteur d'amortissement / Dumping Factor (alpha) entre 0 et 1 : 0.85  
Informations sur le graphe choisi :  
  
Nodes: 9914, Edges: 36854
```

2.3.2 Outputs

Les résultats (les 2 métriques d'analyses) sont enregistrés dans un fichier texte.

- Le temps d'exécution.
- Le nombre d'itérations.



3. Analyse Des Résultats

3.1 Métriques d'analyse

Dans cette partie, nous allons à chaque fois varier :

- Les graphes du web (de 1 à 6).
- Le nombre de nouveaux sommets.
- La probabilité de transitions p entre les nouveaux sommets (entre 0 et 1).
- Le facteur d'amortissement α (entre 0 et 1).

Puis nous observons les comportements des deux métriques :

- Le temps d'exécution en secondes (pour que le PageRank converge).
- Le nombre d'itérations (pour que le PageRank converge).

L'analyse est faite en exécutant à chaque fois notre l'algorithme sur C , puis copier les résultats numériques (outputs) dans une feuille Excel pour générer les graphes, pour ensuite les interpréter.

En effet, pour chacun des 6 graphes du web, nous avons varié 16 différents nombres de nouveaux sommets (de 10 à 500000000 sommets). Ainsi, pour chacun de ces nombre de nouveaux sommets, nous avons varié différentes valeurs de p et de α de telle sorte que : p soit inférieure α , p soit supérieure α , p soit égale à α , dans le but de mieux voir l'effet de ces paramètres sur le comportement général du pagerank.

Pour chaque cas :

- Le paramètre à varier (nombre de nouveaux sommets) est représenté sur l'axe horizontal.
- Les deux métriques (**temps d'exécution** et **nombre d'itérations**) sont représentées sur le même graphique, respectivement sur (**Axe vertical gauche**, **Axe vertical droit**).

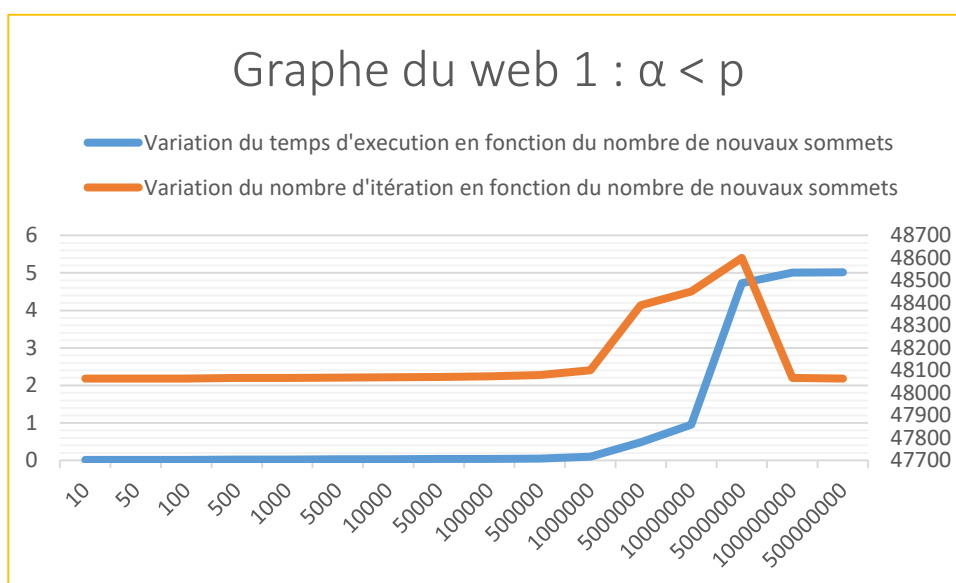
3.2 Résultats numériques et Graphes

3.2.1 Graphe du web 1 (wb-cs-stanford) :

3.2.1.1 valeurs de α inférieures aux valeurs de p ($\alpha < p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	0,01	0,01	0,01	0,02	0,02	0,03	0,03	0,04	0,04
nombre d'itération	48063	48063	48064	48066	48067	48068	48070	48071	48073

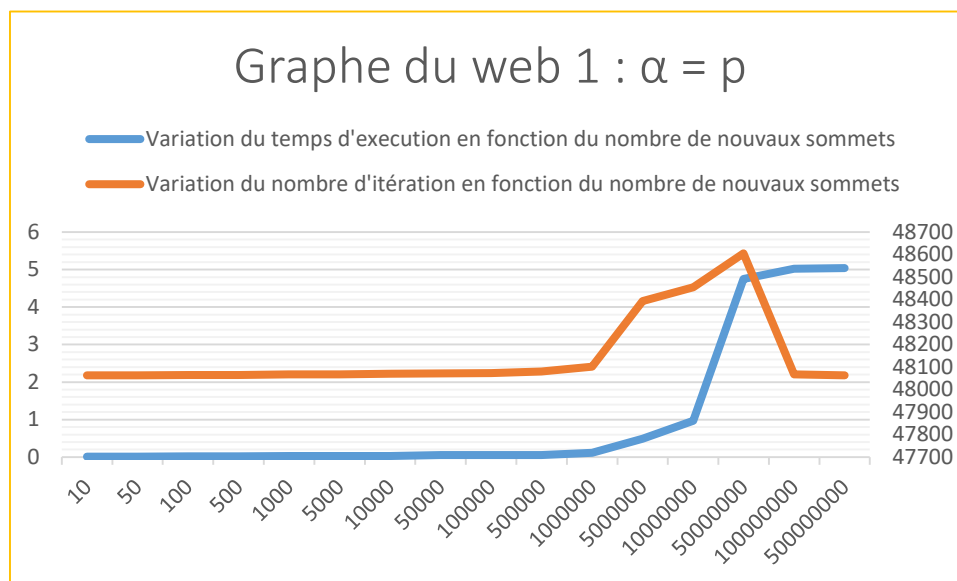
500000	1000000	5000000	10000000	50000000	100000000	500000000
0.05	0.1	0.48	0.95	4,73	5,01	5,02
48080	48100	48390	48450	48600	48067	48063



3.2.1.2 valeurs de α égales aux valeurs de p ($\alpha = p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	0,01	0,01	0,02	0,02	0,03	0,03	0,03	0,05	0,05
nombre d'itération	48064	48064	48065	48065	48067	48068	48070	48072	48073

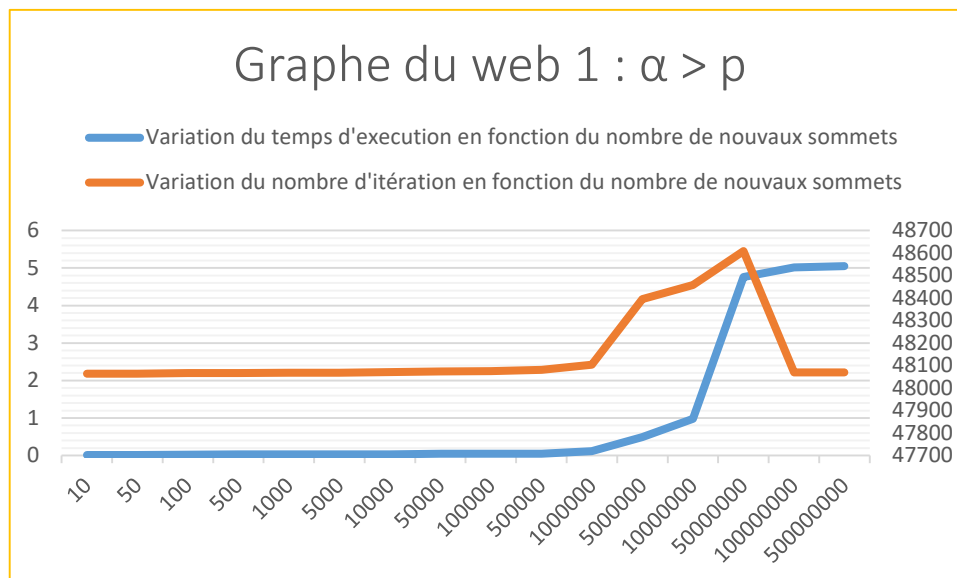
500000	1000000	5000000	10000000	50000000	100000000	500000000
0.05	0.11	0.49	0.97	4,75	5,02	5,04
48080	48102	48393	48455	48606	48068	48064



