SDSC 3001 Course Project Report

Research of PageRank and its modified version
(adaptive PageRank and modified adaptive PageRank)

Group members:
Chan Yuk Yee 56230549
Jin Yu Heng 55966786
Leung Tsan Lim 56211178

Table of contents:

1. Research Problem

In this report, we reproduced the techniques introduced in a paper, Adaptive methods for the computation of PageRank (Kamvar, Golub, 2004), which is about designing a new simple algorithm to speed up the computation of PageRank.

Pagerank, a new and useful computation algorithm designed by Larry Page (the CEO of Google). It uses the Power method to calculate the principal eigenvectors of the matrix described by the hyperlinks in the network while the application is mainly on a search engine, measuring the importance of the website, which indicates an important role in internet searching activity.

The paper indicated that our website faces a huge improvement that needs more computation with a large sheer size. To design a simple algorithm to **speed up the computation of PageRank,** in which the PageRank of pages that **have converged is not recomputed** during each iteration after convergence will be solved by performing adaptive PageRank computation.

Our topic --- Adaptive PageRank and Modified Adaptive PageRank are novel algorithms that have been researched by Prof. Sepandar Kamvar and Gene Golub.

Besides, from this paper, we found out that pages with large PageRank values can slow down the rate of convergence.

2. Challenge of the research problem

Before we move onto the main topic, it is necessary that we state the difficulties we faced during our projects.

First of all, the convergence rate of the Power method is very fast and lots of PageRank **are not being identified** since these pages have been converged too early. This is a special **difficulty whenever we want to compute a matrix** using the Pagerank method.

There are other issues in the usage of adaptive PageRank. Like **recording the**

web matrix at each iteration is usually expensive. That is the reason why a lot of **fast eigensolvers are not feasible** in PageRank because the size and the sparsity of the web matrix make an inversion of factorization prohibitively very expensive.

3. The solution proposed by the paper

1. Adaptive PageRank algorithm

Reducing running time can be decided by eliminating redundant computation. **Adaptive algorithm is for the elimination of redundancies**, like the pages that have already converged to other pages.

Algorithm 3. Adaptive PageRank

function **adaptivePR**$(A, \vec{x}^{(0)}, \vec{v})$ {

repeat

$\vec{x}_N^{(k+1)} = A_N \vec{x}^{(k)}$;

$\vec{x}_C^{(k+1)} = \vec{x}_C^{(k)}$;

$[N, C] = \text{detectConverged}(\vec{x}^{(k)}, \vec{x}^{(k+1)}, \epsilon)$;

periodically, $\delta = \|A\vec{x}^{(k)} - \vec{x}^{k}\|_1$;

until $\delta < \epsilon$;

return $\vec{x}^{(k+1)}$;

}

Xn: the PageRank that has not yet converged

Xc: the PageRank that has already converged.

An: the matrix experienced modification.

detectConverged: the PageRank xi of page i has converged when

$$\left| x_i^{(k+1)} - x_i^{(k)} \right| \Big/ \left| x_i^{(k)} \right| < 10^{-3}$$

We want to reduce the cost of iteration. Reordering the matrix A in each iteration is expensive and slow. We get matrix An that is much smaller than the original matrix A. Since **adaptive PageRank ignores pages already converged**. Therefore after the computation. We will neglect the Xc after each computation and the matrix we use is An will not make each iteration include

Xc — the pages that are too small and already converged. After each recycles in function adaptive PageRank. The space and time consumption will be largely reduced to **reduce the cost and fulfill our destiny**.

Advantages of Adaptive PageRank

Adaptive PageRank has two main improvements to the original PageRank. Reducing the time spent on computation that improves the speed of computation is the main idea of our project and the destination why the adaptive PageRank has been designed. According to our results which we get from coding. The algorithm **speeds up the computation of PageRank by nearly 30%**. (at least 10% and at most 26%)

Besides, identifying pages in each iteration that have converged is not expensive at all. Therefore because the reduction of matrices that reduce the cost and identification of converged pages is not expensive. That will highly reduce our cost of calculation because **the space occupied will be reduced** and no more. Hardware will lower the probability of overhitting.

