Distributed computation of the Fiedler vector with application to topology inference in ad hoc networks

Alexander Bertrand*,† and Marc Moonen*,†

∗ KU Leuven, Dept. Electrical Engineering ESAT, SCD-SISTA
† iMinds-Future Health Department
Kasteelpark Arenberg 10, B-3001 Leuven, Belgium
E-mail: alexander.bertrand@esat.kuleuven.be
marc.moonen@esat.kuleuven.be
Phone: +32 16 321899, Fax: +32 16 321970

***Abstract*—The Fiedler vector of a graph is the eigenvector corresponding to the smallest non-trivial eigenvalue of the Laplacian matrix corresponding to this graph. The entries of the Fiedler vector are known to provide a powerful heuristic for topology inference, e.g., to identify densely connected node clusters, to search for bottleneck links in the information dissemination, or to increase the overall connectivity of the network. In this paper, we consider ad hoc networks where the nodes can process and exchange data in a synchronous fashion, and we propose a distributed algorithm for in-network estimation of the Fiedler vector and the algebraic connectivity of the corresponding network graph. The algorithm is fully scalable with respect to the network size in terms of per-node computational complexity and data transmission. Simulation results demonstrate the performance of the algorithm.**

***Index Terms*—Spectral graph theory, Fiedler vector, wireless sensor networks, distributed algorithms.**

I. Introduction

Ad hoc networks often appear in signal processing applications, mainly in the field of wireless sensor networks (WSNs) [1]. A WSN consists of a collective of sensor nodes that can exchange data amongst each other through wireless links, and where each node has a processing unit to perform local computations. Oftentimes, the network topology is not predefined but created in an ad hoc way, e.g., based on a nearest-neighbor criterion to allow for low-power communication. However, the performance of many distributed algorithms that are operated in such an ad hoc network highly depends on the network topology [2], [3], [4], [5], [6], [7]. For example, densely connected networks generally result in significantly faster convergence due to a more efficient in-network information diffusion.

It is therefore desirable to gain some high-level knowledge about the topology of the network. Spectral graph theory [8], involving the eigenvalue decomposition of the Laplacian matrix corresponding to the network graph, has been demonstrated to be a very powerful tool for topology inference problems. The eigenvalues and/or eigenvectors of the Laplacian matrix allow, e.g., to estimate the connectivity of the network [9], [10], [11], [12], to find topological invariants [13], to find densely connected clusters of nodes [9], [12], [14], [15], [16], [17], [18], [19], [20], [21], to identify bottlenecks or critical links (i.e., the sparse set of links between these densely connected node clusters) [9], [22], and to search for redundant links or potential links that would greatly improve the connectivity if they would be established [9], [23], [24], [25]. These are all non-trivial tasks, especially so if this has to be performed in a distributed fashion. Indeed, a node usually only has a limited view, i.e., it can only see its neighbors, and so it does not immediately know what the rest of the network looks like (unless their neighbors pass on additional information about their respective neighbors and so on). For example, the earlier mentioned node-clustering problem corresponds to a graph-partitioning problem in the corresponding network graph, which is very difficult to solve in a distributed fashion. Furthermore, even in a centralized approach, the graph-partitioning problem is NP-complete [20], indicating the need for heuristic methods, e.g., by means of the spectrum of the Laplacian matrix of the graph.

In [26], a distributed algorithm is proposed to find the n eigenvectors corresponding to the n largest eigenvalues of the Laplacian matrix or the (weighted) adjacency matrix, based on power iteration and random walk techniques. This algorithm requires nested loops (for decentralized orthogonalization of distributed vectors) which severely affects its efficiency in terms of communication requirements and convergence speed. This is because each power iteration requires many lower-level iterations, which also require additional data exchange

The work of A. Bertrand was supported by a Postdoctoral Fellowship of the Research Foundation - Flanders (FWO). This work was carried out at the ESAT Laboratory of KU Leuven, in the frame of KU Leuven Research Council CoE EF/05/006 'Optimization in Engineering' (OPTEC) and PFV/10/002 (OPTEC), Concerted Research Action GOA-MaNet, the Belgian Programme on Interuniversity Attraction Poles initiated by the Belgian Federal Science Policy Office IUAP P6/04 (DYSCO, 'Dynamical systems, control and optimization', 2007-2011), Research Project iMinds, and Research Project FWO nr. G.0763.12 'Wireless acoustic sensor networks for extended auditory communication'. The scientific responsibility is assumed by its authors.

between neighboring nodes. Another family of algorithms let the nodes oscillate at the eigenfrequencies corresponding to the network topology [12], [27], [28]. By computing the discrete-time Fourier transform (DTFT) of these oscillating signals at the different nodes, the eigenstructure of the Laplacian matrix is revealed. These methods typically require less communication than [26]. Although they have been demonstrated to provide good results in tasks where only approximate eigenvector/eigenvalue estimates are sufficient (such as in, e.g., node clustering tasks [12], [27]), these methods usually suffer from a rather poor accuracy and robustness issues. This is because it is difficult to extract accurate estimates of the eigenvectors from the DTFT spectra, mainly due to spectral leakage. Furthermore, the DTFT and the peak detection/extraction in the second step of the algorithm may be too computationally expensive for, e.g., low-power sensor networks.

Although the above procedures allow to compute n dominant eigenvectors, it has been argued and demonstrated that the most useful eigenvector for graph partitioning is the one corresponding to the *second-smallest* eigenvalue λ_2 of the Laplacian matrix. This eigenvalue λ_2 is referred to as the algebraic connectivity and its eigenvector is often referred to as the Fiedler vector [19], [29], after Miroslav Fiedler who developed the original theory related to algebraic connectivity [10]. For example, it is well-known that the entries in the Fiedler vector provide a powerful heuristic to partition a graph in subgraphs [9], [16], [17], [18], [19], [20], [21], [30], [31], which also reveals the critical links within the network graph. The Fiedler vector can also be used to improve the overall algebraic connectivity of the network topology [9], [23], [24].

In this paper, we present a novel distributed algorithm to accurately compute the Fiedler vector and/or the algebraic connectivity[1]. The algorithm is fully scalable in terms of per-node computational complexity and data exchange. Since we only focus on a single eigenvector, we are able to combine an efficient in-network power iteration (PI) with mean correction steps. This avoids the need for nested loops, making it much more efficient (both in terms of convergence and data exchange), more accurate, and easier to implement than the algorithm in [26] when the latter is used for the computation of the Fiedler vector. These claims will be demonstrated with simulations.

The outline of the paper is as follows. Section II provides the definition of algebraic connectivity and the Fiedler vector, as well as some pointers to applications that rely on these quantities and a brief example how the Fiedler vector can be used to identify densely connected clusters and their sparse cross connections in a network graph. In Section III, a distributed algorithm is derived for in-network computation of the Fiedler vector. Section IV provides simulation results. Conclusions are drawn in Section V.

[1]For the sake of completeness, it should be noted that there also exists a distributed algorithm for the estimation of the algebraic connectivity itself, without estimating the Fiedler vector [11].

II. The algebraic connectivity and the Fiedler vector

A. Definition

Consider an ad hoc network with a set of nodes $\mathcal{K} = \{1, \ldots, K\}$. We denote $\mathcal{N}_k$ as the set of neighboring nodes of node k (node k excluded). The Laplacian matrix $\mathbf{L} = (l_{kq})_{K\times K}$ corresponding to the network graph G is defined as

$$l_{kq} = \begin{cases} |\mathcal{N}_k| & \text{if } k = q \\ -1 & \text{if } k \neq q, \text{ and } q \in \mathcal{N}_k \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where $|\mathcal{N}_k|$ denotes the cardinality of $\mathcal{N}_k$, which is equal to the degree of node k. Notice that $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D} = \text{diag}(|\mathcal{N}_1|, \ldots, |\mathcal{N}_K|)$ is the degree matrix and $\mathbf{A} = (a_{kq})_{K\times K}$ is the so-called adjacency matrix where $a_{kq} = 1$ if node k and q are connected and $a_{kq} = 0$ otherwise. It is noted that all results in this paper can straightforwardly be generalized to weighted graphs, where the weighted Laplacian matrix is then used instead:

$$l_{kq} = \begin{cases} \sum_{j\in\mathcal{K}} w_{kj} & \text{if } k = q \\ -w_{kq} & \text{if } k \neq q, \text{ and } q \in \mathcal{N}_k \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where w_{kq} is the weight on the edge between nodes k and q. For example, in a communication network, this weight can be associated to the packet loss or bit-error rate (BER) over a communication link, or the received signal strength (RSS) in case of wireless networks.