3.2.1.3 valeurs de α supérieures aux valeurs de p ($\alpha > p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	0,01	0,01	0,02	0,03	0,03	0,03	0,03	0,05	0,05
nombre d'itération	48064	48064	48066	48066	48068	48068	48071	48073	48075

500000	1000000	5000000	10000000	50000000	100000000	500000000
0.05	0.12	0.49	0.98	4,76	5,02	5,05
48081	48104	48395	48458	48609	48069	48070

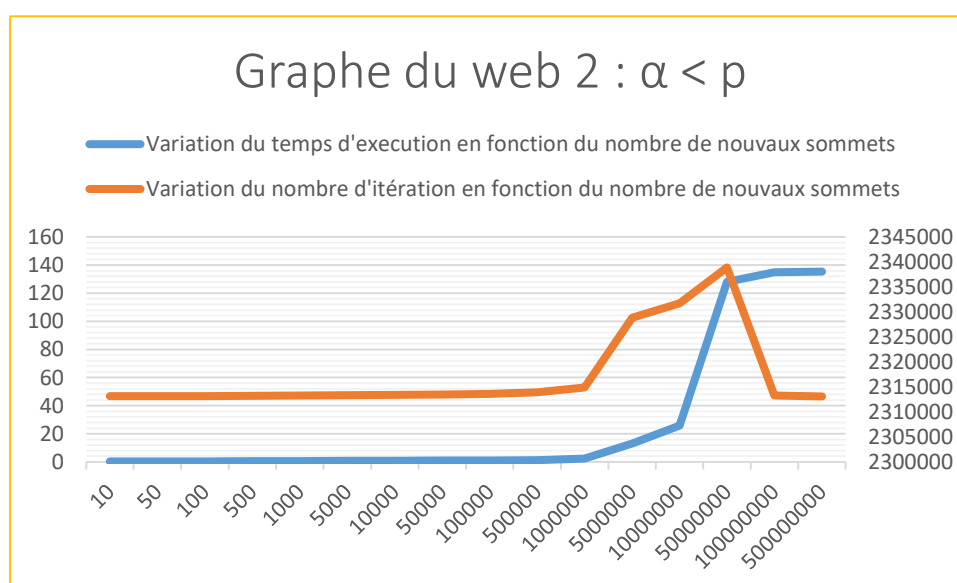


3.2.2 Graphe du web 2 (Stanford) :

3.2.2.1 valeurs de α inférieures aux valeurs de p ($\alpha < p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	0,25	0,25	0,25	0,5	0,51	0,7	0,76	1	1,01
nombre d'itération	2313158	2313159	2313159	2313250	2313296	2313345	2313440	2313486	2313580

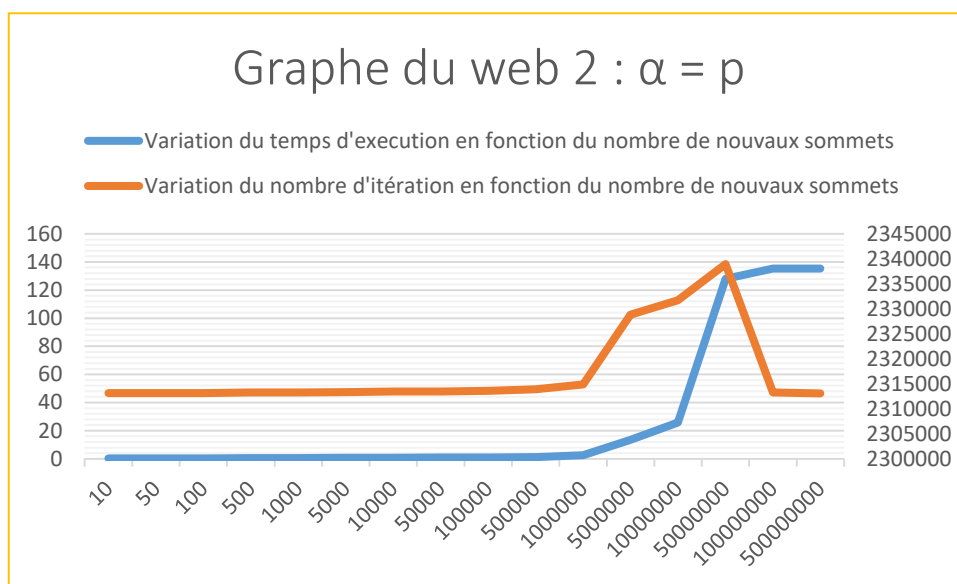
500000	1000000	5000000	10000000	50000000	100000000	500000000
1,25	2,5	13	25,73	128,02	135	135,3
2313915	2314875	2328829	2331710	2338920	2313265	2313070



3.2.2.2 valeurs de α égales aux valeurs de p ($\alpha = p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	0,25	0,26	0,26	0,52	0,53	0,71	0,78	1	1,03
nombre d'itération	2313158	2313160	2313160	2313252	2313299	2313345	2313444	2313489	2313581

500000	1000000	5000000	10000000	50000000	100000000	500000000
1,25	2,6	13,5	25,83	128,09	135,3	135,32
2313917	2314877	2328832	2331715	2338922	2313265	2313071

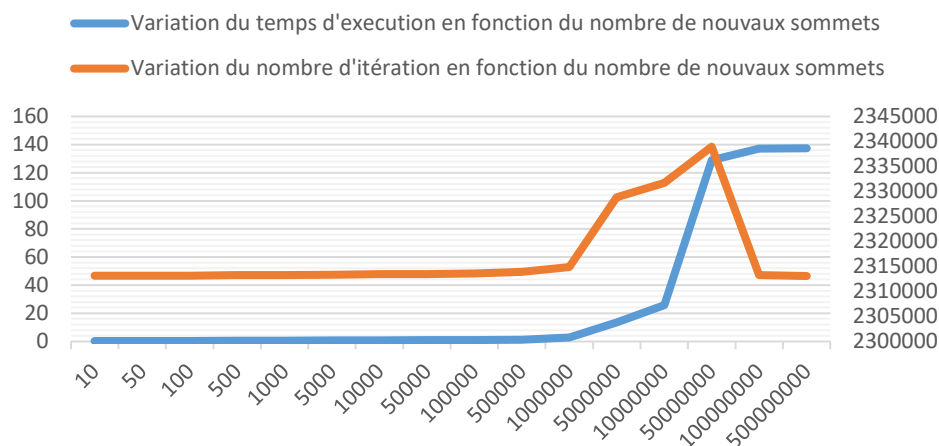


3.2.2.3 valeurs de α supérieures aux valeurs de p ($\alpha > p$) :

Nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	0,25	0,26	0,26	0,53	0,54	0,73	0,8	1,02	1,05
nombre d'itération	2313158	2313160	2313160	2313254	2313302	2313347	2313448	2313491	2313583

500000	1000000	5000000	10000000	50000000	100000000	500000000
1,3	2,7	13,59	25,87	129	137	137,32
2313919	2314879	2328838	2331725	2338926	2313269	2313073

Graphe du web 2: $\alpha > p$



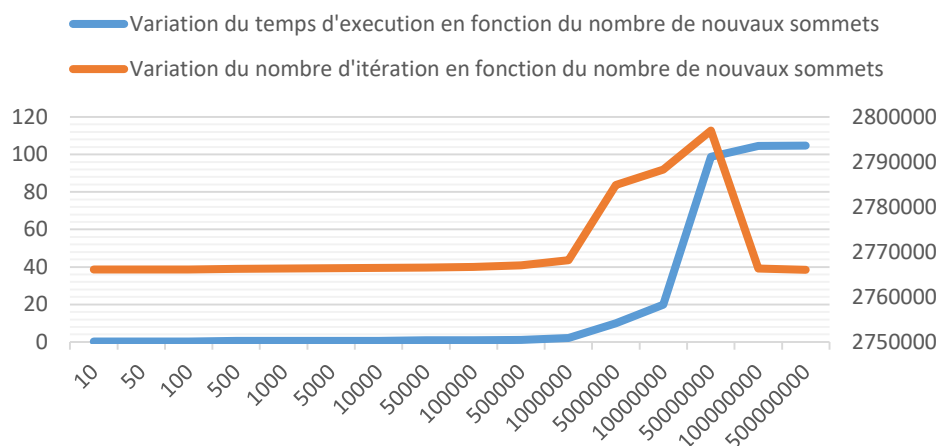
3.2.3 Graphe du web 3 (Stanford_BerkeleyV2) :

3.2.3.1 valeurs de α inférieures aux valeurs de p ($\alpha < p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	0,31	0,31	0,31	0,62	0,63	0,63	0,65	0,85	0,86
nombre d'itération	2766076	2766077	2766132	2766246	2766304	2766361	2766475	2766532	2766645

500000	1000000	5000000	10000000	50000000	100000000	500000000
1,06	2,1	10,04	19,85	98,8	104,5	104,71
2767047	2768196	2784885	2788335	2796966	2766290	2766053

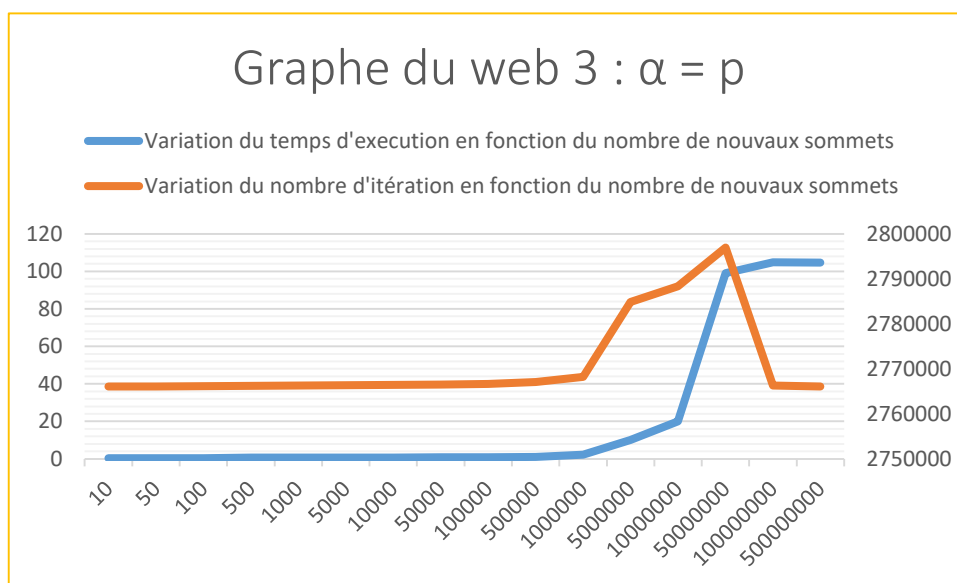
Graphe du web 3 : $\alpha < p$



3.2.3.2 valeurs de α égales aux valeurs de p ($\alpha = p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	0,31	0,32	0,4	0,63	0,64	0,65	0,65	0,87	0,88
nombre d'itération	2766076	2766077	2766137	2766248	2766309	2766369	2766475	2766535	2766650

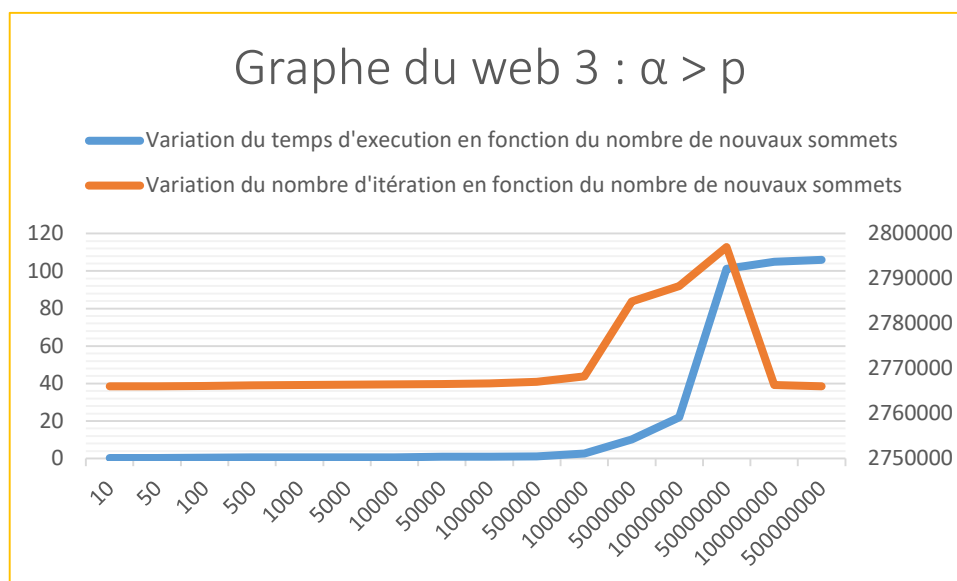
500000	1000000	5000000	10000000	50000000	100000000	500000000
1,1	2,3	10,11	20	99	104,9	104,75
