Efficient and Robust Fully Distributed Power Method with an Application to Link Analysis

Geoffrey Canright Kenth Engø-Monsen Márk Jelasity

Technical Report UBLCS-2005-17

September 2005

Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7
40127 Bologna (Italy)

The University of Bologna Department of Computer Science Research Technical Reports are available in PDF and gzipped PostScript formats via anonymous FTP from the area `ftp.cs.unibo.it:/pub/TR/UBLCS` or via WWW at URL `http://www.cs.unibo.it/`. Plain-text abstracts organized by year are available in the directory `ABSTRACTS`.

Recent Titles from the UBLCS Technical Report Series

2004-17 *Supporting e-Commerce Systems Formalization with Choreography Languages*, Bravetti, M., Guidi, C., Lucchi, R., Zavattaro, G., November 2004.

2004-18 *Decentralized Ranking in Large-Scale Overlay Networks*, Montresor, A., Jelasity, M., Babaoglu, O., December 2004.

2004-19 *Advanced Collective Communication in WDM Optical Rings*, Margara, L., Simon, J., Vassura, V., December 2004.

2005-1 *ARTIS: Design and Implementation of an Adaptive Middleware for Parallel and Distributed Simulation (Ph.D. Thesis)*, D'Angelo, G., March 2005.

2005-2 *Analysis and Prototype of a Metamodeling Environment for Engineering Grid Services (Ph.D. Thesis)*, Moretti, R., March 2005.

2005-3 *On some combinatorial optimization problems arising from computer networks (Ph.D. Thesis)*, Vassura, M., March 2005.

2005-4 *Experiences with Synthetic Network Emulation for Complex IP based Networks (Ph.D. Thesis)*, Cacciaguerra, S., March 2005.

2005-5 *Interactivity Maintenance for Event Synchronization in Massive Multiplayer Online Games (Ph.D. Thesis)*, Ferretti, S., March 2005.

2005-6 *Reasoning with preferences over temporal, uncertain, and conditional statements (Ph.D. Thesis)*, Venable, K. B., March 2005.

2005-7 *Whole Platform (Ph.D. Thesis)*, Solmi, R., March 2005.

2005-8 *Loss Functions and Structured Domains for Support Vector Machines (Ph.D. Thesis)*, Portera, F., March 2005.

2005-9 *A Reasoning Infrastructure to Support Cooperation of Intelligent Agents on the Semantic Grid*, Dragoni, N., Gaspari, M., Guidi, D., April 2005.

2005-10 *Fault Tolerant Knowledge Level Communication in Open Asynchronous Multi-Agent Systems*, Dragoni, N., Gaspari, M., April 2005.

2005-11 *The AEDSS Application Ontology: Enhanced Automatic Assessment of EDSS in Multiple Sclerosis*, Gaspari, M., Saletti, N., Scandellari, C., Stecchi, S., April 2005.

2005-12 *How to cheat BitTorrent and why nobody does*, Hales, D., Patarin, S., May 2005.

2005-13 *Choose Your Tribe! - Evolution at the Next Level in a Peer-to-Peer network*, Hales, D., May 2005.

2005-14 *Knowledge-Based Jobs and the Boundaries of Firms: Agent-based simulation of Firms Learning and Workforce Skill Set Dynamics*, Mollona, E., Hales, D., June 2005.

2005-15 *Tag-Based Cooperation in Peer-to-Peer Networks with Newscast*, Marcozzi, A., Hales, D., Jesi, G., Arteconi, S., Babaoglu, O., June 2005.

2005-16 *Atomic Commit and Negotiation in Service Oriented Computing*, Bocchi, L., Ciancarini, P., Lucchi, R., June 2005.

Efficient and Robust Fully Distributed Power Method with an Application to Link Analysis[1]

Geoffrey Canright[2] **Kenth Engø-Monsen**[2] **Márk Jelasity**[3]

Technical Report UBLCS-2005-17

September 2005

Abstract

Methods for link analysis are a key component in search engines for hyperlinked document networks. Documents are assigned an importance score based on the graph structure of the hyperlinks among the documents. At the heart of link analysis protocols we find the problem of calculating the principal eigenvector of a suitable matrix that is defined based on the hyperlink graph. In this paper we introduce a fully distributed method, inspired by the power method, for the calculation of the principal eigenvector of generic matrices, focusing on link analysis as an application. Theoretical results are given that support the correctness of the approach, and experimental validation is presented based on subsets of the WWW. Unlike other proposals, our protocol matches the sequential power method in speed and accuracy for generic matrices even in extremely hostile failure scenarios. This allows for better scalability, fault tolerance and load balancing. Most importantly, it represents an important step towards a flexible, cheap and fully peer-to-peer search method in networks of hyperlinked documents.

1. Authors are listed in alphabetical order. This work was partially supported by the Future and Emerging Technologies unit of the European Commission through Project BISON (IST-2001-38923) and DELIS (IST-2002-001907).
2. Telenor R&D, N-1331 Fornebu, Norway
3. University of Bologna, Mura Anteo Zamboni 7, I-40126 Bologna, Italy, and MTA RGAI, SZTE, Szeged, Hungary

1 Introduction

Distributed, mostly hierarchical, schemes for searching hyperlinked document networks were proposed very early, see for example [14]. A few years later Brin and Page argued [4] that scalability issues do not represent a problem and indexing is best done in a centralized manner. The raise of Google proved this thesis right spectacularly. Recently, with the emergence of peer-to-peer networks and the unforseen growth rate of the information available on the Internet, the scalability problem is becoming relevant again.

In the heart of most indexing and ranking schemes including the PageRank method of Google lies the calculation of the principal eigenvector of a suitably defined matrix. In the past years a number of protocols have been proposed that in some form or another, rely on the distributed calculation of the principal eigenvector of fully distributed matrices [7, 12, 13]. In these approaches, the matrix is stored in (possibly weighted) links among nodes, and the vector elements are stored at the nodes, one element per node. Matrix-vector multiplication, the kernel operation of the power method, is implemented by updating the weights of the nodes by propagating the weights along the links. Multiplying the propagated weight by the link weight implements the matrix-vector multiplication.

We adopt the same distributed framework. Our contribution is as follows:

- We relax synchronization in a novel way which makes it possible to achieve comparable performance to that of the sequential power iteration, both in terms of speed and precision.
- Our method is able to keep the norm of the vector we operate on constant thereby making it possible to run the method indefinitely and on any (potentially non weight conserving) matrices.
- Our method can normalize the average of the vector elements to a pre-specified value, which allows us to implement the random surfer operator [4] on arbitrary (non weight conserving) matrices.
- Finally, we introduce a buffering technique that enables the method to deal with extreme failure scenarios with minimal performance cost and with no loss in precision.

The outline of the paper is as follows. In Section 2 we describe a set of methods for link analysis. Most importantly, all of these methods are presented as eigenvector calculation problems over a suitably defined matrix. These examples define the application area on which we will evaluate our fully distributed power iteration method. In Section 3 we present a generic method for calculating the dominant eigenvector of matrices that are stored as weighted graphs (e.g. hyperlinked documents, or other overlay networks). Apart from the generic discussion, we focus specifically on the instances of the framework for link analysis. In Section 4 experimental results are discussed, again, in the link analysis context, over subsets of the WWW hyperlink graph. Section 5 concludes the paper.

