Extrapolation Methods for Accelerating PageRank Computations

Sepandar D. Kamvar
Stanford University
sdkamvar@stanford.edu

Taher H. Haveliwala
Stanford University
taherh@cs.stanford.edu

Christopher D. Manning
Stanford University
manning@cs.stanford.edu

Gene H. Golub
Stanford University
golub@stanford.edu

ABSTRACT

We present a novel algorithm for the fast computation of PageRank, a hyperlink-based estimate of the "importance" of Web pages. The original PageRank algorithm uses the Power Method to compute successive iterates that converge to the principal eigenvector of the Markov matrix representing the Web link graph. The algorithm presented here, called Quadratic Extrapolation, accelerates the convergence of the Power Method by periodically subtracting off estimates of the nonprincipal eigenvectors from the current iterate of the Power Method. In Quadratic Extrapolation, we take advantage of the fact that the first eigenvalue of a Markov matrix is known to be 1 to compute the nonprincipal eigenvectors using successive iterates of the Power Method. Empirically, we show that using Quadratic Extrapolation speeds up PageRank computation by 25–300% on a Web graph of 80 million nodes, with minimal overhead. Our contribution is useful to the PageRank community and the numerical linear algebra community in general, as it is a fast method for determining the dominant eigenvector of a matrix that is too large for standard fast methods to be practical.

Keywords

PageRank, link analysis, eigenvector computation

1. INTRODUCTION

The PageRank algorithm for determining the "importance" of Web pages has become a central technique in Web search [18]. The core of the PageRank algorithm involves computing the principal eigenvector of the Markov matrix representing the hyperlink structure of the Web. As the Web graph is very large, containing over a billion nodes, the PageRank vector is generally computed offline, during the preprocessing of the Web crawl, before any queries have been issued.

The development of techniques for computing PageRank efficiently for Web-scale graphs is important for a number of reasons. For Web graphs containing a billion nodes, computing a PageRank vector can take several days. Computing PageRank quickly is necessary to reduce the lag time from when a new crawl is completed to when that crawl can be made available for searching. Furthermore, recent approaches to personalized and topic-sensitive PageRank schemes [11, 20, 14] require computing *many* PageRank vectors, each biased towards certain types of pages. These approaches intensify the need for faster methods for computing PageRank.

Eigenvalue computation is a well-studied area of numerical linear algebra for which there exist many fast algorithms. However, many of these algorithms are unsuitable for our problem as they require matrix inversion, a prohibitively costly operation for a Web-scale matrix. Here, we present a series of novel algorithms devised expressly for the purpose of accelerating the convergence of the iterative PageRank computation. We show empirically on an 80 million page Web crawl that these algorithms speed up the computation of PageRank by 25–300%.

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM 1-58113-680-3/03/0005.

1.1 Preliminaries

In this section we summarize the definition of PageRank [18] and review some of the mathematical tools we will use in analyzing and improving the standard iterative algorithm for computing PageRank.

Underlying the definition of PageRank is the following basic assumption. A link from a page $u \in \mathit{Web}$ to a page $v \in \mathit{Web}$ can be viewed as evidence that v is an "important" page. In particular, the amount of importance conferred on v by u is proportional to the importance of u and inversely proportional to the number of pages u points to. Since the importance of u is itself not known, determining the importance for every page $i \in \mathit{Web}$ requires an iterative fixed-point computation.

To allow for a more rigorous analysis of the necessary computation, we next describe an equivalent formulation in terms of a random walk on the directed Web graph G. Let $u \to v$ denote the existence of an edge from u to v in G. Let $\deg(u)$ be the outdegree of page u in G. Consider a random surfer visiting page u at time k. In the next time step, the surfer chooses a node v_i from among u's out-neighbors $\{v|u \to v\}$ uniformly at random. In other words, at time $k+1$, the surfer lands at node $v_i \in \{v|u \to v\}$ with probability $1/\deg(u)$.

The PageRank of a page i is defined as the probability that at some particular time step $k > K$, the surfer is at page i. For sufficiently large K, and with minor modifications to the random walk, this probability is unique, illustrated as follows. Consider the Markov chain induced by the random walk on G, where the states are given by the nodes in G, and the stochastic transition matrix describing the transition from i to j is given by P with $P_{ij} = 1/\deg(i)$.

For P to be a valid transition probability matrix, every node must have at least 1 outgoing transition; i.e., P should have no rows consisting of all zeros. This holds if G does not have any pages with outdegree 0, which does not hold for the Web graph. P can be converted into a valid transition matrix by adding a complete set of outgoing transitions to pages with outdegree 0. In other words, we can define the new matrix P' where all states have at least one outgoing transition in the following way. Let n be the number of nodes (pages) in the Web graph. Let $\vec{v}$ be the n-dimensional column vector representing a uniform probability distribution over all

$\vec{y} = cP^T\vec{x}$;
$w = ||\vec{x}||_1 - ||\vec{y}||_1$;
$\vec{y} = \vec{y} + w\vec{v}$;

Algorithm 1: Computing $\vec{y} = A\vec{x}$

nodes:

$$\vec{v} = [\frac{1}{n}]_{n\times 1} \tag{1}$$

Let $\vec{d}$ be the n-dimensional column vector identifying the nodes with outdegree 0:

$$d_i = \begin{cases} 1 & \text{if } \deg(i) = 0, \\ 0 & \text{otherwise} \end{cases}$$

Then we construct P' as follows:

$$D = \vec{d} \cdot \vec{v}^T$$
$$P' = P + D$$

In terms of the random walk, the effect of D is to modify the transition probabilities so that a surfer visiting a dangling page (i.e., a page with no outlinks) randomly jumps to another page in the next time step, using the distribution given by $\vec{v}$.

By the Ergodic Theorem for Markov chains [9], the Markov chain defined by P' has a unique stationary probability distribution if P' is aperiodic and irreducible; the former holds for the Markov chain induced by the Web graph. The latter holds iff G is strongly connected, which is generally *not* the case for the Web graph. In the context of computing PageRank, the standard way of ensuring this property is to add a new set of complete outgoing transitions, with small transition probabilities, to *all* nodes, creating a complete (and thus strongly connected) transition graph. In matrix notation, we construct the irreducible Markov matrix P'' as follows:

$$E = [1]_{n\times 1} \times \vec{v}^T$$
$$P'' = cP' + (1-c)E$$

In terms of the random walk, the effect of E is as follows. At each time step, with probability $(1-c)$, a surfer visiting any node will jump to a random Web page (rather than following an outlink). The destination of the random jump is chosen according to the probability distribution given in $\vec{v}$. Artificial jumps taken because of E are referred to as *teleportation*.

By redefining the vector $\vec{v}$ given in Equation 1 to be nonuniform, so that D and E add artificial transitions with nonuniform probabilities, the resultant PageRank vector can be biased to prefer certain kinds of pages. For this reason, we refer to $\vec{v}$ as the *personalization* vector.

For simplicity and consistency with prior work, the remainder of the discussion will be in terms of the transpose matrix, $A = (P'')^T$; i.e., the transition probability distribution for a surfer at node i is given by row i of P'', and column i of A.