2767051	2768202	2784889	2788340	2796969	2766299	2766060



3.2.3.3 valeurs de α supérieures aux valeurs de p ($\alpha > p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	0,32	0,32	0,41	0,64	0,642	0,66	0,68	0,89	0,91
nombre d'itération	2766078	2766078	2766139	2766249	2766311	2766372	2766480	2766539	2766654

500000	1000000	5000000	10000000	50000000	100000000	500000000
1,15	2,6	10,13	22	101,1	105	106,02
2767054	2768222	2784892	2788347	2796972	2766301	2766070

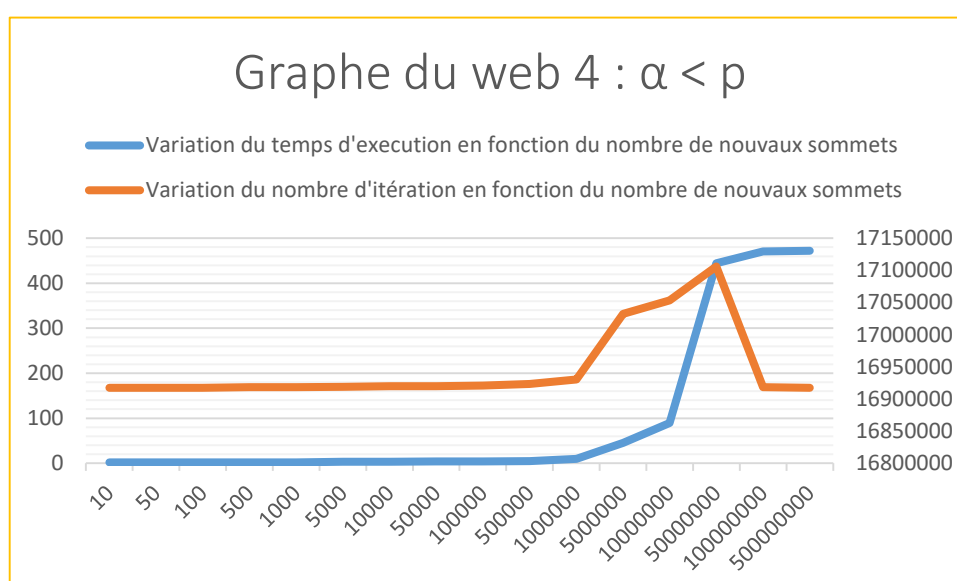


3.2.4 Graphe du web 4 (in-2004v2) :

3.2.4.1 valeurs de α inférieures aux valeurs de p ($\alpha < p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	1,88	1,88	1,88	1,9	1,95	2,92	2,93	3,8	3,84
nombre d'itération	16917459	16917459	16917459	16918163	16918513	16918867	16919569	16919920	16920626

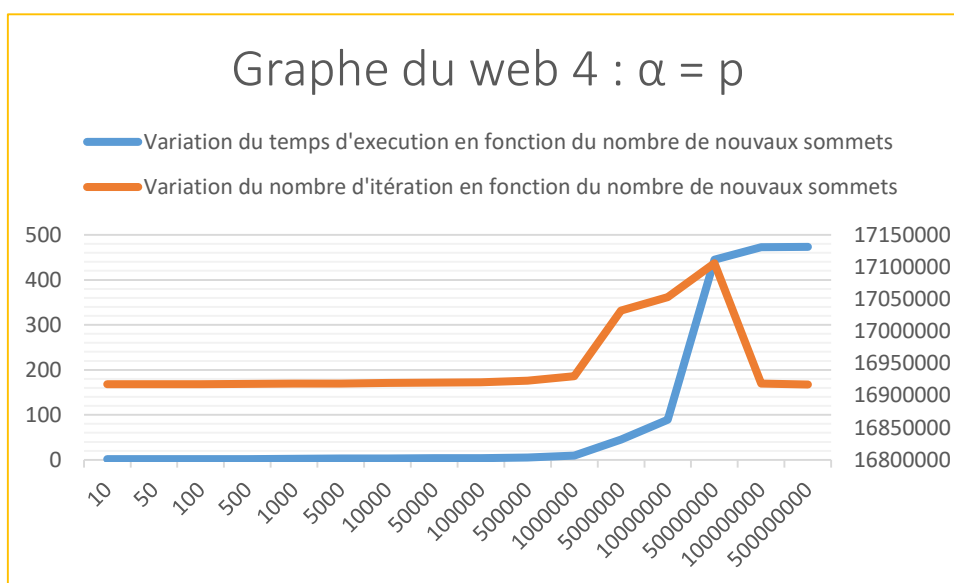
500000	1000000	5000000	10000000	50000000	100000000	500000000
4,7	9,4	45,1	89,3	444,6	471	471,9
16923088	16930130	17032203	17053320	17106120	16918514	16917105



3.2.4.2 valeurs de α égales aux valeurs de p ($\alpha = p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	1,88	1,89	1,891	1,9	1,96	2,94	2,95	3,851	3,86
nombre d'itération	16917459	16917462	16917463	16918165	16918517	16918877	16919575	16919928	16920632

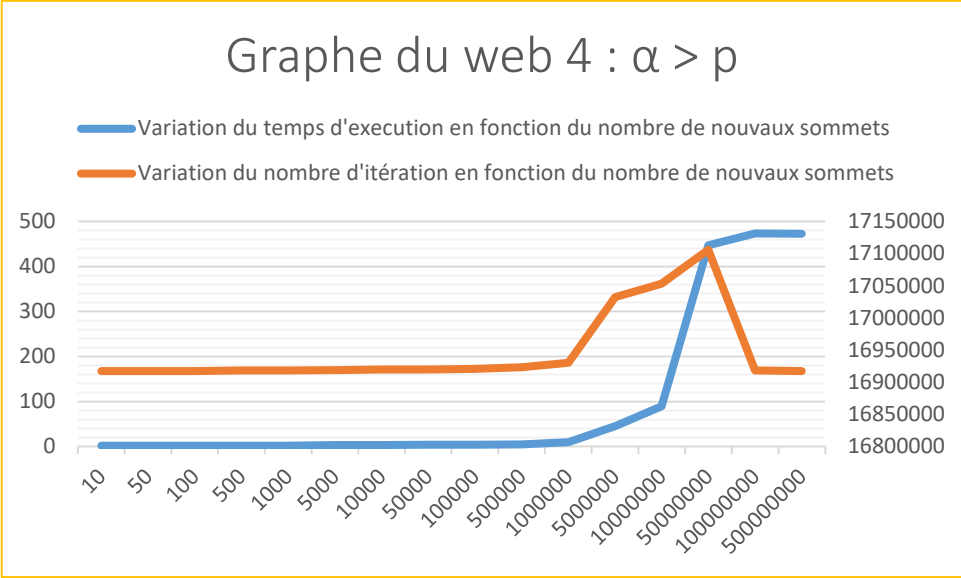
500000	1000000	5000000	10000000	50000000	100000000	500000000
4,82	9,53	45,23	89,41	445	472,3	473,01
16923096	16930141	17032214	17053329	17106128	16918525	16917125



3.2.4.3 valeurs de α supérieures aux valeurs de p ($\alpha > p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	1,88	1,892	1,894	1,91	1,971	2,95	2,97	3,91	3,93
nombre d'itération	16917459	16917464	16917464	16918165	16918519	16918881	16919579	16919932	16920636

500000	1000000	5000000	10000000	50000000	100000000	500000000
4,85	9,55	45,26	89,5	447,01	473,5	473,1
16923100	16930147	17032224	17053339	17106134	16918551	16917150

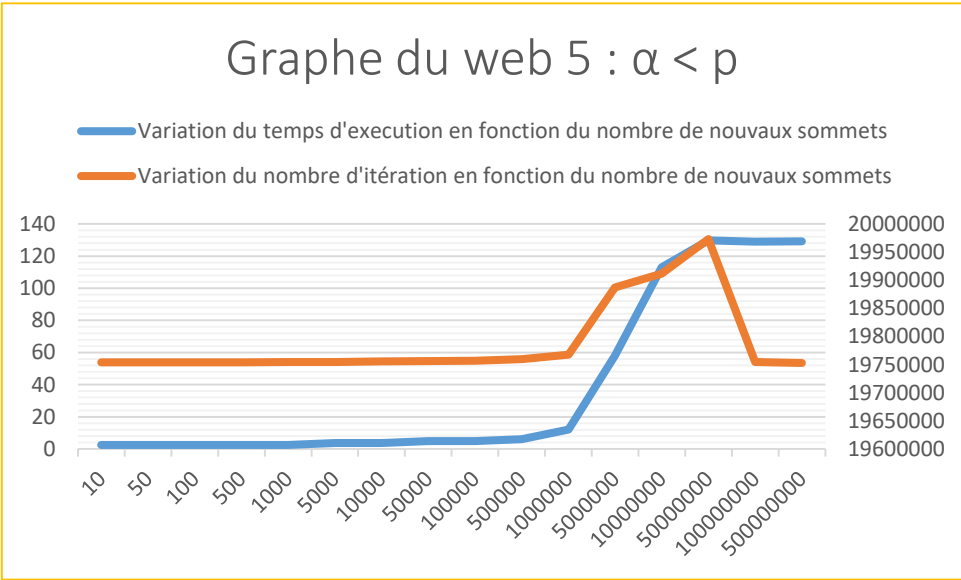


3.2.5 Graphe du web 5 (wikipedia20051105V2) :

3.2.5.1 valeurs de α inférieures aux valeurs de p ($\alpha < p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	2,41	2,41	2,41	2,42	2,43	3,65	3,659	4,86	4,863
nombre d'itération	19753901	19753901	19753902	19753904	19754315	19754726	19755549	19755960	19756788

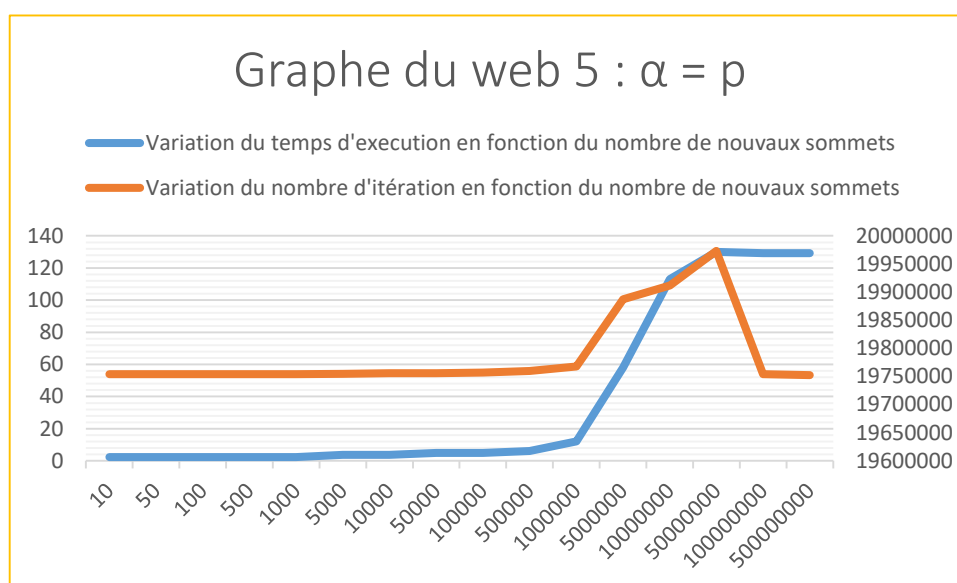
500000	1000000	5000000	10000000	50000000	100000000	500000000
6,05	12,1	58,1	113	129,79	129	129,2
19759657	19767878	19887060	19911718	19973364	19754315	19752670



3.2.5.2 valeurs de α égales aux valeurs de p ($\alpha = p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	2,41	2,41	2,41	2,42	2,43	3,66	3,66	4,861	4,864
nombre d'itération	19753901	19753901	19753902	19753904	19754316	19754728	19755550	19755962	19756789

