Asynchronous Distributed Power Iteration with Gossip-based Normalization*

Márk Jelasity[1], Geoffrey Canright[2], and Kenth Engø-Monsen[2]

[1] University of Szeged, HAS Research Group on AI, Hungary
[2] Telenor R&D, Fornebu, Norway

Abstract. The dominant eigenvector of matrices defined by weighted links in overlay networks plays an important role in many peer-to-peer applications. Examples include trust management, importance ranking to support search, and virtual coordinate systems to facilitate managing network proximity. Robust and efficient asynchronous distributed algorithms are known only for the case when the dominant eigenvalue is exactly one. We present a fully distributed algorithm for a more general case: non-negative square matrices that have an arbitrary dominant eigenvalue. The basic idea is that we apply a gossip-based aggregation protocol coupled with an asynchronous iteration algorithm, where the gossip component controls the iteration component. The norm of the resulting vector is an unknown finite constant by default; however, it can optionally be set to any desired constant using a third gossip control component. Through extensive simulation results on artificially generated overlay networks and real web traces we demonstrate the correctness, the performance and the fault tolerance of the protocol.

1 Introduction

The calculation of the dominant eigenvector of a matrix has long been a fundamental tool in almost all areas of science. In recent years, eigenvector calculation has found new and important applications in fully distributed environments such as peer-to-peer (P2P) overlay networks.

For example, the PageRank algorithm [13] calculates the *importance ranking* for hyperlinked web pages. The calculated ranks are given by the dominant eigenvector of a matrix that can be derived from the adjacency matrix of the graph defined by the hyperlinks. Fully distributed algorithms have already been proposed to implement PageRank [14,16,17]. As another example, *trust assignment* is a key problem in P2P networks. In [9] a method was proposed, that assigns a global trust value to each peer, through calculating the dominant eigenvector of the matrix containing local (pairwise) trust values. Finally, the eigenvectors

* In: Euro-Par 2007, LNCS 4641, pp. 514-525, 2007, Springer. DOI: 10.1007/978-3-540-74466-5_55. This work was completed while Márk Jelasity was with the University of Bologna. This work was partially supported by the Future and Emerging Technologies unit of the European Commission through Project DELIS (IST-2002-001907).

that belong to the largest few absolute eigenvalues also play a role in esthetic low dimensional *graph layout* [11]. This application is relevant in virtual coordinate assignment that allows to map the actual delays among all pairs of nodes onto the distance in the n-dimensional Euclidian space [5].

Motivated by these applications, and firmly believing that new ones will keep emerging, we identify fully distributed eigenvector calculation as an important P2P service that should be studied in its own right.

The environments we target impose special requirements. We assume, that there is a large number of nodes, the connections are volatile and unreliable and the eigenvector needs to be continuously updated and maintained in a decentralized way. Communication is implemented through message passing, where messages can be dropped or delayed. However, nodes have access to a local clock that measures the passage of real time with a reasonable accuracy. We do not assume that the local clocks at different nodes are synchronized.

In this model, we propose a protocol that involves three components. The first is an instantiation of the asynchronous iteration model described in [12]. This algorithm requires that the dominant eigenvalue is exactly one. We extend this protocol with a gossip-based control component that allows the iteration algorithm to converge even if the dominant eigenvalue is less than or greater than one. A third gossip component can be applied to explicitly control the exact value of the vector norm (which is an unspecified finite value without this third component).

These extensions make the asynchronous iteration robust to dynamic change and errors. Traditional methods are very sensitive to the dominant eigenvalue being exactly one: the slightest deviation results in misbehavior on the long run. Besides, the protocol is able to implement algorithms that assume a dominant eigenvalue different from one. A recent promising example is a ranking method using unnormalized web-graphs [4].

We demonstrate the correctness, the performance and the fault tolerance of the protocol through extensive simulation results on artificially generated overlay networks and real web traces.

2 Chaotic Asynchronous Power Iteration

Given a square matrix A, vector x is an *eigenvector* of A with *eigenvalue* λ, if $Ax = \lambda x$. Vector x is a *dominant* eigenvector if there are no other eigenvectors with an eigenvalue larger than $|\lambda|$ in absolute value. In this case λ is a *dominant eigenvalue* and $|\lambda|$ is the *spectral radius* of A.