2 Problem set

In this section, we will define four distinct problems that require the calculation of the principal eigenvector of some matrix in a distributed fashion. These problems will serve as application examples of our distributed power method. Two of these problems are in fact link analysis, while the other two may be viewed as 'warm-up' problems for full link analysis—although one of them (eigenvector centrality) is of significant interest in its own right.

We define our problems in a logical order, formed by combinations and permutations of two basic design choices: **N**ormalized or **U**n-normalized matrices, and **U**ndirected or **D**irected links. We abbreviate the four choices as **NU** (normalized undirected), **UU**, **ND**, and **UD**. Here, by normalized, we mean that the matrix chosen is column normalized (for right multiplication), so that the weight sent out upon multiplication by the matrix is equal to the weight found at the nodes. That is, when the matrix is normalized, the operation of matrix multiplication is weight conserving.

The distinction between directed and undirected links is clear. We note that, as noted above, the undirected cases do not apply directly to (hyper)link analysis; but the UU case is eigenvector centrality, as defined in social science [3], and is of considerable interest.

Now we discuss each of these four problems in more detail.

2.1 Normalized undirected graphs (NU)

This case is analytically soluble. That is, the normalized matrix for an undirected graph is a symmetric stochastic matrix [9], for which the dominant eigenvalue is one. The ith component of the dominant eigenvector is then proportional to the degree k_i of node i (with the constant of proportionality determined by normalization).

One can map this problem to the problem of independent random walks on the undirected graph. The dominant eigenvector, suitably (L1) normalized, then gives, in its ith component, the asymptotic probability of node i being visited by a walk.

Now we address the calculation of this same eigenvector using the power method, visiting some of the problems a distributed implementation will face. We address several elements which are needed for the power method: (i) start vector; (ii) propagation of weights; (iii) vector normalization; (iv) convergence.

2.1.1 Start vector

Calculation of the dominant eigenvector has the nice property that convergence is essentially independent of the start vector: the only requirement is that it is not orthogonal to the dominant eigenvector. This has practically zero probability, especially considering the limited numeric precision available. Therefore, an arbitrary start vector will do, and no global coordination is needed. By the same token, *synchronization* of the start is not crucial, as long as all nodes eventually do start (without excessive delay).

Also, we note that, for the link analysis case, we are most interested in maintaining a running update of the dominant eigenvector. For such applications, starting is a rare event.

2.1.2 Propagation of weights

The start vector places a "weight" at each node. Multiplication by the matrix for the graph then amounts to propagation of these weights over the links; and since the matrix is normalized, the propagation is weight conserving: each nodes divides up its weight among its neighbors (according to possible weightings on the links), and sends the result to them.

The main issue here to deal with in a distributed implementation is the synchronization (or the lack of it) of cycles, that is, propagation has to be done in a way that can be interpreted as consecutive matrix multiplication at the global level.

2.1.3 Vector normalization

Normalization of the weight vector is not strictly necessary for carrying out this (NU) calculation. Ignoring lost messages and other forms of noise, the rate of growth of the length of the weight vector converges to the dominant eigenvalue, which, for the normalized case, is one.

There are nevertheless at least two reasons to wish to be able to normalize the vector in this case. One is that just mentioned: in the case of lost messages, the weight vector will steadily (albeit presumably slowly) shrink in length. Hence one will wish to rescale the length of this vector from time to time.

Another reason to normalize the vector is to give each node a common (global!) scale for interpreting its own weight. This consideration is rather academic for this case, since every node can know its answer without even doing the calculation. However, the problem of lost messages remains, and suggests that normalization of the weight vector is desirable for this case. Normalization of the vector is a global operation. Hence this task represents a challenge to our distributed calculation.

2.1.4 Convergence

Since we are interested in *maintaining* the eigenvalue in the face of potentially changing link-graphs, convergence is a non-issue. Nevertheless, all nodes can decide locally whether convergence has been achieved (i.e., whether there is a significant change in the local value from one update to the next).

2.2 Un-normalized undirected graphs (UU)

Here the appropriate matrix is the "adjacency matrix" A. Typically, this matrix is defined so that (for undirected graphs) $A_{ij} = 1$ if i and j are linked, and zero otherwise. We see that this gives a symmetric, non-negative real matrix that has a positive dominant eigenvector (as in the previous case). Generalizations are also possible, whereby weights (other than 1) may be placed on the links; these do not change the problem significantly.

When the matrix is not normalized, the dominant eigenvalue λ is greater than or equal to one. This is the principal new feature for this problem. It affects one of the four components discussed in the previous section.

2.2.1 Vector normalization

When the calculation has converged, the weight w_i at node i will grow according to $w_i \to \lambda w_i$ after each propagation step. This presents two problems relative to the NU case because it is not practical for the weights to grow without bound.

Because of this first problem, a UU calculation needs to regularly normalize the weight vector—a global operation. We will present a way to do this in a later section.

2.3 Normalized directed graphs (ND)

For the hyperlinked case, links are directed. Hence we generalize the definition of the adjacency matrix A as follows: $A_{ij} = 1$ if $i \longrightarrow j$, and zero otherwise. Here $i \longrightarrow j$ means that i points to j. The resulting adjacency matrix is asymmetric. Normalization of an asymmetric A is as described previously (e.g., for right multiplication, column sums must be one, except when a column is all zero).

Directed graphs introduce a new complication to the use of the power method. That is, in general, directed graphs are not *strongly connected*. In a strongly connected directed graph, there is a directed path from i to j for *any* pair of nodes (i, j). More typically, a directed graph is composed of multiple *strongly connected components* (SCCs). Assuming that the graph is at least not disconnected, then these SCCs are connected by links in such a way that movement with arrows only goes one way between SCCs (e.g., SCC $\longrightarrow$ SCC'). Hence there will be one or more SCCs that are *sinks*—they have inlinks, but no outlinks.

The presence of sink SCCs creates a problem for any calculation of importance from a dominant eigenvector. For the normalized (weight conserving) case, sink SCCs will take all of the weight in the converged (dominant) eigenvector [10]. Hence one needs some sort of "sink remedy" in order to get useful results. The PageRank algorithm [10] uses the normalized adjacency matrix, supplemented with a "random surfer" operator, which sends weight from any node to any other with a uniform probability. The random surfer operator R thus renders the entire graph strongly connected.

A very attractive feature of the random surfer operator is that it sends the *same* weight to every node. Hence there is no need to actually implement the all-to-all links, either in a centralized or in a distributed calculation. In either case, the net effect of R is to add a constant weight (adjusted by a parameter ϵ) to each node at each propagation step. In other words, R times any vector gives a vector which is uniform, and whose value may be known if both ϵ and the L1 norm of the vector are known [10].