We denote the eigenvalues of the Laplacian matrix as

$$\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_K \ . \quad (3)$$

The Laplacian matrix $\mathbf{L}$ is an important matrix in spectral graph theory with several interesting properties. It is a symmetric positive semidefinite matrix, and it has a single[2] eigenvalue equal to zero ($\lambda_1 = 0$), corresponding to the eigenvector $\frac{1}{\sqrt{K}}\mathbf{1}_K$, where $\mathbf{1}_K$ is the K-dimensional vector with all entries equal to one. The second smallest eigenvalue λ_2 is referred to as the *algebraic connectivity*, and its corresponding eigenvector $\mathbf{x}_F$ is usually referred to as the *Fiedler (eigen)vector*.

Remark I: For the sake of an easy exposition, we make the pragmatic assumption that $\lambda_2 \neq \lambda_3$, i.e., there is only one eigenvector associated to λ_2 ($\mathbf{x}_F$ is unique). However, this is without loss of generality, as the algorithm derived in Section III can also cope with the rare cases where this assumption is not satisfied[3]. Indeed, if λ_2 has a multiplicity $n > 1$, the algorithm will converge to an arbitrary vector in the n-dimensional subspace spanned by the n eigenvectors associated to λ_2.

B. Applications

The algebraic connectivity λ_2 contains important information on the connectivity of a graph and its separation properties

[2]If the network graph is not connected, the multiplicity of the zero eigenvalue is larger than 1, i.e., it is equal to the number of disconnected subgraphs.

[3]This is due to the fact that the algorithm relies on power iterations, which also converge to a valid solution if the dominant eigenspace has a dimension larger than one.

[9], [11], [29], [32], and it determines the convergence speed of distributed estimation algorithms that are operated in the same network [9], including consensus- [2], [33], gossip- [34], and diffusion-based [4] techniques. Furthermore, the entries of the Fiedler vector can be used as a heuristic to manipulate the network topology, e.g., to identify redundant links (which can be removed to save energy), to add links that significantly improve the connectivity, or to manipulate the mixing weights in consensus algorithms to improve convergence [9], [23], [24], [25].

Furthermore, it is well-known that the coefficients of the Fiedler vector $\mathbf{x}_F$ form a powerful heuristic to identify densely connected node clusters and their sparse cross connections [9], [16], [17], [18], [19], [20], [21]. Identifying such clusters also reveals the weak points or the critical links in the network, i.e., the sparse links between these dense node clusters [9], [22]. These links can be viewed as bottleneck links to transfer the information between clusters. Furthermore, if these bottleneck links are removed, this has a large impact on the connectivity of the network, as the small number of 'bridges' between these dense node clusters gets even more reduced (which may even result in disconnected subgraphs).

It is noted that a detailed description, performance analysis and comparison of graph-partitioning or node-clustering algorithms is beyond the scope of this paper, as these are only addressed here as an example application of the Fiedler vector, the actual focus of the paper being the in-network distributed computation of this eigenvector. However, as a brief illustration and to demonstrate what type of information the Fiedler vector actually conveys, we will briefly address its use for clustering in the next subsection.

C. Node clustering based on the Fiedler vector

Consider an ad hoc network for which the network topology is described by the connected graph $G(\mathcal{K}, \mathcal{E})$ where $\mathcal{E}$ denotes the set of edges in the graph, i.e., the links in the network. In this network (or graph), we would like to identify several node clusters (or subgraphs) that are internally densely connected, but which have only a few links to other clusters (i.e., perform graph partitioning). For example, in the network depicted in Fig. 1(a), three node clusters can be clearly distinguished (separated by the dotted lines), and each cluster has only two links with the two other clusters. The links cut by the dotted lines form important bridges between the clusters, and can therefore considered to be bottleneck links for the data dissemination in a WSN (this is also important information for resource allocation algorithms). Even though it is immediately clear for a human observer how to cluster the network in Fig. 1(a), the node-clustering problem appears to be a lot harder for the network depicted in Fig. 1(b), even though both networks are actually identical. Designing a node-clustering algorithm that identifies these clusters automatically is a difficult task, especially so in large-scale networks. This becomes even harder in a distributed context since the individual nodes usually only observe a restricted neighborhood of the network graph.

For the time being, assume that we aim to divide a network in two non-overlapping clusters. In graph theory, this is referred to as a graph cut. One of the most common criteria for graph partitioning is the so-called ratio cut, which minimizes the so-called edge density defined as [16]

$$\rho(\mathcal{K}_1, \mathcal{K}_2) = \frac{|\mathrm{E}(\mathcal{K}_1, \mathcal{K}_2)|}{|\mathcal{K}_1||\mathcal{K}_2|} \tag{4}$$

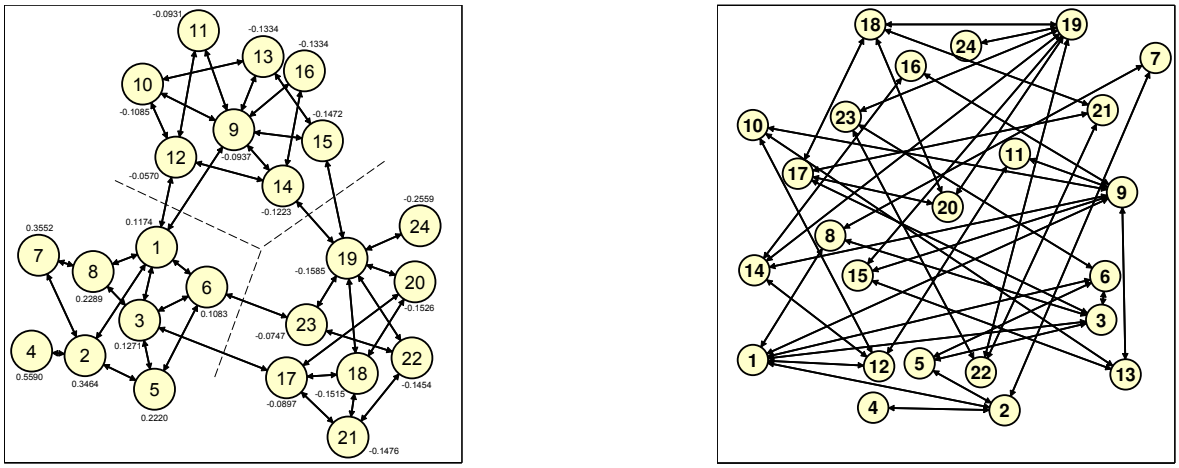
where $|\cdot|$ denotes cardinality, and where $\mathcal{K}_1 \cap \mathcal{K}_2 = \emptyset$, $\mathcal{K}_1 \cup \mathcal{K}_2 = \mathcal{K}$ and $\mathrm{E}(\mathcal{K}_1, \mathcal{K}_2)$ denotes the set of edges in G that are shared between $\mathcal{K}_1$ and $\mathcal{K}_2$, i.e., the edges of the graph that are cut. The intuition behind the ratio cut is that the minimization of the numerator of (4) will minimize the number of edges between the two subgraphs, while the maximization of the denominator is a driving force towards subgraphs of equal size (which avoids trivial solutions). It is noted that the edge density can easily be generalized for weighted graphs by replacing $|\mathrm{E}(\mathcal{K}_1, \mathcal{K}_2)|$ with the sum of the weights of the edges that are cut.

Finding the ratio cut of a graph is not straightforward. The problem is shown to be NP-complete [20], and so one has to rely on heuristics to solve it. A common method is to divide the network in two clusters based on the entries of the Fiedler vector (an intuition why the Fiedler vector has this clustering property can be found in, i.a., [18]). In most cases, the positive entries and negative entries in $\mathbf{x}_F$ are separated to define two subgraphs, although other approaches also exist [17], [19]. If a graph is to be partitioned into more than two subgraphs, these Fiedler-based graph partitioning methods can be applied recursively, i.e., the Fiedler vector is used to partition the network into two subgraphs, then the Fiedler vector of each subgraph is computed, and this is continued recursively until the desired number of subgraphs is found. This approach is called Recursive Spectral Bisection (RSB) [30], [31]. Whereas RSB is a top-down approach, also bottom-up approaches exist (based on similar spectral techniques) where multiple node clusters are grown from certain seed nodes and then merged together [14], [15].

Consider the Fiedler vector of the example network graph depicted in Fig. 1, of which the entries are given in Table I (and also next to the corresponding nodes in Fig. 1(a)). The first important observation is that the nodes in $\mathcal{K}_1 = \{1, \ldots, 8\}$ have positive entries, whereas the other nodes have negative entries. This indicates that the nodes in $\mathcal{K}_1$ form a (densely connected) node cluster with only a small number of links to nodes outside this node cluster. A second -more subtle- observation is that within $\mathcal{K}_1$, nodes 1, 3 and 6 have entries that are significantly closer to zero than the others. This is because these nodes contain the 'bridges' between $\mathcal{K}_1$ and the other node cluster $\mathcal{K}\backslash\mathcal{K}_1$. Similarly, nodes 9, 12, 17 and 23 also have entries that are significantly closer to zero than the other entries in $\mathbf{x}_F$, again due to the fact that these form the connections with $\mathcal{K}_1$.

