Available online at www.sciencedirect.com

ScienceDirect

Journal of Computer and System Sciences 74 (2008) 70–83

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

www.elsevier.com/locate/jcss

A decentralized algorithm for spectral analysis

David Kempe [a,*,1], Frank McSherry [b]

[a] *Computer Science Department, University of Southern California, USA*
[b] *Microsoft Research SVC, USA*

Received 1 February 2005; received in revised form 1 March 2006

Available online 25 April 2007

Abstract

In many large network settings, such as computer networks, social networks, or hyperlinked text documents, much information can be obtained from the network's spectral properties. However, traditional centralized approaches for computing eigenvectors struggle with at least two obstacles: the data may be difficult to obtain (both due to technical reasons and because of privacy concerns), and the sheer size of the networks makes the computation expensive. A decentralized, distributed algorithm addresses both of these obstacles: it utilizes the computational power of all nodes in the network and their ability to communicate, thus speeding up the computation with the network size. And as each node knows its incident edges, the data collection problem is avoided as well.

Our main result is a simple decentralized algorithm for computing the top k eigenvectors of a symmetric weighted adjacency matrix, and a proof that it converges essentially in $O(\tau_{\text{mix}} \log^2 n)$ rounds of communication and computation, where τ_{mix} is the mixing time of a random walk on the network. An additional contribution of our work is a decentralized way of actually detecting convergence, and diagnosing the current error. Our protocol scales well, in that the amount of computation performed at any node in any one round, and the sizes of messages sent, depend linearly on the degree of the node, polynomially on k, but not at all on the (typically much larger) number n of nodes. To achieve independence of n, the coordinates of the computed eigenvectors are held locally by the nodes to which they correspond, enabling many eigenanalyses without distributing complete global state.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Eigenvectors; Spectral analysis; Decentralized algorithm; Markov Chain; Large networks

1. Introduction

One of the most stunning trends of recent years has been the emergence of very large-scale networks. A major driving force behind this development has been the growth and wide-spread usage of the Internet. The structure of hosts and routers—in itself a large network—has facilitated the growth of the World Wide Web, consisting of billions of web pages linking to each other. This in turn has allowed or helped users to take advantage of services such as Instant Messaging (IM) or various sites such as Friendster, Orkut, or BuddyZoo to explore their current

[*] Corresponding author.
E-mail addresses: dkempe@usc.edu (D. Kempe), mcsherry@microsoft.com (F. McSherry).
[1] Work done while the author was supported by an NSF Graduate Fellowship.

0022-0000/$ – see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2007.04.014

social network and develop new social ties. Beyond Internet-based applications, a large amount of effort is now being focused on structures and applications of decentralized Peer-to-Peer (P2P) networks [1–4].

In all of these cases, the (weighted) network structure contains much information that could be beneficial to the nodes. For the router graph of the Internet or P2P networks, we may be interested in sparse cuts, as these may lead to network traffic congestion, or—in the extreme case—network partitioning. For linked web pages, most useful measures of relevance or relatedness (such as PageRank [5] or hub and authority weights [6]) are defined in terms of the eigenvectors of the network's adjacency matrix. In social networks, individuals may be interested in questions such as: Is there a natural clustering among my friends? Which two of my friends are most likely to be compatible, and should therefore be introduced? Which of my friends belong to social circles different from mine, and could therefore introduce me to new people?

For all of the above questions, good solutions can be obtained by spectral analysis of the underlying graph structure, as nodes on different sides of a sparse cut tend to have very different entries in the second eigenvector [7]. In addition, several recent results have also shown how to use spectral techniques for clustering [8–10], characterization [6,11,12], and recommendation/prediction [13].

When trying to apply these techniques to the large network settings described above, one encounters several difficulties. First and foremost, the very size of the networks may be prohibitively large for (efficient, but superlinear) spectral algorithms. Second, the actual network data may be difficult to collect. This may be a result of either technological obstacles (such as implementing an efficient web crawler), or of privacy concerns: users of a P2P network may want to keep their identity concealed, and users of IM or other social network systems may be reluctant to share their social connections.

A solution to both of these problems is to perform the computation in the network. This leverages the computational power of the individual nodes. At the same time, nodes only communicate and share data with their neighbors in the network, which may go a long way toward alleviating privacy concerns. Last but not least, a decentralized design may be more desirable solely on the grounds that it does not offer a single point of failure, and the system as a whole can continue to function even when many of the nodes fail.

1.1. Our contributions

We present a decentralized algorithm for computing eigenvectors of a symmetric matrix, and singular vectors of arbitrary matrices (corresponding to the adjacency matrices of undirected respectively directed graphs). We assume that associated with each edge of the network is a weight a_{ij}, which is known to both endpoints. This weight may be the bandwidth available between two machines, the number of links between two web pages, or an estimate of the strength of a social tie between two individuals.

Our algorithm considers each node of the network as an independent computational entity that can communicate with all of its neighbors. (This assumption is certainly warranted for social networks, P2P networks, or the autonomous systems in the Internet; it can also be simulated fairly easily for web graphs.) The sizes of messages passed between nodes, as well as the computation performed at each node, are nominal; when computing the k principal eigenvectors (or singular vectors), they are $O(k^3)$ in each round. The number of rounds to achieve error ϵ is $O(\log^2(n/\epsilon) \cdot \tau_{\text{mix}}(G))$, where $\tau_{\text{mix}}(G)$ denotes the mixing time of the random walk on the network G. As many of the above-mentioned networks have good expansion (either by design or empirical observation), this time will essentially be logarithmic in the number n of nodes, hence exponentially faster than the centralized algorithms for spectral analysis.

Our algorithm is based on a decentralized implementation of *Orthogonal Iteration*, a simple method for computing eigenvectors. Let $A = (a_{ij})$ denote the weighted adjacency matrix of the graph under consideration. In the Orthogonal Iteration method, k random vectors are chosen initially. In each iteration, all vectors are first multiplied by A; then, the resulting vectors are orthonormalized, and serve as the starting vectors for the next iteration. We show how to approximately implement both the multiplication and orthogonalization phases of an iteration in a decentralized fashion. As this implementation introduces additional errors, we analyze how errors propagate through future iterations.

Our analysis of a single orthogonal iteration shows that the error with respect to a centralized implementation drops to ϵ within time $O(\log \frac{1}{\epsilon} \cdot \tau_{\text{mix}})$. One feature of our approach is that nodes need not (and usually do not) know the entire network structure, and in particular will usually not know the value of τ_{mix}. Hence, we also show how nodes can detect convergence to within error ϵ in a decentralized way without more than a constant factor in overhead.

1.2. Applications

We elaborate briefly on some of the previously mentioned potential applications of spectral methods in a decentralized setting. We restrict our discussion to applications where nodes can make decisions or draw inferences locally, by comparing their own k-tuples to those of their neighbors. This precludes more global uses of eigenvectors, including the prediction of non-existing links (except perhaps when the two nodes are at distance 2, and the comparison could thus be performed by a common neighbor).

1.2.1. Network engineering

One of the main challenges in designing and maintaining networks is to ensure a high bandwidth for concurrent flows between arbitrary sources and sinks. This usually involves detecting bottlenecks, and removing them by increasing the bandwidth along bottleneck edges, or by adding more edges. Bottlenecks can often be detected by considering the principal eigenvectors of the network's adjacency matrix, as the components of nodes on different sides of a sparse cut tend to have different signs in these eigenvectors.

More formally, by combining the theorem of Leighton and Rao [14], on the maximum amount f^* of flow that can be concurrently routed between source/sink pairs (s_i, t_i), with results relating the expansion of a graph to the second-largest eigenvector of its Laplacian matrix L, maximum concurrent flow and eigenvalues relate as follows: $O(\frac{n\lambda_{n-1}(L)}{\log n}) \leqslant f^* \leqslant O(n\sqrt{\lambda_{n-1}(L)})$. Hence, to increase the amount of flow that can be concurrently sent, it suffices to increase $\lambda_{n-1}(L)$—or equivalently, to decrease the second-largest eigenvalue of $I - L$.