Hence, for a suitable (uniform) form for R, the second term in the propagation equation

$$\mathbf{w}^{m+1} = (1-\epsilon)A\mathbf{w}^m + \epsilon R\mathbf{w}^m \tag{1}$$

may be known in advance, as long as the L1 norm of $\mathbf{w}$ is known.

We then assume that we can control the L1 norm of $\mathbf{w}$ using regular normalization. Then we can implement the PageRank R operator—a global operator—using purely local operations (addition of $\epsilon(R\mathbf{w})_i$ at each node i, with the quantity in parentheses known). Specifically, a uniform R that satisfies weight conservation with the above equation will have $R_{ij} = 1/N$, where N is the number of nodes in the graph. This gives

$$(\epsilon R\mathbf{w})_i = (\epsilon/N)\sum_i w_i \ . \tag{2}$$

As we will show below, the average weight, $\sum_i w_i/N$ can be calculated by the set of nodes in the same process that gives normalization. In short, using the normalization protocol, we can implement a sink remedy in a distributed fashion, as we shall describe.

2.4 Un-normalized directed graphs (UD)

This case combines the problems of the UU case with those of the ND case. That is: because the adjacency matrix is not normalized, besides, there can be sink nodes as well, the dominant eigenvalue does not equal to one, most often it is larger than one. Hence procedures like those used in the UU case, to handle (i) normalization and (ii) convergence, must also be used here.

Secondly, because the graph is directed, it is in general not strongly connected, and in particular can have sink SCCs. These must have a "remedy" like that one described in the previous section. We propose again to use the R operator, in the same form as that for the ND case. This allows us (again) to implement R locally. Since weight conservation is not required here, the propagation equation need not take the exact form of (1)—any weighted sum of A and R would remove all sinks—but it is convenient to do so, and we will in the following.

2.5 Common problems

Summarizing, we find several challenges which must be addressed by a distributed calculation of a dominant eigenvector for a graph:

Synchronization A method needs to be find that can implement or replace the synchronization required by the consecutive matrix multiplications.

Normalization To achieve a stable, constant vector length, a method needs to be developed to keep the vector norm constant.

Sinks For directed graphs, the problem of sink SCCs must be remedied. We have shown a way to do so above, using the PageRank random surfer operator. Implementation of the latter requires the knowledge of the average of the weights.

In the following, we will present detailed solutions to this set of problems. These solutions represent a step towards enabling the distributed implementation of importance ranking for linked documents.

3 Fully Distributed Power Method

In this section we address the problem of calculating the principal eigenvector of a matrix, in a fully distributed way. On "fully distributed" we mean that the elements of the vector are held by individual network nodes, one vector element per one node. The matrix is represented by *links* between the network nodes, and *weights* assigned to these links. In networking terms, the links between the nodes can be physical or virtual (application level), although in our applications we will only consider virtual links. A virtual link from i to node j is represented by storing the address of j in node i. Assuming an underlying routing service, this means that node i can send messages addressed to node j.

For a matrix A, the element A_{ij} is represented by a link from node j to node i with the assigned weight of A_{ij}. If there is no link from j to i then $A_{ij} = 0$.

Our goal is to implement an iterative method, inspired by the *power method* [2], for finding the principal eigenvector, that is, the eigenvector with the largest absolute eigenvalue. To implement the power method, the matrix needs to be accessed only in the form of a vector multiplication, that is, we need to be able to calculate the product $A\mathbf{x}$ for a vector $\mathbf{x}$.

In this setting, the matrix A is stored in the links and their assigned weights in a distributed fashion, and the vector elements are stored by the nodes as mentioned above. In notations, the element $w_i^{(k)}$ of the weight vector in iteration k is stored in node i. As a result of matrix multiplication $\mathbf{w}^{(k+1)} = A\mathbf{w}^{(k)}$, we get

$$w_i^{(k+1)} = \sum_j A_{ij} w_j^{(k)}, \tag{3}$$

which can be computed if all nodes "propagate weights" through the links starting from them, that is, they multiply their own value by the weight of the outgoing link, and send a message to the target of the link containing this product. All nodes sum up the weights they receive which gives the new vector elements.

```
1: let $\mathbf{x}^{(0)}$ be a random vector ($\|\mathbf{x}^{(0)}\| = 1$)
2: for $i = 1, 2, \ldots$ do
3:     $\mathbf{y} \leftarrow A\mathbf{x}^{(i-1)}$
4:     $\mathbf{x}^{(i)} \leftarrow \mathbf{y}/\|\mathbf{y}\|$
5:     if $|1 - {\mathbf{x}^{(i)}}^T\mathbf{x}^{(i-1)}| < \epsilon$ then
6:         return $\mathbf{x}^{(i)}$
```

Figure 1. Power method

Based on this idea, we will gradually build up the iterative method, with the goal of making it robust, fully distributed and keeping it simple. The sub-problems to solve are: eliminating the need for synchronization points and solving the problem of normalization.

3.1 The synchronized approach

The algorithm of the power method [2] is shown in Figure 1. First of all, we will not address the stopping problem here, because in the applications we have in mind it is required to *maintain* the eigenvector in the face of possible changes of the matrix A continuously. In other words, if A changes in time, than the power method will continuously update the eigenvector accordingly, and this is exactly what we aim for.

Besides, as mentioned previously, we do not need to worry about the starting vector either because the power method only requires that it is not orthogonal to the principal eigenvector. In fact, due to loss of precision in numeric calculations, even that is not a strict criterion.

Let us start with the description of a naive distributed implementation that will involve in each iteration one synchronization point and an oracle for obtaining the normalization factor. Let each node i have a buffer variable b_i beside the weight variable w_i. This variable will be used to temporarily store and sum up weights contained in the incoming messages. This way, we have in fact defined another vector $\mathbf{b}$. A naive distributed implementation of this algorithm would involve one synchronized event: NEXTCYCLE. The NEXTCYCLE event signaling the beginning of cycle $m + 1$ would also contain the normalization factor in variable NEXTCYCLE.FACTOR. When receiving event NEXTCYCLE, node i would set $w_i = b_i/\text{NEXTCYCLE.FACTOR}$, and $b_i = 0$, followed by sending out weights wrapped in messages to the targets of its links according to the matrix-vector multiplication described above. These messages would be delivered as another event, MESSAGE, which would cause the node to add the value wrapped in the message to its buffer variable: $b_j = b_j + \text{MESSAGE.W}$.

Vector $\mathbf{b}$ can be initialized arbitrarily, just like the weight vector, considering the arguments we have given previously.

The drawback of this method is that it is very important that all nodes receive the NEXTCYCLE event in a way that guarantees that all messages sent in the previous cycle to the given node have been received; at that point vector $\mathbf{b}^{(m+1)}$ must hold the next (unnormalized) vector in the iteration, to get the dynamics of the power method:

$$\mathbf{b}^{(m+1)} = A\mathbf{w}^{(m)} \quad (4)$$

$$\mathbf{w}^{(m+1)} = \frac{\mathbf{b}^{(m+1)}}{\|\mathbf{b}^{(m+1)}\|} \quad (5)$$

Even though this event need not be synchronized according to a global clock, (the nodes need not receive the event that signals the beginning of the same cycle at the same time), it requires non-trivial mechanisms to keep track of the actual cycle in a consistent way throughout the system, and being able to read the current sate of the iteration. While it is clear that a solution could be developed to solve this cycle-synchronization problem, we will argue that we can simply ignore the problem of synchronization, and the protocol will still produce the correct result in a robust way.

