CPSC 517 Final Project
The inner-outer solver for Pagerank combined with a subsampling scheme

Name: Brock Hargreaves

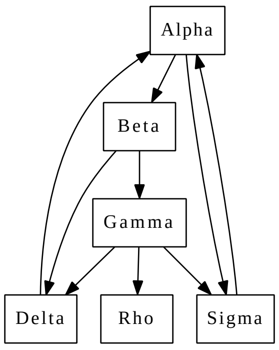
December 18, 2013

Abstract

We investigate what the effect of a low rank approximation for the transition matrix has on the power method and an inner-outer iteration for solving the Pagerank problem. The purpose of the low rank approximation is two fold: (1) to reduce memory requirements (2) to decrease computational time. We show that we see an improvement in storage requirements and a decrease in computational time if we discard the time it takes to perform the low rank approximation, however at the sacrifice of accuracy.

1 Introduction to Pagerank

The problem of Pagerank is a simple one to state: Given a collection of websites, how do we rank them? The primary way of formulating this as a mathematical problem is to represent the collection of websites as a graph, where each websites is considered as a node(vertex) and their links are considered edges between the nodes. This is a directed graph, as it is not necessary for a webpage to link back to a linking neighbor. Let us consider the following graph and it's associated adjacency matrix:



$$\begin{array}{c|cccccc} & Alpha & Beta & Gamma & Delta & Rho & Sigma \\ \hline Alpha & 0 & 0 & 0 & 1 & 0 & 1 \\ Beta & 1 & 0 & 0 & 0 & 0 & 0 \\ Gamma & 0 & 1 & 0 & 0 & 0 & 0 \\ Delta & 0 & 1 & 1 & 0 & 0 & 0 \\ Rho & 0 & 0 & 1 & 0 & 0 & 0 \\ Sigma & 1 & 0 & 1 & 0 & 0 & 0 \end{array}$$

It will be convenient for us to define the out-degree of a node in the graph. The out-degree, n_j, of node w_j is the number of outgoing links from w_j. This can be calculated as the column sums of the adjacency matrix. For example, the outdegrees for Gamma, Delta and Rho, are 3, 1, and 0 respectively.

What makes a webpage more important than another? The idea behind Pagerank is that a page is considered important, if other important pages link to it. It's not hard to imagine then, that the Pagerank π_i of a webpage w_i is defined recursively[2] with weighted sums. The weights of which are the inverse out-degrees:

$$\pi_i = \sum_{j \to i} \frac{\pi_j}{n_j} \text{ ; } n_j \text{ is the out-degree of page j}$$

This can be reformulated in matrix notation, so that the problem is to find a vector π that satisfies $\pi = \bar{P}\pi$ where

$$\bar{P}_{j,i} = \begin{cases} \frac{1}{n_i} & : i \to j \\ 0 & : i \not\to j \end{cases}$$

Currently there are issues with this formulation, one of which can be seen in our toy example. Note that Rho has a column of all zeroes. From a model perspective, if a surfer clinks on a link to Rho, he will never be able to leave this site. This can be fixed if we perturb those nodes with no outgoing links. Nodes with no outgoing links are referred to as dangling nodes in the literature and we perturb the transition matrix P with a rank-1 matrix:

$$P = \bar{P} + ud^T \text{ ; } d_i = \begin{cases} 1 & \text{if } n_i = 0 \\ 0 & otherwise \end{cases}$$

where u is a probability vector.

A generalization of this issue is that a surfer might wander into a subgraph and never leave the subgraph since there no outgoing links from the subgraph to other parts of the grpah. We fix this by allowing surfers to teleport anywhere in the graph. We essentially give them a probability to follow a link, or to teleport somewhere else in the graph in which case we define:

$$A = \alpha P + (1 - \alpha)\, ve^T \tag{1}$$

where v is a probability vector and e is a vector of ones. Then we wish to find x such that $Ax = x$ or rather, $\alpha Px + (1-\alpha)\, ve^T x = x$. There are a variety of properties that A now satisifes, one of which is that it is an irreducible column stochastic matrix. According to Perron-Frobenius Theory, an irreducible column-stochastic matrix has 1 as the largest eigenvalue. If we give uniform probability vectors u and v, for our toy example we obtain:

$$A = \begin{array}{c|cccccc} & Alpha & Beta & Gamma & Delta & Rho & Sigma \\ \hline Alpha & .025 & .025 & .025 & .875 & .166 & .875 \\ Beta & .450 & .025 & .025 & .025 & .166 & .025 \\ Gamma & .025 & .450 & .025 & .025 & .166 & .025 \\ Delta & .025 & .450 & .308 & .025 & .166 & .025 \\ Rho & .025 & .025 & .308 & .025 & .166 & .025 \\ Sigma & .450 & .025 & .308 & .025 & .166 & .025 \end{array}$$

Notice that it's column sums are 1. It is also important to note that we never actually compute this matrix in practice, which we will see in the following sections.

2 Iterative methods for the Pagerank Problem

The power method is a well studied algorithm for calculating the dominant eigenvector of a linear system. This method consists of generating the sequence of vectors $A^k v_0$ where v_0 is some nonzero initial vector. Under relatively mild conditions[5], this sequence indeed converges to the dominant eigenvector of the linear system $Ax = x$.

2.1 Power Method

Algorithm 1 Power Method for Pagerank (PI)

```
1: Input: An $n \times n$ matrix A, an initial vector $x_0$, tolerence $\epsilon$, maximum iterations
2: for k = 1 to maximum iterations do
3:     $y_k = Ax_k$
4:     $x_k = \frac{y_k}{||y_k||_1}$
5:     if $||x_k - x_{k-1}||_1 < \epsilon$ then
6:         return
7:     end if
8: end for
```

Note that we never explicitly form A and pass it into the Power method. If we make the convention that our solution should have $||x||_1 = 1$, we are left with $\alpha Px + (1-\alpha)\, v = x$. And the left hand side can be computed quickly using fast matrix vector products. So then in practice we pass A as a function handle, which computes Ax quickly. In our toy example A only had 36 entries and is small enough to store and compute explicitly, however in our numerical experiment section we deal with matrices for which

2.2 An inner-outer iteration

In this section we briefly outline the inner-outer stationary method of Greif et al [3]. As we saw in the previous section, we can reformulate $Ax = x$ as $\alpha Px + (1-\alpha)\, v = x$. After rearranging we obtain:

$$(I - \alpha P)\, x = (1-\alpha) v$$

Now supposing that β is a positive scalar. Notice that the Pagerank problem, $(I - \beta P)\, x = (1-\beta)v$ is easier to solve than the original if $\beta < \alpha$. This can be seen by investigating the eigenvalues of 1. Inspired by this fact, consider the stationary iteration:

$$(I - \beta P)x_{k+1} = (\alpha - \beta)Px_k + (1-\alpha)v \tag{2}$$

which will be referred to as the outer iteration. However solving this system with $I - \beta P$ is still computationally difficult. Thus an inner Richardson iteration is done. The right hand side of 2 is fixed to

$$f = (\alpha - \beta)Px_k + (1-\alpha)v$$

We then solve(approximately) an easier subproblem:

$$(I - \beta P)y = f$$

via the inner iteration $y_{j+1} = \beta P y_j + f$. Then set $x_{k+1} = y$, update the right hand side and repeat. This is outlined in Algorithm 2.

Algorithm 2 Basic inner-outer iteration (IO)

```
 1: Input: P, $\alpha$,$\beta$,$\tau$,$\eta$,$v$
 2: Ouput: x
 3: : $x \leftarrow v$
 4: $y \leftarrow Px$
 5: while $||\alpha y + (1-\alpha)v - x||_1 \geq \tau$ do
 6:     $f \leftarrow (\alpha - \beta)u + (1-\alpha)v$
 7:     repeat
 8:         $x \leftarrow f + \beta y$
 9:         $y \leftarrow Px$
10:     until $||f + \beta y - x||_1 < \eta$
11: end while
12: $x \leftarrow \alpha y + (1-\alpha)v$
```

This algorithm can be improved via running the power method after the inner iterations are close to convergence. However in this paper we only use the basic inner outer scheme for reasons we will describe in the discussion. There are a variety of implementation issues that are discussed in depth in the original paper [3].