One approach to attempt to minimize $\lambda_2(I - L)$ is to consider the eigenvalue characterization $\lambda_2(I - L) = \max_{\vec{x}\perp\vec{x}_1} \frac{\vec{x}^T(I-L)\vec{x}}{\vec{x}^T\vec{x}}$, where $\vec{x}_1$ denotes the principal eigenvector of $I - L$. The second eigenvector is the $\vec{x}$ attaining the maximum. By increasing a_{ij} for nodes i, j with opposite signs in the vector $\vec{x}$ (and decreasing a_{ij} for nodes with equal signs), the ratio on the right-hand side is reduced, corresponding to the above intuition that the bandwidth should be increased between nodes with different signs in their eigenvector entries. Notice that this will not necessarily reduce $\lambda_2(I - L)$, as the maximum may be attained by a different vector $\vec{x}$ now. However, at worst, this is a good practical heuristic; in fact, we conjecture that by extending this technique to multiple eigenvectors, $\lambda_2(I - L)$ can be provably reduced. As all non-zero entries of $I - L$ coincide with non-zero entries of A, they correspond to edges of the network, and the computation can thus be performed by our decentralized algorithm.

1.2.2. Social engineering and weak ties

The importance of spectral methods in the analysis of networks in general, and social networks in particular, results from the fact that they assign to each point a vector in $\mathbb{R}^k$ for some small k, and that proximity in this space $\mathbb{R}^k$ corresponds to a similarity of the two nodes in terms of their positions within the network. For social networks, this means that individuals with similar (or compatible) social circles will be mapped to close points.

A first application of this observation would lie in link prediction or "social engineering": introducing individuals who do not know each other (do not share an edge), even though their mappings into $\mathbb{R}^k$ are close. This requires the existence of a node to observe the proximity, for instance a "common friend" (a node adjacent to both); a more sophisticated solution might let a node broadcast its k-dimensional vector to other nodes, and let them choose to contact this possibly compatible node (with small inner product of the two vectors [13]).

A second, and perhaps more interesting, application is the detection of weak ties. Sociologists have long distinguished between "strong" and "weak" social ties—see the seminal paper by Granovetter [15] on the subject. The notions of weak and strong ties refer to the frequency of interaction between individuals, but frequently coincide with ties between individuals of different respectively similar social circles. The distinction of social ties into different classes is important in that [15] reports that a disproportionately large fraction of employment contracts are the result of weak tie interaction. One may expect similar phenomena for other aspects of life. An individual may therefore want to discover which of his ties are weak, in order to seek introduction to potential employers, new friends, etc.

Using the mapping into $\mathbb{R}^k$, we can define a precise notion of a weak tie, by comparing the distance between the two endpoints of an edge. (A weak tie between individuals will thus correspond intuitively to adjacent nodes on different sides of a sparse cut in the sense discussed above.) What is more, the two endpoints themselves can determine whether their tie is weak, and act accordingly.

1.3. Related work

For a general introduction to spectral techniques, see [16]. There has been a large body of work on parallelizing matrix operations—see for instance [17] for a comprehensive overview. These approaches assume a fixed topology of the parallel computer which is unrelated to the matrix to be decomposed; our approach, on the other hand, has a network of processors analyze its own adjacency matrix.

Our work relates to other recent work that tries to infer global properties of a graph by simple local processes on it. In particular, Benjamini and Lovász [18] show how to determine the genus of a graph from a simple random walk-style process.

Our implementation of Orthogonal Iteration is based on a recent decentralized protocol for computing aggregate data in networks, due to Kempe, Dobra, and Gehrke [19]. Here, we show how to extend the ideas to compute significantly more complex properties of the network itself.

Both the above-mentioned paper [19] and our paper draw connections between computing the sum or average of numbers, and the mixing speed of random walks. In recent work, Boyd et al. [20,21] have made this connection even more explicit, showing that the two are essentially identical under additional assumptions.

The equivalence between averaging and Markov Chains suggests that in order for these decentralized algorithms to be efficient, they should use a Markov Chain with as small mixing time as possible. Boyd, Diaconis, and Xiao [22] show that the fastest mixing Markov Chain can be computed in polynomial time, using semi-definite programming. For the special case of random geometric graphs (which are reasonable models for sensor networks), Boyd et al. [23] show that the fastest mixing Markov Chain mixes at most by a constant factor faster than the random walk, in time $\Theta(r^{-2}\log n)$ (where all n points are randomly placed in a unit square, and considered adjacent if they are within distance r). In essence, this shows that slow convergence is inherent in decentralized averaging algorithms on random geometric graphs.

2. The algorithm

We consider the problem of computing the eigenvectors of a weighted graph, where the computation is performed at the nodes in the graph. Each node has access to the weights on incident edges, and is able to communicate along edges of non-zero weight, exchanging messages of small size. The goal is for each node to compute its value in each of k principal eigenvectors. For simplicity, we will assume that each node can perform an amount of computation and communication proportional to its degree in each round.

2.1. Orthogonal iteration

Our algorithm emulates the behavior of Orthogonal Iteration, a simple algorithm for computing the top k eigenvectors of a (graph adjacency) matrix $A = (a_{ij})_{i,j}$.

```
Algorithm 1. Orthogonal Iteration ($A$)

1: Choose a random $n \times k$ matrix $Q$.
2: loop
3:     Let $V = AQ$.
4:     Let $Q = \text{Orthonormalize}(V)$.
5: end loop
6: Return $Q$ as the eigenvectors.
```

Once the eigenvectors have been computed, it is easy to obtain from them the projections of each node onto the eigenspace, as it is captured by the rows of V.

Orthogonal Iteration converges quickly: the error in the approximation to the true Q decreases exponentially in the number t of iterations, as characterized by Theorem 3 below.

We adapt Orthogonal Iteration to a decentralized environment. Each node i takes full responsibility for the rows of V and Q associated with it, denoted V_i and Q_i. The choice of a random matrix is easy to implement in a decentralized fashion. Similarly, when the matrix Q is already known, then $V = AQ$ can be computed locally: each node j sends its row Q_j to all of its neighbors; then, node i can compute its row V_i as a linear combination (with coefficients a_{ij}) of all vectors Q_j received from its neighbors j. The key aspect of the decentralization is therefore how to perform the orthonormalization of V in a decentralized way.

2.2. Decentralized orthonormalization

The orthonormalization in Orthogonal Iteration is typically performed by computing the QR factorization of V, i.e. matrices Q, R such that $V = QR$, the k columns of Q are orthonormal, and the $k \times k$ matrix R is upper triangular. Orthonormalization is thus performed by applying R^{-1} to V, yielding Q. If each node had access to R, each could locally compute the inverse R^{-1} and apply it to its copy of V_i. The resulting collection of vectors would then form an orthonormal Q.

However, it is not obvious how to compute R directly. Therefore, we use the fact that if $K = V^T V$, then R is the unique $k \times k$ upper triangular matrix with $K = R^T R$. This holds because if Q is orthonormal, then $Q^T Q$ is the identity matrix, so

$$K = V^T V = R^T Q^T Q R = R^T R.$$

(Here, we are using the fact that the QR-factorization $V = QR$ and the Cholesky factorization $K = R^T R$ are both unique.) Once each node i has access to the $k \times k$ matrix K, each can compute the Cholesky factorization $K = R^T R$ locally, invert R, and apply R^{-1} to its row V_i.

Unfortunately, it is unclear how to provide each node with the precise matrix K. Instead, each node computes an approximation to K. To see how, observe that $K = \sum_i V_i^T V_i$. Each node i is capable of producing $K^{(i)} = V_i^T V_i$ locally, and if we can, in a decentralized manner, sum up these matrices, each node can obtain a copy of K.