Motivated by the various application areas mentioned in the Introduction, in this paper we concentrate of the abstract problem of calculating the dominant eigenvector of a weighted neighborhood matrix of some large network, in a fully distributed way. By "fully distributed" we mean the worst case, when the elements of the vector are held by individual network nodes, one vector element per one node. The matrix A is defined by physical or overlay *links* between the network nodes, more precisely, the *weights* assigned to these links: let matrix

```
1: loop {Active Thread}
2:    wait(Δ)
3:    for each j ∈ out-neighbors_i do
4:       send A_ji w_i to j
5:    b_i ← Σ_{k∈in-neighbors_i} b_ki
6:    w_i ← b_i
```

```
1: loop {Passive Thread}
2:    x ← receive(∗)
3:    k ← sender(x)
4:    b_ki ← x
```

Fig. 1. Asynchronous iteration executed at node i.

element A_{ij} be the weight of the link from node j to node i. If there is no link from j to i then $A_{ij} = 0$.

In [12], Lubachevsky and Mitra present a chaotic asynchronous family of message passing algorithms to calculate the dominant eigenvector of a non-negative irreducible matrix, that has a spectral radius of one. Figure 1 shows an instantiation of this framework, that we will apply in this paper.

In the algorithm, the values w_i represent the elements of the vector that converges to the dominant eigenvector. The values b_{ki} are buffered incoming weighted values from incoming neighbors in the graph. These values are not necessarily up-to-date, but, as shown in [12], the only assumption about message failure is that there is a finite upper bound on the age of these values. The age of value b_{ki} is defined by the time that elapsed since k sent the last update successfully received by i. This bound can be very large, so delays and message drop are tolerated extremely well. In addition, the values b_{ki} have to be initialized to be positive.

In dynamic scenarios, when nodes or network links are added or removed, the algorithm is still functional. Temporary node failures, churn, and link failures are all regarded as message failures, and are therefore covered by the assumption of the finite upper bound on update delay. Permanent changes can be dealt with as well: after the change the vector will start converging to the new eigenvector, provided simple measures are taken to make sure nodes remove dead links and take new ones into consideration. Due to lack of space, we omit further details on the dynamic scenarios.

3 Adding Normalization

Let λ_1 be a dominant eigenvalue of A. We can assume that $\lambda_1 \geq 0$ since A was assumed to be non-negative. The asynchronous method described above is known to work correctly if $\lambda_1 = 1$, but if $\lambda_1 > 1$ or $\lambda_1 < 1$, then the vector elements will grow indefinitely or tend to zero, respectively. This motivates us to propose a control component that continuously approximates the *average growth rate* of the vector elements, and normalizes each updated component with this value. Note that after we achieve convergence, the growth rate of every single vector element becomes λ_1. This suggests that approximating the global average using local, limited information is a viable plan.

We adopt the gossip protocol described in [8] to approximate the average growth rate. More precisely, we will use this algorithm to approximate the geo-

```
1: loop {Active Thread}
2:    wait(Δ_r)
3:    j ← getRandomPeer()
4:    send r_i to j
5:    r_j ← receive(j)
6:    r_i ← (r_i + r_j)/2
```

```
1: loop {Passive Thread}
2:    r_j ← receive(∗)
3:    send r_i to sender(r_j)
4:    r_i ← (r_i + r_j)/2
```

Fig. 2. Gossip based averaging protocol executed by node i. The local state of i is denoted as r_i.

metric mean of the local growth rates $b_i^{(m+1)}/w_i^{(m)}$ over all nodes i, where $b_i^{(m+1)}$ is the value calculated in line 5 in Figure 1 and $w_i^{(m)}$ is the value of w_i before executing line 6. The geometric mean is a more natural choice since we average multiplicative factors.

The averaging protocol (shown in Figure 2) is run by all nodes in parallel with the distributed power iteration. The local state r_i of node i is the current approximation of the average at node i. 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 [7]. 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 (sending messages to the same nodes at the same time). This means the newscast adds no extra cost.

To calculate the geometric mean, each node i, when updating w_i, 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 $r_i = \log(b_i^{(m+1)}/w_i^{(m)})$. The approximation of the growth rate is therefore $e^{r_i(t)}$ at node i at time t. This value is used to normalize b_i, that is, we replace line 6 by $w_i = b_i/e^{r_i(t)}$ in the active thread of the iteration algorithm.