Note that the edges artificially introduced by D and E never need to be explicitly materialized, so this construction has no impact on efficiency or the sparsity of the matrices used in the computations. In particular, the matrix-vector multiplication $\vec{y} = A\vec{x}$ can be implemented efficiently using Algorithm 1.

Assuming that the probability distribution over the surfer's location at time 0 is given by $\vec{x}^{(0)}$, the probability distribution for the surfer's location at time k is given by $\vec{x}^{(k)} = A^k\vec{x}^{(0)}$. The unique stationary distribution of the Markov chain is defined as $\lim_{k\to\infty} x^{(k)}$, which is equivalent to $\lim_{k\to\infty} A^k x^{(0)}$, and is independent of the initial distribution $\vec{x}^{(0)}$. This is simply the principal eigenvector of the matrix $A = (P'')^T$, which is exactly the PageRank vector we would like to compute.

The standard PageRank algorithm computes the principal eigenvector by starting with $\vec{x}^{(0)} = \vec{v}$ and computing successive iterates $\vec{x}^{(k)} = A\vec{x}^{(k-1)}$ until convergence. This is known as the Power Method, and is discussed in further detail in Section 3.

While many algorithms have been developed for fast eigenvector computations, many of them are unsuitable for this problem because of the size and sparsity of the Web matrix (see Section 7.1 for a discussion of this).

In this paper, we develop a fast eigensolver, based on the Power Method, that is specifically tailored to the PageRank problem and Web-scale matrices. This algorithm, called Quadratic Extrapolation, accelerates the convergence of the Power Method by periodically subtracting off estimates of the nonprincipal eigenvectors from the current iterate $\vec{x}^{(k)}$. In Quadratic Extrapolation, we take advantage of the fact that the first eigenvalue of a Markov matrix is known to be 1 to compute estimates of the nonprincipal eigenvectors using successive iterates of the Power Method. This allows seamless integration into the standard PageRank algorithm. Intuitively, one may think of Quadratic Extrapolation as using successive iterates generated by the Power Method to extrapolate the value of the principal eigenvector.

2. EXPERIMENTAL SETUP

In the following sections, we will be introducing a series of algorithms for computing PageRank, and discussing the rate of convergence achieved on realistic datasets. Our experimental setup was as follows. We used two datasets of different sizes for our experiments. The STANFORD.EDU link graph was generated from a crawl of the `stanford.edu` domain created in September 2002 by the Stanford WebBase project. This link graph contains roughly 280,000 nodes, with 3 million links, and requires 12MB of storage. We used STANFORD.EDU while developing the algorithms, to get a sense for their performance. For real-world, Web-scale performance measurements, we used the LARGEWEB link graph, generated from a large crawl of the Web that had been created by the Stanford WebBase project in January 2001 [13]. LARGEWEB contains roughly 80M nodes, with close to a billion links, and requires 3.6GB of storage. Both link graphs had dangling nodes removed as described in [18]. The graphs are stored using an adjacency list representation, with pages represented by 4-byte integer identifiers. On an AMD Athlon 1533MHz machine with a 6-way RAID-5 disk volume and 2GB of main memory, each application of Algorithm 1 on the 80M page LARGEWEB dataset takes roughly 10 minutes. Given that computing PageRank generally requires up to 100 applications of Algorithm 1, the need for fast methods is clear.

We measured the relative rates of convergence of the algorithms that follow using the L_1 norm of the residual vector; i.e.,

$$||Ax^{(k)} - x^{(k)}||_1$$

We describe why the L_1 residual is an appropriate measure in Section 6.

3. POWER METHOD

3.1 Formulation

One way to compute the stationary distribution of a Markov chain is by explicitly computing the distribution at successive time steps, using $\vec{x}^{(k)} = A\vec{x}^{(k-1)}$, until the distribution converges.

```
function x⃗^(n) = PowerMethod() {
x⃗^(0) = v⃗;
k = 1;
repeat
    x⃗^(k) = A x⃗^(k−1);
    δ = ||x^(k) − x^(k−1)||_1;
    k = k + 1;
until δ < ε;
}
```

Algorithm 2: Power Method

This leads us to Algorithm 2, the Power Method for computing the principal eigenvector of A. The Power Method is the oldest method for computing the principal eigenvector of a matrix, and is at the heart of both the motivation and implementation of the original PageRank algorithm (in conjunction with Algorithm 1).

The intuition behind the convergence of the power method is as follows. For simplicity, assume that the start vector $\vec{x}^{(0)}$ lies in the subspace spanned by the eigenvectors of A.[1] Then $\vec{x}^{(0)}$ can be written as a linear combination of the eigenvectors of A:

$$\vec{x}^{(0)} = \vec{u}_1 + \alpha_2\vec{u}_2 + \ldots + \alpha_m\vec{u}_m \tag{2}$$

Since we know that the first eigenvalue of a Markov matrix is $\lambda_1 = 1$,

$$\vec{x}^{(1)} = A\vec{x}^{(0)} = \vec{u}_1 + \alpha_2\lambda_2\vec{u}_2 + \ldots + \alpha_m\lambda_m\vec{u}_m \tag{3}$$

and

$$\vec{x}^{(n)} = A^n\vec{x}^{(0)} = \vec{u}_1 + \alpha_2\lambda_2^n\vec{u}_2 + \ldots + \alpha_m\lambda_m^n\vec{u}_m \tag{4}$$

Since $\lambda_n \leq \ldots \leq \lambda_2 < 1$, $A^{(n)}\vec{x}^{(0)}$ approaches $\vec{u}_1$ as n grows large. Therefore, the Power Method converges to the principal eigenvector of the Markov matrix A.

3.2 Operation Count

A single iteration of the Power Method consists of the single matrix-vector multiply $A\vec{x}^{(k)}$. Generally, this is an $O(n^2)$ operation. However, if the matrix-vector multiply is performed as in Algorithm 1, the matrix A is so sparse that the matrix-vector multiply is essentially $O(n)$. In particular, the average outdegree of pages on the Web has been found to be around 7 [16]. On our datasets, we observed an average of around 8 outlinks per page.

It should be noted that if λ_2 is close to 1, then the power method is slow to converge, because n must be large before λ_2^n is close to 0, and vice versa.

3.3 Results and Discussion

As we show in [12], the eigengap $1 - |\lambda_2|$ for the Web Markov matrix A is given exactly by the teleport probability $1 - c$. Thus, when the teleport probability is large, and the personalization vector $\vec{v}$ is uniform over all pages, the Power Method works reasonably well. However, for a large teleport probability (and with a uniform personalization vector $\vec{v}$), the effect of link spam is increased, and pages can achieve unfairly high rankings.[2] In the extreme case, for a teleport probability of $1 - c = 1$, the assignment of rank to pages becomes uniform. Chakrabarti et al. [5] suggest that c should be tuned based on the connectivity of topics on the Web. Such tuning has generally not been possible, as the convergence of PageRank slows down dramatically for small values of $1 - c$ (i.e., values of c close to 1).

[1] This assumption does not affect convergence guarantees.

[2] A high teleport probability means that every page is given a fixed "bonus" rank. Link spammers can make use of this bonus to generate local structures to inflate the importance of certain pages.