3.2 A note on the normalization factor

In the synchronized version we assumed that the normalization factor is provided by an oracle so that the norm of the weight vector is kept constant. There is however another approach to normalization, that is based on knowing the largest absolute eigenvalue, $|\lambda_1|$. In this case the normalization factor can be always $|\lambda_1|$, a constant. It can be proven that the power iteration still converges to a dominant eigenvector. However, the norm of this dominant eigenvector is not know in advance, in particular it is not guaranteed that it will be one.

It might seem more complicated to obtain an approximation of the dominant eigenvalue than to normalize a vector. However, in a distributed implementation, it is more convenient to work with this second approach. The intuition behind this is that after convergence the eigenvalue can be observed fully locally: the eigenvalue is the growth factor of a vector element from one cycle to the next, and it is equal for all vector elements. Besides, it does not change from cycle to cycle either. These properties suggest that approximating the dominant eigenvalue can be done in a more robust way than approximating the norm of the vector.

Note that we *will* propose a method for driving the average of the vector elements towards one, thereby controlling the norm. However, at the heart of the protocol, convergence will primarily be achieved through approximating the average growth rate of the vector elements.

3.3 Outline of the distributed implementation

In the following sections we describe a fully distributed and robust implementation of an iterative method inspired by the power method.

First, in Section 3.4 we relax the assumption that all the nodes need to be aware of the current cycle and the global state of the computation. The new version is not mathematically equivalent to the power iteration. Assuming that the dominant eigenvalue is known and that messages are delivered instantly and without error, we present convergence results for the modified distributed method.

In Section 3.5 we describe a method to approximate the average growth rate of the vector, thereby replacing the assumption about the knowledge of the dominant eigenvalue with a practical distributed implementation. We show experimentally that the fully distributed protocol has similar convergence behavior to that of the power iteration, still under the assumption that messages are delivered instantly and without error.

In Section 3.6 we introduce a method to approximate the actual average of the vector. We use this approximation to modify the protocol to converge to an eigenvector that has a unit average, besides, we use this approximation also to implement the random surfer operator described in Section 2.3. We show experimentally that the protocol still provides stable convergence with very similar characteristics to that of the power method.

Finally, in Section 3.7 we analyze the fault tolerance of the protocol and introduce a last simple modification to deal with message delay jitter and message omissions. We experimentally demonstrate that this final version of the protocol has stable convergence properties under realistic failure scenarios as well.

3.4 Relaxing synchronization

Instead of developing a rigorous cycle-management protocol, we propose to relax this requirement and replace it with an extremely simple solution. According to this solution, all nodes simply have a parameter: Δ, the *cycle length*. As a result, the NEXTCYCLE event is simply generated locally at each node, in each Δ time units, measured by to the local clock of each node. It is easy to achieve that all nodes know Δ, in the simplest case it can simply be a fixed parameter of the protocol.

This means that in any time interval of length Δ, all nodes will generate the NEXTCYCLE event exactly once. This further means that (assuming that message delay is constant for a link), all nodes receive exactly one message in each interval of length Δ through all their incoming links.

We will still talk about global cycles, although it will simply mean consecutive time intervals of length Δ, starting from an arbitrary point in global time. The nodes need not know which cycle is being executed, or where our arbitrary starting point is, this notion of cycle simply provides a convenient way of describing the dynamics of the system. Accordingly, when we will refer to $\mathbf{w}^{(m)}$, we will refer to the snapshot of the distributed system at the time point that signals the beginning of our notion of cycle m.

In this section we assume that the dominant eigenvalue is known (λ_1) and the normalization factor is the constant $|\lambda_1|$. As a result of the relaxation of synchronization, we notice that the dynamics of the system will no longer be described by the power iteration, that is, it is not always true that $\mathbf{w}^{(m+1)} = A\mathbf{w}^{(m)}/|\lambda_1|$.

Instead, the new dynamics is given as follows, if a node sends all the messages at the same time, the messages are instantly delivered and the node index reflects the order of starting cycle $m+1$ locally[4]:

$$
\begin{aligned}
b_i^{(m+1)} &= \sum_{k=i}^{n} a_{ki} w_k^{(m)} &(6)\\
w_i^{(m+1)} &= \frac{b_i^{(m)} + \sum_{k=1}^{i-1} a_{ki} w_k^{(m)}}{|\lambda_1|} &(7)
\end{aligned}
$$

If we do not assume that the nodes send the messages on each outgoing link at the same time, the dynamics is very similar. Indeed, let as assume that the messages that belong to each outgoing link are sent independently, at random times, but in a way that for a fixed outgoing link the consecutive messages are sent regularly separated by Δ time units. One can think of this as if all outgoing links were autonomous but would work according to the same cycle length as the node itself. Then the model is

$$
\begin{aligned}
b_i^{(m+1)} &= \sum_{k=1}^{n} (1-\chi_{ki}) a_{ki} w_k^{(m)} &(8)\\
w_i^{(m+1)} &= \frac{b_i^{(m)} + \sum_{k=1}^{n} \chi_{ki} a_{ki} w_k^{(m)}}{|\lambda_1|} &(9)
\end{aligned}
$$

where χ_{ki} is zero or one for all k, i.

Now, let us assume that $\lambda_1 = 1$. This is only to simplify notation, since we assumed that λ_1 is known, but then the elements of A can be divided by λ_1. The resulting matrix A/λ_1 has 1 as dominant eigenvalue, and the dynamics of the protocol has exactly the same mathematical description as that of the system defined above.

We will prove that the asynchronous version of the protocol can converge only to the dominant eigenvector of A, if A is non-negative. As a first step, let us observe the both models above can be described in the following form:

$$B = \begin{pmatrix} O & A_1 \\ I & A_2 \end{pmatrix} \tag{10}$$

$$\begin{pmatrix} \mathbf{b}^{m+1} \\ \mathbf{w}^{m+1} \end{pmatrix} = B \begin{pmatrix} \mathbf{b}^{m} \\ \mathbf{w}^{m} \end{pmatrix} \tag{11}$$

where $A_1 + A_2 = A$, O is the all zero matrix and I is the identity matrix. In other words, the protocol is equivalent to the power method executed on the block matrix B.