In order to compute this sum of matrices in a decentralized fashion, we employ a technique proposed in [19]: the idea is to have the value (or, in this case, matrix) from each node perform a deterministic simulation of a random walk. Once this "random walk" has mixed well, each node i will hold roughly a π_i fraction of the value from each other node j (where π_i denotes the stationary probability for node i of the random walk). Hence, if we also compute π_i and divide by it, then each node calculates approximately the sum of all values. (For matrices, all of this computation applies entry-wise.) Hence, let $B = (b_{ij})$ be an arbitrary stochastic matrix, such that the corresponding Markov Chain is ergodic and reversible,[2] and $b_{ij} = 0$ whenever there is no edge from i to j in the network.[3] Then, the algorithm for summing is given by Algorithm 2 below.

Algorithm 2. Push-Sum $(B, (K^{(i)}))$

```
1: One node $\hat{\imath}$ starts with $w_{\hat{\imath}} = 1$, all others with $w_i = 0$.
2: All nodes set $S_i = K^{(i)}$.
3: **loop**
4:     Set $S_i = \sum_{j \in N(i)} b_{ji} S_j$
5:     Set $w_i = \sum_{j \in N(i)} b_{ji} w_j$
6: **end loop**
7: Return $\frac{S_i}{w_i}$.
```

At each node, the ratio $\frac{S_i}{w_i}$ converges to the sum $\sum_i K^{(i)}$ at essentially the same speed as the Markov Chain defined by B converges to its stationary distribution. The exact bound and analysis are given as Theorem 5.

Combining this orthonormalization process with the decentralized computation of AV, we obtain the following decentralized algorithm for eigencomputation, as executed at each node i:

[2] Recall that a Markov Chain is called *reversible* if it satisfies the *detailed balance condition* $\pi_i B_{ij} = \pi_j B_{ji}$ for all i and j.

[3] A natural choice is the random walk on the underlying network, i.e. $b_{ij} = \frac{1}{\deg(i)}$. However, our results hold in greater generality, and the additional flexibility may be useful in practice when the random walk on the network itself does not mix well.

Algorithm 3. DecentralizedOI (k)

```
1: Choose a random $k$-dimensional vector $Q_i$.
2: loop
3:     Set $V_i = \sum_{j \in N(i)} a_{ij} Q_j$.
4:     Compute $K^{(i)} = V_i^T V_i$.
5:     Set $K = \text{Push-Sum}(B, K^{(i)})$.
6:     Compute the Cholesky factorization $K = R^T R$.
7:     Set $Q_i = V_i R^{-1}$.
8: end loop
9: Return $Q_i$ as the $i$th component of each eigenvector.
```

We have been fairly casual about the number of iterations that should occur, and how a common consensus on this number is achieved by the nodes. One simplistic approach is to have the initiator specify a number of iterations, and keep this amount fixed throughout the execution. A more detailed analysis, showing how nodes can estimate the approximation error in a decentralized way, is given in Section 3.3.

The main technical result of our paper is a careful analysis of the convergence speed and properties of the DecentralizedOI Algorithm 3. In order to state the convergence properties formally, we describe the subspaces in terms of projection matrices, instead of a specific set of basis vectors. This simplifies the presentation by avoiding technical issues with ordering and rotations among the basis vectors. For a subspace S with orthonormal basis $\{\vec{b}_1, \ldots, \vec{b}_k\}$, the *projection matrix* onto S is $P_S = \sum_i \vec{b}_i \vec{b}_i^T$. Our main theorem is then:

Theorem 1. *Let A be a symmetric matrix, and $\lambda_1, \lambda_2, \ldots$ its eigenvalues, such that $|\lambda_1| \geqslant |\lambda_2| \geqslant \ldots$. Let P_Q denote the projection onto the space spanned by the top k eigenvectors of A and let $P_{Q'}$ denote the projection onto the space spanned by the eigenvectors computed after t iterations of Decentralized Orthogonal Iteration.*

If DecentralizedOI runs Push-Sum for $\Omega(t\tau_{\text{mix}} \cdot \log(\frac{kc}{\epsilon} \cdot \|A\|_2))$ steps in each of its iterations, and $\|R^{-1}\|_2$ is consistently less than c, then with high probability,

$$\|P_Q - P_{Q'}\|_2 \leqslant O\left(\left|\frac{\lambda_{k+1}}{\lambda_k}\right|^t \cdot n\right) + 3\epsilon^{4t}.$$

Remark 2 *(Vector and matrix norm notation).* For any probability distribution $\vec{\mu}$, we write $\|\vec{x}\|_{p,\vec{\mu}} = (\sum_i |x_i|^p \cdot \mu_i)^{1/p}$, and $\|\vec{x}\|_{\infty,\vec{\mu}} = \max_i |x_i|$. When $\vec{\mu}$ is omitted, we mean the norm $\|\vec{x}\|_p = (\sum_i |x_i|^p)^{1/p}$.

For vector norms $\|\cdot\|_a$, $\|\cdot\|_b$, the matrix operator norm of a matrix A is defined as $\|A\|_{a\to b} = \max_{\|\vec{x}\|_a=1} \|A\vec{x}\|_b$. We most frequently use $\|A\|_2 := \|A\|_{2\to 2}$. In addition to the operator norms induced by L_p norms on vectors, we define the *Frobenius norm* of a matrix A as $\|A\|_F := (\sum_{i,j} a_{ij}^2)^{1/2}$. These two norms relate in the following useful ways: for any two matrices A, B, we have that $\|A\|_2 \leqslant \|A\|_F \leqslant \sqrt{\text{rank}(A)}\|A\|_2$, and $\|AB\|_F \leqslant \|A\|_2\|B\|_F$.

3. Analysis

In this section, we analyze the convergence properties of our decentralized algorithm, and prove Theorem 1. The proof must take into account two sources of error: (1) The Orthogonal Iteration algorithm itself does not produce an exact solution, but instead converges to the true eigenvectors, and (2) our decentralized implementation DecentralizedOI introduces additional error.

The convergence of Orthogonal Iteration itself has been analyzed extensively in the past (Theorem 8.2.2 in [24]); the relevant results are stated as Theorem 3.

Theorem 3. *Let P_Q describe the projection onto the space spanned by the top k eigenvectors of a symmetric matrix A, and let $P_{Q'}$ be the projection onto the space spanned by the approximate Q' obtained after t iterations of Orthogonal Iteration, starting from a uniformly random projection. With high probability,*

$$\|P_Q - P_{Q'}\|_2 \leqslant O\left(\left|\frac{\lambda_{k+1}}{\lambda_k}\right|^t \cdot n\right).$$

Remark 4. Interpreted, this theorem implies that the space found by orthogonal iteration is close to the true space, so the projections $V_i = AQ_i$ are nearly perfect. Furthermore, not many iterations are required to achieve good accuracy. To bring this error bound to $O(\epsilon)$, we need to perform $t = \log(\frac{n}{\epsilon})/\log(|\frac{\lambda_k}{\lambda_{k+1}}|)$ iterations.

Notice that the bound of Theorem 8.2.2 in [24] characterizes the error in terms of the tangent of the angle between the initial subspace and the span of the top k eigenvectors. Here, we bound the tangent by $O(n)$ with high probability, having started from a uniformly random initial subspace. The probability is not exponentially small, though one can make it so through repeated or parallel application of the algorithm, as is common in the centralized setting. If a better bound on the tangent of the angle between subspaces is available, e.g., if we are starting from a solution that is known a priori to be nearly accurate, the $O(n)$ term can be replaced by a better bound.

For analyzing the second type of error—introduced by the inaccurate computations of sums, we first analyze the error introduced by one iteration of Push-Sum in Section 3.1, and then analyze the propagation of such errors through multiple iterations of DecentralizedOI in Section 3.2.