3 Sampling scheme for low rank approximation

A largely studied topic in numerical linear algebra is low rank matrix approximation. Consider an $m \times n$ matrix A, we seek a solution of the following optimization problem

$$\arg\min_{\tilde{A}} ||A - \tilde{A}||_F$$
$$\text{subject to } \operatorname{rank}(\tilde{A}) \leq r$$

This solution of which can be obtained analytically from the singular value decomposition of A. For large matrices, the singular value decomposition becomes computationally intractable and the optimal low rank approximation cannot be obtained thus attention must be shifted to finding a *near-optimal* low rank approximation of A. For the remainder of this section, we focus on the results of Achlioptas et al 2007 [1] and Liu et al 2013 [4]. They focus on a random sampling technique which produces a low rank approximation $\hat{A}$ of A.

Theorem 1 (Achlioptas et al 2007 [1]). *Given any $m \times n$ matrix A with $m \leq n$ and any fixed $\epsilon > 0$, let $\hat{A}$ be a random matrix whose entries are independent random variables such that for all i, j : $\mathbb{E}[\hat{A}] = A_{i,j}$, $Var(\hat{A}_{i,j}) \leq \sigma^2$, and $\hat{A}_{i,j}$ takes values in an interval of length K, where*

$$K = \left(\frac{log(1+\epsilon)}{2log(m+n)} \right)^2 \times \sigma\sqrt{m+n}$$

. For any $\theta > 0$ and $m + n \geq 152$,

$$Pr[||A - \hat{A}||_2 \geq 2(1+\epsilon+\theta)\sigma\sqrt{m+n}] < 2exp\left(-\frac{16\theta^2}{\epsilon^4}(log(n)^4) \right)$$

This theorem gives rise to the notion of sampling the original matrix according to some distribution. The important question is how to we decide which distribution. The answer lies in the error of the approximation and is explained in detail in Achlioptas et al 2007 [1], but we'll give the argument again here which essentially argues why we don't just use a uniform distribution. Suppose we sparsify A by keeping every entry with the same probability p, we see that for every i, j

$$\text{Var}(\hat{A_{i,j}}) = \frac{1-p}{p} A_{i,j}$$

So then small entries of A give rise to random variables with comparatively small variance in $\hat{A}$. However Theorem 1 relies only on the maximum variance of the entries of $\hat{A}$. Then our goal would be to decrease the probabiliity of keeping each entries $A_{i,j}$ to some $p_{i,j} < p$, so that the entries in $\hat{A}$ have essentially the same variance. This will increase the number of omitted entries when entry magnitudes vary without affecting the bound on $||A - \hat{A}||_2$.

1. Let $\hat{A}_{i,j} = \frac{A_{i,j}}{p_{i,j}}$ with probability $p_{i,j}$ and 0 otherwise. Then $\mathbb{E}[\hat{A}_{i,j}] = A_{i,j}$ and $\text{Var}(\hat{A}_{i,j})$
2. Take $p_{i,j} = p(\frac{A_{i,j}}{b})^2$, where $p \in (0, 1]$ and $b = \max_{i,j} |A_{i,j}|$. Then $\text{Var}(\hat{A}_{i,j}) = \frac{b^2}{p} - A_{i,j}^2 \approx \frac{b^2}{p}$
3. The expected number of retained values is $\sum_{i,j} p(\frac{A_{i,j}}{b})^2 = p\frac{||A||_F^2}{b}$
4. Then we retain $(pmn) \times \text{Avg}[(\frac{A_{i,j}}{b})^2]$ entries in expectation, compared to pmn entries for uniform sampling.

This idea, and theorem, are manifested into an one-pass matrix sampling scheme[1] . Here we state the scheme given by Liu et al 2013 [4] but we should state that it is the same algorithm given by Achlioptas et al 2007 [1] but with an extra parameter for which they choose the parameter $\theta = \frac{(8log(n))^2}{\sqrt{n}}$ to match that of Achlioptas et al. Please excuse the abuse of notation, the α used in this algorithm is not the same α used as a model parameter for damping.