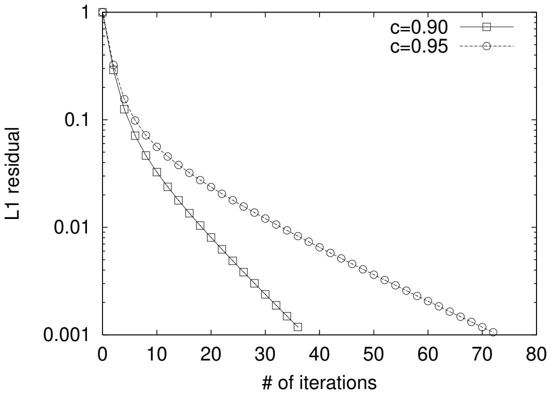

Figure 1: Comparison of convergence rate for the standard Power Method on the LargeWeb dataset for $c = 0.90$ and $c = 0.95$.

In Figure 1, we show the convergence on the LargeWeb dataset of the Power Method for $c \in \{0.90, 0.95\}$ using a uniform $\vec{v}$. Note that increasing c slows down convergence. Since each iteration of the Power Method takes 10 minutes, computing 100 iterations requires over 16 hours. As the full Web is estimated to contain over two billion static pages, using the Power Method on Web graphs close to the size of the Web would require several days of computation.

In the next sections, we describe how to remove the error components of $x^{(k)}$ along the direction of $\vec{u}_2$ and $\vec{u}_3$, thus increasing the effectiveness of Power Method iterations.

4. AITKEN EXTRAPOLATION

4.1 Formulation

We begin by introducing an algorithm which we shall call Aitken Extrapolation. We develop Aitken Extrapolation as follows. We assume that the iterate $\vec{x}^{(k-2)}$ can be expressed as a linear combination of the first two eigenvectors. This assumption allows us to solve for the principal eigenvector $\vec{u}_1$ in closed form using the successive iterates $\vec{x}^{(k-2)}, \ldots, \vec{x}^{(k)}$.

Of course, $\vec{x}^{(k-2)}$ can only be approximated as a linear combination of the first two eigenvectors, so the $\vec{u}_1$ that we compute is only an estimate of the true $\vec{u}_1$. However, it can be seen from section 3.1 that this approximation becomes increasingly accurate as k becomes larger.

We begin our formulation of Aitken Extrapolation by assuming that $\vec{x}^{(k-2)}$ can be expressed as a linear combination of the first two eigenvectors.

$$\vec{x}^{(k-2)} = \vec{u}_1 + \alpha_2\vec{u}_2 \tag{5}$$

Since the first eigenvalue λ_1 of a Markov matrix is 1, we can write the next two iterates as:

$$\vec{x}^{(k-1)} = A\vec{x}^{(k-2)} = \vec{u}_1 + \alpha_2\lambda_2\vec{u}_2 \tag{6}$$

$$\vec{x}^{(k)} = A\vec{x}^{(k-1)} = \vec{u}_1 + \alpha_2\lambda_2^2\vec{u}_2 \tag{7}$$

Now, let us define

$$g_i = (x_i^{(k-1)} - x_i^{(k-2)})^2 \quad (8)$$
$$h_i = x_i^{(k)} - 2x_i^{(k-1)} + x_i^{(k-2)} \quad (9)$$

where x_i represents the ith component of the vector $\vec{x}$. Doing simple algebra using equations 6 and 7 gives:

$$g_i = \alpha_2^2(\lambda_2 - 1)^2(u_2)_i^2 \quad (10)$$
$$h_i = \alpha_2(\lambda_2 - 1)^2(u_2)_i \quad (11)$$

Now, let us define f_i as the quotient g_i/h_i:

$$f_i = \frac{g_i}{h_i} = \frac{\alpha_2^2(\lambda_2 - 1)^2(u_2)_i^2}{\alpha_2(\lambda_2 - 1)^2(u_2)_i} \quad (12)$$
$$= \alpha_2(u_2)_i \quad (13)$$

Therefore,

$$\vec{f} = \alpha_2\vec{u}_2 \quad (14)$$

Hence, from equation 5, we have a closed-form solution for $\vec{u}_1$:

$$\vec{u}_1 = \vec{x}^{(k-2)} - \alpha_2\vec{u}_2 = \vec{x}^{(k-2)} - \vec{f} \quad (15)$$

However, since this solution is based on the assumption that $\vec{x}^{(k-2)}$ can be written as a linear combination of $\vec{u}_1$ and $\vec{u}_2$, equation 15 gives only an approximation to $\vec{u}_1$. Algorithm 3 and Algorithm 4 show how to use Aitken Extrapolation in conjunction with the Power Method to get consistently better estimates of $\vec{u}_1$.

Aitken Extrapolation is equivalent to applying the well-known Aitken Δ^2 method for accelerating linearly convergent sequences [1] to each component of the iterate $\vec{x}^{(k-2)}$. What is novel here is this derivation of Aitken acceleration, and the proof that Aitken acceleration computes the principal eigenvector of a Markov matrix in one step under the assumption that the power-iteration estimate $\vec{x}^{(k-2)}$ can be expressed as a linear combination of the first two eigenvectors.

As a sidenote, let us briefly develop a related method. Rather than using equation 8, let us define g_i alternatively as:

$$g_i = (x_i^{(k-1)} - x_i^{(k-2)})(x_i^{(k)} - x_i^{(k-1)}) = \alpha_2^2\lambda_2(\lambda_2 - 1)^2(u_2)_i^2$$

We define h as in equation 9, and f_i now becomes

$$f_i = \frac{g_i}{h_i} = \frac{\alpha_2^2\lambda_2(\lambda_2 - 1)^2(u_2)_i^2}{\alpha_2(\lambda_2 - 1)^2(u_2)_i} = \alpha_2\lambda_2(u_2)_i$$

By equation 6,

$$\vec{u}_1 = x^{(k-1)} - \alpha_2\lambda_2\vec{u}_2 = \vec{x}^{(k-1)} - \vec{f}$$

Again, this is an approximation to $\vec{u}_1$, since it's based on the assumption that $\vec{x}^{(k-2)}$ can be expressed as a linear combination of $\vec{u}_1$ and $\vec{u}_2$. What is interesting here is that this is equivalent to performing a second-order epsilon acceleration algorithm [22] on each component of the iterate $\vec{x}^{(k-2)}$. For this reason, we call this algorithm Epsilon Extrapolation.

4.2 Operation Count

In order for an extrapolation method such as Aitken Extrapolation or Epsilon Extrapolation to be useful, the overhead should be minimal. By overhead, we mean any costs in addition to the cost of applying Algorithm 1 to generate iterates. It is clear from inspection that the operation count of the loop in Algorithm 3 is $O(n)$, where n is the number of pages on the Web. The operation count of one extrapolation step is less than the operation count of a single iteration of the Power Method, and since Aitken Extrapolation may be applied only periodically, we say that Aitken Extrapolation has minimal overhead. In our implementation, the additional cost of each application of Aitken Extrapolation was negligible – about 1% of the cost of a single iteration of the Power Method (i.e., 1% of the cost of Algorithm 1).

```
function x* = Aitken(x^(k-2), x^(k-1), x^(k)) {
for i = 1 : n do
    g_i = (x_i^(k-1) - x_i^(k-2))^2 ;
    h_i = x_i^(k) - 2x_i^(k-1) + x_i^(k-2);
    x_i* = x_i^(k) - g_i/h_i;
end
}
```

Algorithm 3: Aitken Extrapolation

```
function x^(n) = AitkenPowerMethod() {
x^(0) = v;
k = 1;
repeat
    x^(k) = A x^(k-1);
    δ = ||x^(k) - x^(k-1)||_1;
    periodically, x^(k) = Aitken(x^(k-2), x^(k-1), x^(k));
    k = k + 1;
until δ < ε;
}
```

Algorithm 4: Power Method with Aitken Extrapolation

4.3 Experimental Results

In Figure 2, we show the convergence of the Power Method with Aitken Extrapolation applied once at the 10th iteration, compared to the convergence of the unaccelerated Power Method for the STANFORD.EDU dataset. The x-axis denotes the number of times a multiplication $A\vec{x}$ occurred; i.e., the number of times Algorithm 1 was needed. Note that there is a spike at the acceleration step, but speedup occurs nevertheless. This spike is caused by the poor approximation for u_2.