3.1. Analysis of Push-Sum

We begin by analyzing the error introduced by using Push-Sum for adding matrices instead of obtaining accurate sums. We define the mixing time τ_{mix} of the Markov Chain associated with B in terms of the $\|\cdot\|_2$ norm, namely as the smallest t such that $\|\vec{e}_i^T B^t - \vec{\pi}^T\|_2 \leqslant \frac{1}{2}$ for all i. Then, we can prove the following theorem about the convergence speed of Push-Sum:

Theorem 5. *Let $S_{t,i}$ be the $k \times k$ matrix held by node i after the tth iteration of Push-Sum, $w_{t,i}$ its weight at that time, and S the correct matrix. Define $M = \sum_i |S_{0,i}|$ to be the matrix whose (r, c) entry is the sum of absolute values of the initial matrices $S_{0,i}$ at all nodes i. Then, for any ϵ, the approximation error is $\|\frac{S_{t,i}}{w_{t,i}} - S\|_F \leqslant \epsilon\|M\|_F$, after $t = O(\tau_{\text{mix}} \cdot \log\frac{1}{\epsilon})$ rounds.*

The proof of this theorem rests mainly on Lemma 6 below, relating the approximation quality for every single entry of the matrix to the convergence of B^t to the stationary distribution of B. In the formulation of the lemma, we are fixing a single entry (r, c) of all matrices involved. We write $x_i = K_{rc}^{(i)}$, and $s_{t,i} = (S_{t,i})_{rc}$.

Lemma 6. *Let t be such that $\|\frac{\vec{e}_j^T B^t - \vec{\pi}}{\vec{\pi}}\|_\infty \leqslant \frac{\epsilon}{2+\epsilon}$ for all j.*[4] *Then, for any node i, the approximation error $|\frac{s_{t,i}}{w_{t,i}} - \sum_j x_j|$ at time t is at most $\epsilon \sum_j |x_j|$.*

Proof. Let $\vec{s}_t$ and $\vec{w}_t$ denote the vector of all $s_{t,i}$ respectively $w_{t,i}$ values at time t. Thus, $\vec{s}_0 = \vec{x}$, and $\vec{w}_0 = \vec{e}_{\hat{\imath}}$, for the special node $\hat{\imath}$. Then, it follows immediately from the definition of Push-Sum that $\vec{s}_{t+1}^T = \vec{s}_t^T B$, and $\vec{w}_{t+1}^T = \vec{w}_t^T B$. By induction, we obtain that $\vec{s}_t^T = \vec{x}^T B^t = \sum_j x_j \cdot \vec{e}_j B^t$, and $\vec{w}_t^T = \vec{e}_{\hat{\imath}}^T B^t$.

Node i's estimate of the sum at time t is $\frac{s_{t,i}}{w_{t,i}} = \sum_j x_j \cdot \frac{(\vec{e}_j B^t)_i}{(\vec{e}_{\hat{\imath}} B^t)_i}$. Because both the numerator and denominator converge to π_i, the right-hand side converges to $\sum_j x_j$. Specifically, let t be such that $\|\frac{\vec{e}_j^T B^t - \vec{\pi}}{\vec{\pi}}\|_\infty \leqslant \frac{\epsilon}{2+\epsilon}$ for all j. Then, a straightforward calculation shows that $1 - \epsilon \leqslant \frac{(\vec{e}_j B^t)_i}{(\vec{e}_{\hat{\imath}} B^t)_i} \leqslant 1 + \epsilon$ for all i, j.

Finally, by a simple application of the Triangle Inequality, we obtain that $|\frac{s_{t,i}}{w_{t,i}} - \sum_j x_j| \leqslant \epsilon \sum_j |x_j|$, completing the proof. □

The lemma gives bounds on the error in terms of the mixing speed of the Markov Chain, as measured in the $\|\cdot\|_\infty$ norm. Most analysis of Markov Chains is done in terms of the $\|\cdot\|_{2,\vec{\pi}}$ norm, or the total variation distance. For this reason, we give the discrete time analogue of Lemma 2.4.6 from [25], which relates $\|\cdot\|_\infty$ and $\|\cdot\|_{2,\vec{\pi}}$ for reversible Markov Chains.

[4] When we write a fraction of vectors, we mean the vector whose entries are the component-wise fractions.

Lemma 7. *Let B be a stochastic matrix whose associated Markov Chain is ergodic and reversible, with stationary probability $\vec{\pi}$. Then, for any time t, we have that* $\max_i \|\frac{\vec{e}_i^T B^{2t} - \vec{\pi}^T}{\vec{\pi}}\|_\infty \leqslant (\max_i \|\frac{\vec{e}_i^T B^t - \vec{\pi}^T}{\vec{\pi}}\|_{2,\vec{\pi}})^2$.

Proof. Substituting the definition of $\|\cdot\|_\infty$, and noticing that $\vec{\pi}^T = \vec{e}_i^T \vec{1}\vec{\pi}^T$, we can rewrite the quantity to be bounded as $\max_{i,j} \vec{e}_i^T (B^{2t} - \vec{1}\vec{\pi}^T)\frac{\vec{e}_j}{\vec{\pi}}$. Then, it is easy to see that this quantity is equal to $\max_{\|\vec{x}\|_{1,\vec{\pi}}=1} \|(B^{2t} - \vec{1}\vec{\pi}^T)\vec{x}\|_\infty$ (as the maximum in the second version is attained when only one coordinate of $\vec{x}$ is non-zero). This is, by definition, the operator norm $\|B^{2t} - \vec{1}\vec{\pi}^T\|_{1,\vec{\pi}\to\infty}$.

Because B (and hence B^t) is stochastic with stationary probability $\vec{\pi}$, we have that $\vec{\pi}^T \cdot B^t = \vec{\pi}^T$, and $B^t \cdot \vec{1} = \vec{1}$. Furthermore, the fact that $\vec{\pi}$ is a probability measure implies that $\vec{\pi}^T\vec{1} = 1$, so we obtain that $B^{2t} - \vec{1}\vec{\pi}^T = (B^t - \vec{1}\vec{\pi}^T)^2$. Now, applying the submultiplicativity of operator norms to $B^{2t} - \vec{1}\vec{\pi}^T$ gives us that $\|B^{2t} - \vec{1}\vec{\pi}^T\|_{1,\vec{\pi}\to\infty} \leqslant \|B^t - \vec{1}\vec{\pi}^T\|_{1,\vec{\pi}\to 2,\vec{\pi}} \cdot \|B^t - \vec{1}\vec{\pi}^T\|_{2,\vec{\pi}\to\infty}$.

For ease of notation, we write $K = B^t - \vec{1}\vec{\pi}^T$. Because B satisfies the detailed balance condition $\pi_i b_{ij} = \pi_j b_{ji}$ for all i, j, so does B^t (which can be shown by a simple inductive proof). Therefore, K also satisfies the detailed balance condition. Using the fact that $\|K\|_{1,\vec{\pi}\to 2,\vec{\pi}} = \max_{\|\vec{x}\|_{1,\vec{\pi}}=1, \|\vec{y}\|_{2,\vec{\pi}}=1} \sum_i (K\vec{x})_i y_i \pi_i$ (one direction of which is proved using the Cauchy–Schwartz Inequality, the other by appropriate choice of $\vec{x}$ and $\vec{y}$), the detailed balance property of K yields $\|K\|_{1,\vec{\pi}\to 2,\vec{\pi}} = \|K\|_{2,\vec{\pi}\to\infty}$. Finally, $\|K\|_{1,\vec{\pi}\to 2,\vec{\pi}} = \max_i (\sum_j \frac{K_{ij}^2}{\pi_j})^{1/2} = \max_i \|\frac{\vec{e}_i^T B^t - \vec{\pi}^T}{\vec{\pi}}\|_{2,\vec{\pi}}$, again by the detailed balanced condition. □

By combining Lemma 6 and Lemma 7, we can prove Theorem 5.