Finally, the larger node cluster $\mathcal{K}\backslash\mathcal{K}_1$ can be further divided in two separate node clusters by computing the Fiedler vector of the subgraph corresponding to this node cluster. This finally results in the three node clusters indicated by the dotted lines in Fig. 1.

It is noted that the powerful clustering properties of the Fiedler vector have mostly been empirically demonstrated.



(a) Ordened node placement

(b) Random node placement

Fig. 1. Two different visualizations of an ad hoc network consisting of $K = 24$ nodes.

TABLE I
ENTRIES OF THE FIEDLER VECTOR OF THE NETWORK DEPICTED IN FIG. 1.

1	2	3	4	5	6	7	8
0.1174	0.3464	0.1271	0.5590	0.2220	0.1083	0.3552	0.2289
9	10	11	12	13	14	15	16
-0.0937	-0.1085	-0.0931	-0.0570	-0.1334	-0.1223	-0.1472	-0.1334
17	18	19	20	21	22	23	24
-0.0897	-0.1515	-0.1585	-0.1526	-0.1476	-0.1454	-0.0747	-0.2559

For planar graphs in particular, there exists theoretical proof showing that the Fiedler vector almost always reveals a good clustering [19], although some contrived counterexamples exist where the Fiedler vector (or any other spectral clustering method) cannot lead to good results [19], [35]. Finally, it should be noted that Fiedler-based clustering is just one possible approach to tackle the clustering problem, and many others exist (see, e.g., [14] for an extensive overview of clustering methods).

III. DISTRIBUTED COMPUTATION OF THE FIEDLER VECTOR

In this section, we explain how the Fiedler vector $\mathbf{x}_F$ can be computed inside the network in a distributed fashion, such that each node $k \in \mathcal{K}$ eventually has access to its corresponding entry in $\mathbf{x}_F$. Throughout this paper, we assume that the communication links of the network are ideal and we assume a synchronous setting where there is a common network-wide iteration index that is incremented deterministically at regular time intervals. During each iteration, the nodes are assumed to perform a pre-defined task and share the result with their neighbors.

An obvious but important observation is that the Laplacian matrix is implicitely coded inside the network itself, allowing to perform multiplications with the Laplacian matrix in a distributed fashion. Indeed, for the matrix-vector product $\mathbf{y} = \mathbf{L}\mathbf{x}$, assuming that node k stores the k-th entry of $\mathbf{x}$ (denoted by x_k) and has access to the x_q's of its neighbors ($q \in \mathcal{N}_k$), the k-th entry of $\mathbf{y}$ can be computed at node k as

$$y_k = |\mathcal{N}_k| x_k - \sum_{q \in \mathcal{N}_k} x_q \tag{5}$$

(in the case of an unweighted graph). This allows to use a distributed power iteration (PI) method for the in-network computation of the eigenvector corresponding to λ_K, i.e., the largest eigenvalue of $\mathbf{L}$. Basically the PI method amounts to a repeated matrix-vector product computation

$$\mathbf{x}^{(i+1)} = \mathbf{L}\mathbf{x}^{(i)} \tag{6}$$

where i is an iteration index, and $\mathbf{x}^{(0)}$ is initialized with a random non-zero vector. To avoid $\lim_{i\to\infty} \|\mathbf{x}^{(i)}\| = 0$ (when $\lambda_K < 1$) or $\lim_{i\to\infty} \|\mathbf{x}^{(i)}\| = \infty$ (when $\lambda_K > 1$), intermediate normalization steps have to be performed, which require full knowledge of $\mathbf{x}^{(i)}$, hindering the distributed computation. In principle, the norm of $\mathbf{x}^{(i)}$ (be it the 1-norm, 2-norm or infinity norm) can be computed in a distributed fashion by means of consensus averaging [2], [3], [36], [37] or gossip techniques [38] that run in parallel with (6). However, if $\lambda_K \gg 1$ or $\lambda_K \ll 1$, the norm of $\mathbf{x}^{(i)}$ will change significantly in each iteration of (6), such that these distributed averaging techniques may be too slow to track it. In this case, it is helpful to also estimate the growth or shrinking rate of $\|\mathbf{x}^{(i)}\|$ in (6) and compensate for it [39].

We will use a similar PI-based distributed algorithm for the in-network computation of the Fiedler vector. To this end, we

define the matrix

$$\mathbf{M} = \mathbf{I}_K - \frac{1}{\alpha}\mathbf{L} \tag{7}$$

where $\mathbf{I}_K$ denotes the $K \times K$ identity matrix and where α is chosen as a strictly positive value (see below). It is noted that the matrix-vector product $\mathbf{y} = \mathbf{M}\mathbf{x}$ can again be computed in a distributed fashion, i.e.,

$$y_k = \left(1 - \frac{1}{\alpha}|\mathcal{N}_k|\right) x_k + \frac{1}{\alpha} \sum_{q \in \mathcal{N}_k} x_q \ . \tag{8}$$

If α is chosen large enough, the eigenvectors corresponding to the two smallest eigenvalues of $\mathbf{L}$, i.e., $\lambda_1 = 0$ and λ_2, are equal to the eigenvectors corresponding to the largest eigenvalues of $\mathbf{M}$, i.e., $\mu_K = 1$ and $\mu_{K-1} = 1 - \frac{\lambda_2}{\alpha}$. A sufficient condition to guarantee this is to choose[4] $\alpha \geq \lambda_K$. An upper bound for λ_K is given in [40]:

$$\lambda_K \leq K \ . \tag{9}$$

Gershgrorin's circle theorem [41], yields a second upper bound which is usually tighter than (9):

$$\lambda_K \leq 2\Delta \tag{10}$$

where Δ is the maximum degree of the network, i.e.,

$$\Delta = \max_{k \in \mathcal{K}} |\mathcal{N}_k| \ . \tag{11}$$

This bound is improved[5] in [42], where it shown that

$$\lambda_K \leq \Phi \tag{12}$$

where

$$\Phi = \max_{k \in \mathcal{K}} \left(|\mathcal{N}_k| + \frac{1}{|\mathcal{N}_k|} \sum_{q \in \mathcal{N}_k} |\mathcal{N}_q| \right) \tag{13}$$

which can be easily computed in a distributed fashion. Hence, if we set

$$\alpha = K, \ \alpha = 2\Delta, \text{ or } \alpha = \Phi \tag{14}$$

then the matrix $\mathbf{M}$ is guaranteed to be positive semidefinite. It is noted that the bounds (10) and (12) can be generalized to the case of weighted graphs by replacing $|\mathcal{N}_k|$ with $\sum_{q \in \mathcal{N}_k} w_{kq}$.

In the sequel, we assume that the nodes are able to agree on a common value of α. This is easiest for $\alpha = 2\Delta$ or $\alpha = \Phi$ as it merely requires a min/max-consensus over locally computable quantities (see, e.g., [43]).

The largest eigenvalue of $\mathbf{M}$ is $\mu_K = 1$ and its corresponding eigenvector is $\frac{1}{\sqrt{K}}\mathbf{1}_K$. This means that the matrix $\mathbf{M}$ is mean-preserving, i.e., the entries of $\mathbf{M}\mathbf{x}$ will have the same mean value as the entries in $\mathbf{x}$ (notice that $\frac{1}{K}\mathbf{1}_K^T\mathbf{M}\mathbf{x} = \frac{1}{K}\mathbf{1}_K^T\mathbf{x}$). Therefore, if[6] $\mathbf{1}_K^T\mathbf{x}^{(0)} = 0$, then the PI-sequence $\{\mathbf{x}^{(i)}\}_{i \in \mathbb{N}}$, generated as

$$\mathbf{x}^{(i+1)} = \frac{\mathbf{M}\mathbf{x}^{(i)}}{\|\mathbf{M}\mathbf{x}^{(i)}\|} \tag{15}$$

has $\mathbf{1}_K^T\mathbf{x}^{(i)} = 0$ for all i and hence will converge to $\mathbf{x}_F$ instead of $\frac{1}{\sqrt{K}}\mathbf{1}_K$. Unfortunately, this is not a stable procedure since rounding errors will yield $\mathbf{x}^{(i)}$'s that have a non-zero mean, which will eventually result in convergence to the vector $\frac{1}{\sqrt{K}}\mathbf{1}_K$ instead of $\mathbf{x}_F$. Therefore, a mean correction step is required to remove the mean of the entries in $\mathbf{x}^{(i)}$ (and also in $\mathbf{x}^{(0)}$), i.e., the PI-sequence is generated as

$$\mathbf{v} \leftarrow \frac{\mathbf{M}\mathbf{x}^{(i)}}{\|\mathbf{M}\mathbf{x}^{(i)}\|} \tag{16}$$

$$\mathbf{x}^{(i+1)} = \mathbf{v} - \frac{\mathbf{v}^T\mathbf{1}_K}{K}\mathbf{1}_K \ . \tag{17}$$

Note that it is usually not necessary to perform this mean correction in each iteration.