For $c = 0.99$, Aitken Extrapolation takes 38% less time to reach an iterate with a residual of 0.01. However, after this initial speedup, the convergence rate for Aitken slows down, so that to reach an iterate with a residual of 0.002, the time savings drops to 13%. For lower values of c, Aitken provided much less benefit. Since there is a spike in the residual graph, if Aitken Extrapolation is applied too often, the power iterations will not converge. In experiments, Epsilon Extrapolation performed similarly to Aitken Extrapolation.

4.4 Discussion

In this section, we presented Aitken Extrapolation, and a closely related method called Epsilon Extrapolation. Aitken Extrapolation is equivalent to applying the well-known Aitken Δ^2 method [1] to each component of the iterate $\vec{x}^{(k-2)}$, and Epsilon Extrapolation is equivalent to applying a second-order epsilon acceleration method to each component of the iterate $\vec{x}^{(k-2)}$ [22]. What is novel here is this derivation of these methods, and the proof that these methods compute the principal eigenvector of a Markov matrix in one step under the assumption that the power-iteration estimate $\vec{x}^{(k-2)}$ can be expressed as a linear combination of the first two eigenvec-

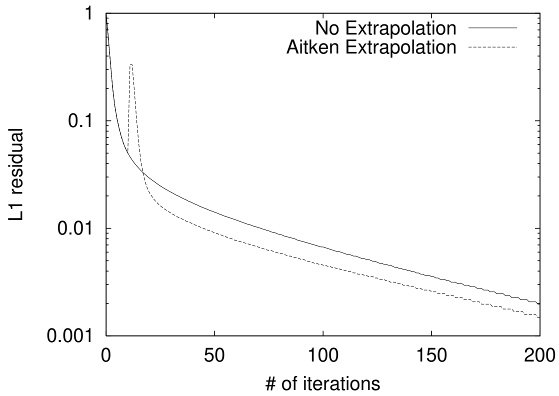

Figure 2: Comparison of convergence rate for unaccelerated Power Method and Aitken Extrapolation on the STANFORD.EDU **dataset, for $c = 0.99$. Extrapolation was applied at the 10th iteration.**

tors. Furthermore, these methods have not been used thus far to accelerate eigenvector computations.

These methods are very different from standard fast eigensolvers, which generally rely strongly on matrix factorizations or matrix inversions. Standard fast eigensolvers do not work well for the PageRank problem, since the web hyperlink matrix is so large and sparse. For problems where the matrix is small enough for an efficient inversion, standard eigensolvers such as inverse iteration are likely to be faster than these methods. The Aitken and Epsilon Extrapolation methods take advantage of the fact that the first eigenvalue of the Markov hyperlink matrix is 1 to find an approximation to the principal eigenvector.

In the next section, we present Quadratic Extrapolation, which assumes the iterate can be expressed as a linear combination of the first *three* eigenvectors, and solves for $\vec{u}_1$ in closed form under this assumption. As we shall soon discuss, the Quadratic Extrapolation step is simply a linear combination of successive iterates, and thus does not produce spikes in the residual.

5. QUADRATIC EXTRAPOLATION

5.1 Formulation

We develop the Quadratic Extrapolation algorithm as follows. We assume that the Markov matrix A has only 3 eigenvectors, and that the iterate $\vec{x}^{(k-3)}$ can be expressed as a linear combination of these 3 eigenvectors. These assumptions allow us to solve for the principal eigenvector $\vec{u}_1$ in closed form using the successive iterates $\vec{x}^{(k-3)}, \ldots, \vec{x}^{(k)}$.

Of course, A has more than 3 eigenvectors, and $\vec{x}^{(k-3)}$ can only be approximated as a linear combination of the first three eigenvectors. Therefore, the $\vec{u}_1$ that we compute in this algorithm is only an estimate for the true $\vec{u}_1$. We show empirically that this estimate is a better estimate to $\vec{u}_1$ than the iterate $\vec{x}^{(k-3)}$, and that our estimate becomes closer to the true value of $\vec{u}_1$ as k becomes larger. In Section 5.3 we show that by periodically applying Quadratic Extrapolation to the successive iterates computed in PageRank, for values of c close to 1, we can speed up the convergence of PageRank by a factor of over 3.

We begin our formulation of Quadratic Extrapolation by assuming that A has only three eigenvectors $\vec{u}_1, \ldots, \vec{u}_3$ and approximating $\vec{x}^{(k-3)}$ as a linear combination of these three eigenvectors.

$$\vec{x}^{(k-3)} = \vec{u}_1 + \alpha_2\vec{u}_2 + \alpha_3\vec{u}_3 \tag{16}$$

We then define the successive iterates

$$\vec{x}^{(k-2)} = A\vec{x}^{(k-3)} \tag{17}$$
$$\vec{x}^{(k-1)} = A\vec{x}^{(k-2)} \tag{18}$$
$$\vec{x}^{(k)} = A\vec{x}^{(k-1)} \tag{19}$$

Since we assume A has 3 eigenvectors, the characteristic polynomial $p_A(\lambda)$ is given by:

$$p_A(\lambda) = \gamma_0 + \gamma_1\lambda + \gamma_2\lambda^2 + \gamma_3\lambda^3 \tag{20}$$

A is a Markov matrix, so we know that the first eigenvalue $\lambda_1 = 1$. The eigenvalues of A are also the zeros of the characteristic polynomial $p_A(\lambda)$. Therefore,

$$p_A(1) = 0 \Rightarrow \gamma_0 + \gamma_1 + \gamma_2 + \gamma_3 = 0 \tag{21}$$

The Cayley-Hamilton Theorem states that any matrix A satisfies it's own characteristic polynomial $p_A(A) = 0$ [8]. Therefore, by the Cayley-Hamilton Theorem, for any vector z in $\mathbb{R}^n$,

$$p_A(A)z = 0 \Rightarrow [\gamma_0 I + \gamma_1 A + \gamma_2 A^2 + \gamma_3 A^3]z = 0 \tag{22}$$

Letting $z = \vec{x}^{(k-3)}$,

$$[\gamma_0 I + \gamma_1 A + \gamma_2 A^2 + \gamma_3 A^3]\vec{x}^{(k-3)} = 0 \tag{23}$$