Proof of Theorem 5. Given a desired approximation quality ϵ, we define $\epsilon' = \frac{\epsilon}{(2+\epsilon)}$. By definition of the mixing time τ_{mix}, the $\|\cdot\|_2$ distance at time τ_{mix} is at most $\|\vec{e}_i^T B^{\tau_{\text{mix}}} - \vec{\pi}^T\|_2 \leqslant \frac{1}{2}$ for any i. Therefore, by a simple geometric convergence argument, at time $t = O(\log \frac{1}{\sqrt{\epsilon'}}\tau_{\text{mix}}) = O(\log \frac{1}{\epsilon}\tau_{\text{mix}})$, the error is at most $\|\vec{e}_i^T B^t - \vec{\pi}^T\|_2 \leqslant \sqrt{\epsilon'}$, for any i.

By Lemma 7, $\max_i \|\frac{\vec{e}_i^T B^{2t} - \vec{\pi}^T}{\vec{\pi}}\|_\infty \leqslant \epsilon' = \frac{\epsilon}{2+\epsilon}$. For any node i and each (r, c) pair, Lemma 6 therefore shows that $|\frac{(S_{2t,i})_{rc}}{w_{2t,i}} - \sum_j x_j| \leqslant \epsilon \cdot \sum_j |x_j| = \epsilon m_{rc}$. Hence, we can bound the Frobenius norm

$$\|S_{2t,i} - S\|_F \leqslant \sqrt{\sum_{r,c} \epsilon^2 \sum_j m_{rc}^2} = \epsilon \|M\|_F,$$

completing the proof. □

3.2. Error of Orthogonal Iteration

The more challenging part is to analyze the effect of the errors introduced by Push-Sum on the aggregation. In Section 3.1, we showed that the error for each entry of the matrix K at each node i drops exponentially in the number of steps that Push-Sum is run. Still, after any finite number of steps, each node i is using a (different) approximation $\widehat{K}_i$ to the correct matrix K, from which it computes $\widehat{R}_i^{-1}$ and then its new vector Q_i. We therefore need to analyze the effects that the error introduced into the matrix $\widehat{K}_i$ will have on future (approximate) iterations, and show that it does not hinder convergence. Specifically, we want to know how many iterations of Push-Sum need to be run to make the error so small that even the accumulation over the iterations of Orthogonal Iteration keeps the total error bounded by ϵ.

In order to bound the growth of error for the decentralized Orthogonal Iteration algorithm, we first analyze the effects of a single iteration. Recall that a single iteration, in the version that we use to decentralize, looks as follows: It starts with an orthonormal matrix Q, determines $V = AQ$ and $K = V^T V$, and from this computes a Cholesky factorization $K = R^T R$, where R is a $k \times k$ matrix. Finally, the output of the iteration is $Q' = VR^{-1}$, which is used as input for the next iteration.

The decentralized implementation will start from a matrix $\widehat{Q}$ which is perturbed due to approximation errors from previous iterations. The network computes $\widehat{V} = A\widehat{Q}$, and we can hence define $\widehat{K} = \widehat{V}^T\widehat{V}$. However, due to the approximate nature of Push-Sum, node i will not use $\widehat{K}$, but instead use a matrix $\widehat{K}_i = \widehat{K} + E_i$, for some error matrix E_i. Node i then computes $\widehat{R}_i$ such that $\widehat{K}_i = \widehat{R}_i^T \widehat{R}_i$, and applies $\widehat{R}_i^{-1}$ to its row $\widehat{V}_i$ of the matrix $\widehat{V}$. Hence, the resulting matrix $\widehat{Q}'$ has as its ith row the vector $\widehat{V}_i \widehat{R}_i^{-1}$.

Lemma 8. *Let Q and $\widehat{Q}$ be matrices where Q is orthonormal, satisfying the bound $\|Q - \widehat{Q}\|_F + \epsilon k \leqslant (2\|A\|_2 \|R^{-1}\|_2)^{-3}$. If Q' and $\widehat{Q}'$ are respectively the results of one step of Orthogonal Iteration applied to Q and Decentralized Orthogonal Iteration applied to $\widehat{Q}$, and the number of steps run in Push-Sum is $t = \Omega(\tau_{mix} \log(1/\epsilon))$, then*

$$\|Q' - \widehat{Q}'\|_F \leqslant \sqrt{k}\big(2\|A\|_2 \big\|R^{-1}\big\|_2\big)^4 \big(\|Q - \widehat{Q}\|_F + \epsilon k\big).$$

Proof. The proof consists of two parts: First, we apply perturbation results for the Cholesky decomposition and matrix inverse to derive a bound on $\|R^{-1} - \widehat{R}_i^{-1}\|_2$. Second, we analyze the effect of applying the (different) matrices $\widehat{R}_i^{-1}$ to the rows of $\widehat{V}$.

Throughout, we will be making repeated use of the relationship between the matrix norms of A, V, R, K. Because $V = Q'R$, and $V = AQ$, we can use the submultiplicativity of matrix norms, together with the fact that Q and Q' are orthonormal (and hence have norm 1) to observe that $\|V\|_F \leqslant \|R\|_F$, $\|V\|_2 \leqslant \|R\|_2$, and $\|V\|_2 \leqslant \|A\|_2$. Finally, because $K = R^T R$, its norms satisfy $\|K\|_2 = \|R\|_2^2$, and $\|K\|_F \leqslant \|R\|_F^2$.

The perturbation bound will have four steps, which respectively bound the terms $\|K - \widehat{K}\|_F$, then $\|K - \widehat{K}_i\|_F$, then $\|R - \widehat{R}_i\|_F$, then $\|R^{-1} - \widehat{R}_i^{-1}\|_2$. We start by applying the Triangle Inequality to $\|K - \widehat{K}\|_F$, followed by some rearrangement,

$$\begin{aligned}\|K - \widehat{K}\|_F &= \big\|V^T V - \widehat{V}^T \widehat{V}\big\|_F \leqslant \big\|V^T V - \widehat{V}^T V\big\|_F + \big\|\widehat{V}^T V - \widehat{V}^T \widehat{V}\big\|_F \\ &\leqslant \|V\|_2 \big\|V^T - \widehat{V}^T\big\|_F + \|\widehat{V}\|_2 \|V - \widehat{V}\|_F = \big(\|V\|_2 + \|\widehat{V}\|_2\big)\|V - \widehat{V}\|_F.\end{aligned}$$

Next, we want to bound the distance between K and the approximation $\widehat{K}_i$ used by node i. Because we chose the number t of iterations large enough, Theorem 5 implies that $\|\widehat{K}_i - \widehat{K}\|_F \leqslant \epsilon\|M\|_F$, where $M = (m_{rc})_{r,c}$ denotes the matrix with entries $m_{rc} = \sum_i |(\widehat{V}_i^T \widehat{V}_i)_{rc}|$. Applying the Cauchy–Schwartz Inequality after expanding the definition of $\|\cdot\|_F$ bounds $\|M\|_F \leqslant \|\widehat{V}\|_F^2$, so

$$\|K - \widehat{K}_i\|_F \leqslant \|K - \widehat{K}\|_F + \|\widehat{K} - \widehat{K}_i\|_F \leqslant \big(\|V\|_2 + \|\widehat{V}\|_2\big)\|V - \widehat{V}\|_F + \epsilon\|\widehat{V}\|_F^2 .$$

We apply two well-known theorems to bound the propagation of errors in the Cholesky factorization and matrix inversion steps. First, a theorem by Stewart [26] states that if $K = R^T R$ and $\widehat{K} = \widehat{R}^T \widehat{R}$ are Cholesky factorizations of symmetric matrices, then $\|R - \widehat{R}\|_F \leqslant \|K^{-1}\|_2 \|R\|_2 \|\widehat{K} - K\|_F$. Applying this theorem to our setting, using $\|R\|_2 = \|V\|_2$

$$\|R - \widehat{R}_i\|_F \leqslant \|K^{-1}\|_2 \|V\|_2 \big(\big(\|V\|_2 + \|\widehat{V}\|_2\big)\|V - \widehat{V}\|_F + \epsilon\|\widehat{V}\|_F^2\big). \tag{1}$$