In the basic PI procedure (16)-(17), the normalization and the mean correction hinder distributed computation since these require that each node has access to the entries of a network-wide vector. In the next subsections, we will explain how these problems can be circumvented.

A. Normalization

In this subsection, we assume infinite precision, hence the mean correction step (17) can be ignored (this will be addressed in the next subsection). Since we are not necessarily interested in a normalized Fiedler vector, we can omit the full normalization as in (15). However, we still have to avoid that the norm of $\mathbf{x}^{(i)}$ converges to zero (the norm cannot diverge since the spectral radius of $\mathbf{M}$ is equal to $\mu_K = 1$). To this end, we will estimate the rate at which the norm decreases in each iteration, i.e.,

$$r^{(i)} = \frac{\|\mathbf{M}\mathbf{x}^{(i)}\|}{\|\mathbf{x}^{(i)}\|} \tag{18}$$

and compensate for it such that the norm stops shrinking. This results in the following updating procedure

$$\tilde{\mathbf{x}}^{(i+1)} = \mathbf{M}\mathbf{x}^{(i)} \tag{19}$$

$$\mathbf{x}^{(i+1)} = \frac{1}{p}\tilde{\mathbf{x}}^{(i+1)} \tag{20}$$

where p is preferably $p \approx r^{(i)}$ (we do not add an iteration index i to p since p is not necessarily updated in each iteration, as explained later). Since (19)-(20) causes $\mathbf{x}^{(i)}$ to converge to the eigenvector corresponding to eigenvalue μ_{K-1} of $\mathbf{M}$, independently from p, it follows from (18) that

$$\lim_{i \to \infty} r^{(i)} = \mu_{K-1} \ . \tag{21}$$

For the same reason, we know that $\tilde{x}_k^{(i+1)} = \mu_{K-1}x_k^{(i)}$ when $i \to \infty$. Therefore, if each node $k \in \mathcal{K}$ has a local estimate of $r^{(i)}$, denoted as $r_k^{(i)}$, this estimate can be updated with the

[4]This guarantees that the matrix $\mathbf{M}$ is positive semidefinite, which avoids existence of a dominant negative eigenvalue to which the PI would then converge.

[5]There exist tighter upper bounds for λ_K (see, e.g., [40]), but we use (9), (10), and (12) since these are elegant and feasible to compute in a distributed fashion.

[6]We will explain later how such initialization can be achieved in only one iteration, i.e., a single mean correction step in which nodes share only one scalar value with their neighbors.

normalized least mean squares (NLMS) algorithm [44] as

$$r_k^{(i+1)} = r_k^{(i)} + \frac{\sigma}{\max\left(\delta, \left(x_k^{(i)}\right)^2\right)} x_k^{(i)} e_k^{(i)} \tag{22}$$

where $\sigma > 0$ is the learning rate, δ is a small positive value to avoid a division by zero, and

$$e_k^{(i)} = \tilde{x}_k^{(i+1)} - r_k^{(i)} x_k^{(i)} \tag{23}$$

is the a-priori error signal. NLMS is an adaptive estimator and it is widely used because of its simplicity and robustness. It is a normalized version of the well-known LMS algorithm [44], where the normalization is added to guarantee convergence if $\sigma < 2$. In the sequel, we use the learning rate $\sigma = 1$, which is the optimal learning rate in noise-free NLMS [44].

It is noted that the $r_k^{(i)}$'s will generally be different for different nodes if $i < \infty$, whereas each node should use an identical value such that the generated vector are proportional to those in (19)-(20). This can be achieved by means of consensus averaging (CA) algorithms, as done in, e.g., [39]. However, by using CA techniques, exact consensus can only be reached after an infinite amount of iterations. To ensure that each node uses exactly the same value for p in a finite number of iterations, we propose the use of a beacon node. This beacon node can be an arbitrary node q who will determine the actual p, i.e.,

$$p \leftarrow r_q^{(i-P)} \tag{24}$$

in (20) at regular intervals of P iterations. Assuming that a new future value for p can be disseminated over the network at one hop[7] per iteration i, then it takes H_q iterations before all nodes have access to this new value of p, where

$$H_q = \max_{k \in \mathcal{K}} \min_{\Pi \in \mathcal{P}_{qk}} |\Pi| \tag{25}$$

with $\mathcal{P}_{qk}$ denoting the set of all paths between node q and node k, and with $|\Pi|$ denoting the length of path Π. Therefore, it is assumed that

$$P \geq H_q \tag{26}$$

which can be easily guaranteed by either computing H_q itself[8], or an upperbound for H_q (e.g., K). It is recommended (but not required) to choose a central node (with small H_q) to serve as the beacon node, such that P can be kept small. Note that the assignment of a beacon node does not contradict the distributed nature of the algorithm (in principle, any node can be selected as a beacon node, and the choice of q can even change during operation of the algorithm). Furthermore, as the beacon node is only used to determine a shrinking rate, it only has an influence on the dynamic range of the entries in $\mathbf{x}^{(i)}$, but not on the convergence speed of the PI towards $\mathbf{x}_F$.

Finally, to obtain a better (and more robust) estimate of $r^{(i)}$ at the beacon node q, we use a cooperative diffusion strategy [4], where the nodes exchange their $r_k^{(i)}$'s with their neighbors and average these after each iteration of (22):

$$\psi_k^{(i+1)} = r_k^{(i)} + \frac{x_k^{(i)}\left(\tilde{x}_k^{(i+1)} - r_k^{(i)} x_k^{(i)}\right)}{\max\left(\delta, \left(x_k^{(i)}\right)^2\right)} \tag{27}$$

$$r_k^{(i+1)} = \frac{1}{1+|\mathcal{N}_k|}\left(\psi_k^{(i+1)} + \sum_{n \in \mathcal{N}_k} \psi_n^{(i+1)}\right) . \tag{28}$$

It is empirically found that this provides a far more robust estimate of $r^{(i)}$ compared to the case where an isolated NLMS algorithm is used at the beacon node q (in particular when i is small). Using an isolated NLMS algorithm at node q often yields a signficant under- or overestimation of $r^{(i)}$, resulting in under- or overflow in the $x_k^{(i)}$'s.

In [10], Fiedler shows that

$$\lambda_2 \leq \theta = \frac{K}{K-1} \min_{k \in \mathcal{K}} |\mathcal{N}_k| \tag{29}$$

and therefore

$$1 - \frac{\theta}{\alpha} \leq \mu_{K-1} \leq 1 . \tag{30}$$

For large networks where K is usually not known[9], the factor $\frac{K}{K-1} \approx 1$ in (29) and can therefore be omitted, in which case θ can be computed by means of a simple min-consensus algorithm [43], allowing us to exploit (30) in the diffusion NLMS algorithm. Based on (21), the $r_k^{(i)}$'s should ideally converge to μ_{K-1}, and therefore we can use the bounds in (30) to correct the values in (27), i.e., (27)-(28) is replaced with

$$\psi_k^{(i+1)} = \mathcal{B}_{[1-\frac{\theta}{\alpha},1]}\left(r_k^{(i)} + \frac{x_k^{(i)}\left(\tilde{x}_k^{(i+1)} - r_k^{(i)} x_k^{(i)}\right)}{\max\left(\delta, \left(x_k^{(i)}\right)^2\right)}\right) \tag{31}$$

$$r_k^{(i+1)} = \frac{1}{1+|\mathcal{N}_k|}\left(\psi_k^{(i+1)} + \sum_{n \in \mathcal{N}_k} \psi_n^{(i+1)}\right) . \tag{32}$$

where $\mathcal{B}_{[a,b]}(x)$ denotes the projection operator

$$\mathcal{B}_{[a,b]}(x) = \min\left(b, \max(x,a)\right) . \tag{33}$$

It is noted that this projection operator can be omitted if θ is not known.