A cycle length $\Delta_r < \Delta$ is chosen so that a sufficiently accurate average is calculated, in spite of the continuous updates of r_i external to the averaging protocol. According to preliminary experiments, setting $\Delta_r = \Delta/5$ is already a safe choice on all the problems we examined. This is because, based on the results from [8], the approximation error decreases exponentially fast, besides, the growth rate is similar at all vector elements, as mentioned before.

4 Controlling the Vector Norm

The iteration component combined with gossip-based normalization is designed to achieve convergence, however, the norm of the converged vector is not known in advance. In some applications this might not be sufficient, since interpreting a single vector element becomes impossible, only relative values carry information.

Besides, in scenarios when the matrix A constantly and frequently changes, the vector norm can grow without bounds or can tend to zero without explicitly controlling the vector norm. Finally, knowing a suitable vector norm makes it possible to implement some algorithms that require global knowledge. We will describe the random surfer operator of the PageRank algorithm as an example.

To address these issues, we apply a second gossip component for calculating two measures: the maximum and the average of the absolute value of the vector elements. The maximum is also known as the norm $\|.\|_\infty$, while the average is equal to $\|.\|_1/N$, where N is the dimension of the vector.

To get the maximum, in the protocol in Figure 2 we simply have to replace lines 6 and 4 in the active and passive threads, respectively, with calculating the maximum instead: $r_i \leftarrow \max(r_i, r_j)$.

It must be noted that in the case of norm calculation, $\Delta_r = \Delta/30$ appears to be necessary according to preliminary experiments, since, unlike growth rates, the vector elements themselves are not guaranteed to be similar.

Let us now assume that $n_i(t)$ is the approximation of either the maximum or the average of the vector at node i. To push this value towards one, we propose the following heuristic control factor to modify the normalization factor to introduce a bias towards the vector of which the average or maximum, respectively, is one. Intuitively, if $n_i(t)$ is too large, we decrease the local value a little more, and if it is too low, we increase a little more. More formally, we calculate a factor c as

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

and subsequently replace line 6 with $w_i = b_i/c$. The factor c in (1) 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 is from one.

As a relatively more complex example of the possibilities this framework offers, we present the implementation of the random surfer operator used in the PageRank algorithm [13]. This operator will in turn allow us to implement PageRank as well.

The PageRank algorithm is concerned with the normalized adjacency matrix of a directed graph (e.g., the WWW link graph). Apart from this directed graph, the PageRank algorithm uses a "random surfer" operator R as well, defined as $R_{ij} = 1/N$, for all $i, j = 1, \ldots, N$, where N is the number of nodes. This corresponds to the definition of R as being a uniform random walk on the fully connected graph (hence the name "random surfer"). The net effect of R is to add a constant weight 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 the average of the vector is known [13]. Hence, we can effectively replace matrix A with the PageRank operator $(1-\epsilon)A + \epsilon R$ where the second term involving R may be known as long as the average of $\mathbf{w}$ is known. Note that ϵ is a parameter of the PageRank algorithm, and defines the weight of the random surfer operator.

As described above, we can in fact obtain an approximation of the vector average. Then we can implement the PageRank R operator—a global operator—using purely local operations: node i now has the update rule

$$w_i = (1-\epsilon)\frac{b_i}{e^{r_i(t)}} + \epsilon n_i(t) \tag{2}$$

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

Finally, note that controlling the average of the vector and applying the random surfer operator can be done simultaneously as well, using the update rule

$$w_i = (1-\epsilon)\frac{b_i}{c} + \epsilon n_i(t). \tag{3}$$

where c is the same as in (1).

5 Experimental Results

We performed extensive event-based simulation experiments using the PeerSim simulator developed at the University of Bologna [15]. The goal of the experiments was to demonstrate that our method, is both efficient and robust to failure.

5.1 Notes on the Implementation

In the case of one of the components—the gossip-based protocol that continuously calculates the *average* of the current vector approximation, described in Section 4—we applied two modifications to increase robustness.

First, instead of the protocol in Figure 2, we applied a variant presented in [10]. This variant is very similar; the main difference is that it is slightly modified so that it can apply the "push only" communication model, while the original version is based on the "push-pull" model. In the push model, the nodes only send messages, but need not answer them. In the push-pull model all messages must be answered immediately. We apply the push variant because it is more robust to message delays: while in the push-pull version the state of the nodes are inconsistent for a short time (between the sending and reception of the answer), this problem does not exist in the push version.