Now is a good time to reduce these terms somewhat. Recall that because $\widehat{V} = V + A(\widehat{Q} - Q)$, we have that

$$\|V - \widehat{V}\|_F \leqslant \|A\|_2 \|Q - \widehat{Q}\|_F \quad \text{and} \quad \|\widehat{V}\|_2 \leqslant \|V\|_2 + \|A\|_2 \|Q - \widehat{Q}\|_2.$$

Recalling our assumption that $\|Q - \widehat{Q}\|_F \leqslant (2\|A\|_2 \|R^{-1}\|_2)^{-3}$, and noting that by submultiplicativity $\|A\|_2 \|R^{-1}\|_2 \geqslant \|AQR^{-1}\|_2 = \|Q'\|_2 = 1$, we have

$$\|V - \widehat{V}\|_F \leqslant \|A\|_2/8 \quad \text{and} \quad \|\widehat{V}\|_2 \leqslant \|V\|_2 + \|A\|_2/8.$$

Introducing these bounds into (1), as well as $\|V\|_2 \leqslant \|A\|_2$ and $\|\widehat{V}\|_F^2 \leqslant k\|\widehat{V}\|_2^2$, followed by a sequence of rearrangement of terms, we bound

$$\begin{aligned}\|R - \widehat{R}_i\|_F &\leqslant \big\|K^{-1}\big\|_2 \|A\|_2^3 \big((17/8)\|Q - \widehat{Q}\|_F + \epsilon k(9/8)^2\big) \\ &\leqslant (17/8)\big\|K^{-1}\big\|_2 \|A\|_2^3 \big(\|Q - \widehat{Q}\|_F + \epsilon k\big).\end{aligned}$$

As $\|K^{-1}\|_2 = \|R^{-1}\|_2^2$, and using the lemma's assumption on $\|Q - \widehat{Q}\|_F + \epsilon k$,

$$\|R - \widehat{R}_i\|_F \leqslant (17/64)\big\|R^{-1}\big\|_2^{-1}.$$

As the final step in our perturbation bounds, we apply Wedin's Theorem [27], which states that for non-singular matrices $R, \widehat{R}_i$,

$$\|R^{-1} - \widehat{R}_i^{-1}\|_2 \leqslant \frac{1+\sqrt{5}}{2}\|R - \widehat{R}_i\|_2 \max\{\|R^{-1}\|_2^2, \|\widehat{R}_i^{-1}\|_2^2\}.$$

To bound $\|\widehat{R}_i^{-1}\|_2$, we use a perturbation bound on the singular values of a matrix: $\sigma_k(R) - \sigma_k(\widehat{R}_i) \leqslant \|R - \widehat{R}_i\|_2$. As $\sigma_k(R) = \|R^{-1}\|_2^{-1}$ and $\sigma_k(\widehat{R}_i) = \|\widehat{R}_i^{-1}\|_2^{-1}$, we have that

$$\|R^{-1}\|_2^{-1} - \|\widehat{R}_i^{-1}\|_2^{-1} \leqslant \|R - \widehat{R}_i\|_2.$$

Substituting our bound on $\|R - \widehat{R}_i\|_F$, we obtain that

$$\|R^{-1}\|_2^{-1} - \|\widehat{R}_i^{-1}\|_2^{-1} \leqslant \frac{17}{64}\|R^{-1}\|_2^{-1}.$$

Rearranging these terms yields $\frac{47}{64}\|R^{-1}\|_2^{-1} \leqslant \|\widehat{R}_i^{-1}\|_2^{-1}$, and reciprocating then gives $\|\widehat{R}_i^{-1}\|_2 \leqslant \frac{64}{47}\|R^{-1}\|_2$. Using this bound in Wedin's theorem, we obtain

$$\|R^{-1} - \widehat{R}_i^{-1}\|_2 \leqslant \frac{1+\sqrt{5}}{2} \cdot \frac{64}{47} \cdot \|R^{-1}\|_2^2 \|R - \widehat{R}_i\|_2 \leqslant 5\|A\|_2^3 \|R^{-1}\|_2^4 \cdot (\|Q - \widehat{Q}\|_F + \epsilon k).$$

In the second part of the proof, we want to analyze the effect obtained by each node i applying its own matrix $\widehat{R}_i^{-1}$ to its row $\widehat{V}_i$ of the matrix $\widehat{V}$. Notice that this is a non-linear operation, so we cannot argue in terms of matrix products as above. Instead, we perform the analysis on a row-by-row basis. We can write $Q'_i - \widehat{Q}'_i$ as

$$Q'_i - \widehat{Q}'_i = V_i R^{-1} - \widehat{V}_i \widehat{R}_i^{-1} = V_i(R^{-1} - \widehat{R}_i^{-1}) + (V_i - \widehat{V}_i)\widehat{R}_i^{-1}.$$

We let C be the matrix whose ith row is $(V_i - \widehat{V}_i)\widehat{R}_i^{-1}$, and D the matrix whose ith row is $V_i(R^{-1} - \widehat{R}_i^{-1})$. We bound the Frobenius norms $\|C\|_F$, $\|D\|_F$ separately. To bound $\|C\|_F$, observe that

$$\begin{aligned}\|C\|_F^2 &= \sum_i \|(V_i - \widehat{V}_i)\widehat{R}_i^{-1}\|_2^2 \leqslant \sum_i \|V_i - \widehat{V}_i\|_2^2 \|\widehat{R}_i^{-1}\|_2^2 \leqslant \max_i \|\widehat{R}_i^{-1}\|_2^2 \cdot \sum_i \|V_i - \widehat{V}_i\|_2^2 \\ &= \max_i \|\widehat{R}_i^{-1}\|_2^2 \cdot \|V - \widehat{V}\|_F^2.\end{aligned}$$

Similarly, to bound the Frobenius norm of D:

$$\|D\|_F^2 = \sum_i \|V_i(R^{-1} - \widehat{R}_i^{-1})\|_2^2 \leqslant \|V\|_F^2 \cdot \max_i \|R^{-1} - \widehat{R}_i^{-1}\|_2^2.$$

We take square roots on both sides of these bounds, and combine them using the Triangle Inequality, getting

$$\|Q' - \widehat{Q}'\|_F \leqslant \|\widehat{V} - V\|_F \cdot \max_i \|\widehat{R}_i^{-1}\|_2 + \|V\|_F \cdot \max_i \|R^{-1} - \widehat{R}_i^{-1}\|_2.$$

Finally, inserting our bounds on $\|\widehat{R}_i^{-1}\|_2$ and $\|R^{-1} - \widehat{R}_i^{-1}\|_2$ yields that

$$\begin{aligned}\|Q' - \widehat{Q}'\|_F &\leqslant \|A\|_2 \|\widehat{Q} - Q\|_F \cdot \frac{64}{47}\|R^{-1}\|_2 + 5\sqrt{k}\|A\|_2 \|A\|_2^3 \|R^{-1}\|_2^4 (\|Q - \widehat{Q}\|_F + \epsilon k) \\ &\leqslant 7\sqrt{k} \cdot \|R^{-1}\|_2^4 \cdot \|A\|_2^4 \cdot (\|Q - \widehat{Q}\|_F + \epsilon k),\end{aligned}$$

completing the proof. □

Proof of Theorem 1. Lemma 8 establishes that the approximation error $\|Q - \widehat{Q}\|_F$ grows by at most a factor of $\sqrt{k} \cdot (2\|R^{-1}\|_2 \|A\|_2)^4$ with each iteration, plus an additional ϵk error. While this worst-case exponential growth is worrisome, the initial error is 0, and ϵ decreases exponentially with the number of Push-Sum steps performed. In particular, if we perform $\Omega(t\tau_{\text{mix}} \log(\frac{kc}{\epsilon} \cdot \|A\|_2))$ steps of Push-Sum in each iteration, then by Theorem 5, the new error introduced by the Push-Sum approximation in each iteration is at most δk, with $\delta = (\frac{\epsilon}{2kc} \frac{1}{\|A\|_2})^{4t}$.