We end this subsection by showing that the iterations (31)-(32) will eventually converge to μ_{K-1} in every node. If the sequence $\{\mathbf{x}^{(i)}\}_{i \in \mathbb{N}}$ converges to a vector that is proportional to the Fiedler vector $\mathbf{x}_F$ (see next subsection), then (19) implies that

$$\forall\, k \in \mathcal{K} : \lim_{i \to \infty}\left(\tilde{x}_k^{(i+1)} - \mu_{K-1} x_k^{(i)}\right) = 0 . \tag{34}$$

Therefore, $\tilde{x}_k^{(i+1)}$ can be replaced with $\mu_{K-1} x_k^{(i)}$ in the righthand side of (31) when $i \to \infty$, which yields

$$\mathcal{B}_{[1-\frac{\theta}{\alpha},1]}\left(\left(1-\epsilon_k^{(i)}\right) r_k^{(i)} + \epsilon_k^{(i)} \mu_{K-1}\right) \tag{35}$$

[7] It is noted that faster dissemination protocols, i.e., decoupled from the iteration index i, may improve performance of the algorithm in Subsection III-C.

[8] For example, by letting each node forward a copy of a time-stamped message (originating at node q) to each of its neighbors. Each node can then check how many iterations have passed before it receives a copy of this message. H_q can then be computed by means of a max-consensus iteration [43].

[9] The distributed computation of the network size K is in itself a problem [45].

where

$$\epsilon_k^{(i)} = \frac{\left(x_k^{(i)}\right)^2}{\max\left(\delta, \left(x_k^{(i)}\right)^2\right)} . \tag{36}$$

If δ is chosen small enough, i.e., $\delta < \left(x_k^{(i)}\right)^2$, then (35) reduces to $\mathcal{B}_{[1-\frac{\varrho}{\alpha},1]}(\mu_{K-1}) = \mu_{K-1}$ (see also (30)). Therefore, (31) will result in the same value at all nodes, in which case the diffusion step (32) has no influence anymore, i.e.,

$$\forall\, k \in \mathcal{K} : \lim_{i\to\infty} r_k^{(i)} = \mu_{K-1} . \tag{37}$$

In the rare cases where $\delta > \left(x_k^{(i)}\right)^2$ for an infinite amount of iterations, both the update (31) and the diffusion step (32) will then correct this value over time since the $\psi_n^{(i)}$'s of the other nodes independently converge to μ_{K-1} when $i \to \infty$. This shows that the (constrained) diffusion NLMS algorithm (31)-(32) will converge to the same value $r_k^{(\infty)} = \mu_{K-1}$ in each node $k \in \mathcal{K}$.

B. Mean correction

In this subsection, we explain how a distributed mean correction can be performed on a vector $\mathbf{x}^{(i)}$ in a single iteration without requiring all nodes to know the mean of $\mathbf{x}^{(i)}$. For the sake of an easy exposition, we ignore the normalization and we use the notation '$\propto$' to denote proportionality (i.e., equality up to a non-zero scaling).

Because $\frac{1}{\sqrt{K}}\mathbf{1}_K$ is the eigenvector corresponding to $\lambda_1 = 0$ of $\mathbf{L}$, it follows that

$$\mathbf{1}_K^T\mathbf{L} = \mathbf{0} \tag{38}$$

and therefore

$$\mathbf{1}_K^T\mathbf{L}\mathbf{x} = 0,\ \forall\, \mathbf{x} \in \mathbb{R}^K \tag{39}$$

which means that the sum of the entries in $\mathbf{L}\mathbf{x}$ is always zero. Therefore, a mean-corrected vector $\mathbf{x}^{(i+1)}$ can be computed by applying the update

$$\mathbf{x}^{(i+1)} = \mathbf{L}\mathbf{x}^{(i)} . \tag{40}$$

It is noted that (40) generally shifts the vector $\mathbf{x}^{(i)}$ towards the eigenvector corresponding to λ_K instead of $\mathbf{x}_F$. Hence, applying (40) too often may result in a non-converging algorithm. The following theorem quantifies the maximum frequency with which (40) can be applied such that the overall PI procedure still converges to the Fiedler vector $\mathbf{x}_F$.

Theorem III.1. *Consider the following iterative procedure:*

$$\mathbf{x}^{(i+1)} = \begin{cases} \mathbf{L}\mathbf{x}^{(i)} & \textit{if } (i \mod N) = 0 \\ \mathbf{M}\mathbf{x}^{(i)} & \textit{otherwise} \end{cases} \tag{41}$$

Assuming that $\mathbf{1}^T\mathbf{x}^{(0)} \neq 0$*, then* $\mathbf{x}^{(i)} \propto \mathbf{x}_F$ *when* $i \to \infty$ *if and only if*

$$N > \frac{\log\left(\frac{\lambda_3}{\lambda_2}\right)}{\log\left(\frac{\alpha-\lambda_2}{\alpha-\lambda_3}\right)} + 1 . \tag{42}$$

Proof: It is noted that the sequence generated by (41) contains a subsequence that can be generated by the following PI process

$$\mathbf{y}^{(j+1)} = \mathbf{V}\mathbf{y}^{(j)} \tag{43}$$

where

$$\mathbf{V} = \mathbf{M}^{N-1}\mathbf{L} \tag{44}$$

and where $\mathbf{y}^{(0)} = \mathbf{x}^{(0)}$. We will prove that $\mathbf{y}^{(j)} \propto \mathbf{x}_F$ when $j \to \infty$, which also implies $\mathbf{x}^{(i)} \propto \mathbf{x}_F$ when $i \to \infty$ in (41), since $\mathbf{M}\mathbf{x}_F \propto \mathbf{L}\mathbf{x}_F \propto \mathbf{x}_F$ (the latter holds because $\mathbf{x}_F$ is an eigenvector of both $\mathbf{M}$ and $\mathbf{L}$).

Let the eigenvalue decomposition of $\mathbf{L}$ be given as

$$\mathbf{L} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T \tag{45}$$

with $\mathbf{\Lambda} = \mathrm{Diag}(0, \lambda_2, \ldots, \lambda_K)$ (eigenvalues in increasing order) and $\mathbf{Q}$ an orthogonal matrix with the corresponding eigenvectors of $\mathbf{L}$ in its columns. From (7), it is observed that

$$\mathbf{V} = \mathbf{M}^{N-1}\mathbf{L} = \mathbf{Q}\left(\mathbf{I}_K - \frac{1}{\alpha}\mathbf{\Lambda}\right)^{N-1}\mathbf{\Lambda}\mathbf{Q}^T . \tag{46}$$

Therefore, we find that $\mathbf{V}$ has the set of eigenvalues

$$\nu_n = f(\lambda_n),\ n = 1, \ldots, K \tag{47}$$

where

$$f(\lambda) = \lambda(1 - \frac{\lambda}{\alpha})^{N-1} \tag{48}$$

and the eigenvector of $\mathbf{V}$ corresponding to ν_n can be found in the n-th column of $\mathbf{Q}$. Note that $\nu_1 = 0$, which is the eigenvalue of $\mathbf{V}$ corresponding to the eigenvector $\frac{1}{\sqrt{K}}\mathbf{1}_K$, hence the sequence $\{\mathbf{y}^{(j)}\}_{j\in\mathbb{N}}$ generated by (43) will have zero mean. Furthermore, $\mathbf{x}_F$ is also an eigenvector of $\mathbf{V}$, corresponding to the eigenvalue ν_2. To obtain $\mathbf{x}_F$ as the solution of the PI process (43), ν_2 must be the dominant eigenvalue, i.e.,

$$f(\lambda_2) > \max_{n\in\{3,\ldots,K\}} f(\lambda_n) . \tag{49}$$

The function $f(\lambda)$ is continuous and has one stationary point (a maximum at $\lambda = \frac{\alpha}{N}$) within the interval $(0, \alpha)$. This is illustrated in Fig. 2 for $N = 10$ and $\alpha = 5$. If $f(\lambda_2) > f(\lambda_3)$ holds (see (49)), then λ_3 must be in the decreasing part of the function f, and therefore

$$f(\lambda_2) > f(\lambda_3) \Rightarrow \forall n \in \{3, \ldots, K\} : f(\lambda_2) > f(\lambda_n) . \tag{50}$$

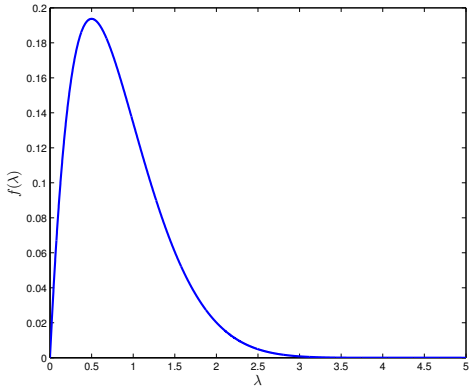


Fig. 2. The shape of $f(\lambda)$ when $N = 10$ and $\alpha = 5$.