Algorithm 3 Non uniform sampling(A,α,θ) (A arbitrary)

```
1: Initialize: An empty priority queue, Q, $Z = 0$, $s = \frac{N}{\alpha^2}$
2: for each $A_{i,j}$ do
3:     Set $Z \leftarrow Z + A_{i,j}^2$
4:     draw $r_{i,j}$ uniformly and independently from $[0, 1]$
5:     insert $A_{i,j}$ in $Q$ with key $k_{i,j} = max\{\frac{sA_{i,j}^2}{r_{i,j}}, \frac{sA_{i,j}^2}{\theta^2 r_{i,j}^2}\}$
6:     remove all elements with key smaller than Z
7: end for
8: Set $\tilde{A}$ as the contents of Q
```

4 Experimental Results

We now perform the power method and the inner-outer iteration on several datasets, once with the original transition matrix, and once with the matrix obtain by by subsampling the transition matrix via Non uniform sampling(A,α,θ). All experiments in this section were conducted on a laptop Intel(R) Core(TM) i7 Quad Core with 8 gigabytes of memory. Each method was written Matlab, however the non uniform sampling scheme requires an efficient implementation of a queue. The queue itself was written in C++ and then mex compiled and called inside the Matlab code. The following is a list of notation used to describe the experiment results along with parameters:

Notation:

1. Power iterations: PI
2. Directly subsampled power iterations: DSPI
3. Inner outer iterations: IO
4. Directly subsampled inner outer iterations: DSIO
5. Relative error: $\frac{||x-\tilde{x}||_1}{||x||_1}$ where we take the solution from PI as x ie: as the true solution.
6. Pearson coefficients: $\rho(x,y) = \frac{\sum_i (x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_i (x_i-\bar{x})^2 \sum_i (y_i-\bar{y})^2}}$

Parameter selection:

1. Damping parameter: $\alpha = 0.85$
2. IO and DSIO parameters: $\beta = 0.5$, $\eta = 10e^{-4}$, and $\tau = 10e^{-5}$
3. PI and DSPI parameter: $\epsilon = 10e^{-5}$
4. Associate dangling and teleportation vectors: uniform probability vectors
5. Non uniform subsampling: $\alpha = \sqrt{2}$

Table 1: Summary of datasets

Dataset	Size	nonzeros	nonzeros sampled	Time to subsample (seconds)	Percentage of samples retained
ubc-cs	51,681	673,010	481,685	40	71.57
ubc	339,147	4,203,811	1,489,877	270	35.44
eu-2005	862,664	19,235,140	9,197,348	1296	47.82

Quantifying experimental results

Table 2: Wall time (seconds)

Dataset	PI	DSPI	IO	DSIO
ubc	.3159	.2997	.2078	.1864
ubc-cs	3.579	.1688	1.654	.942
eu-2005	9.275	7.06	4.95	3.49

Table 3: Relative 1-norm error

Dataset	PI	DSPI	IO	DSIO
ubc	-	.1926	.0001	.1644
ubc-cs	-	.3112	.0004	.2512
eu-2005	-	.1685	.0001	.1655

Table 4: Pearson Coefficients on rankings

Dataset	PI	DSPI	IO	DSIO
ubc	-	.9635	1	.9681
ubc-cs	-	.8925	1	.8934
eu-2005	-	.9677	1	.9684