From equations 17–19,

$$\gamma_0\vec{x}^{(k-3)} + \gamma_1\vec{x}^{(k-2)} + \gamma_2\vec{x}^{(k-1)} + \gamma_3\vec{x}^{(k)} = 0 \tag{24}$$

From equation 21,

$$\vec{x}^{(k-3)}(-\gamma_1 - \gamma_2 - \gamma_3) + \gamma_1\vec{x}^{(k-2)} + \gamma_2\vec{x}^{(k-1)} + \gamma_3\vec{x}^{(k)} = 0 \tag{25}$$

We may rewrite this as,

$$(\vec{x}^{(k-2)} - \vec{x}^{(k-3)})\gamma_1 + (\vec{x}^{(k-1)} - \vec{x}^{(k-3)})\gamma_2 + (\vec{x}^{(k)} - \vec{x}^{(k-3)})\gamma_3 = 0 \tag{26}$$

Let us make the following definitions:

$$\vec{y}^{(k-2)} = \vec{x}^{(k-2)} - \vec{x}^{(k-3)} \tag{27}$$
$$\vec{y}^{(k-1)} = \vec{x}^{(k-1)} - \vec{x}^{(k-3)} \tag{28}$$
$$\vec{y}^{(k)} = \vec{x}^{(k)} - \vec{x}^{(k-3)} \tag{29}$$

We can now write equation 26 in matrix notation:

$$\begin{pmatrix} \vec{y}^{(k-2)} & \vec{y}^{(k-1)} & \vec{y}^{(k)} \end{pmatrix} \vec{\gamma} = 0 \tag{30}$$

We now wish to solve for $\vec{\gamma}$. Since we're not interested in the trivial solution $\vec{\gamma} = 0$, we constrain the leading term of the characteristic polynomial γ_3:

$$\gamma_3 = 1 \tag{31}$$

We may do this because constraining a single coefficient of the polynomial does not affect the zeros.[3] Equation 30 is therefore written:

$$\begin{pmatrix} \vec{y}^{(k-2)} & \vec{y}^{(k-1)} \end{pmatrix} \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix} = -\vec{y}^{(k)} \tag{32}$$

[3] I.e., equation 31 fixes a scaling for $\vec{\gamma}$.

function $\vec{x}^* = QuadraticExtrapolation(\vec{x}^{(k-3)}, \ldots, \vec{x}^{(k)})$ {
for $j = k-2 : k$ **do**
  $\vec{y}^{(j)} = \vec{x}^{(j)} - \vec{x}^{(k-3)}$;
end
$Y = \begin{pmatrix} \vec{y}^{(k-2)} & \vec{y}^{(k-1)} \end{pmatrix}$;
$\gamma_3 = 1$;
$\begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix} = -Y^+\vec{y}^{(k)}$;
$\gamma_0 = -(\gamma_1 + \gamma_2 + \gamma_3)$;
$\beta_0 = \gamma_1 + \gamma_2 + \gamma_3$;
$\beta_1 = \gamma_2 + \gamma_3$;
$\beta_2 = \gamma_3$;
$\vec{x}^* = \beta_0\vec{x}^{(k-2)} + \beta_1\vec{x}^{(k-1)} + \beta_2\vec{x}^{(k)}$;
}

Algorithm 5: Quadratic Extrapolation

This is an overdetermined system, so we solve the corresponding least-squares problem.

$$\begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix} = Y^+\vec{y}^{(k)} \tag{33}$$

where Y^+ is the pseudoinverse of the matrix $Y = \begin{pmatrix} \vec{y}^{(k-2)} & \vec{y}^{(k-1)} \end{pmatrix}$. Now, equations 31, 33, and 21 completely determine the coefficients of the characteristic polynomial $p_A(\lambda)$ (equation 20).

We may now divide $p_A(\lambda)$ by $\lambda - 1$ to get the polynomial $q_A(\lambda)$, whose roots are λ_2 and λ_3, the second two eigenvalues of A.

$$q_A(\lambda) = \frac{\gamma_0 + \gamma_1\lambda + \gamma_2\lambda^2 + \gamma_3\lambda^3}{\lambda - 1} = \beta_0 + \beta_1\lambda + \beta_2\lambda^2 \tag{34}$$

Simple polynomial division gives the following values for β_0, β_1, and β_2:

$$\beta_0 = \gamma_1 + \gamma_2 + \gamma_3 \tag{35}$$
$$\beta_1 = \gamma_2 + \gamma_3 \tag{36}$$
$$\beta_2 = \gamma_3 \tag{37}$$

Again, by the Cayley-Hamilton Theorem, if z is any vector in $\mathbb{R}^n$,

$$q_A(A)z = \vec{u}_1 \tag{38}$$

where $\vec{u}_1$ is the eigenvector of A corresponding to eigenvalue 1 (the principal eigenvector). Letting $z = \vec{x}^{(k-2)}$,

$$\vec{u}_1 = q_A(A)\vec{x}^{(k-2)} = [\beta_0 I + \beta_1 A + \beta_2 A^2]\vec{x}^{(k-2)} \tag{39}$$

From equations 17–19, we get a closed form solution for $\vec{u}_1$:

$$\vec{u}_1 = \beta_0\vec{x}^{(k-2)} + \beta_1\vec{x}^{(k-1)} + \beta_2\vec{x}^{(k)} \tag{40}$$

However, since this solution is based on the assumption that A has only 3 eigenvectors, equation 40 gives only an approximation to $\vec{u}_1$.

Algorithms 5 and 6 show how to use Quadratic Extrapolation in conjunction with the Power Method to get consistently better estimates of $\vec{u}_1$.

5.2 Operation Count

The overhead in performing the extrapolation shown in Algorithm 5 comes primarily from the least-squares computation of γ_1 and γ_2:

$$\begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix} = -Y^+\vec{y}^{(k)}$$

It is clear that the other steps in this algorithm are either $O(1)$ or $O(n)$ operations.

Since Y is an $n \times 2$ matrix, we can do the least-squares solution cheaply in just 2 iterations of the Gram-Schmidt algorithm [21]. Therefore, γ_1 and γ_2 can be computed in $O(n)$ operations. While a presentation of Gram-Schmidt is outside of the scope of this paper, we show in Algorithm 7 how to apply Gram-Schmidt to solve for $[\gamma_1\gamma_2]^T$ in $O(n)$ operations. Since the extrapolation step is on the order of a single iteration of the Power Method, and since Quadratic Extrapolation is applied only periodically during the Power Method, we say that Quadratic Extrapolation has minimal overhead. In our experimental setup, the overhead of a single application of Quadratic Extrapolation is half the cost of a standard power iteration (i.e., half the cost of Algorithm 1). This number includes the cost of storing on disk the intermediate data required by Quadratic Extrapolation (such as the previous iterates), since they may not fit in main memory.

function $\vec{x}^{(n)} = QuadraticPowerMethod()$ {
$\vec{x}^{(0)} = \vec{v}$;
$k = 1$;
repeat
  $\vec{x}^{(k)} = A\vec{x}^{(k-1)}$;
  $\delta = ||x^{(k)} - x^{(k-1)}||_1$;
  periodically,
    $\vec{x}^{(k)} = QuadraticExtrapolation(\vec{x}^{(k-3)}, \ldots, \vec{x}^{(k)})$;
  $k = k + 1$;
until $\delta < \epsilon$;
}

Algorithm 6: Power Method with Quadratic Extrapolation

1. Compute the reduced QR factorization $Y = QR$ using 2 steps of Gram-Schmidt.
2. Compute the vector $-Q^T y^{(k)}$.
3. Solve the upper triangular system:
$R\begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix} = -Q^T y^{(k)}$
for $\begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix}$ using back substitution.