The second modification of this component is, that we apply the *restarting* technique described in [8]. According to this technique the computation of the average is performed in consecutive *epochs*. During one epoch the local values of the nodes (r_i) are not allowed to be updated to allow for convergence. The start of the new epoch is fully automatic and distributed. All messages are tagged with an epoch identifier, that is increased when a given node has completed a specified number of cycles. The node also updates its epoch counter if it detects an incoming message that is tagged by a larger epoch identifier than its own. This way the start of the new epoch is propagated very quickly in an epidemic fashion. The length of the epoch is defined as 30 cycles in our experiments. Recall that one cycle is Δ_r time units, and $30\Delta_r{=}\Delta$.

5.2 Artificially Generated Matrices

For evaluating the protocol we applied a set of artificially generated matrices with controlled properties. To model real applications mentioned in the Introduction, all matrices are sparse and are derived from the adjacency matrix of a link graph. First let us define the graphs that were used to define the matrices.

All graphs have 5000 nodes. The baseline case is a directed random graph, according to the k-out model. In this model, k random out-links are added to each node. We generated an instance of this model with $k = 8$.

The second graph is a scale free graph generated by the Barabási-Albert model [1]. Most importantly, the degree distribution of this graph follows a power law, that is extremely unbalanced with many low degree nodes and a few high degree nodes, and that is known to describe many interesting emergent networks such as the WWW or social relationships [1]. The parameter of the model was set to two. In this case the Barabási-Albert model defines an undirected graph by starting with two disconnected nodes, and subsequently adding nodes one by one, linking each new node to two existing nodes. These two nodes are selected with a probability proportional to their degree (preferential attachment). The average degree in the graph is thus four.

The third graph was generated starting with an undirected ring, and adding two random out-links from all nodes (note that this procedure follows a modified version of the Watts-Strogatz model [18]). The motivation behind using this graph is that, as we will see, its adjacency matrix has a small eigenvalue gap, which results in a slow convergence of the power iteration. This graph was chosen to test whether our method is sensitive to a small eigenvalue gap.

The matrices were derived from the adjacency matrices of these graphs. We note for completeness that the specific instances of the directed graphs we used (the random k-out and the small gap graphs) were all strongly connected. This is not a sufficient condition for convergence but makes it very likely in the case of random graphs. Since our convention in this paper has been that the element A_{ij} describes the weight for network link (j, i)—so that matrix vector multiplication can be defined by sending messages along the outgoing (and not incoming) links—the adjacency matrices were first transposed. The first set of matrices consists of the transposed adjacency matrices. The second set contains the column normalized versions of the matrices in the first set. The normalized versions are such that the weights of the outgoing links sum up to one for all nodes, therefore these matrices describe random walks on the graphs.

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) determines the convergence speed of the power iteration [3], and thus it is a good indicator of the speed of our method as well. For a small gap, convergence is slow. With a zero gap, the power iteration does not converge at all. Since all matrix elements are real and non-negative, the largest eigenvalue is real and non-negative as well.

	random k-out		scale free		small gap	
	normalized	unnormalized	normalized	unnormalized	normalized	unnormalized
λ_1	1.0000	8.0000	1.0000	1.3981	1.0000	4.1938
$\|\lambda_2\|$	0.3573	2.8345	0.8373	1.1737	0.9754	3.9976

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

5.3 Results

Each experiment was carried out as follows. First, each node i was initialized to have $w_i = 1$ and $b_i = 1$. In the simulations we started each node at a random time within the first Δ time units counted from the first snapshot time t_0.

Two versions of the method were run for each problem. In the first, we do not apply the algorithm described in Section 4. In this case we expect the vector norm to converge to a previously unknown value, given that we do not change the underlying matrices in these experiments. In the second version we do apply vector normalization. In particular, we apply the *maximum* of the vector for this purpose, and therefore we expect the maximum to converge to one.