2. Modified Adaptive PageRank algorithm

Because of the links from these pages in C, the algorithm does not recalculate the PageRanks component of these pages in N, thereby **further reducing redundant calculations**.

Algorithm 5. Modified Adaptive PageRank

function **modifiedAPR**$(A, \vec{x}^{(0)}, \vec{v})$ {
repeat

$\vec{x}_N^{(k+1)} = A_{NN}\vec{x}_N^{(k)} + \vec{y}$;

$\vec{x}_C^{(k+1)} = \vec{x}_C^{(k)}$;

periodically,

$[N, C] = \text{detectConverged}(\vec{x}^{(k)}, \vec{x}^{(k+1)}, \epsilon)$;

$\vec{y} = A_{CN}\vec{x}_C^{(k)}$;

periodically, $\delta = \|A\vec{x}^{(k)} - \vec{x}^{k}\|_1$;

until $\delta < \epsilon$
return $\vec{x}^{(k+1)}$;
}

ANN: links from pages that have not converged to pages that have not converged

ACN: links from pages that have converged to pages that have not converged

Advantages of Modified Adaptive PageRank

By reducing redundant computation by not recomputing the PageRanks of those pages in C, it pops out those pages which are converging. Therefore the total computation memory is reduced, which reduces the cost of computation memory and the computation time also.

4. Our experiments

We adopted three large-scale network datasets to implement in adaptive PageRank and modified adaptive PageRank. Dataset is Com_DBLP, Web-google, and Web Berkstan respectively.

Description of three datasets as below:

Datasets	Nodes	Edges	Description	Type
com-DBLP	**317,080**	**1,049,866**	DBLP collaboration network	**Undirected**
web-Google	**875,713**	**5,105,039**	Web graph from Google	**Directed**
web-BerkStan	**685,230**	**7,600,595**	Web graph of Berkeley and Stanford	**Directed**

Source code:

1. Original PageRank

Algorithm 2. PageRank

function **pageRank**$(A, \vec{x}^{(0)}, \vec{v})\{$
repeat
$\vec{x}^{(k+1)} = A\vec{x}^{(k)}$;
$\delta = \|\vec{x}^{(k+1)} - \vec{x}^{(k)}\|_1$;
until $\delta < \epsilon$;
return $\vec{x}^{(k+1)}$;
$\}$

```
for t in range(max_iter):
    xlast = x
    x = alpha * (x * M ) + (1 - alpha) * p
    # check convergence, l1 norm
    err = np.absolute(x - xlast).sum()
    print(f"Iteration {t+1}:",x)
    if err < N * tol:
        end = time.time()
        NM_time = end-start
        print('PageRank running time:' ,NM_time)
        return dict(zip(nodelist, map(float, x)))
print()
raise nx.PowerIterationFailedConvergence(max_iter)
```

2. Adaptive PageRank

Algorithm 3. Adaptive PageRank

function **adaptivePR**$(A, \vec{x}^{(0)}, \vec{v})$ {

repeat

$$\vec{x}_N^{(k+1)} = A_N \vec{x}^{(k)};$$

$$\vec{x}_C^{(k+1)} = \vec{x}_C^{(k)};$$

$$[N, C] = \text{detectConverged}(\vec{x}^{(k)}, \vec{x}^{(k+1)}, \epsilon);$$

periodically, $\delta = \|A\vec{x}^{(k)} - \vec{x}^{k}\|_1;$

until $\delta < \epsilon$;

return $\vec{x}^{(k+1)}$;