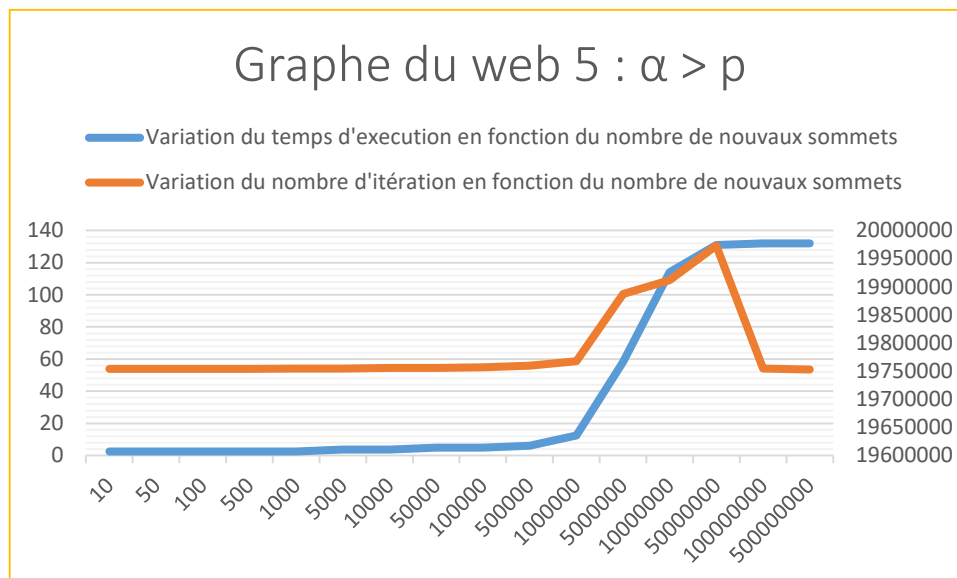
500000	1000000	5000000	10000000	50000000	100000000	500000000
6,08	12,12	58,2	113,1	129,99	129,3	129,31
19759660	19767880	19887062	19911720	19973367	19754325	19752671



3.2.5.3 valeurs de α supérieures aux valeurs de p ($\alpha > p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	2,41	2,41	2,41	2,42	2,44	3,662	3,664	4,87	4,9
nombre d'itération	19753901	19753901	19753902	19753904	19754317	19754730	19755552	19755964	19756792

500000	1000000	5000000	10000000	50000000	100000000	500000000
6,1	12,5	58,3	114	131	131,9	131,93
19759664	19767889	19887067	19911725	19973371	19754353	19752673

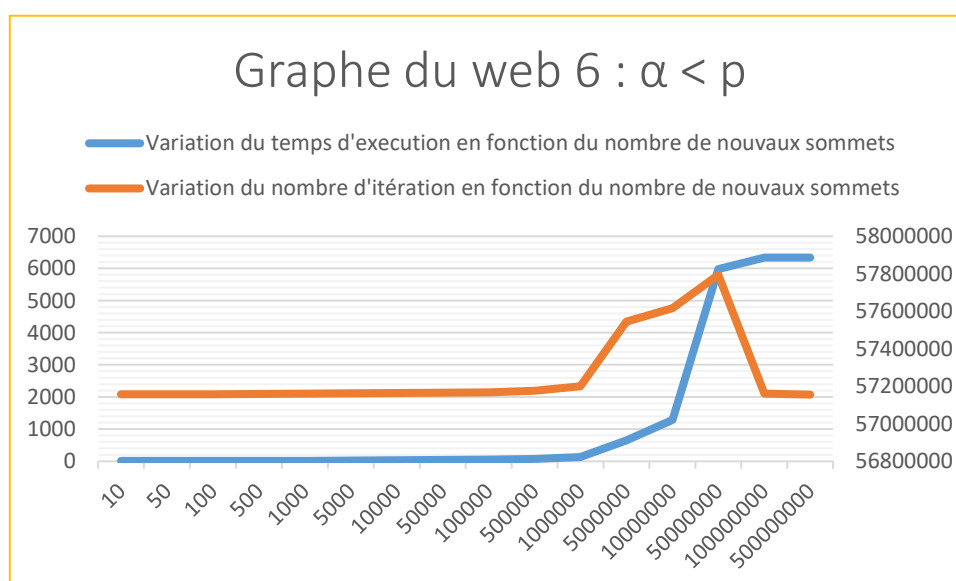


3.2.6 Graphe du web 6 (wb-edu) :

3.2.6.1 valeurs de α inférieures aux valeurs de p ($\alpha < p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	6,75	6,75	6,75	13,2	13,4	20,25	30,37	40,5	54
nombre d'itération	57156563	57156563	57156563	57158941	57160130	57161321	57163698	57164888	57167266

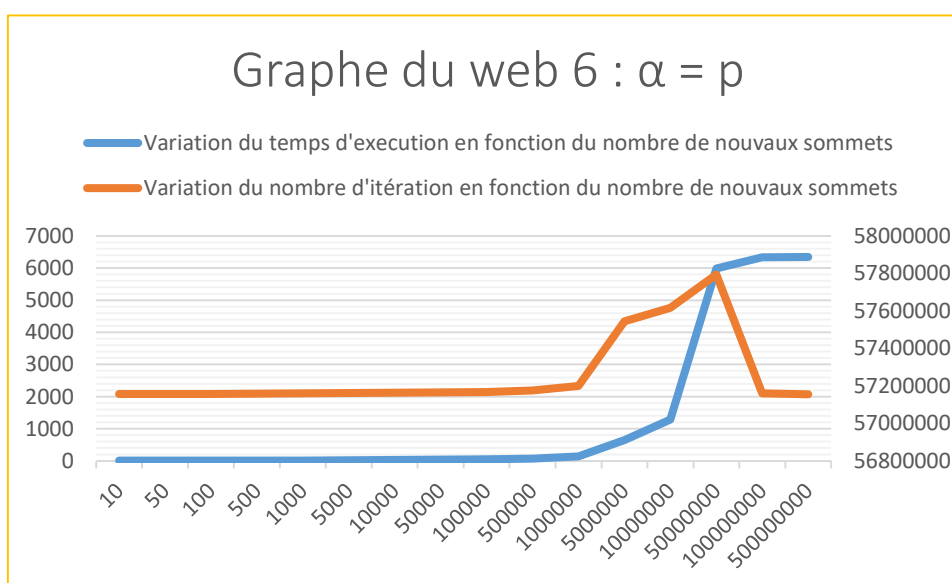
500000	1000000	5000000	10000000	50000000	100000000	500000000
67,5	135	648	1282,2	5980,1	6334,5	6340
57175590	57199372	57544236	57615582	57793960	57160131	57155370



3.2.6.2 valeurs de α égales aux valeurs de p ($\alpha = p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	6,75	6,75	6,77	13,5	13,8	20,28	30,39	41	54,6
nombre d'itération	57156563	57156563	57156564	57158949	57160135	57161327	57163699	57164890	57167269

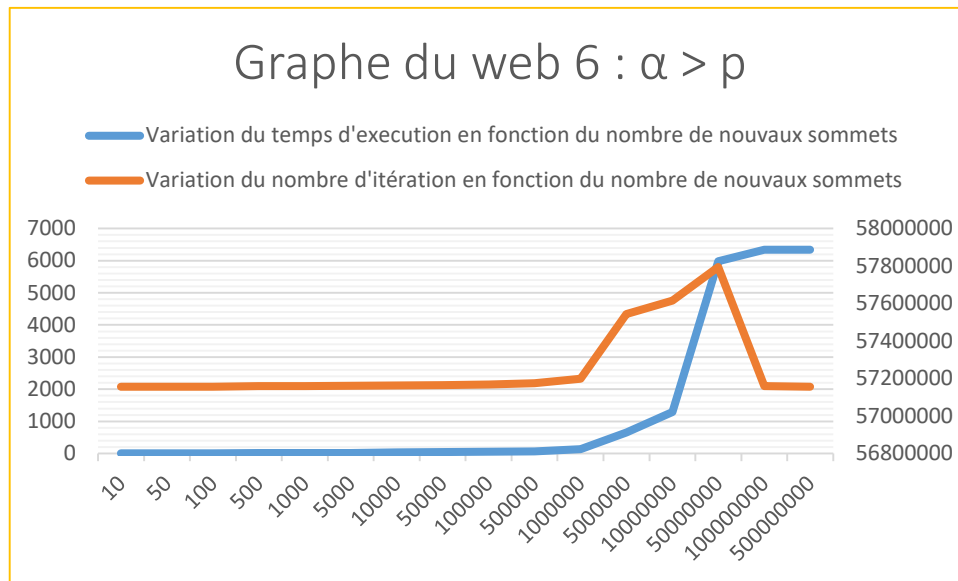
500000	1000000	5000000	10000000	50000000	100000000	500000000
68,5	137	650	1284	5982,1	6334,9	6341
57175599	57199375	57544246	57615589	57793964	57160135	57155371



3.2.6.3 valeurs de α supérieures aux valeurs de p ($\alpha > p$) :

nouveaux sommets	10	50	100	500	1000	5000	10000	50000	100000
temps d'exécution	6,75	6,75	6,77	14,01	14,93	20,7	31	42,01	55,5
nombre d'itération	57156563	57156563	57156564	57158952	57160138	57161330	57163701	57164897	57167273

500000	1000000	5000000	10000000	50000000	100000000	500000000
69,4	138,15	656	1286,2	5984,6	6339	6344
57175601	57199379	57544256	57615593	57793984	57160139	57155373



3.3 Interprétation des résultats

Après notre exécution de l'algorithme (qui a nécessité des heures) notamment pour les grands graphes du web, nous avons remarqué que son comportement est presque semblable pour les 6 graphes du web donnés y compris le plus grand, i.e les courbes épousent des formes semblables.

D'après les deux courbes obtenues pour chacun des 6 graphes du web, on remarque aussi que le temps d'exécution et le nombre d'itérations varient d'une façon cohérente, telle que lorsque l'un des deux croît d'une façon exponentielle l'autre croît d'une façon exponentielle aussi, lorsque l'un est constant l'autre est constant aussi, lorsque l'un croît d'une façon légère l'autre croît également d'une façon légère.