With this, expression (49) can be replaced by

$$\lambda_2(1-\frac{\lambda_2}{\alpha})^{N-1} > \lambda_3(1-\frac{\lambda_3}{\alpha})^{N-1} \ . \tag{51}$$

By taking the logarithm of both sides (exploiting the monotonicity of the logarithmic function), and using some algebraic manipulations, we straightforwardly find that

$$N > \frac{\log\left(\frac{\lambda_3}{\lambda_2}\right)}{\log\left(\frac{\alpha-\lambda_2}{\alpha-\lambda_3}\right)} + 1 \tag{52}$$

which proves the theorem. ■

Remark II: Since the function $f(\lambda)$ as defined in (48) has a maximum in $\lambda = \frac{\alpha}{N}$ (which is the only stationary point in the interval $(0, \alpha)$), we know that $\lambda_2 > \frac{\alpha}{N}$ also implies that $f(\lambda_2) > f(\lambda_3)$ (See Fig. 2). Therefore, we can also define the (less tight) lower bound

$$N > \frac{\alpha}{\lambda_2} \tag{53}$$

to guarantee that $\mathbf{x}^{(i)} \propto \mathbf{x}_F$ when $i \to \infty$ in (41).

Remark III: The choice of N has an influence on the convergence speed of the overall PI algorithm since it affects the eigenvalues of $\mathbf{V}$, as defined in the proof of Theorem III.1. Indeed, the convergence speed of a PI process depends on the ratio between the largest and the one but largest eigenvalue. Since one multiplication with $\mathbf{V}$ requires N multiplications in (41), we should consider the ratio

$$\left(\frac{\nu_2}{\nu_3}\right)^{\frac{1}{N}} \tag{54}$$

to measure the convergence speed with respect to the iterations in (41). With (47), we find that the convergence speed is determined by the quantity

$$\left(\frac{\lambda_2}{\lambda_3}\right)^{\frac{1}{N}} \left(\frac{\alpha-\lambda_2}{\alpha-\lambda_3}\right)^{\frac{N}{N-1}} \ . \tag{55}$$

We observe that for $N \to \infty$, the convergence speed will be determined by $\frac{\alpha-\lambda_2}{\alpha-\lambda_3}$, independent of N.

C. Final algorithm

The core of the algorithm is the iteration (41) with the inclusion of a proper compensation for the shrinking of the norm of $\mathbf{x}^{(i)}$, i.e.,

$$\mathbf{x}^{(i+1)} = \begin{cases} \frac{1}{\alpha|1-p|}\mathbf{L}\mathbf{x}^{(i)} & \text{if } (i \mod N) = 0 \\ \frac{1}{p}\mathbf{M}\mathbf{x}^{(i)} & \text{otherwise} \end{cases} \tag{60}$$

where p is updated by means of the shrinking rate estimator $r_q^{(i)}$ as computed in a beacon node q using a diffusion NLMS algorithm (see Subsection III-A). The exact per-node tasks are described in the final algorithm description in Table II, where (60) is represented in (56). The vector $\mathbf{g}^{(i)}$ is introduced to store the most recent value of the r_q that is currently disseminated over the network (where q is the beacon node). The initialization $r_k^{(0)} = 1 - \frac{1}{3}\frac{\theta}{\alpha}$ is inspired by (30), and is empirically found to be a good initialization (although not crucial). It is noted that the NLMS algorithm does not update during the mean correction step.

If N satisfies (42) or (53) and if P satisfies (26), then the sequence $\{\mathbf{x}^{(i)}\}_{i\in\mathbb{N}}$ generated by the algorithm in Table II will converge to a stable vector $\mathbf{x}^{(\infty)}$ which is proportional to the Fiedler vector $\mathbf{x}_F$ of the network graph, i.e., $\mathbf{x}^{(\infty)} \propto \mathbf{x}_F$. Indeed, the condition (26) implies that the p_k's,$\forall\, k \in \mathcal{K}$, are equal in each iteration. Applying the scaling with $\frac{1}{p_k}$ in (56) is then equivalent to a scaling of the full vector $\mathbf{x}^{(i)}$, which does not harm the PI process (41). Once $r_q^{(i)}$ is approximately equal to μ_{K-1} (see (37)), the norm of $\mathbf{x}^{(i)}$ will not change anymore, yielding a stable solution.

Remark IV: The convergence speed of $\mathbf{x}^{(i)}$ towards the Fiedler vector is completely independent of the convergence speed of the diffusion NLMS steps (57) and (58). Indeed, the latter only determines how well the shrinking rate can be tracked which only influences the norm of $\mathbf{x}^{(i)}$.

Remark V: It is noted that the algebraic connectivity, i.e., the value of λ_2, is available in each node after convergence of the algorithm. Indeed, since the p_k's converge to $p_k = \mu_{K-1} = 1 - \frac{\lambda_2}{\alpha}$, each node can compute λ_2.

Remark VI: Since the algorithm converges for any initialization, it can adapt to changes in the network topology if these changes are slow compared to the convergence speed of the algorithm.

Remark VII: It is noted that the above algorithm does not result in a normalized Fiedler vector $\mathbf{x}_F$, but a scaled version thereof. However, if a normalized version is required, a consensus averaging procedure [2] or a push-sum algorithm [26] can be used to compute the norm of $\mathbf{x}^{(i)}$ in a distributed fashion (see, e.g., [39]).

Remark VIII: The worst-case computational complexity at node k is equal to $21 + 2|\mathcal{N}_k|$ floating point operations (flops) per iteration.

Remark IX: The proposed algorithm is not robust against recurring random node or link failures because these partially annihilate the mean-preserving characteristic of the matrix $\mathbf{M}$, which is an important requirement in the algorithm derivation (the algorithm can only recover from such errors after each mean correction step, i.e., every N iterations). Therefore, the proposed algorithm requires robust communication protocols (e.g., with re-transmissions if a data packet has not been received by its neighbors).

IV. Simulations

In this section, we provide results of a Monte-Carlo (MC) simulation[10] of the distributed algorithm presented in Section III. In each MC trial, we construct a random network graph with $K = 10 + L_1$ nodes and with an average[11] of $3 + L_2$ links per node where L_1 and L_2 are random integers drawn from $\{1, 2, \ldots, 10\}$ and $\{1, 2, 3\}$, respectively. The random network is created by first constructing a random spanning tree and then adding links between randomly chosen node pairs until the predetermined average number of links per node is achieved. The value of N is set to $N = 50 + L_3$ where L_3 is a random integer drawn from $\{1, 2, \ldots, 50\}$.

[10] Matlab code to reproduce these results is freely available at *http://homes.esat.kuleuven.be/~abertran/software.html*.

[11] If $3 + L_2 > \frac{K}{3}$, the average number of links per node is set to the integer closest to $\frac{K}{3}$ to avoid that the network is too densely connected.

TABLE II
DISTRIBUTED COMPUTATION OF THE FIEDLER VECTOR

1) Initialization: choose a beacon node $q \in \mathcal{K}$, set $i \leftarrow 0$ and set $r_k^{(0)} = g_k^{(0)} = 1 - \frac{1}{3}\frac{\theta}{\alpha}$, $p_k \leftarrow r_k^{(0)}$, $\forall\, k \in \mathcal{K}$. Initialize $\mathbf{x}^{(0)}$ with random entries.
2) If $(i \mod P) = P - 1$: at all nodes $k \in \mathcal{K}$, set $p_k \leftarrow g_k^{(i)}$
3) Each node $k \in \mathcal{K}$ transmits $x_k^{(i)}$ to its neighbors in $\mathcal{N}_k$ and computes

$$x_k^{(i+1)} = \begin{cases} \frac{1}{\alpha|1-p_k|}\left(|\mathcal{N}_k| x_k^{(i)} - \sum_{q\in\mathcal{N}_k} x_q^{(i)}\right) & \text{if } (i \mod N) = 0 \\ \frac{1}{p_k}\left(\left(1 - \frac{1}{\alpha}|\mathcal{N}_k|\right) x_k^{(i)} + \frac{1}{\alpha}\sum_{q\in\mathcal{N}_k} x_q^{(i)}\right) & \text{otherwise} \end{cases} \tag{56}$$

$$\psi_k^{(i+1)} = \begin{cases} r_k^{(i)} & \text{if } (i \mod N) = 0 \\ \mathcal{B}_{[1-\frac{\theta}{\alpha},1]}\left(r_k^{(i)} + \frac{x_k^{(i)}\left(p_k x_k^{(i+1)} - r_k^{(i)} x_k^{(i)}\right)}{\max\left(\delta, \left(x_k^{(i)}\right)^2\right)}\right) & \text{otherwise} \end{cases} \tag{57}$$

4) Each node $k \in \mathcal{K}$ transmits $\psi_k^{(i+1)}$ to its neighbors in $\mathcal{N}_k$ and computes

$$r_k^{(i+1)} = \frac{1}{1 + |\mathcal{N}_k|}\left(\psi_k^{(i+1)} + \sum_{n\in\mathcal{N}_k} \psi_n^{(i+1)}\right) \tag{58}$$

5) If $(i \mod P) = 0$: node q sets $g_q^{(i+1)} = r_q^{(i)}$ and transmits $g_q^{(i+1)}$ to its neighbors in $\mathcal{N}_q$.
6) At all nodes $k \in \mathcal{K}$: if a node $n \in \mathcal{N}_k$ has transmitted $g_n^{(i)} \neq p_k$, then

$$g_k^{(i+1)} = g_n^{(i)} \tag{59}$$

and node k transmits $g_k^{(i+1)}$ to its neighbors in $\mathcal{N}_k$. Otherwise, $g_k^{(i+1)} = g_k^{(i)}$.
7) $i \leftarrow i + 1$
8) Return to step 2.