Lemma 3.1. *The value 1 is a dominant eigenvalue of matrix B (given in* (10)*), provided that $\lambda_1 = 1$ is a dominant eigenvalue of A.*

Proof. The proof is the application of the Perron-Frobenius theorem. First of all, $B \geq 0$, so according to the Perron-Frobenius theorem the spectral radius of B is an eigenvalue, in other words, there is a real positive dominant eigenvalue. Let λ be this real positive eigenvalue, and $(\mathbf{b}^*, \mathbf{w}^*)^T$ a corresponding eigenvector. Let us assume that the Lemma does not hold, that is, $\lambda > 1$. Then from (11) we have $\lambda \mathbf{b}^* = A_1 \mathbf{w}^*$, $\lambda \mathbf{w}^* = \mathbf{b}^* + A_2 \mathbf{w}^*$. Substituting $\mathbf{b}^*$ into the second equation we have

$$\lambda \mathbf{w}^* = (A_1/\lambda + A_2)\mathbf{w}^* \tag{12}$$

Due to the assumptions, $(A_1/\lambda + A_2) \leq A$, so according to the Perron-Frobenius theorem, the spectral radius of $(A_1/\lambda + A_2)$ has to be less than or equal to $\lambda_1 = 1$. This leads to a contradiction, since according to (12) $\mathbf{w}^*$ is an eigenvector of $(A_1/\lambda + A_2)$ with an eigenvalue $\lambda > 1$. This proves that $\lambda \leq 1$.

We now show that $\lambda \geq 1$. Indeed, let $A\mathbf{w}^* = \mathbf{w}^*$. Then it can be verified that $(A\mathbf{w}^*, \mathbf{w}^*)^T$ is an eigenvector of B with eigenvalue 1. □

4. The last assumption is just to simplify notation

```
1: for each consecutive $\Delta_r$ time units at
   a randomly picked time do
2:     $q \leftarrow$ GETRANDOMPEER()
3:     send $r_p$ to $q$
4:     $r_q \leftarrow$ receive($q$)
5:     $r_p \leftarrow (r_p + r_q)/2$
```

(a) active thread

```
1: loop
2:     $r_q \leftarrow$ receive($*$)
3:     send $r_p$ to sender($r_q$)
4:     $r_p \leftarrow (r_p + r_q)/2$
```

(b) passive thread

Figure 2. Gossip based averaging protocol executed by node p. The local state of p is denoted as r_p.

3.5 Distributed calculation of the eigenvalue

The protocols discussed so far assume an oracle service that can provide the nodes with the correct normalization factor, that is, $|\lambda_1|$. We propose the implementation of that service with the help of our gossip-based averaging protocol [6]. For the sake of completeness, we first briefly summarize the basic idea behind this protocol (shown in Figure 2), and then describe how it is applied for approximating the growth rate of the approximation of the eigenvector.

The averaging protocol is run by all nodes in parallel with the distributed power iteration. The local state r_p of node p is the current approximation of the average at node p. As a result of the protocol, at all nodes these approximations quickly converge to the average of the initial values of the local approximations. The protocol relies on a *peer sampling service*, accessed by GETRANDOMPEER, that returns a random node from the system. We use NEWSCAST to implement this service, a detailed description can be found in [5]. The details of the NEWSCAST protocol are not required for understanding the present work, the only important aspect is the incurred communication cost. Fortunately, the averaging protocol and newscast can be implemented to use the same connections (talking to the same nodes at the same frequency) merging the messages so no overhead is introduced.

The calculated average can be accessed locally by simply reading the local approximation. To calculate the growth rate of the vector from one cycle to another, each node, when the regularly generated NEXTCYCLE event is generated, overwrites the local approximation of the growth rate by the *logarithm* of the locally observed growth rate of the vector element held by the node. That is, node i sets $r_i = \log b_i^{m+1}/w_i^m$. This way, we in effect calculate the *geometric mean* of the growth rates, which is a natural choice for a mean of multiplicative factors.

The approximation of the growth rate is therefore $e^{r_i(t)}$ at node i at time t. This value is used to normalize the element b_i^{m+1}, that is,

$$w_i^{m+1} = \frac{b_i^{m+1}}{e^{r_i(t)}}. \tag{13}$$

After the normalization the local approximation of the growth factor $r_i(t)$ is updated as described above.

A cycle length $\Delta_r < \Delta$ is chosen so that sufficient mixing is achieved. We set $\Delta_r = \Delta/30$, based on the results from [6], where it is shown that 30 cycles already provide an extremely accurate approximation independent of network size.

3.6 Approximating the vector average

Approximating the growth rate (and thereby the dominant eigenvalue), solves the problem of convergence, however, knowing the average of the vector locally is still useful for two purposes. The first, and more important, is the implementation of the random surfer operator, as described in Section 2.3. As we have seen, the random surfer operator with parameter ϵ can be simulated by simply adding to the local weight the average of all the weights in the system multiplied by the factor ϵ. The second application is pushing the vector average towards one, which can increase numeric stability and can support some specific application requirements too.

To implement the calculation of the vector average, we apply another instance of the averaging protocol described above. This protocol will use the same Δ_r cycle length. Furthermore, it will not represent communication overhead, because, like merging the NEWSCAST and the growth rate averaging messages, the vector average calculation messages can also be merged with these previous two. Since all three protocols (NEWSCAST and the two averaging protocols) generate small messages, the actual communication overhead is the same as if we ran only one of them.

To calculate the vector average, we will apply a slightly more sophisticated version of the averaging protocol, that involves periodic restarts. This technique is also described in [6]. In a nutshell, the averaging protocol works in *epochs*, that consist of 30 consecutive cycles. Since we set $\Delta_r = \Delta/30$, one epoch corresponds to one cycle of the power method. We do not describe the restarting method in detail here: it is completely automatic and transparent to the application. That is, in each cycle, a node informs the averaging protocol about the current local value, and reads out the current converged approximation of the average.

The (automatic) restarting technique is necessary because we have to make sure that all nodes see the same approximation. Since the vector values can radically differ in magnitude, the averaging protocol needs the 30 cycles of undisturbed convergence. In the case of the approximation of the growth rate, nodes constantly update the approximations, so, strictly speaking, no convergence could ever be achieved if the local growth rates did not converge to the same value themselves. The price of organizing averaging into a series of epochs is that the vector average is known only with a delay of at least the epoch length Δ. Intuitively, this is not a problem because the protocol ensures that the average converges to a constant value, so a delay quickly becomes irrelevant.

Using the approximation of the average, and the parameter ϵ, the random surfer operator is implemented by the modified update rule

$$w_i^{m+1} = (1-\epsilon)\frac{b_i^{m+1}}{e^{s_i(t)}} + \epsilon n_i(t) \tag{14}$$

where $n_i(t)$ is the locally known converged approximation of the average at time t.

Finally, if the average needs to be pushed towards one, then we propose the following heuristic rule, that involves the modification of the growth rate approximation by a small factor that introduces a bias towards the unit average vector. Intuitively, if the average is too large, we decrease the local value a little more, and if it is too low, we increase a little more. More formally,

$$c = e^{s_i(t)} \cdot \left(\frac{0.2}{1+1/n_i(t)} + 0.9\right) \tag{15}$$

$$w_i^{m+1} = (1-\epsilon)\frac{b_i^{m+1}}{c} + \epsilon n_i(t). \tag{16}$$

The factor in (15) is a sigmoid function over the logarithm of $n_i(t)$, transformed to have range $[0.9, 1, 1]$. This means that the growth rate approximation is never altered by more than 10% no matter how far the average from one. This feature is essential to maintain the stability of convergence. Taking the logarithm to define the sigmoid function is necessary to handle the both the case when the average is less than one, and when it is larger than one equally.