4.1 Sparsity graphs of associated experiments

Sparsity graphs: ubc-cs

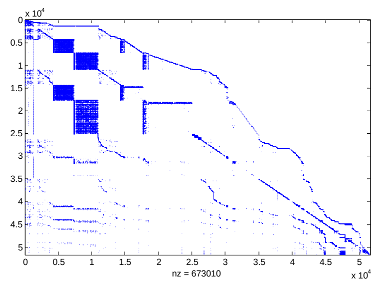


Figure 1: Non-zeros: 673,010

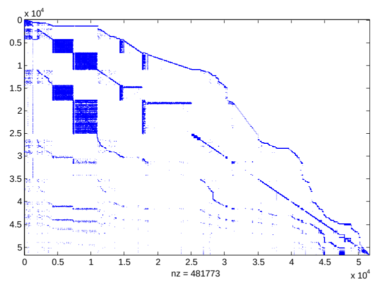


Figure 2: Non-zeros: 481,685

Sparsity graphs: ubc

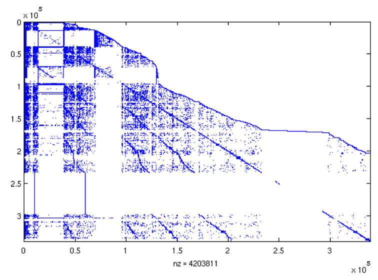


Figure 3: Non-zeros: 4,203,811

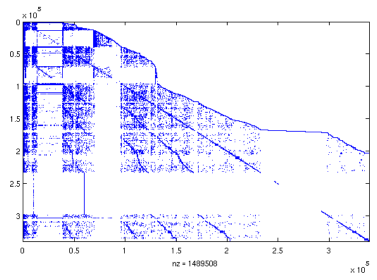


Figure 4: Non-zeros: 1,489,877

Sparsity graphs: eu-2005

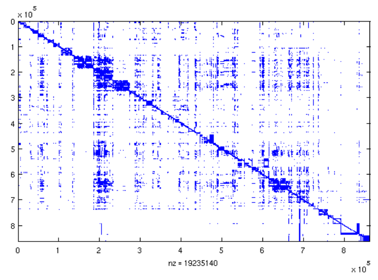


Figure 5: Non-zeros: 19,235,140

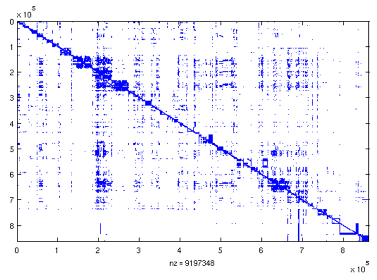


Figure 6: Non-zeros: 9,197,348

5 Discussion

If we discard the time it takes to sample the transition matrix, we indeed see an improvement in walltime for the methods for which we used subsampling. This is directly a result of reducing the number of non-zeroes in the transition matrix, thus speeding up matrix vector products. In applications where this subsampling only takes place once, and then solves for a variety of parameters, the time it takes to subsample should be negligible. However if this sampling needs to be done repeatedly, then it loses it's efficiency. Also, though it is not mentioned, the number of retained values from subsampling is significantly smaller than the original and will thus reduce the memory required for storage. Another method which is described in [1] is to sample the transition matrix inside the Power Method and adaptively adding samples to the transition matrix, however at least with the implementation we used for the sampling process, we find that this is not a feasible process.

It is relieving to see a reflection of the poor relative error in the Pearson coefficient for the ubc-cs dataset, though it's large value is hard to interpret as a sign that the method converged to the correct solution.

An interesting result was that we obtain approximately the same relative error for the eu-2005 and ubc dataset, even though they had significantly different subsampling rates. We think this can be related to the theoretical results behind the non uniform subsampling scheme in that as the size of the matrix grows, we obtain higher probabilities of obtaining a good low rank approximation of the matrix.

As mentioned in an earlier section, we use the inner-outer scheme, not the inner-outer scheme with power iterations. This is a result of not having an appropriate convergence criteria for the power iteration on the power iteration at the end of the inner loop when using subsampling techniques, and found that it only slowed our convergence down, this requires further investigation as the power iteration is intended(and has been shown) to to increase performance.

References

[1] Dimitris Achlioptas and Frank Mcsherry. Fast computation of low-rank matrix approximations. *J. ACM*, 54(2), April 2007.

[2] Erik Andersson and Per-Anders Ekström. Investigating googles pagerank algorithm. 2004.

[3] David F. Gleich, Andrew P. Gray, Chen Greif, and Tracy Lau. An inner-outer iteration for PageRank. *SIAM Journal of Scientific Computing*, 32(1):349–371, February 2010.

[4] Wenting Liu, Guangxia Li, and James Cheng. Fast pagerank approximation by adaptive sampling. *Knowledge and Information Systems*, pages 1–20, 2013.

[5] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003.