Algorithm 7: Using Gram-Schmidt to solve for γ_1 and γ_2.

5.3 Experimental Results

Of the algorithms we have discussed for accelerating the convergence of PageRank, Quadratic Extrapolation performs the best empirically. In particular, Quadratic Extrapolation considerably improves convergence relative to the Power Method when the damping factor c is close to 1. We measured the performance of Quadratic Extrapolation under various scenarios on the LARGEWEB dataset. Figure 3 shows the rates of convergence when $c = 0.90$; after factoring in overhead, Quadratic Extrapolation reduces the time needed to reach a residual of 0.001 by 23%.[4] Figure 4 shows the rates of convergence when $c = 0.95$; in this case, Quadratic Ex-

[4]The time savings we give factor in the overhead of applying extrapolation, and represent "wall-clock" time savings.

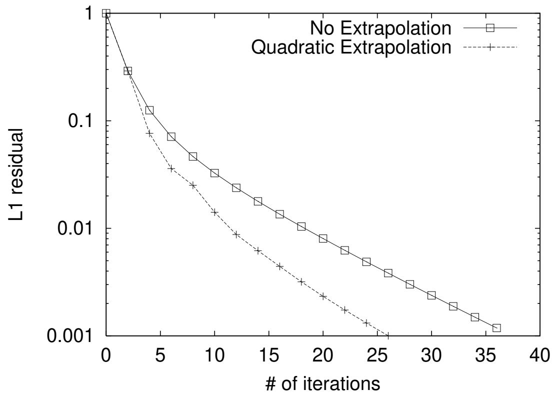

Figure 3: Comparison of convergence rates for Power Method and Quadratic Extrapolation on LARGEWEB for $c = 0.90$. Quadratic Extrapolation was applied the first 5 times that three successive power iterates were available.

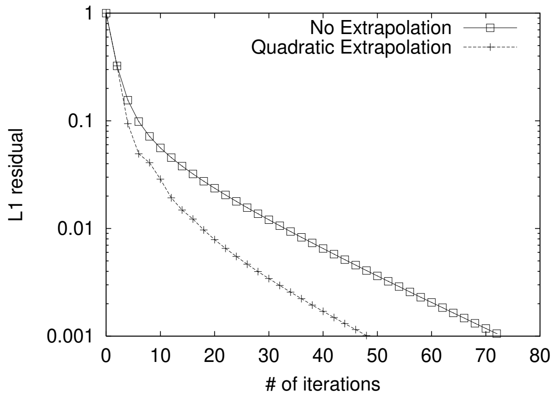

Figure 4: Comparison of convergence rates for Power Method and Quadratic Extrapolation on LARGEWEB for $c = 0.95$. Quadratic Extrapolation was applied 5 times.

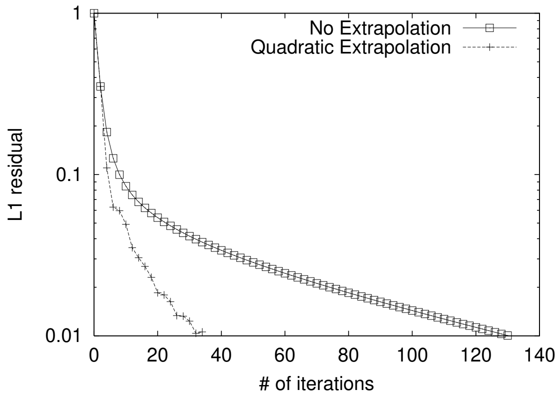

Figure 5: Comparison of convergence rates for Power Method and Quadratic Extrapolation on LARGEWEB when $c = 0.99$. Quadratic Extrapolation was applied all 11 times possible.

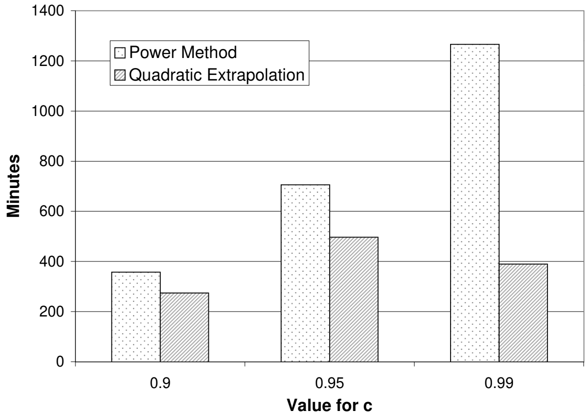

Figure 6: Comparison of wallclock times taken by Power Method and Quadratic Extrapolation on LARGEWEB for $c = \{0.90, 0.95, 0.99\}$. For $c = \{0.90, 0.95\}$, the residual tolerance ϵ was set to 0.001, and for $c = 0.99$, it was set to 0.01.

trapolation speeds up convergence more significantly, saving 31% in the time needed to reach a residual of 0.001. Finally, in the case where $c = 0.99$, the speedup is more dramatic. Figure 5 shows the rates of convergence of the Power Method and Quadratic Extrapolation for $c = 0.99$. Because the Power Method is so slow to converge in this case, we plot the curves until a residual of 0.01 is reached. The use of extrapolation saves 69% in time needed to reach a residual of 0.01; i.e., the unaccelerated Power Method took over 3 times as long as the Quadratic Extrapolation method to reach the desired residual. The wallclock times for each of these scenarios are summarized in Figure 6.

Figure 7 shows the convergence for the Power Method, Aitken Extrapolation, and Quadratic Extrapolation on the STANFORD.EDU dataset; each method was carried out to 200 iterations. To reach a residual of 0.01, Quadratic Extrapolation saved 59% in time over the Power Method, as opposed to a 38% savings for Aitken Extrapolation.

An important observation about Quadratic Extrapolation is that it does not necessarily need to be applied too often to achieve maximum benefit. By contracting the error in the current iterate along the direction of the second and third eigenvectors, Quadratic Extrapolation actually enhances the convergence of future applications of the standard Power Method. The Power Method, as discussed previously, is very effective in annihilating error components of the iterate in directions along eigenvectors with small eigenvalues. By subtracting off approximations to the second and third eigenvectors, Quadratic Extrapolation leaves error components primarily along the smaller eigenvectors, which the Power Method is better equipped to eliminate.

For instance, in Figure 8, we plot the convergence when Quadratic Extrapolation is applied 5 times compared with when it is applied as often as possible (in this case, 14 times), to achieve a residual of 0.001. Note that the additional applications of Quadratic Extrapolation do not lead to much further improvement. In fact, once we factor in the 0.5 iteration-cost of each application of Quadratic Extrapolation, the case where it was applied 5 times ends up being faster.