3.7 Fault tolerance

So far we have described a fully distributed protocol that has favorable convergence properties. However, our assumption has always been that the messages are delivered instantly (zero delay) without any error.

In a realistic scenario, we need to deal with three sources of error: message delay, message delay jitter and message omissions. On jitter we mean the variation of the arrival time of messages with respect to the expected arrival time. Note that the expected arrival time of messages from a given source is in regular intervals of length Δ.

The protocol in its unmodified form turns out to be very robust to constant message delay, but it is very sensitive to both delay jitter and message omission. A similar problem exists in multimedia applications where the solution is to buffer the incoming messages and mask the jitter buy presenting the messages to the application in regular intervals. We adopt this solution for our protocols, only we deal with omissions using the same buffer as well.

In our very simple implementation, all incoming sources are assigned a separate buffer. This does not assume that nodes know the incoming links beforehand; the buffer can be created when the first message is

	math				albert			
	NU	UU	ND	UD	NU	UU	ND	UD
λ_1	1.0000	38.3298	0.80128	2.734	1.000000	185.67	0.84648	121.6063
$\|\lambda_2\|$	0.9355	36.8392	0.80000	2.653	0.999982	162.46	0.80000	121.6055

Table 1. The first and second largest magnitude eigenvalues. Note that the largest magnitude eigenvalue is guaranteed to be real and positive.

detected from a given source. The buffer simply remembers the last value received and returns that value when the applications requests it (in regular intervals of length Δ).

This simple mechanism also solves the omission problem automatically: when a message does not arrive, the previous message is used. The intuition is that in the converged phase all messages contain exactly the same value, so it is completely undetectable if one was lost. During convergence the consecutive values received are also similar. Finally, note that this mechanism can be extended to deal with churn, that is, the dynamic join and leave operations of nodes in a natural way: the buffer can gradually decrease the value it stores if it does not get refreshed eventually reaching zero when it is removed completely.

Finally, it is also possible to optimize communication overhead: when a node feels its value has converged, it simply stops transmitting messages, and its neighbors will switch to the buffer instead. In case of any change in the local value, the node can adaptively resume message transfer. To avoid the timeout effect described below, a requested timeout can be attached to all messages sent out, which can be increased in stable periods.

4 Experimental Results

We performed extensive simulation experiments using the PEERSIM simulator developed at the University of Bologna [11]. The goal of the simulations is twofold. First, we compare the convergence characteristics of the method to our baseline: the power iteration. Second, we explore the fault tolerance of the protocol in extreme failure scenarios.

We used two subsets of the WWW as test data. The first one was collected based on the methodology described in [8]. In a nutshell, one popular keyword was searched for in Google, in our case, "mathematics". The first 150 top ranked hits are recorded: they form the core of the graph. Then all their outlinks and all their inlinks were added up to a prescribed maximum (40) and finally all links that connect the node set obtained this way are added. This data is available from us upon request. The sample contains 4586 nodes. We will call this sample MATH. The second one is the same dataset used in [1], available from the authors. It was generated by a crawler, starting from one page within the domain of the University of Notre Dame. This sample has 325729 nodes. We will call this sample ALBERT.

Both datasets define a directed graph. As described in Section 2, we generate the four problem instances NU, UU, ND and UD. For the directed cases, the parameter of the random surfer operator is $\epsilon = 0.2$. Table 1 shows the first two largest magnitude eigenvalues for all the problem instances. Note that the eigenvalue gap (the difference between the first and second largest eigenvalues) predicts the convergence speed of the power iteration. For a small gap, convergence is slow. With a zero gap, the power iteration does not converge at all.

Figure 3 shows the convergence of the power method on these problems. Note that some cases show poor convergence, which is predicted by the small eigenvalue gap. In this paper we are not interested in "fixing" this problem and transform the graphs so that the eigenvalue gap becomes larger. Here we are concerned with showing that the distributed power method we propose has comparable performance to the sequential power method, illustrated in Figure 3.

4.1 Convergence in the absence of failure

Here we assume that messages are delivered instantly and without error. Experiments were performed with the protocols described in Section 3.6. As mentioned several times previously, the starting vector (and therefore the bootstrap procedure) is non-critical, as long as all nodes start eventually. We assume that each

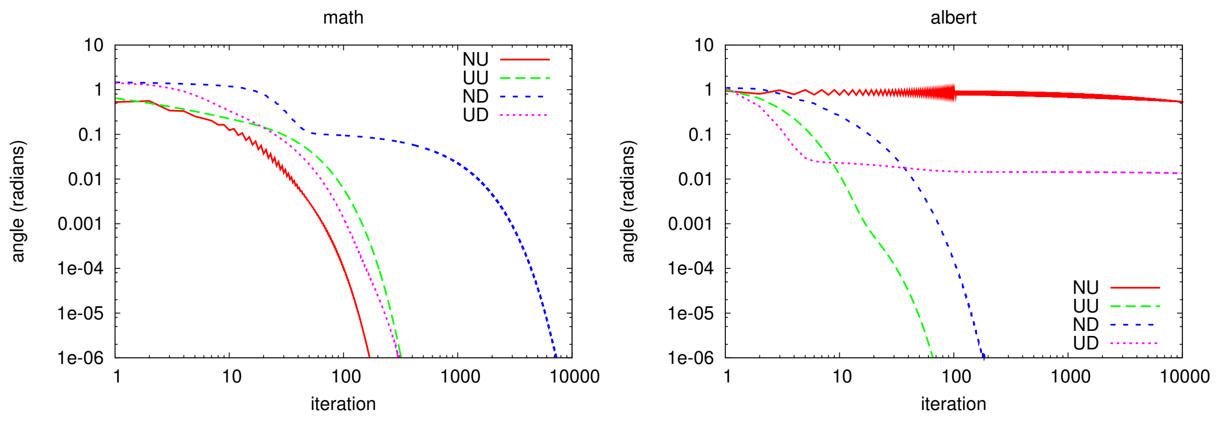

Figure 3. Convergence of power iteration. Plots show the angle between the vector in the given iteration and the principal eigenvector.