The evaluation metrics were as follows. We first computed the correct dominant eigenvector ($\mathbf{x}$) using a centralized algorithm. Following general practice in matrix computations, we measured the angle of the actual approximation and the correct vector to characterize convergence. That is, we computed the cosine of the angle $\cos\alpha^{(i)} = \|\mathbf{x}^T\mathbf{w}^{(\mathbf{i})}\|_2/\|\mathbf{x}\|_2/\|\mathbf{w}^{(\mathbf{i})}\|_2$, and used the angle $\alpha^{(i)}$ as a metric, which tends to zero during the iteration, as i increases. As a second metric, we measured the maximum of the vector elements to verify normalization.

The failure scenarios involved varying message drop probabilities, and varying message delays. Message drop was modeled by dropping all messages with a given probability, and message delay and delay jitter was modeled by drawing a delay value from a specified interval uniformly, for all messages. Obviously, these settings were applied for all messages sent by any of the components of the protocol equally.

Figure 3 shows the results of the experiments. First of all, even the more moderate failure scenario can be considered pessimistic, not to mention the more severe scenario. This is because in the application scenarios we envision, the interval Δ can be rather long, in the range of ten to thirty seconds, so a delay of 10% of Δ is already large. Most importantly, from the point of view of the averaging and maximum finding protocols, that have a much shorter cycle length of $\Delta_r = \Delta/30$, these delay values are extreme.

From the experiments we can conclude that when the vector norm is not controlled explicitly, then convergence is fast, comparable to that of the centralized power iteration. Our preliminary experiments (not shown) suggest that message delay has virtually no effect on the convergence results, when $P_{drop} = 0$. Higher drop rates slow down convergence but do not change its characteristics significantly.