}

```python
for t in range(max_iter):
    xlast = x.copy()
    x_change = alpha * (x * M[:,list_N]) + (1 - alpha) * p[list_N]
    x[list_N] = x_change
    print(f"Iteration {t+1}:",x)

    # check convergence, l1 norm
    err = np.absolute(x - xlast).sum()
    list_N = np.where(((abs(x-xlast)/abs(x))) > 1e-3)[0]
    if err < N * tol or not any(list_N):
        end = time.time()
        AP_time = end-start
        print('AdaptovePR runnign time: ',AP_time)
        return dict(zip(nodelist, map(float, x)))
print()
raise nx.PowerIterationFailedConvergence(max_iter)
```

3. Modified Adaptive PageRank

Algorithm 5. Modified Adaptive PageRank

function **modifiedAPR**$(A, \vec{x}^{(0)}, \vec{v})$ {

repeat

$$\vec{x}_N^{(k+1)} = A_{NN}\vec{x}_N^{(k)} + \vec{y};$$

$$\vec{x}_C^{(k+1)} = \vec{x}_C^{(k)};$$

periodically,

$$[N, C] = \text{detectConverged}(\vec{x}^{(k)}, \vec{x}^{(k+1)}, \epsilon);$$

$$\vec{y} = A_{CN}\vec{x}_C^{(k)};$$

periodically, $\delta = \|A\vec{x}^{(k)} - \vec{x}^{k}\|_1;$

until $\delta < \epsilon$

return $\vec{x}^{(k+1)}$;

}

```python
for t in range(max_iter):
    xlast = x.copy()
    x_change = alpha * (x[list_N]* M[:,list_N][list_N] + y ) + (1 - alpha) * p[list_N]

    x[list_N] = x_change
    print(f"Iteration {t+1}:",x)

    # check convergence, l1 norm
    err = np.absolute(x - xlast).sum()

    list_N = np.where((abs(x-xlast)/abs(x)) > 1e-3)[0]
    list_C = np.where((abs(x-xlast)/abs(x)) <= 1e-3)[0]
    if err < N * tol :
        end = time.time()
        MP_time = end-start
        print('ModifiedPR running time: ', MP_time)
        return dict(zip(nodelist, map(float, x)))
    y = x[list_C]* M[:,list_N][list_C]
print()
raise nx.PowerIterationFailedConvergence(max_iter)
```

5. Experimental Results and Performances

1. Com-DBLP dataset

```
DBLP collaboration network:
Nodes:  317080
Edges:  1049866

Iteration 1: [3.60748554e-06 7.49855665e-06 4.73966604e-06 ... 2.82713691e-06
 3.01185821e-06 3.01185821e-06]
Iteration 2: [3.63280289e-06 7.54611876e-06 4.65304129e-06 ... 2.83660366e-06
 3.00583224e-06 3.00583224e-06]
Iteration 3: [3.63248821e-06 7.55001258e-06 4.65684154e-06 ... 2.83552969e-06
 3.00484213e-06 3.00484213e-06]
Iteration 4: [3.63264339e-06 7.55118308e-06 4.65671608e-06 ... 2.83560544e-06
 3.00476818e-06 3.00476818e-06]
PageRank running time: 15.864495277404785

Iteration 1: [3.60748554e-06 7.49855665e-06 4.73966604e-06 ... 2.82713691e-06
 3.01185821e-06 3.01185821e-06]
Iteration 2: [3.63280289e-06 7.54611876e-06 4.65304129e-06 ... 2.83660366e-06
 3.00583224e-06 3.00583224e-06]
Iteration 3: [3.63248821e-06 7.55001258e-06 4.65684154e-06 ... 2.83552969e-06
 3.00484213e-06 3.00484213e-06]
Iteration 4: [3.63248821e-06 7.55001258e-06 4.65684154e-06 ... 2.83552969e-06
 3.00484213e-06 3.00484213e-06]
AdaptovePR runnign time:  14.03819990158081

Iteration 1: [3.60748554e-06 7.49855665e-06 4.73966604e-06 ... 2.82713691e-06
 3.01185821e-06 3.01185821e-06]
Iteration 2: [3.63280289e-06 7.54611876e-06 4.65304129e-06 ... 2.83660366e-06
 3.00583224e-06 3.00583224e-06]
Iteration 3: [3.63248821e-06 7.55001258e-06 4.65684154e-06 ... 2.83552969e-06
 3.00484213e-06 3.00484213e-06]
Iteration 4: [3.63248821e-06 7.55001258e-06 4.65684154e-06 ... 2.83552969e-06
 3.00484213e-06 3.00484213e-06]
ModifiedPR running time:  12.540035486221313
```