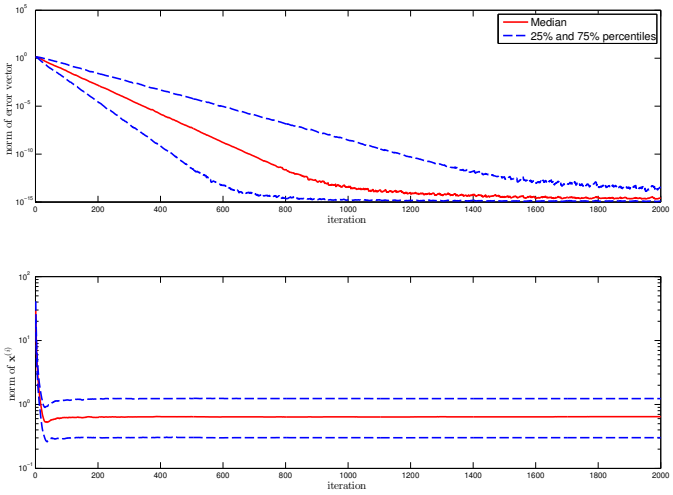


Fig. 3. Convergence properties of the distributed algorithm that computes the Fiedler vector in small-scale networks up to 20 nodes (results based on 1000 MC trials).

Fig. 3 shows the convergence properties of the algorithm over 2000 iterations. The three lines indicate the quartiles over 1000 MC trials (the dashed lines correspond to the 25% and 75% percentiles, and the full line corresponds to the median, i.e., the 50% percentile). The upper plot shows the norm of the error vector between $\mathbf{x}^{(i)}$ and $\mathbf{x}_F$ over the different iterations, i.e.,

$$\left\| \frac{\mathbf{x}^{(i)}}{\|\mathbf{x}^{(i)}\|} \pm \frac{\mathbf{x}_F}{\|\mathbf{x}_F\|} \right\| \tag{61}$$

where $\pm$ resolves the sign ambiguity. It is observed that this error converges to zero (up to machine precision), hence the algorithm indeed finds the exact Fiedler vector (notice that full convergence is often not even required, e.g., for node clustering based on the sign of the entries in $\mathbf{x}_F$). The lower plot shows the norm $\|\mathbf{x}^{(i)}\|$ over the different iterations, demonstrating that it does neither vanish nor diverge.

Fig. 4 shows similar results for a large-scale network of $K = 150$ nodes over 1000 MC trials. In each MC trial, three networks of 50 nodes each and an average of 4 links per node are randomly generated using the same procedure as mentioned earlier. The three networks are then linked together to form a single network with 3 distinct clusters. Two of these clusters are connected with each other by 20 random links, and the third cluster is connected to the other two with 5 random links each. The value of N is set to $N = 150$ in each MC trial. It is observed that, even in large networks, the algorithm converges relatively quickly to the Fiedler vector. The sawtooth shape demonstrates the effect of the mean correction after every $N = 150$ iterations.

Fig. 5 compares the performance of the proposed algorithm and the orthogonal iteration (OI) algorithm in [26], in the same MC experiment as in Fig. 4. The OI algorithm can be used to compute the two dominant eigenvectors of $\mathbf{M}$ (where

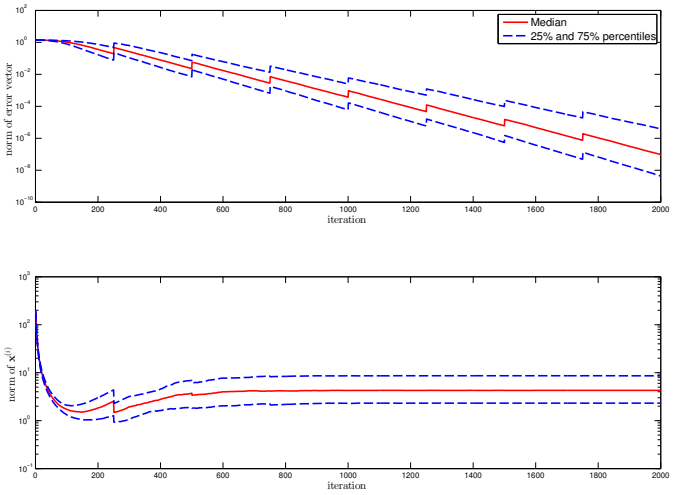


Fig. 4. Convergence properties of the distributed algorithm that computes the Fiedler vector in a large-scale network of 150 nodes (results based on 1000 MC trials).

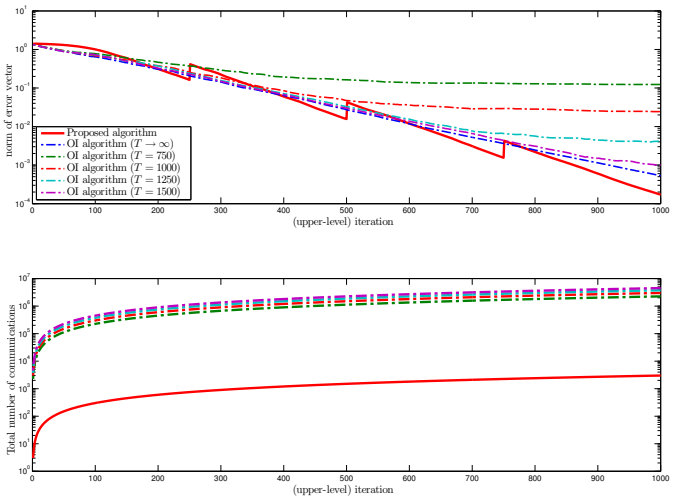


Fig. 5. Comparison of the convergence properties and communication requirements of the proposed algorithm and the OI algorithm in [26] (results based on 200 MC trials).

the second eigenvector corresponds to the Fiedler vector). The OI algorithm computes a PI on a two-column matrix ($\mathbf{X}^{(i+1)} = \mathbf{M}\mathbf{X}^{(i)}$), where a distributed orthogonalization procedures is performed on the two columns of $\mathbf{X}^{(i+1)}$ after every in-network multiplication with $\mathbf{M}$. This requires a nested loop of T iterations after each upper-level iteration. The required value of T will depend on the network size and its connectivity (in this experiment, it is empirically found that T should be larger than 750 to obtain sufficiently accurate results, see Fig. 5). Although it appears that the proposed algorithm and the OI have a similar convergence speed[12], it should be emphasized that Fig. 5 only shows the upper-level iterations. Since the OI algorithm actually performs T iterations for each upper-level iteration increment in Fig. 5, it has a much slower overall convergence (T times slower). The lower plot shows the total number of transmissions (over all nodes) after a certain number of iterations. Notice that the OI algorithm requires significantly more data transmission compared to the proposed algorithm (three orders of magnitude higher). This is again due to the T nested iterations after each upper-level iteration.

To obtain a perfect orthogonalization, T should in principle be infinitely large (the performance for $T \to \infty$ is also depicted in Fig. 5). In practice however, T will have a finite value, which will introduce cut-off errors which propagate to the upper-level PIs. This will result in a saturation behavior in terms of accuracy, which can be seen for all curves in Fig. 5 for which $T < \infty$.

[12]This is not surprising as both algorithms perform PIs based on the same matrix $\mathbf{M}$.

V. Conclusions

We have addressed how the Fiedler vector of a network graph, i.e., the eigenvector corresponding to the smallest non-trivial eigenvalue of the Laplacian matrix, can be used for topology inference in ad hoc networks, e.g., as a heuristic for node clustering or to identify bottleneck links in the data diffusion over the network. We have proposed a distributed algorithm for in-network computation of the Fiedler vector of the corresponding network graph, which is based on a combination of power iterations with occasional mean correction steps. We have explained how the growth or shrinking rate can be estimated to counteract the diverging or vanishing tendency of the vector norm. Monte-Carlo simulation results have confirmed that the algorithm converges to the Fiedler vector.

In future work, we will consider the non-ideal case where links may temporarily fail at random. The current version of the algorithm is not robust agains this problem due to the fact that link failures may introduce errors that destroy the orthogonality with the all-ones vector, i.e., the zero-mean property. A next step is then a modification of the algorithm towards an asynchronous setting, e.g., based on principles explained in [39].