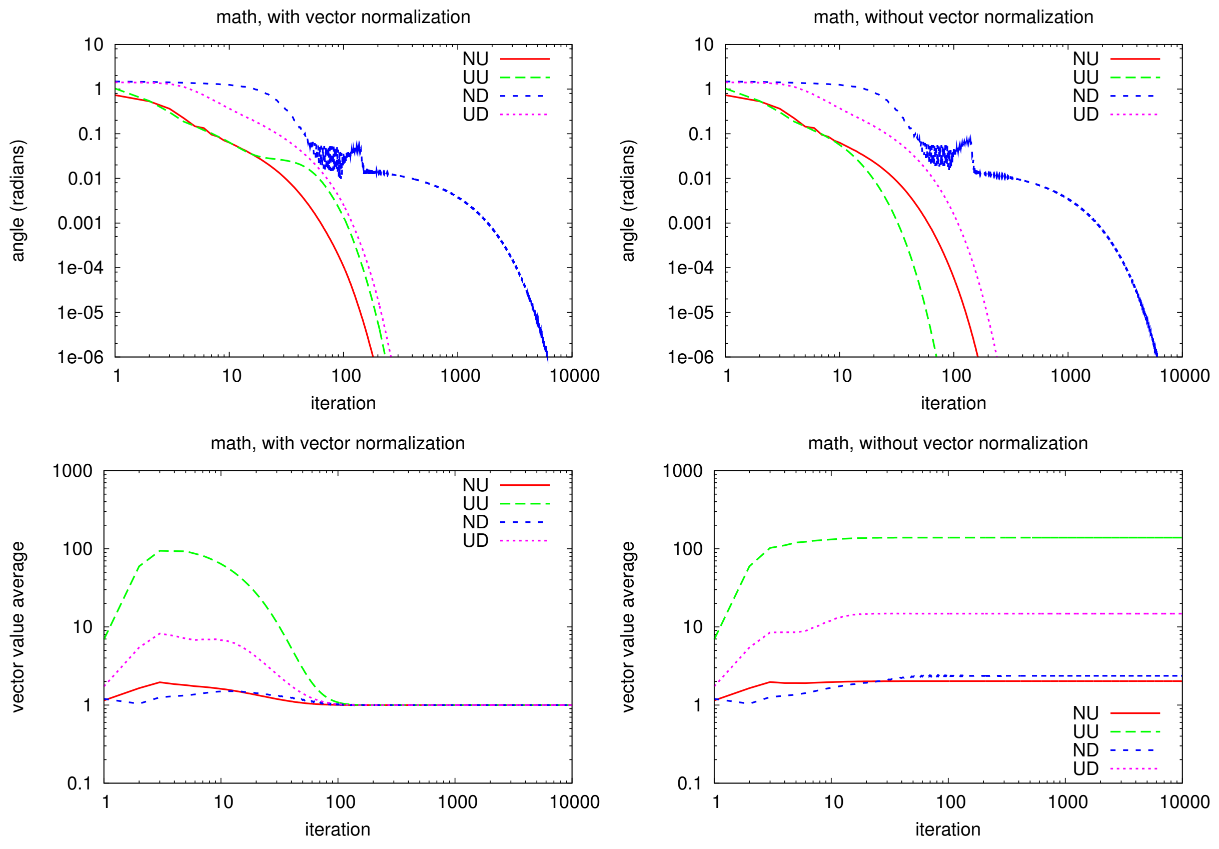

Figure 4. Convergence in the absence of failure over the MATH problem set.

nodes starts at a random time between a zero time point and time point Δ (that is, the cycle length). All vector elements ($\mathbf{w}^0$ and $\mathbf{b}^0$) are initialized to one.

Figure 4 shows the convergence of the angle between the principal eigenvalue and the approximation, and the average vector value of the approximation. The performance is very similar to that of the power method. In fact, when there is no normalization of the average of the vector, the UU case converges faster

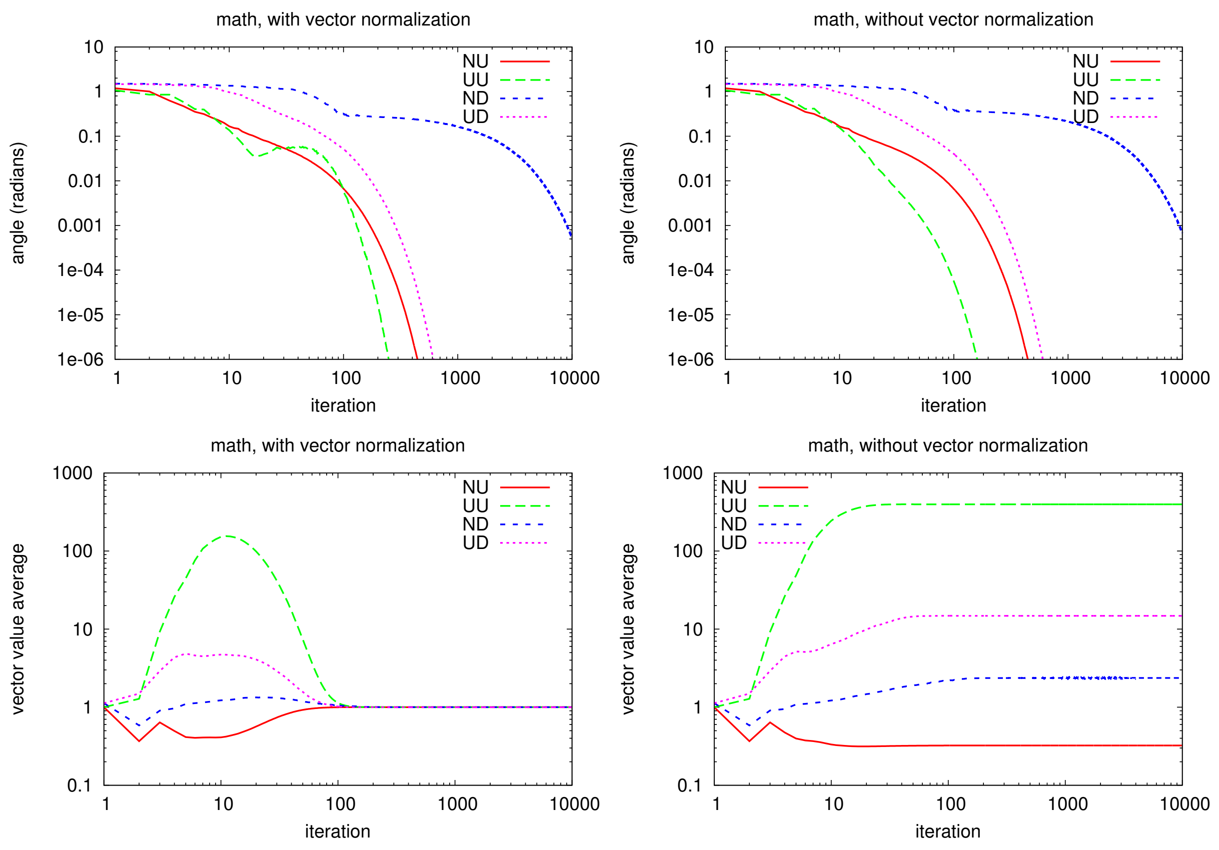

Figure 5. Convergence in an extreme failure scenario over the MATH problem set.

than with power iteration. When there is normalization, then convergence is temporarily slowed down until the normalization is finished. The ND case also converges faster both with and without normalization.

4.2 Convergence in the presence of failure

As already mentioned in Section 3.7, preliminary experiments show that the protocol is very sensitive to message delay jitter and message omission, and we suggested a solution involving buffers to solve it.