- Pour le temps d'exécution :

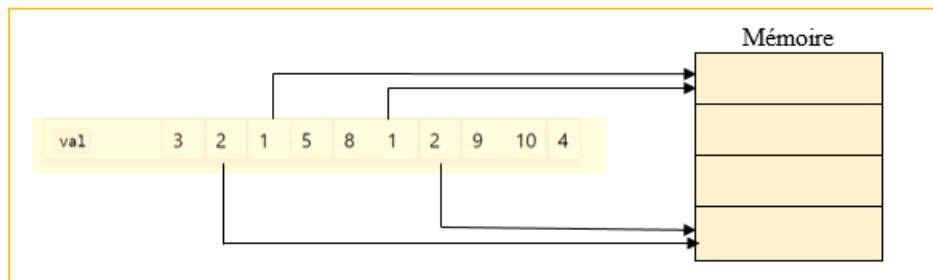
Jusqu'à l'ajout d'un certain nombre de nouveaux sommets (exemple 1000000 pour le graphe du web 1) le temps d'exécution subit une croissance très légère (exemple de 0.01 à 0.1 secondes pour le graphe du web 1). Au-delà de ce nombre de nouveaux sommets, le temps d'exécution croît très rapidement pour qu'il atteigne (exemple 4,73 secondes pour le graphe du web 1) à l'ajout d'un certain nombre de nouveaux sommets plus grand (exemple 50000000 pour le graphe du web 1). Au-delà, de ce nombre de nouveaux sommets le temps d'exécution ré-subit à nouveau une croissance légère.

Le temps d'exécution reste convenable pour les grands graphes du web.

- Pour le nombre d'itérations :

De même, que pour le temps d'exécution. Jusqu'à l'ajout d'un certain nombre de nouveaux sommets (exemple 1000000 pour le graphe du web 1) le nombre d'itérations subit une croissance très légère. Au-delà de ce nombre de nouveaux sommets, le nombre d'itérations croît très rapidement pour qu'il atteigne (exemple 48600 itérations pour le graphe du web 1) à l'ajout d'un certain nombre de nouveaux sommets plus grand (exemple 50000000 pour le graphe du web 1). Au-delà, de ce nombre de nouveaux sommets, le nombre d'itérations subit une décroissance claire vers la fin (cohérence avec la très légère croissance du temps d'exécution vers la fin aussi).

Cette décroissance du nombre d'itérations, lorsque le nombre de nouveaux sommets devient considérable est due à la structure de CSR.



En effet, Erdos attribue des probabilités de transitions égales aux liens ajoutés dans le graphe du web (valeurs redondantes). Ainsi, plus le nombre de nouveaux sommets augmente, plus on aura des valeurs redondantes. Et comme expliqué précédemment, dans la structure CSR, les valeurs non-nulles redondantes (les probabilités de transitions redondantes dans notre cas) vont être stockées qu'une seule fois, avec des pointeurs se pointant vers la même adresse mémoire. De ce fait, l'algorithme accède à ces valeurs rapidement ce qui diminue le nombre d'itérations du PageRank (convergence plus rapide lorsque le nombre de nouveaux nœuds est très grand).

Cette dernière optimisation nous a été très utile car en examinant les graphes du web notamment avec Erdos nous constatons qu'il y a beaucoup de valeurs de probabilités de transitions redondantes.

NB : Le PageRank et la structure creuse vue durant le TP 1 et 2 (i.e sans l'implémentation d'Erdos) a convergé au bout de 55 itérations seulement pour le graphe du web 2 stanford. Mais,