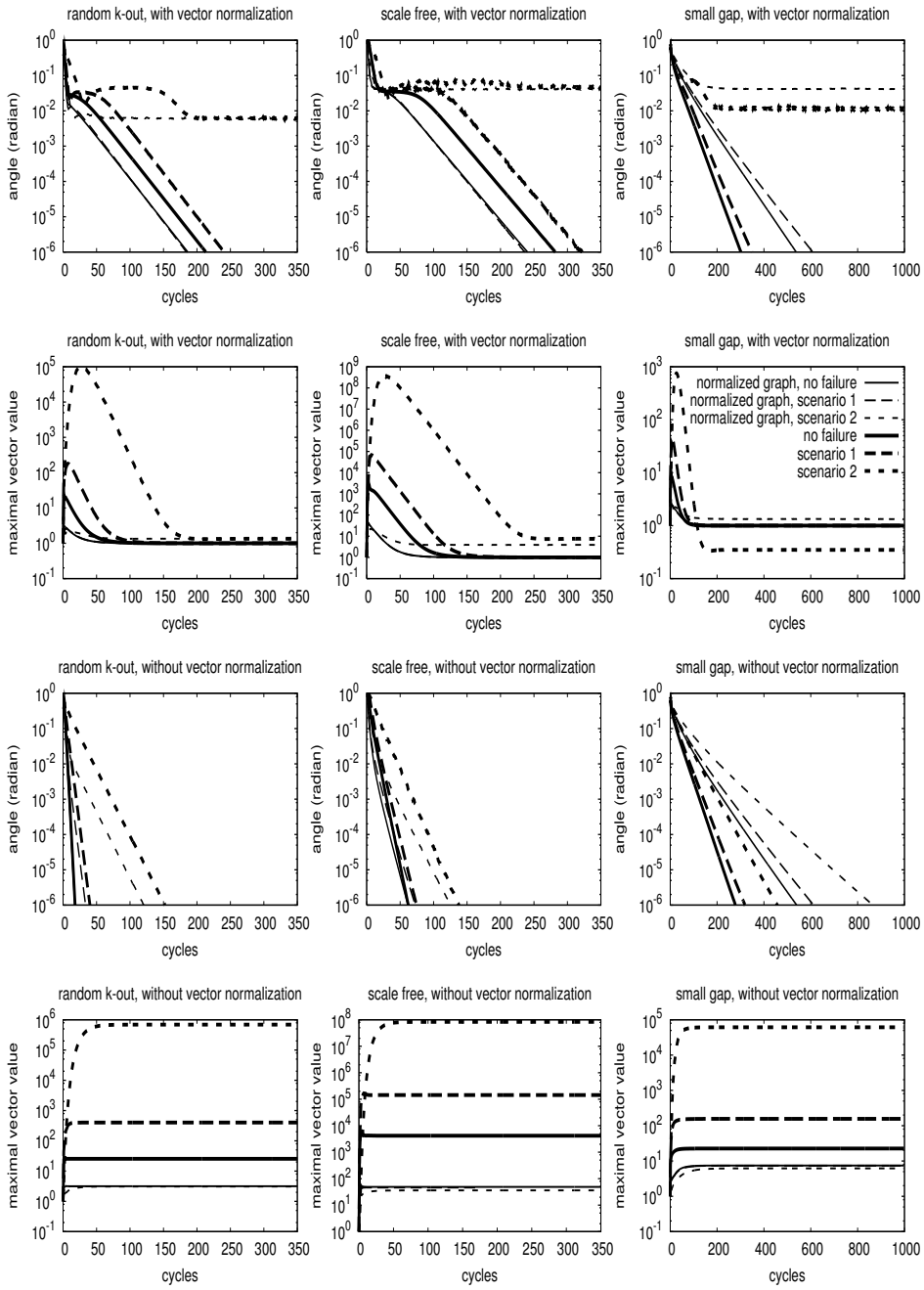

Fig. 3. Simulation results. Scenario 1 involves $P_{drop} = 0.1$, and a random message delay drawn from $[0, \Delta/10]$ uniformly. In scenario 2, $P_{drop} = 0.3$ and the message delay is drawn from $[\Delta/10, \Delta/2]$.

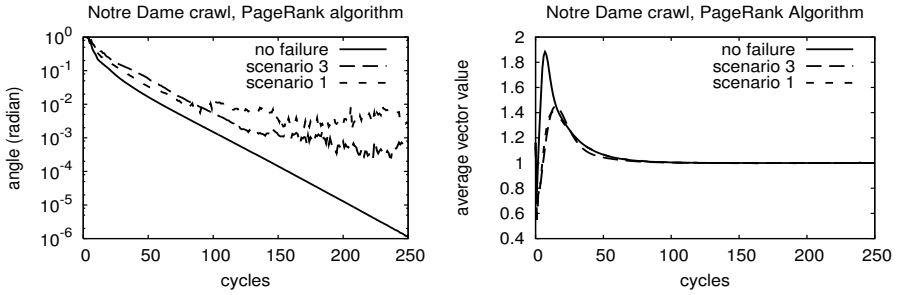

Fig. 4. Simulation results with the PageRank algorithm. Scenario 1 involves $P_{drop} = 0.1$, and a random message delay drawn from $[0, \Delta/10]$ uniformly. In scenario 3, $P_{drop} = 0$ and the message delay is the same as in scenario 1.

When we do apply vector normalization, convergence slows down somewhat due to the interference of vector normalization and asynchronous iteration. In the extreme failure scenario we don't achieve full convergence. The reason is that the extremely high delay and message drop rate prevents the propagation of the current maximum of the vector to all nodes during the interval Δ, and so different nodes might normalize with a different value. As a side effect, the maximum does not converge to one, and there is a constant noise factor in the approximation of the eigenvector. However, in the less severe, but still pessimistic scenario we do achieve convergence.

5.4 PageRank on WWW Crawl Data

As a more realistic case study, we tested our method on the same dataset used in [2], 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. On this dataset we executed the PageRank algorithm, as described in Section 4. The weight of the random surfer operator was $\epsilon = 0.2$. This way, for the complete linear operator of the PageRank algorithm, we have $\lambda_1 = 0.84648, \lambda_2 = 0.8$.

The results of the method are shown in Figure 4 in various failure scenarios. We can observe that the protocol is now more sensitive to failure than in the case of the previous experiments, although the achieved accuracy is still satisfactory (note the logarithmic scale of the plots). The reason is that in this case the vector average is used for controlling the norm of the vector, that is, it is guaranteed that the average of the vector stays one. The average is used to implement the random surfer operator as well. However, the calculation of the average is more sensitive to failure than the calculation of the maximum. This way, the approximation of the actual average of the vector has a small noise factor, that is inherited by the approximation of the ranks.

We can also note that the protocol scales well: the network examined here is two orders of magnitude larger than the previously examined networks, while convergence speed is still similar.

