Adaptive methods for the computation of PageRank

1.

Induction

Let’s start with the first set of slides

Agenda

- Research Problem
- Challenges of the resaerch problem
- The solution proposed by the paper
- Our experiment
- Evaluation

Research paper induction

- Topic: *Adaptive methods for the computation of PageRank*
- Aim: To design a simple algorithm to **speed up the computation of PageRank**, in which the PageRank of pages that have converged **are not recomputed** at each iterations after convergence.

Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Linear Algebra and its Applications 386 (2004) 51–65

LINEAR ALGEBRA AND ITS APPLICATIONS

www.elsevier.com/locate/laa

Adaptive methods for the computation of PageRank

Sepandar Kamvar [a,*], Taher Haveliwala [b], Gene Golub [a]

[a] *Scientific Computing and Computational Mathematics, Stanford University, P.O. Box 18544, Stanford, CA 94305, USA*

[b] *Department of Computer Science, Stanford University, Stanford, CA 94305, USA*

Received 12 August 2003; accepted 13 December 2003

Submitted by C. Meyer

Abstract

We observe that the convergence patterns of pages in the PageRank algorithm have a non-uniform distribution. Specifically, many pages converge to their true PageRank quickly, while relatively few pages take a much longer time to converge. Furthermore, we observe that these

Reasons for using adaptive pagerank

1. Speed up the computation speed for reducing time spending.

2. require method to calculate many pagerank vector

2.research problem

1.High Pagerank cause converge slow

2.Whether adaptive pagerank can give us faster speed than original method with large datasets

3.Challenge of research problem

1.Recording the web matrix at each iteration is expensive

2.Convergence rate of the Power method is very fast make lots of pagerank are not being identified

3.Many fast eigen solvers are not feasible

5.adaptive pagerank solution

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

}

6.Advantage of adaptive pagerank

1.Reduce the time spend on computation(faster speed)

2.identifying pages in each iteration that have coverged is cheap

Our Experiment - source code

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)
```

Our Experiment - source code

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

}

```
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)
```

Our Experiment - source code

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)
```

Our Experiment - dataset

Dataset Implemented:

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**

Performance - running time

1. com-DBLP

```
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:

PageRank: 15.86
Adaptive PR: 14.03
Modified Adaptive PR: 12.54

2. web-Google

```
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:

PageRank: 37.57
Adaptive PR: 28.39
Modified Adaptive PR: 22.67

3. web-BerkStan

```
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:

PageRank: 51.49
Adaptive PR: 40.56
Modified Adaptive PR: 41.12

Performance - percentage of acceleration

Improve percentage of acceleration of computation :

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

Conclusion

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%.**

Thank you!