avec l'implémentation d'Erdos ce dernier crachait (bug du programme) lors d'utilisation des grands graphes du web à cause de la mémoire.

En effet, avec Erdos le nombre d'itérations augmente beaucoup car à chaque itération on rajoute un grand nombre d'autres nouveaux sommets (et donc arcs aussi) aléatoirement, ce qui fait, on réitère l'algorithme encore jusqu'à la convergence. Mais une fois le nombre de nouveaux sommets augmente beaucoup et devient considérable, ce nombre d'itérations diminue.

Le nombre de nouveaux sommets influence sur la vitesse de convergence. Tels que plus le nombre de nouveaux sommets augmente plus la vitesse de convergence est lente. Mais, le fait d'augmenter beaucoup le nombre de nouveaux sommets permet à l'algorithme de converger plus vite (grâce à la structure CSR).

On remarque que notre solution est efficace et adaptée pour les graphes du web très grands.

NB : Le fait de varier un très grand nombre de nouveaux sommets (jusqu'à cinq cent millions 500000000) nous a pris du temps mais nous a bien permis d'observer le comportement du pagerank. En revanche l'ajout de petits nombre de nouveaux sommet ne permet pas d'observer un tel comportement.

En ce qui concerne les deux paramètres restants α et p , nous les avons variés avec chaque nombre de nouveaux sommets de telle sorte que **α soit inférieure à p** , exemple : $\alpha = 0.1$; $p = 0.85$, $\alpha = 0.1$; $p = 0.75$, $\alpha = 0.5$; $p = 0.9$...etc. nous avons remarqué une petite influence sur les résultats obtenus, tels que plus on diminue α , plus le nombre d'itérations et le temps d'exécution diminuent, ainsi, plus l'algorithme converge plus vite.

Pour chaque graphe du web, plus α soit proche de 1 cela ralentit la convergence. plus elle sera proche de 0 cela accélère la convergence. Donc on se permet de dire que :

α influence sur la vitesse de convergence. Ainsi, diminuer sa valeur permet à l'algorithme de converger plus vite.

À la base α est utilisé dans le surfer aléatoire (Google) lorsque une page ne contient aucune url vers aucune autre page (cas où la somme d'une ligne de la matrice de transition soit égale à 0).

En revanche, nous n'avons remarqué aucune influence particulière de p sur les résultats mis à part le fait d'avoir beaucoup de valeurs redondantes dans la matrice de transition (générées par

Erdos) permet aussi à l'algorithme de converger plus vite (comme expliqué plus haut dans la structure de CSR).

4. Conclusion