6 Related Work

Due to its importance, the distributed calculation of the dominant eigenvalue of large matrices has an extensive literature. In the area of parallel and cluster computing, the focus has largely been the optimization of existing, often iterative, methods on parallel computers and clusters (for a summary, see [3]). Such optimizations include partitioning; for example, different parts of the vector can be freely assigned to different processors in order to minimize message exchange and to maximize speedup. Besides, due to the reliable computing platform, synchronization can be efficiently implemented. This model is radically different from ours: in our case the assignment is fixed and given *a priory*, and the main goal is to achieve robustness to high rates of message delivery failures.

Asynchronous protocols have also been proposed for implementing iterative methods, and important convergence results are available as well (see [6] for a summary). These protocols are extremely fault tolerant and also efficient, but so far no algorithms are known that can deal with the case when the dominant eigenvalue is different from one. This introduces a certain sensitivity to dynamic environments even if $\lambda_1 \approx 1$, besides, many interesting applications where $\lambda_1 \neq 1$ cannot be tackled, for example [4].

Finally, in the context of P2P systems the main focus is on distributed PageRank implementations, where in all cases $\lambda_1 = 1$ is assumed, for example, [14, 16, 17]. The EigenTrust protocol in [9] also applies a similar implementation, but the authors assume all values are updated in each round, presumably unaware of the advantages of the long existing asynchronous version of the protocol, and thereby offering a rather fragile algorithm.

7 Conclusions

In this paper we have addressed the problem of designing a fully distributed and robust algorithm for finding the dominant eigenvector of large and sparse matrices, that are represented as weights of links between nodes of a network. Our contribution can be summarized as follows. First of all, our algorithm does not require the dominant eigenvalue to be one. This is an important feature even if the problem involves a dominant eigenvalue of one (like PageRank does). In PageRank, sophisticated techniques for "fixing" the graph are required to make sure the dominant eigenvalue is one, which are not needed in our case, as we demonstrated. Besides, the protocol opens the door for applications where the dominant eigenvalue is known to be different from one [4].

Second, the norm of the approximation of the dominant eigenvector can be controlled as well. In other words, in addition to guaranteeing that the norm of the vector converges to a finite value, we can define this value explicitly using

an additional gossip-component. This also means that the algorithm can be run indefinitely in a continuously changing environment.

Finally, we demonstrated the robustness of the algorithm through event-based simulation experiments, both on artificially generated graphs and on web-crawl data.

References

1. R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, January 2002.
2. R. Albert, H. Jeong, and A.-L. Barabási. Diameter of the world wide web. *Nature*, 401:130–131, 1999.
3. Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors. *Templates for the Solution of Algebraic Eigenvalue Problems: a Practical Guide*. SIAM, Philadelphia, 2000.
4. M. Burgess, G. Canright, and K. Engø-Monsen. Importance-ranking functions derived from the eigenvectors of directed graphs. Technical Report DELIS-TR-0325, DELIS Project, 2006.
5. F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proc. SIGCOMM 2004*. ACM Press, 2004.
6. A. Frommer and D. B. Szyld. On asynchronous iterations. *Journal of Computational and Applied Mathematics*, 123(1-2):201–216, 2000.
7. M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In H.-A. Jacobsen, editor, *Middleware*, LNCS 3231, pages 79–98. Springer, 2004.
8. M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, August 2005.
9. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proc. WWW*. ACM, 2003.
10. D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proc. FOCS'03*, pages 482–491. IEEE Computer Society, 2003.
11. Y. Koren. On spectral graph drawing. In *Proc. COCOON'03*, number 2697 in LNCS, pages 496–508. Springer, 2003.
12. B. Lubachevsky and D. Mitra. A chaotic asynchronous algorithm for computing the fixed point of a nonnegative matrix of unit radius. *J. of the ACM*, 33(1):130–150, 1986.
13. L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.
14. J. X. Parreira, D. Donato, S. Michel, and G. Weikum. Efficient and decentralized PageRank approximation in a peer-to-peer web search network. In *Proc. VLDB*, pages 415–426, 2006.
15. PeerSim. http://peersim.sourceforge.net/.
16. K. Sankaralingam, S. Sethumadhavan, and J. C. Browne. Distributed pagerank for p2p systems. In *Proc. HPDC-12*, pages 58–69, 2003.
17. S. Shi, J. Yu, G. Yang, and D. Wang. Distributed page ranking in structured p2p networks. In *Proc. ICPP03*, pages 179–186, October 2003.
18. D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.