References

[1] D. Estrin, L. Girod, G. Pottie, and M. Srivastava, "Instrumenting the world with wireless sensor networks," *Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on*, vol. 4, pp. 2033–2036 vol.4, 2001.

[2] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems and Control Letters*, vol. 53, no. 1, pp. 65 – 78, 2004.

[3] L. Xiao, S. Boyd, and S. Lall, "A scheme for robust distributed sensor fusion based on average consensus," in *Proc. International Symposium on Information Processing in Sensor Networks (ISPN)*, 2005, pp. 63 – 70.

[4] F. S. Cattivelli and A. H. Sayed, "Diffusion LMS strategies for distributed estimation," *IEEE Transactions on Signal Processing*, vol. 58, pp. 1035–1048, March 2010.

[5] A. Bertrand, M. Moonen, and A. H. Sayed, "Diffusion bias-compensated RLS estimation over adaptive networks," *IEEE Transactions on Signal Processing*, vol. 59, no. 11, pp. 5212 –5224, Nov. 2011.

[6] G. Mateos, I. D. Schizas, and G. B. Giannakis, "Performance analysis of the consensus-based distributed LMS algorithm," *EURASIP Journal on Advances in Signal Processing*, vol. 2009, Article ID 981030, 19 pages, 2009. doi:10.1155/2009/981030.

[7] A. Bertrand and M. Moonen, "Consensus-based distributed total least squares estimation in ad hoc wireless sensor networks," *IEEE Trans. Signal Processing*, vol. 59, no. 5, pp. 2320–2330, May 2011.

[8] F. Chung, *Spectral Graph Theory*. American Mathematical Society, 1997.

[9] A. Bertrand and M. Moonen, "Seeing the bigger picture: How nodes can learn their place within a complex ad hoc network topology," *IEEE Signal Processing Magazine*, 2013 (in press).

[10] M. Fiedler, "Algebraic connectivity of graphs," *Czechoslovak Mathematical Journal*, vol. 23, no. 98, pp. 298–305, 1973.

[11] R. Aragues, G. Shi, D. V. Dimarogonas, C. Sagues, and K. H. Johansson, "Distributed algebraic connectivity estimation for adaptive event-triggered consensus," in *Proc. American Control Conference*, Fairmont Queen Elizabeth, Montreal, Canada, June 2012, pp. 32–37.

[12] T. Sahai, A. Speranzon, and A. Banaszuk, "Hearing the clusters of a graph: A distributed algorithm," *Automatica*, vol. 48, no. 1, pp. 15 – 24, 2012.

[13] A. Muhammad and A. Jadbabaie, "Decentralized computation of homology groups in networks by gossip," in *American Control Conference (ACC)*, july 2007, pp. 3438 –3443.

[14] S. E. Schaeffer, "Graph clustering," *Computer Science Review*, vol. 1, pp. 27–64, 2007.

[15] P. Orponen and S. Schaeffer, "Local clustering of large graphs by approximate Fiedler vectors," in *Experimental and Efficient Algorithms*, ser. Lecture Notes in Computer Science, S. Nikoletseas, Ed. Springer Berlin Heidelberg, 2005, vol. 3503, pp. 524–533.

[16] M. Bojan, "Laplace eigenvalues of graphs - a survey," *Discrete Mathematics*, vol. 109, no. 13, pp. 171 – 183, 1992.

[17] T. F. Chan, T. C. Ciarlet, and W. K. Szeto, "On the optimality of the median cut spectral bisection graph partitioning method," *SIAM Journal on Scientific Computing*, vol. 18, pp. 943–948, 1997.

[18] M. Holzrichter and S. Oliveira, "A graph based method for generating the fiedler vector of irregular problems," in *In Lecture Notes in Computer Science*. Lecture, 1999, pp. 978–985.

[19] D. A. Spielman and S.-H. Teng, "Spectral partitioning works: Planar graphs and finite element meshes," *Linear Algebra and its Applications*, vol. 421, no. 23, pp. 284 – 305, 2007.

[20] L. Hagen and A. Kahng, "New spectral methods for ratio cut partitioning and clustering," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 9, pp. 1074 –1085, sep 1992.

[21] U. Brandes and S. Cornelsen, "Visual ranking of link structures," *Journal of Graph Algorithms and Applications*, vol. 7, no. 2, pp. 181–201, 2003.

[22] C. Gkantsidis, G. Goel, M. Mihail, and A. Saberi, "Towards topology aware networks," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, may 2007, pp. 2591–2595.

[23] A. Ghosh and S. Boyd, "Growing well-connected graphs," in *IEEE Conference on Decision and Control*, dec. 2006, pp. 6605 –6611.

[24] C. Asensio-Marco and B. Beferull-Lozano, "A greedy perturbation approach to accelerating consensus algorithms and reducing its power consumption," in *IEEE Statistical Signal Processing Workshop (SSP)*, june 2011, pp. 365 –368.

[25] M. De Gennaro and A. Jadbabaie, "Decentralized control of connectivity for multi-agent systems," in *Proc. IEEE Conference on Decision and Control*, dec. 2006, pp. 3628 –3633.

[26] D. Kempe and F. McSherry, "A decentralized algorithm for spectral analysis," in *Proc. ACM symposium on Theory of computing*, ser. STOC '04. New York, NY, USA: ACM, 2004, pp. 561–568.

[27] T. Sahai, A. Speranzon, and A. Banaszuk, "Wave equation based algorithm for distributed eigenvector computation," in *IEEE Conference on Decision and Control (CDC)*, dec. 2010, pp. 7308 –7315.

[28] M. Franceschelli, A. Gasparri, A. Giua, and C. Seatzu, "Decentralized laplacian eigenvalues estimation for networked multi-agent systems," in *Proc. IEEE Conference on Decision and Control*, dec. 2009, pp. 2717 –2722.

[29] M. Newman, "The laplacian spectrum of graphs," University of Manitoba, Winnipeg, Canada, July 2000.

[30] S. Barnard and H. Simon, "Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," *Concurrency and computation: Practice and Experience*, vol. 6, p. 101117, 1994.

[31] H. Simon, "Partitioning of unstructured problems for parallel processing," *Computing Systems in Engineering*, vol. 2, no. 23, pp. 135 – 148, 1991.

[32] J. Martin and M. Bojan, "Optimal linear labelings and eigenvalues of graphs," *Discrete Applied Mathematics*, vol. 36, no. 2, pp. 153 – 168, 1992.

[33] R. Olfati-Saber, J. Fax, and R. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215 –233, jan. 2007.

[34] J. Lavaei and R. Murray, "On quantized consensus by means of gossip algorithm - part II: Convergence time," in *American Control Conference*, june 2009, pp. 2958 –2965.

[35] S. Guattery and G. L. Miller, "On the quality of spectral separators," *SIAM Journal on Matrix Analysis and Applications*, vol. 19, no. 3, pp. 701–719, July 1998.

[36] M. S. Talebi, M. Kefayati, B. H. Khalaj, and H. R. Rabiee, "Adaptive consensus averaging for information fusion over sensor networks," in *Proc. IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, 2006.

[37] P. Braca, S. Marano, and V. Matta, "Running consensus in wireless sensor networks," in *Proc. Int. Conf. on Information Fusion*, July 2008, pp. 1 –6.

[38] D. Shah, "Gossip algorithms," *Foundations and Trends in Networking*, vol. 3, pp. 1–125, 2009.

[39] M. Jelasity, G. Canright, and K. Engo-Monsen, *Asynchronous distributed power iteration with gossip-based normalization*. Springer, 2007, vol. 4641, pp. 514–525.

[40] X.-D. Zhang, "The laplacian eigenvalues of graphs: a survey," in *Linear Algebra Research Advance*. Nova Science Publishers, INC., 2007, pp. 201–228.

[41] R. Horn and C. Johnson, *Matrix Analysis*. Cambride, UK: Cambridge University Press, 1999.

[42] R. Merris, "A note on laplacian graph eigenvalues," *Linear Algebra and its Applications*, vol. 285, no. 13, pp. 33 – 35, 1998.

[43] A. Tahbaz-Salehi and A. Jadbabaie, "A one-parameter family of distributed consensus algorithms with boundary: From shortest paths to mean hitting times," in *Proc. IEEE Conference on Decision and Control*, dec. 2006, pp. 4664 –4669.

[44] S. Haykin, *Adaptive Filter Theory*, 4th ed. Prentice Hall, 2002.

[45] J. Cardoso, C. Baquero, and P. Almeida, "Probabilistic estimation of network size and diameter," in *Proc. Latin-American Symposium on Dependable Computing*, sept. 2009, pp. 33 –40.