La réalisation de ce projet était une chance pour nous de mettre en pratique les concepts théoriques vus en cours durant tout le semestre, que ce soit dans l'aspect de la programmation de l'algorithme, ou même l'interprétation et l'analyse des résultats. Ça traitait aussi un sujet très intéressant. Nous avons aussi certes rencontré plusieurs difficultés dans la réalisation de ce travail, mais notre enseignant était toujours présent, attentif et très réactif à nos questions avec des réponses très rapides et précises à nos besoins. Sur ce nous remercions l'enseignant du module « méthodes de ranking et recommandations » pour son implication et les efforts fournies avec nous.

Avant de conclure, nous avons tirés quelques conclusions de ce petit travail :

Pas toute structure matricielle creuse supporte l'implémentation d'Erdos. En effet, La représentation matricielle creuse vue en TP fonctionnait uniquement pour les 3 premiers graphes du web, ce qui nous a poussé à implémenter une nouvelle représentation matricielle plus efficace (tableau de pointeurs CSR)

Notre implémentation ne consomme pas beaucoup de mémoire et s'exécute en un temps acceptable par rapport aux grand nombre de sommets ajoutés. De ce fait, notre solution optimise la complexité spatiale et temporelle pour les grands graphes du web (jusqu'à 1 Go).

Pour bien apprécier les résultats de notre structure, il faut augmenter le nombre de nouveaux sommets et diminuer la valeur de α .

Pour bien observer le comportement de PageRank avec Erdos, il faut varier un grand nombre de nouveaux sommets.

NB1 : Pour tester et exécuter notre implémentation nous avons utilisé des laptop SSD performants avec 16Go de Ram et des processeurs Intel i7.

NB2 : Nos résultats numériques sont dans le fichier **Analyses.xlsx**, le code est dans le fichier **main3.c** et les données sont dans le dossier **data**.

5. Bibliographie

Matrices creuses :

<https://bthierry.pages.math.cnrs.fr/course-fem/lecture/elements-finis-triangulaires/matrice-creuse/>

<https://bthierry.pages.math.cnrs.fr/course-fem/lecture/elements-finis-triangulaires/matrice-creuse/>

https://fr.wikipedia.org/wiki/Matrice_creuse

PageRank :

Cours (pagerank 98) / TDs / TPs

<https://fr.wikipedia.org/wiki/PageRank>

<https://www.webrankinfo.com/dossiers/pagerank/formule>

Erdos et graphes aléatoires :

https://www.geeksforgeeks.org/erdos-renyi-model-generating-random-graphs/?fbclid=IwAR2Xu_GmgQZRAJR9KbSqzfPSEBuOdW_0F5Cby6SvMjqv4aVvsYJsc2dUVeo

<https://pagespro.isae-superaero.fr/IMG/pdf/notes-eisc-102.pdf?fbclid=IwAR3D7iHD5vd54MtKA6pD63fLt08wRbGT5PSOKUKuJoO46sAk18nkQzMH4M8>

http://math.uchicago.edu/~may/REU2017/REUPapers/Tesliuc.pdf?fbclid=IwAR1bSyaWb6AJ0T88tH50l5G_LfxysZefGz86D0i1b7vBxI5YPSMUma86DP4

<http://perso.ens-lyon.fr/eric.thierry/Graphes2010/vincent-picard.pdf?fbclid=IwAR326fOPGqPFjUPbAc5qadjjgQBJ0ClCSRhBsWogy8qpc2Id7SNWGA7AEfc>

6. Annexe