In this section we present simulation results with this solution in the following scenario. All messages are delayed according to the formula $\delta + \delta_j$ where $\delta = \Delta/2$ is a constant, and δ_j is a uniform random variable from the interval $[-\Delta/2, \Delta/2]$. If we consider that in a real application Δ can be expected to be in the range of minutes, this delay model is extremely harsh. It is safe to assume that all realistic delay will be lower than this, with less jitter, at least for messages that are not dropped.

The case of infinite delay (that is, message omission) is also considered: in addition to the above delay, we drop every message independently with a probability 50%. This can also be considered a safe upper bound on realistic drop rate, even if no reliable transfer protocol, such as TCP, is used.

Results are shown in Figures 5 and 6. For the ALBERT data set fewer iterations are shown due to the resource requirements of the simulations. Over such a large network, it takes days to generate 1000 iterations with one problem instance on an up-to-date workstation.

We can observe that convergence proportionally slows down in some of the cases with respect to the power method. This is not unexpected, given that 50% of the messages are dropped. However, the protocol achieves the same accuracy as before. This is encouraging given that, without the application of buffers, as little as 1% jitter destroys convergence even if there are no message omissions.

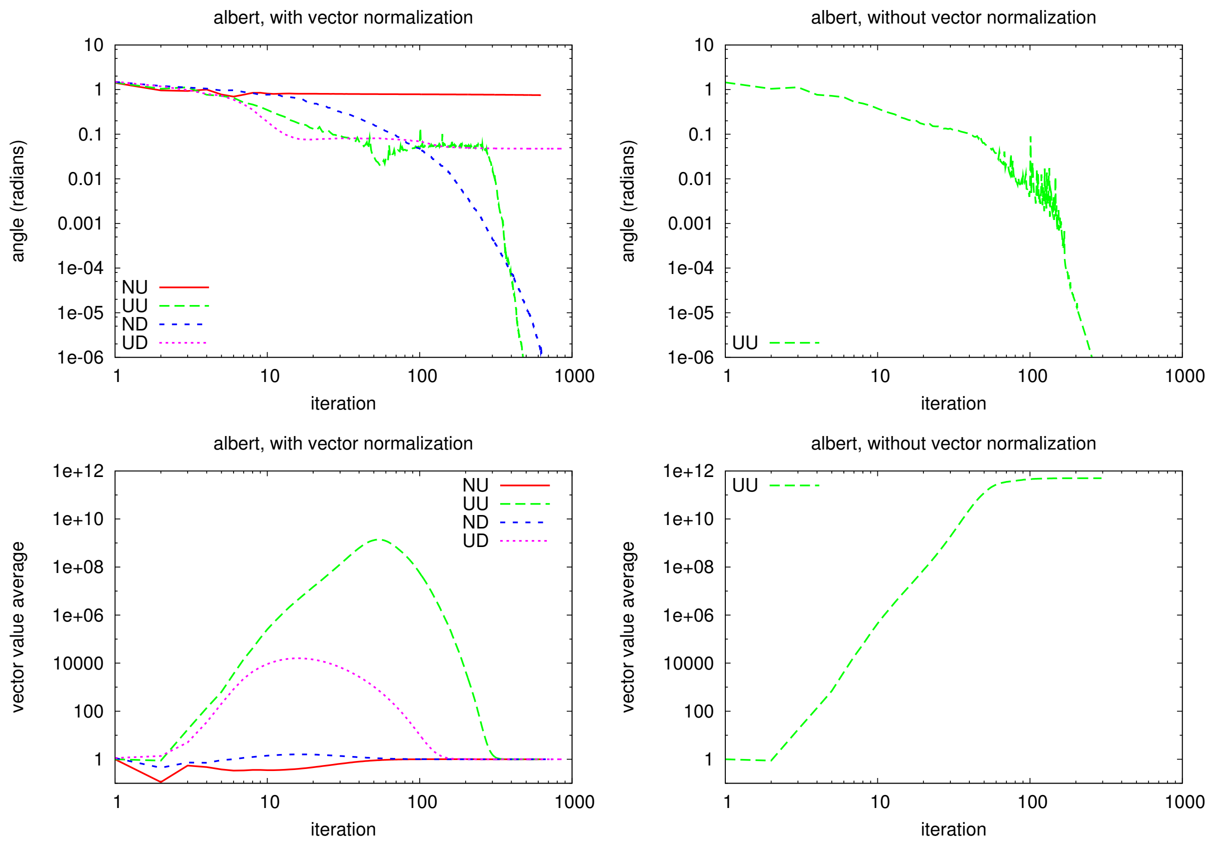

Figure 6. Convergence in an extreme failure scenario over the ALBERT problem set.

5 Conclusions and Future Work

In this paper we have proposed a fully distributed power method for finding the principal eigenvalue of arbitrary matrices, that are stored in weighted links of an overlay network. We addressed the problem of synchronization and normalization. Through the example of link analysis we demonstrated both theoretically and experimentally that the proposed method is competitive with the power method. We have also demonstrated extreme fault tolerance.

A large number of ideas remain unexplored. In particular, the version of the protocol presented here is unoptimized. For example, there are several possibilities to reduce the communication overhead. This would need a thorough investigation of some of the parameters such as the ratio Δ/Δ_r, and a full exploitation of the possibilities buffering introduces, such as the adaptive change of message sending rate, masked by the buffers at the receiver side. Our future work includes investigating additional failure scenarios, such as churn, and looking at the properties of the protocol in dynamic environments, where the links may change.

References

[1] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Diameter of the world wide web. *Nature*, 401:130–131, 1999.

[2] Zhaojun Bai, James Demmel, Jack Dongarra, Axel Ruhe, and Henk van der Vorst, editors. *Templates for the Solution of Algebraic Eigenvalue Problems: a Practical Guide*. SIAM, Philadelphia, 2000.

[3] P. Bonacich. Factoring and weighting approaches to status scores and clique identification. *Journal of Mathematical Sociology*, 2:113–120, 1972.

[4] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[5] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In Hans-Arno Jacobsen, editor, *Middleware 2004*, volume 3231 of *Lecture Notes in Computer Science*, pages 79–98. Springer-Verlag, 2004.

[6] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, August 2005.

[7] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web (WWW'03)*, pages 640–651, New York, NY, USA, 2003. ACM Press.

[8] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[9] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 1995.

[10] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[11] PeerSim. http://peersim.sourceforge.net/.

[12] Karthikeyan Sankaralingam, Simha Sethumadhavan, and James C. Browne. Distributed pagerank for p2p systems. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12 '03)*, pages 58–69, 2003.

[13] ShuMing Shi, Jin Yu, GuangWen Yang, and DingXing Wang. Distributed page ranking in structured p2p networks. In *Proceedings of the 2003 International Conference on Parallel Processing (ICPP03)*, pages 179–186, October 2003.

[14] Ron Weiss, Bienvenido Vélez, Mark A. Sheldon, Chanathip Nemprempre, Peter Szilagyi, Andrzej Duda, and David K. Gifford. HyPursuit: a hierarchical network search engine that exploits content-link hypertext clustering. In *Proceedings of the the seventh ACM conference on Hypertext (HYPERTEXT '96)*, pages 180–193. ACM Press, 1996.