5.4 Discussion

Like Aitken and Epsilon Extrapolation, Quadratic Extrapolation makes the assumption that an iterate can be expressed as a linear combination of a subset of the eigenvectors of A in order to find an approximation to the principal eigenvector of A. In Aitken and Epsilon Extrapolation, we assume that $\vec{x}^{(k-2)}$ can be written as a

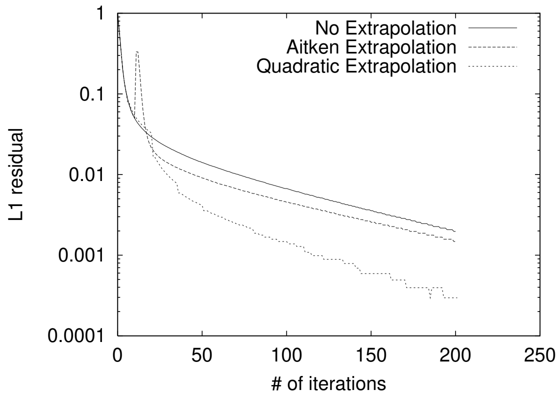

Figure 7: Comparison of convergence rates for Power Method, Aitken Extrapolation, and Quadratic Extrapolation on the STANFORD.EDU dataset for $c = 0.99$. Aitken Extrapolation was applied at the 10th iteration, Quadratic Extrapolation was applied every 15th iteration. Quadratic Extrapolation performs the best by a considerable degree. Aitken suffers from a large spike in the residual when first applied.

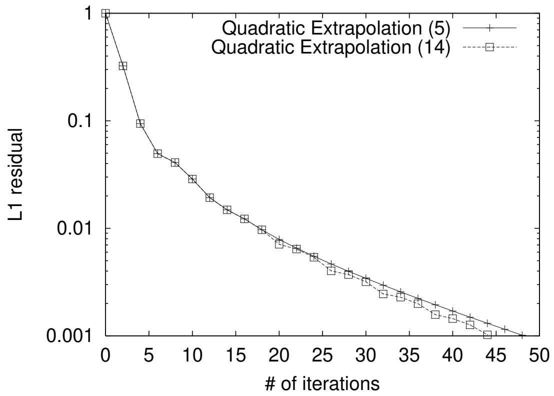

Figure 8: Comparison of convergence rates for Quadratic Extrapolation on LARGEWEB for $c = 0.95$, under two scenarios: Extrapolation was applied the first 5 possible times in one case, and all 14 possible times in the other. Applying it only 5 times achieves nearly the same benefit in this case.

linear combination of the first two eigenvectors, and in Quadratic Extrapolation, we assume that $\vec{x}^{(k-3)}$ can be written as a linear combination of the first three eigenvectors. Since the assumption made in Quadratic Extrapolation is closer to reality, the resulting approximations are closer to the true value of the principal eigenvector of A.

While Aitken and Epsilon Extrapolation are logical extensions of existing acceleration algorithms, Quadratic Extrapolation is completely novel. Furthermore, all of these algorithms are general purpose. That is, they can be used to compute the principal eigenvector of any large, sparse Markov matrix, not just the web graph. They should be useful in any situation where the size and sparsity of the matrix is such that a QR factorization is prohibitively expensive.

One thing that is interesting to note is that since acceleration may be applied periodically during any iterative process that generates iterates $\vec{x}^{(k)}$ that converge to the principal eigenvector $\vec{u}_1$, it is straightforward to use Quadratic Extrapolation in conjunction with other methods for accelerating PageRank, such as Gauss-Seidel [8, 2].

6. MEASURES OF CONVERGENCE

In this section, we present empirical results demonstrating the suitability of the L_1 residual, even in the context of measuring convergence of *induced document rankings*. In measuring the convergence of the PageRank vector, prior work has usually relied on $\delta_k = ||Ax^{(k)} - x^{(k)}||_p$, the L_p norm of the residual vector, for $p = 1$ or $p = 2$, as an indicator of convergence. Given the intended application, we might expect that a better measure of convergence is the distance, using an appropriate measure of distance, between the rank orders for query results induced by $Ax^{(k)}$ and $x^{(k)}$. We use two measures of distance for rank orders, both based on the the Kendall's-τ rank correlation measure: the KDist measure, defined below, and the K_{min} measure, introduced by Fagin et al. in [7]. To see if the residual is a "good" measure of convergence, we compared it to the KDist and K_{min} of rankings generated by $Ax^{(k)}$ and $x^{(k)}$.

We show empirically that in the case of PageRank computations, the L_1 residual δ_k is closely correlated with the KDist and K_{min} distances between query results generated using the values in $Ax^{(k)}$ and $x^{(k)}$.

We define the distance measure, KDist as follows. Consider two partially ordered lists of URLs, τ_1 and τ_2, each of length k. Let U be the union of the URLs in τ_1 and τ_2. If ρ_1 is $U - \tau_1$, then let τ_1' be the extension of τ_1, where τ_1' contains ρ_1 appearing after all the URLs in τ_1.[5] We extend τ_2 analogously to yield τ_2'. KDist is then defined as:

$$\mathrm{KDist}(\tau_1, \tau_2) = \frac{|\{(u,v) : \tau_1', \tau_2' \text{ disagree on order of } (u,v), u \neq v\}|}{(|U|)(|U|-1)} \quad (41)$$

In other words, $\mathrm{KDist}(\tau_1, \tau_2)$ is the probability that τ_1' and τ_2' disagree[6] on the relative ordering of a randomly selected pair of distinct nodes $(u, v) \in U \times U$.

To measure the convergence of PageRank iterations in terms of induced rank orders, we measured the KDist distance between the induced rankings for the top 100 results, averaged across 27 test queries, using successive power iterates for the LARGEWEB dataset, with the damping factor c set to 0.9.[7] The average residuals using the KDist, K_{min}, and L_1 measures are plotted in Figure 9.[8] Surprisingly, the L_1 residual is almost perfectly correlated with KDist, and is closely correlated with K_{min}.[9] A rigorous explanation for the close match between the L_1 residual and the Kendall's τ based residuals is an interesting avenue of future investigation.

7. RELATED WORK

7.1 Fast Eigenvector Computation

The field of numerical linear algebra is a mature field, and many algorithms have been developed for fast eigenvector computations. However, many of these algorithms are unsuitable for this problem, because they require matrix inversions or matrix decompositions

[5]The URLs in ρ are placed with the *same* ordinal rank at the end of τ.

[6]A pair ordered in one list and tied in the other is considered a disagreement.

[7]Computing Kendall's τ over the complete ordering of all of LARGEWEB is expensive; instead we opt to compute KDist and K_{min} over query results.

[8]The L_1 residual δ_k is normalized so that δ_0 is 1.

[9]We emphasize that we have shown close agreement between L_1 and KDist for measuring residuals, not for distances between arbitrary vectors.