```
#include <math.h>
#include <stdio.h>
#include <sys/time.h>
#include <stdlib.h>
#include <string.h>

int main(){

    /***** Ouverture et manipulation du fichier *****/

    printf(" Graphes du Web : \n");
    printf("-----\n");
    printf(" 1 - wb-cs-standford      0.6Mo\n");
    printf(" 2 - Stanford              37Mo\n");
    printf(" 3 - Stanford_BerkeleyV2   120Mo\n");
    printf(" 4 - in-2004v2             277Mo\n");
    printf(" 5 - wikipedia-20051105V2  309Mo\n");
    printf(" 6 - wb-edu                1006Mo\n");
    printf("-----\n");

    int choice;
    printf(" Choisir un des 6 graphes du web sur lequel effectuer les analyses : ");
    scanf("%d", &choice);

    char filename[] = "";

    switch(choice){
        case 1:
            strcpy(filename,"data/wb-cs-standford.txt");
            //filename = "data/wb-cs-standford.txt";
            break;
        case 2:
            strcpy(filename,"data/Stanford.txt");
            break;
        case 3:
            strcpy(filename,"data/Stanford_BerkeleyV2.txt");
            break;
        case 4:
            strcpy(filename,"data/in-2004v2.txt");
            break;
        case 5:
            strcpy(filename, "data/wikipedia-20051105V2.txt");
            break;
        case 6:
            strcpy(filename,"data/wb-edu.txt");
            break;
        default:
            printf("Erreur de choix !");
    }

    FILE *fp;
    if((fp = fopen(filename,"r")) == NULL)
```

```

{
    fprintf(stderr, "[Error] Cannot open the file");
    exit(1);
}

// CHOIX DES PARAMETRES
printf(" Choisir le nombre de nouveaux sommets : ");
int nbr;
scanf("%d", &nbr);

printf(" Choisir la probabilite de transition (p) entre 0 et 1 : ");
float prob;
scanf("%f", &prob);

printf(" Choisir le facteur d'amortissement / Dumping Factor (alpha)
entre 0 et 1 : ");
float alpha;
scanf("%f", &alpha);

/***** Les variables *****/

// Initialisation des variables de temps d'execution
clock_t begin, end;
double time_spent;
begin = clock();

// Lecture des donnees
int n, e;
char ch;
char str[100];
//ch = getc(fp);

/* while(ch == '#') {
    fgets(str, 100-1, fp);

    //printf("%s", str);
    sscanf (str, "%*s %d %*s %d", &n, &e); //nombres de noeud
    ch = getc(fp);
}*/

fgets(str, 100-1, fp);
sscanf(str, "%d", &n);
fgets(str, 100-1, fp);
sscanf(str, "%d", &e);
//ungetc(ch, fp);

// DEBUG: Print the number of nodes and edges, skip everything else
printf("\nInformations sur le graphe choisi :\n\n Nodes: %d, Edges: %d
\n\n", n, e);
printf("WE REACHED THIS PART OF CODE 1");
/***** CSR Structure de la matrice *****/

/* Compressed sparse row format:
    - Val vector: contains 1.0 if an edge exists in a certain row
    - Col_ind vector: contains the column index of the corresponding value
in 'val'
    - Row_ptr vector: points to the start of each row in 'col_ind'
*/

```



```

float *val = calloc(e, sizeof(float));
int *col_ind = calloc(e, sizeof(int));
int *row_ptr = calloc(n+1, sizeof(int));

// The first row always starts at position 0
row_ptr[0] = 0;

int fromnode, tonode;
int cur_row = 0;
int i = 0;
int j = 0;
// Elements for row
int elrow = 0;
// Cumulative numbers of elements
int curel = 0;

FILE *wptr;
wptr = fopen("iter.txt", "w");
while(!feof(fp)){

    fscanf(fp,"%d%d",&fromnode,&tonode);

    // printf("From: %d To: %d\n",fromnode, tonode);

    if (fromnode > cur_row) { // change the row
        curel = curel + elrow;
        int k = 0;
        for (k = cur_row + 1; k <= fromnode; k++) {
            row_ptr[k] = curel;
        }
        elrow = 0;
        cur_row = fromnode;
    }

    val[i] = 1.0;
    col_ind[i] = tonode;
    elrow++;
    i++;
    int nods;
    int save = 0;

    for(nods=0; nods<nbr; nods=nods+100){
        save = elrow;
    }
    end = clock();
    time_spent = (double)(end - begin);

    fprintf(wptr, "Nombre d'iteration: %d\n", i);
    fprintf(wptr, "Temp d'execution: %.2fs\n", time_spent/10000);
}

row_ptr[cur_row+1] = curel + elrow - 1;

// Fixer la stochastization
int out_link[n];
for(i=0; i<n; i++){
    out_link[i] = 0;
}

int rowel = 0;

```

```

for(i=0; i<n; i++){
    if (row_ptr[i+1] != 0) {
        rowel = row_ptr[i+1] - row_ptr[i];
        out_link[i] = rowel;
    }
}

int curcol = 0;
for(i=0; i<n; i++){
    rowel = row_ptr[i+1] - row_ptr[i];
    for (j=0; j<rowel; j++) {
        val[curcol] = val[curcol] / out_link[i];
        curcol++;
    }
}

/***** INITIALIZATION DE P ET D DAMPING FACTOR *****/

// Set the damping factor 'd'
float d = alpha;

// Initialize p[] vector
float p[n];
for(i=0; i<n; i++){
    //p[i] = 1.0/n;
    p[i] = prob;
}
//SOME CHANGES COULD BE DONE HERE

/***** PageRank BOUCLE *****/

// configuration de la condition de sortie et le nombre d'iterations
(max) 'k'
int looping = 1;
int k = 0;

// initialisation d'un nouveau vecteur p
float p_new[n];

while (looping){

    for(i=0; i<n; i++){
        p_new[i] = 0.0;
    }

    int curcol = 0;

    // Pagerank modifiee algorithm
    for(i=0; i<n; i++){
        rowel = row_ptr[i+1] - row_ptr[i];
        for (j=0; j<rowel; j++) {
            p_new[col_ind[curcol]] = p_new[col_ind[curcol]] + val[curcol] *
p[i];
            curcol++;
        }
    }
}

```

```

// DANGELING ELEMENTS
for(i=0; i<n; i++){
    p_new[i] = d * p_new[i] + (1.0 - d) / n;
}

// CHECK EXIT CONDITION
float error = 0.0;
for(i=0; i<n; i++) {
    error = error + fabs(p_new[i] - p[i]);
}
// EPSILON
if (error < 0.000001){
    looping = 0;
}

// mettre a jour p[]
for (i=0; i<n;i++){
    p[i] = p_new[i];
}

// Nombre d'iterations
k = k + 1;
}

/***** CONCLUSIONS *****/

// STOP TIMER
end = clock();
time_spent = (double)(end - begin); // CLOCKS_PER_SEC;

//
//Sleep(500);

// Print resultats
printf ("\nNumber d'iteration pour converger %d \n\n", k);
printf ("Valeur finale de pagerank:\n\n");
for (i=0; i<n; i++){
    printf("%f ", p[i]);
    if(i!=(n-1)){ printf(", "); }
}
printf("\n\nTEMPS D'EXECUTION': %f seconds.\n", time_spent);

return 0;
}

```