As the matrices $Q, \widehat{Q}$ always satisfy the conditions of Lemma 8 (an easy proof by induction), the total error introduced over all t iterations is

$$\delta k \cdot \sum_{i=0}^{t-1} \left(\sqrt{k} \cdot \left(2\left\|R^{-1}\right\|_2 \|A\|_2\right)^4\right)^i = O\left(\delta k \cdot \left(\sqrt{k} \cdot \left(2\left\|R^{-1}\right\|_2 \|A\|_2\right)^4\right)^t\right).$$

Substituting the value of δ shows that if $\|R^{-1}\|_2 \leqslant c$ in each iteration, then the difference $\|Q - \widehat{Q}\|_F$ is bounded by ϵ^{4t} after t iterations.

To transform this bound into the claimed bound on the difference in projections $\|P_Q - P_{\widehat{Q}}\|_F$, note that

$$\|P_Q - P_{\widehat{Q}}\|_F = \left\|QQ^T - \widehat{Q}\widehat{Q}^T\right\|_F \leqslant \left\|QQ^T - Q\widehat{Q}^T\right\|_F + \left\|Q\widehat{Q}^T - \widehat{Q}\widehat{Q}^T\right\|_F \leqslant \left(\|Q\|_2 + \|\widehat{Q}\|_2\right)\|Q - \widehat{Q}\|_F.$$

By the argument in the proof of Lemma 8, the first factor is at most $17/8 \leqslant 3$, so that $\|P_Q - P_{\widehat{Q}}\|_F \leqslant 3\epsilon^{4t}$. Finally, combining this bound with the one from Theorem 3 completes the proof of Theorem 1. □

The main assumption of Theorem 1, that $\|R^{-1}\|_2$ is bounded, raises an interesting point. $\|R^{-1}\|_2$ becoming unbounded corresponds to the columns of Q becoming linearly dependent, an event that is unlikely to happen outside of matrices A of rank less than k. Should it happen, the decentralized algorithm will deal with this in the same manner that the centralized algorithm does: The final column of Q will be filled with garbage values. This garbage will then serve as the basis for a new attempt at convergence for this column. The difference between the centralized and decentralized approaches is precisely which garbage is used. Clearly if the error is adversarial, the new columns of Q could be chosen to be orthogonal to the top k eigenvectors, and correct convergence will not occur.

Notice that even if $\|R^{-1}\|_2$ is large for some value of k, it may be bounded for smaller values k'. Orthogonal iteration is a nested process, meaning that the results hold for $k' < k$, where we examine the matrices restricted to the first k' eigenvectors. This means that while we can no longer say that the final $k - k'$ columns necessarily track the centralized approach, we *can* say that the first k' are still behaving properly.

3.3. Detecting convergence in Push-Sum

In our discussion thus far, we have glossed over the issue of termination by writing "Run Push-Sum until the error drops below ϵ." We have yet to address the issue of how the nodes in the network know how many rounds to run. If the nodes knew τ_{mix}, the problem would be easy—however, this would require knowledge and a detailed analysis of the graph topology, which we cannot assume nodes to possess.

Instead, we would like nodes to detect convergence to within error ϵ themselves. We show how to achieve this goal under the assumption that each node knows (a reasonable upper bound on) the diameter $\text{diam}(G)$ of the graph G. In order to learn the diameter to within a factor of 2, a node may simply initiate a BFS at the beginning of the computation, and add the length of the two longest paths found this way.

Assume now that nodes know an upper bound d on the diameter, as well as a target upper bound ϵ on the relative error. For the purpose of error detection, the nodes, in addition to the matrices S_i from before, compute the sum of the non-negative matrices A_i, with $(A_i)_{rc} = |(S_i)_{rc}|$. When the nodes want to test whether the error has dropped below ϵ, they compute the values $a_{rc}^{\max} = \max_i \frac{(A_i)_{rc}}{w_i}$, $a_{rc}^{\min} = \min_i \frac{(A_i)_{rc}}{w_i}$, $s_{rc}^{\max} = \max_i \frac{(S_i)_{rc}}{w_i}$, and $s_{rc}^{\min} = \min_i \frac{(S_i)_{rc}}{w_i}$. (Notice that the maximum and minimum can be computed by using flooding, and only sending one value for each position (r, c), as both operations are idempotent.) The nodes decide to stop if the values for all matrix positions (r, c) satisfy $a_{rc}^{\min} \geqslant \frac{1}{1+\epsilon} a_{rc}^{\max}$, and $s_{rc}^{\max} - s_{rc}^{\min} \leqslant \frac{\epsilon}{1+\epsilon} a_{rc}^{\max}$. Otherwise, the nodes continue with Push-Sum.

We will show in Theorem 9 below that this rule essentially terminates when the maximum error is less than ϵ. As the computation of the maximum and minimum takes time $\Theta(\text{diam}(G))$, testing the error after each iteration would cause a slowdown by a multiplicative factor of $\Theta(\text{diam}(G))$. However, the BFS need only be performed every d steps, in which case at most an additional d rounds are run, while the amortized cost is at most a constant factor. Whenever $d = \Theta(\text{diam}(G))$, the overall effect is only a constant factor.

For our theorem below, we focus only on one matrix entry (r, c), as taking the conjunction over all entries does not alter the problem. We let x_i denote the value held by node i before the first iteration, and write $s_i = (S_i)_{rc}$, and $a_i = (A_i)_{rc}$ for the entries at the time under consideration. We define $a^{\max}, a^{\min}, s^{\max}$, and $s^{\min}$ in the obvious way.

In line with the error analysis above, we say that the error at node i is bounded by ϵ if $|\frac{s_i}{w_i} - \sum_j x_j| \leqslant \epsilon \sum_j |x_j|$. The error is bounded by ϵ if it is bounded by ϵ at all nodes i.

Theorem 9.

(1) *When the computation stops, the error is at most* ϵ.
(2) *After the number* t *of steps specified in Lemma* 6 *to obtain error at most* $\frac{\epsilon}{2(1+\epsilon)}$, *the computation will stop.*

Notice that there is a gap of $\frac{1}{2(1+\epsilon)}$ between the actual desired error and the error bound that ensures that the protocol will terminate. However, this is only a constant factor, so only a constant number of additional steps is required (after the actual error has dropped below ϵ) until the nodes actually detect that it is time to terminate.

Proof. (1) When the computation stops, the stopping requirement ensures that

$$a^{\min} \geqslant \frac{1}{1+\epsilon} a^{\max}, \tag{2}$$

$$s^{\max} - s^{\min} \leqslant \frac{\epsilon}{1+\epsilon} a^{\max}. \tag{3}$$

Because $\sum_j w_j = 1$, we obtain that $\sum_j a_j = \sum_j w_j \frac{a_j}{w_j}$ is in fact a convex combination of $\frac{a_j}{w_j}$ terms, and in particular $a^{\min} \leqslant \sum_i a_i \leqslant a^{\max}$. Thus, inequality (2) implies that $a^{\max} \leqslant (1+\epsilon) \cdot \sum_j a_j$.

Inequality (3) therefore implies that $s^{\max} - s^{\min} \leqslant \epsilon \sum_j a_j$. The same convexity argument, applied this time to $\sum_j s_j$, as well as the facts that $\sum_j a_j = \sum_j |x_j|$ and $\sum_j s_j = \sum_j x_j$, now ensures that $|\frac{s_i}{w_i} - \sum_j x_j| \leqslant \epsilon \cdot \sum_j |x_j|$ for all nodes i, i.e. the desired error bound.