Run Time:

Original PageRank: 15.86

Adaptive PageRank: 14.03

Modified Adaptive PageRank: 12.54

2. Web-Google dataset

```
Web graph from Google network:
Nodes:  875713
Edges:  5105039

Iteration 1: [4.84933278e-06 4.63393238e-06 1.06150406e-06 ... 9.99237060e-07
 1.06921100e-06 1.07618814e-06]
Iteration 2: [4.56483311e-06 4.37841984e-06 1.08105103e-06 ... 1.00197547e-06
 1.07276657e-06 1.08064677e-06]
Iteration 3: [4.59309155e-06 4.40423382e-06 1.07986034e-06 ... 1.00163544e-06
 1.07274988e-06 1.08050814e-06]
Iteration 4: [4.59115744e-06 4.40242496e-06 1.08001767e-06 ... 1.00164795e-06
 1.07277279e-06 1.08055149e-06]
PageRank running time: 37.5762095451355

Iteration 1: [4.84933278e-06 4.63393238e-06 1.06150406e-06 ... 9.99237060e-07
 1.06921100e-06 1.07618814e-06]
Iteration 2: [4.56483311e-06 4.37841984e-06 1.08105103e-06 ... 1.00197547e-06
 1.07276657e-06 1.08064677e-06]
Iteration 3: [4.59309155e-06 4.40423382e-06 1.07986034e-06 ... 1.00163544e-06
 1.07274988e-06 1.08050814e-06]
Iteration 4: [4.59118108e-06 4.40244983e-06 1.08001765e-06 ... 1.00163544e-06
 1.07274988e-06 1.08050814e-06]
AdaptovePR runnign time:  28.393141508102417

Iteration 1: [4.84933278e-06 4.63393238e-06 1.06150406e-06 ... 9.99237060e-07
 1.06921100e-06 1.07618814e-06]
Iteration 2: [4.56483311e-06 4.37841984e-06 1.08105103e-06 ... 1.00197547e-06
 1.07276657e-06 1.08064677e-06]
Iteration 3: [4.59309155e-06 4.40423382e-06 1.07986034e-06 ... 1.00163544e-06
 1.07274988e-06 1.08050814e-06]
Iteration 4: [4.59118108e-06 4.40244983e-06 1.08001765e-06 ... 1.00163544e-06
 1.07274988e-06 1.08050814e-06]
ModifiedPR running time:  22.67695116996765
```

Run Time:

Original PageRank: 37.57

Adaptive PageRank: 28.39

Modified Adaptive PageRank: 22.67

3. Web-BerkStan dataset

```
Web graph of Berkeley and Stanford :
Nodes:  685230
Edges:  7600595

Iteration 1: [7.41128243e-06 1.39760851e-06 1.42320225e-06 ... 1.28910487e-06
 1.35198491e-06 1.35198491e-06]
Iteration 2: [6.80845847e-06 1.40535946e-06 1.42581459e-06 ... 1.28552557e-06
 1.35855436e-06 1.35855436e-06]
Iteration 3: [6.88400199e-06 1.40465732e-06 1.42560282e-06 ... 1.28574455e-06
 1.35622582e-06 1.35622582e-06]
Iteration 4: [6.87536433e-06 1.40466048e-06 1.42556395e-06 ... 1.28566694e-06
 1.35647518e-06 1.35647518e-06]
PageRank running time: 51.494282245635986

Iteration 1: [7.41128243e-06 1.39760851e-06 1.42320225e-06 ... 1.28910487e-06
 1.35198491e-06 1.35198491e-06]
Iteration 2: [6.80845847e-06 1.40535946e-06 1.42581459e-06 ... 1.28552557e-06
 1.35855436e-06 1.35855436e-06]
Iteration 3: [6.88400199e-06 1.40465732e-06 1.42560282e-06 ... 1.28574455e-06
 1.35622582e-06 1.35622582e-06]
Iteration 4: [6.87552557e-06 1.40465732e-06 1.42560282e-06 ... 1.28574455e-06
 1.35647521e-06 1.35647521e-06]
AdaptovePR runnign time:  40.56775784492493

Iteration 1: [7.41128243e-06 1.39760851e-06 1.42320225e-06 ... 1.28910487e-06
 1.35198491e-06 1.35198491e-06]
Iteration 2: [6.80845847e-06 1.40535946e-06 1.42581459e-06 ... 1.28552557e-06
 1.35855436e-06 1.35855436e-06]
Iteration 3: [6.88400199e-06 1.40465732e-06 1.42560282e-06 ... 1.28574455e-06
 1.35622582e-06 1.35622582e-06]
Iteration 4: [6.87552557e-06 1.40465732e-06 1.42560282e-06 ... 1.28574455e-06
 1.35647521e-06 1.35647521e-06]
ModifiedPR running time:  41.123403549194336
```

Run Time:

Original PageRank: 51.49

Adaptive PageRank: 40.56

Modified Adaptive PageRank: 41.12

Improve the percentage of acceleration of computation :

Datasets	Adaptive PageRank	Modified Adaptive PageRank
com-DBLP	**11.5%**	**20.9%**
web-Google	**24.4%**	**39.65%**
web-BerkStan	**21.2%**	**20.13%**

Differences between the original PageRank and its adaptive version:

(Total sum of Square(original PageRank - adaptive version))

Datasets	Adaptive PageRank	Modified Adaptive PageRank
com-DBLP	**4.768e-13**	**4.768e-13**
web-Google	**1.025e-18**	**1.025e-18**
web-BerkStan	**1.847e-13**	**1.847e-13**

We can see that adaptive and modified adaptive PageRank perform better than original PageRank even though web Berkstan and Web google are more complex than DBLP. Adaptive Pagerank performs **very well especially for directed graphs** even with a larger number of nodes and edges.

Modified PageRank has accelerated the computation of Google's website by **almost 40%**.

6. Influence on real-world application

Increasing the computation rate plays an important role on both client-side and the server-side. On the client-side, although the waiting time is not long on the scale of 40%, the iteration loading times can release more memory in the RAM of the computer, as much as 8GB, which is especially important given that multitasking is the trending of the teenagers (i.e. listening to music while surfing the internet). On the server side, it can also improve the waiting duration for the task to be performed, as the total computation time is decreased by 40%, which means the server performance can be improved in this 40% time. This benefits the large company by reducing the cost/effort of building larger server hardware and upgrading the server-side applications.

7. Conclusion

We can see that adaptive PageRank is very useful for computation. The two methods we adapted are called adaptive PageRank and modified adaptive PageRank. Reducing time for computation by avoiding redundant computation— the pages with low PageRank values since these pages converged to faster than pages with high PageRank values. These two methods perform better than the original PageRank especially for directed graphs whether those graphs are more complicated or not. By using adaptive PageRank, the RAM will release more memory and thus fasten the loading time, reducing hardware overhitting. Moreover, since the time for computation is reduced. It reduces many costs such as hardware improvement and enlargement.

Reference

1. Kamvar, S.D., Haveliwala, T.H., & Golub, G.H. (2004). Adaptive methods for the computation of PageRank. *Linear Algebra and its Applications, 386*, 51-65.