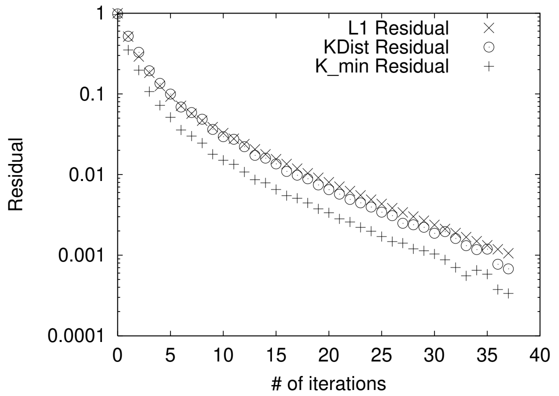

Figure 9: Comparison of the L_1 residual vs. KDist and K_{min} for PageRank iterates. Note that the two curves nearly perfectly overlap, suggesting that in the case of PageRank, the easily calculated L_1 residual is a good measure for the convergence of query-result rankings.

that are prohibitively expensive (both in terms of size and space) for a matrix of the size and sparsity of the Web-link matrix. For example, *inverse iteration* will find the principal eigenvector of A in one iteration, since we know the first eigenvalue. However, inverse iteration requires the inversion of A, which is an $O(n^3)$ operation. The *QR Algorithm with shifts* is also a standard fast method for solving nonsymmetric eigenvalue problems. However, the QR Algorithm requires a QR factorization of A at each iteration, which is also an $O(n^3)$ operation. The *Arnoldi* algorithm is also often used for nonsymmetric eigenvalue problems. However, the strength of Arnoldi is that it quickly computes estimates to the first few eigenvalues. Once it has a good estimate of the eigenvalues, it uses inverse iteration to find the corresponding eigenvectors. In the PageRank problem, we know that the first eigenvalue of A is 1, since A is a Markov matrix, so we don't need Arnoldi to give us an estimate of λ_1. For a comprehensive review of these methods, see [8].

However, there is a class of methods from numerical linear algebra that are useful for this problem. We may rewrite the eigenproblem $A\vec{x} = \vec{x}$ as the linear system of equations: $(I - A)\vec{x} = 0$, and use the classical iterative methods for linear systems: Jacobi, Gauss-Seidel, and Successive Overrelaxation (SOR). For the matrix A in the PageRank problem, the Jacobi method is equivalent to the Power method, but Gauss-Seidel is guaranteed to be faster. This has been shown empirically for the PageRank problem [2]. Note, however, that to use Gauss-Seidel, we would have to sort the adjacency-list representation of the Web graph, so that back-links for pages, rather than forward-links, are stored consecutively. The myriad of multigrid methods are also applicable to this problem. For a review of multigrid methods, see [17].

7.2 PageRank

Seminal algorithms for graph analysis for Web-search were introduced by Page et al. [18] (PageRank) and Kleinberg [15] (HITS). Much additional work has been done on improving these algorithms and extending them to new search and text mining tasks [4, 6, 19, 3, 20, 11]. More applicable to our work are several papers which discuss the computation of PageRank itself [10, 2, 14]. Haveliwala [10] explores memory-efficient computation, and suggests using induced orderings, rather than residuals, to measure convergence. Arasu et al. [2] uses the Gauss-Seidel method to speed up convergence, and looks at possible speed-ups by exploiting structural properties of the Web graph. Jeh and Widom [14] explore the use of dynamic programming to compute a large number of personalized PageRank vectors simultaneously. Our work is the first to exploit extrapolation techniques specifically designed to speed up the convergence of PageRank, with very little overhead.

8. CONCLUSION

Web search has become an integral part of modern information access, posing many interesting challenges in developing effective and efficient strategies for ranking search results. One of the most well-known Web-specific ranking algorithms is PageRank – a technique for computing the authoritativeness of pages using the hyperlink graph of the Web. Although PageRank is largely an offline computation, performed while preprocessing and indexing a Web crawl before any queries have been issued, it has become increasingly desirable to speed up this computation. Rapidly growing crawl repositories, increasing crawl frequencies, and the desire to generate multiple topic-based PageRank vectors for each crawl are all motivating factors for our work in speeding up PageRank computation.

Quadratic Extrapolation is an implementationally simple technique that requires little additional infrastructure to integrate into the standard Power Method. No sorting or modifications of the massive Web graph are required. Additionally, the extrapolation step need only be applied periodically to enhance the convergence of PageRank. In particular, Quadratic Extrapolation works by eliminating the bottleneck for the Power Method, namely the second and third eigenvector components in the current iterate, thus boosting the effectiveness of the simple Power Method itself.

9. ACKNOWLEDGEMENTS

We would like to acknowledge Ronald Fagin for useful discussions regarding the K_{min} measure.

This paper is based on work supported in part by the National Science Foundation under Grant No. IIS-0085896 and Grant No. CCR-9971010, and in part by the Research Collaboration between NTT Communication Science Laboratories, Nippon Telegraph and Telephone Corporation and CSLI, Stanford University (research project on Concept Bases for Lexical Acquisition and Intelligently Reasoning with Meaning).

10. REFERENCES

[1] A. Aitken. On Bernoulli's numerical solution of algebraic equations. *Proc. Roy. Soc. Edinburgh*, 46:289–305, 1926.

[2] A. Arasu, J. Novak, A. Tomkins, and J. Tomlin. PageRank computation and the structure of the web: Experiments and algorithms. In *Proceedings of the Eleventh International World Wide Web Conference, Poster Track*, 2002.

[3] K. Bharat and M. R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of the ACM-SIGIR*, 1998.

[4] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource compilation by analyzing hyperlink structure and associated text. In *Proceedings of the Seventh International World Wide Web Conference*, 1998.

[5] S. Chakrabarti, M. M. Joshi, K. Punera, and D. M. Pennock. The structure of broad topics on the web. In *Proceedings of the Eleventh International World Wide Web Conference*, 2002.

[6] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific web resource

discovery. In *Proceedings of the Eighth International World Wide Web Conference*, 1999.

[7] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[8] G. H. Golub and C. F. V. Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1996.

[9] G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, 1989.

[10] T. H. Haveliwala. Efficient computation of PageRank. *Stanford University Technical Report*, 1999.

[11] T. H. Haveliwala. Topic-sensitive PageRank. In *Proceedings of the Eleventh International World Wide Web Conference*, 2002.

[12] T. H. Haveliwala and S. D. Kamvar. The second eigenvalue of the Google matrix. *Stanford University Technical Report*, 2003.

[13] J. Hirai, S. Raghavan, H. Garcia-Molina, and A. Paepcke. WebBase: A repository of web pages. In *Proceedings of the Ninth International World Wide Web Conference*, 2000.

[14] G. Jeh and J. Widom. Scaling personalized web search. In *Proceedings of the Twelfth International World Wide Web Conference*, 2003.

[15] J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[16] J. Kleinberg, S. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The web as a graph: Measurements, models, and methods. In *Proceedings of the International Conference on Combinatorics and Computing*, 1999.

[17] U. Krieger. Numerical solution of large finite markov chains by algebraic multigrid techniques. In *Proceedings of the 2nd International Workshop on the Numerical Solution of Markov Chains*, 1995.

[18] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. *Stanford Digital Libraries Working Paper*, 1998.

[19] D. Rafiei and A. O. Mendelzon. What is this page known for? Computing web page reputations. In *Proceedings of the Ninth International World Wide Web Conference*, 2000.

[20] M. Richardson and P. Domingos. The intelligent surfer: Probabilistic combination of link and content information in pagerank. In *Advances in Neural Information Processing Systems*, volume 14. MIT Press, Cambridge, MA, 2002.

[21] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM Press, Philadelphia, 1997.

[22] P. Wynn. On the convergence and stability of the epsilon algorithm. *SIAM Journal of Numerical Analysis*, 33:91–122, 1966.