(2) For the second part, we first apply Lemma 6, yielding for all nodes i that

$$\left|\frac{a_i}{w_i} - \sum_j |x_j|\right| \leqslant \frac{\epsilon}{2(1+\epsilon)} \sum_j |x_j|,$$

$$\left|\frac{s_i}{w_i} - \sum_j x_j\right| \leqslant \frac{\epsilon}{2(1+\epsilon)} \sum_j |x_j|.$$

By the Triangle Inequality and the above convexity argument,

$$a^{\max} - a^{\min} \leqslant 2\frac{\epsilon}{2(1+\epsilon)} \sum_j |x_j| \leqslant \frac{\epsilon}{1+\epsilon} a^{\max},$$

so the first stopping criterion is satisfied. Similarly,

$$s^{\max} - s^{\min} \leqslant 2\frac{\epsilon}{2(1+\epsilon)} \sum_j |x_j| \leqslant \frac{\epsilon}{1+\epsilon} a^{\max},$$

so the second criterion is met as well, and the protocol will terminate. □

4. Conclusions

In this paper, we have presented and analyzed a decentralized algorithm for the computation of a graph's spectral decomposition. The approach is based on a simple algorithm called Push-Sum for summing values held by nodes in a network [19].

We have presented a worst-case error analysis; one that is far more pessimistic than those performed in bounding the (similar) effects of floating point errors on numerical linear algebra algorithms. Nonetheless, our analysis shows that t iterations of orthogonal iteration can be performed without central control in time $O(t^2 \tau_{\text{mix}})$, where τ_{mix} is the mixing time of any Markov Chain on the network under consideration.

We believe that our algorithm represents a starting point for a large class of distributed data mining algorithms, which leverage the structure and participants of the network. This suggests the more general question of which data mining services really need to be centralized. For example, Google's primary service is not the computation of Pagerank, but rather computing and serving a huge text reverse-index. Can such a task be decentralized, and can a web search system be designed without central control?

Above, we argue informally that one of the advantages of our algorithm is a greater protection of nodes' privacy. An exciting direction for future work is to investigate in what sense decentralized algorithms can give formal privacy guarantees.

The convergence of our algorithm depends on the mixing speed of the underlying Markov Chain. For a fixed network, different Markov Chains may have vastly different mixing speeds [22]. Boyd et al. [22] show how to compute the fastest mixing Markov Chain by using semi-definite programming; however, this approach requires knowledge of the entire network and is inherently centralized. In more recent work, Boyd, Ghosh, et al. [21], using the techniques of this paper for distributed eigenvector computation, give a fully decentralized implementation of a subgradient approximation algorithm for convex optimization, and use it to compute a (nearly) fastest mixing Markov Chain. Nevertheless, it would be interesting whether a simpler and more direct approach based on the eigenvectors can be used to compute an approximately fastest mixing Markov Chain more efficiently. Such an algorithm would have applications to routing of concurrent flows (by removing bottlenecks), and allow the network to "self-diagnose" and speed up future invocations of our decentralized algorithm.

Another question related to self-diagnosis is the error estimate in the Push-Sum algorithm. At the moment, we assume that all nodes know the diameter, and can run an error estimation protocol after appropriately chosen intervals. Is there a decentralized stopping criterion that does not require knowledge of $\operatorname{diam}(G)$ or n?

Acknowledgments

We would like to thank Alin Dobra, Johannes Gehrke, Sharad Goel, Jon Kleinberg, and Laurent Saloff-Coste for useful discussions. We would also like to thank two anonymous referees for useful suggestions.

References

[1] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proc. ACM–SIGCOMM Conference, 2001, pp. 161–172.
[2] A. Rowstron, P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, in: Proc. 18th IFIP/ACM Intl. Conf. on Distributed Systems Platforms, Middleware 2001, 2001, pp. 329–350.
[3] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for Internet applications, in: Proc. ACM–SIGCOMM Conference, 2001, pp. 149–160.
[4] B. Zhao, J. Kubiatowicz, A. Joseph, Tapestry: An infrastructure for fault-tolerant wide-area location and routing, Tech. Rep. UCB/CSD-01-1141, UC Berkeley, 2001.
[5] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, Comput. Netw. ISDN Systems 30 (1998) 107–117.
[6] J. Kleinberg, Authoritative sources in a hyperlinked environment, J. ACM 46 (1999) 604–632.
[7] M. Fiedler, A property of eigenvectors of nonnegative symmetric matrices and its applications to graph theory, Czechoslovak Math. J. 25 (1975) 619–633.
[8] R. Kannan, S. Vempala, A. Vetta, On clusterings: Good, bad and spectral, in: Proc. 41st IEEE Symp. on Foundations of Computer Science, 2000, pp. 367–377.
[9] A. Ng, M. Jordan, Y. Weiss, On spectral clustering: Analysis and an algorithm, in: Proc. 14th Advances in Neural Information Processing Systems, 2002, pp. 849–856.
[10] F. McSherry, Spectral partitioning of random graphs, in: Proc. 42nd IEEE Symp. on Foundations of Computer Science, 2001, pp. 529–537.
[11] S. Deerwester, S. Dumais, T. Landauer, G. Furnas, R. Harshman, Indexing by latent semantic analysis, J. Amer. Soc. Inform. Sci. 41 (1990) 391–407.
[12] D. Achlioptas, A. Fiat, A. Karlin, F. McSherry, Web search via hub synthesis, in: Proc. 42nd IEEE Symp. on Foundations of Computer Science, 2001, pp. 500–509.
[13] Y. Azar, A. Fiat, A. Karlin, F. McSherry, J. Saia, Spectral analysis of data, in: Proc. 33rd ACM Symp. on Theory of Computing, 2001, pp. 619–626.
[14] F. Leighton, S. Rao, Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms, J. ACM 46 (1999) 787–832.
[15] M. Granovetter, The strength of weak ties, Amer. J. Sociol. 78 (1973) 1360–1380.
[16] F. Chung, Spectral Graph Theory, American Mathematical Society, 1997.
[17] K. Gallivan, M. Heath, E. Ng, B. Peyton, R. Plemmons, J. Ortega, C. Romine, A. Sameh, R. Voigt, Parallel Algorithms for Matrix Computations, Society for Industrial and Applied Mathematics, 1990.

[18] I. Benjamini, L. Lovász, Global information from local observation, in: Proc. 43rd IEEE Symp. on Foundations of Computer Science, 2002, pp. 701–710.
[19] D. Kempe, A. Dobra, J. Gehrke, Computing aggregate information using gossip, in: Proc. 44th IEEE Symp. on Foundations of Computer Science, 2003, pp. 482–491.
[20] S. Boyd, A. Ghosh, B. Prabhakar, D. Shah, Analysis and optimization of randomized gossip algorithms, in: Proc. 43rd IEEE Conference on Decision and Control, 2004, pp. 5310–5315.
[21] S. Boyd, A. Ghosh, B. Prabhakar, D. Shah, Gossip algorithms: Design, analysis and applications, in: Proc. 24th IEEE INFOCOM Conference, 2005, pp. 1653–1664.
[22] S. Boyd, P. Diaconis, L. Xiao, Fastest mixing Markov Chain on a graph, SIAM Rev. 46 (2004) 667–689.
[23] S. Boyd, A. Ghosh, B. Prabhakar, D. Shah, Mixing times for random walks on geometric random graphs, in: Proc. 2nd SIAM Workshop on Analytic Algorithms and Combinatorics, 2005.
[24] G. Golub, C. van Loan, Matrix Computations, third ed., Johns Hopkins University Press, 1996.
[25] L. Saloff-Coste, Lectures on finite Markov Chains, in: Lecture Notes in Math., vol. 1665, Springer, 1997, pp. 301–408, École d'été de St. Flour 1996.
[26] G. Stewart, On the perturbation of LU and Cholesky factors, IMA J. Numer. Anal. 17 (1997) 1–6.
[27] P. Wedin, Perturbation theory for pseudo-inverses, BIT 13 (1973) 217–232.