EUSIPCO 2013 1569743291

TOPOLOGY-AWARE DISTRIBUTED ADAPTATION OF LAPLACIAN WEIGHTS FOR IN-NETWORK AVERAGING

Alexander Bertrand and Marc Moonen

KU Leuven, Dept. of Electrical Engineering-ESAT, SCD-SISTA
Kasteelpark Arenberg 10, B-3001 Leuven, Belgium
E-mail: alexander.bertrand@esat.kuleuven.be
marc.moonen@esat.kuleuven.be

ABSTRACT

Laplacian weights are often used in distributed algorithms to fuse intermediate estimates of linked agents or nodes in a network. We propose a topology-aware (TA) distributed algorithm for on-line adaptation of the Laplacian weighting rule, when applied in an in-network averaging procedure. We demonstrate that the particular structure of the Laplacian weighting rule indeed allows for a distributed convergence rate optimization, based on the in-network computation of two eigenvectors of the Laplacian matrix and their corresponding eigenvalues. Although the proposed TA distributed algorithm cannot always reach the same (optimal) weights as its centralized equivalent, simulations demonstrate that it still provides a significant improvement on the convergence speed when compared to more general combination weights.

Index Terms— Distributed learning, Fiedler vector, Laplacian weights, consensus averaging

1. INTRODUCTION

Distributed learning and distributed estimation have become important topics within the field of signal processing [1–6], mainly due to the increased popularity of multi-agent systems and wireless sensor networks and their application in, a.o., environmental monitoring, surveillance, robotics, etc.

The performance of a distributed learning or estimation algorithm generally depends on the topology[1] of the network in which it is operated. If the topology is known a priori, it is sometimes possible to optimize the algorithm's parameter settings with respect to the topology (see, e.g., [3]). However, as the network is often deployed in an ad hoc fashion, its resulting topology may be unknown at design time, and sometimes the topology may even change during operation of the algorithm. In this case, distributed algorithms usually rely on general 'topology-unaware' (TU) parameter settings, which are suitable for any possible topology. For example, in distributed algorithms where the nodes compute a weighted average between their local estimate and the estimates of their neighbors [1–6], the weighting rule usually only depends on simple quantities that can be easily evaluated on-line, such as the node degrees, the total number of nodes, etc. Examples of such general weighting rules are the Laplacian rule, the maximum-degree rule, the Metropolis rule, etc. (see [1] for an overview). Although these are easy to use, they are not tuned to the actual topology of the network in which they are applied, and hence suboptimal.

It is a non-trivial task to design topology-aware (TA) distributed algorithms that do not require prior knowledge on the network topology. However, by using concepts from spectral graph theory, the nodes can learn some topology-related properties, which can then be used to tune certain parameters of the distributed algorithm [7]. This paper focuses on the use of such techniques to improve the so-called Laplacian weighting rule, which is based on the Laplacian matrix of the network graph. We propose an on-line (adaptive) TA distributed algorithm that optimizes the Laplacian weights at each individual node, in particular for application in the consensus averaging (CA) algorithm [3]. The CA algorithm iteratively computes the network-wide sum and/or average over quantities that are distributed over the different nodes of the network. The CA algorithm and its variations is a common subroutine in many distributed algorithms (see, e.g., [6, 8, 9]), for which a fast convergence is often crucial, especially so when it is used in a nested iteration.

Our algorithm for on-line updating of the Laplacian weights is based on a projected subgradient algorithm, which

Acknowledgements: 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 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 P7/23 (BESTCOM, 2012-2017), Flemish Government iMinds 2013, and Research Project FWO nr. G.0763.12 'Wireless acoustic sensor networks for extended auditory communication'. The scientific responsibility is assumed by its authors.

[1] It is noted that, in stochastic estimation frameworks, e.g., diffusion adaptation [1], the convergence does not only depend on the topology of the network, but also on the stochastic properties of the observations.

requires a distributed computation of two eigenvectors of the Laplacian matrix (the principal eigenvector and the so-called Fiedler (eigen)vector). In contrast to an equivalent centralized implementation of the algorithm, our distributed algorithm cannot always reach the optimal Laplacian weights. However, we will demonstrate that it still provides a significant improvement of the CA convergence speed compared to a CA that adopts frequently used TU combination weights.

2. PROBLEM STATEMENT

2.1. Definitions and notation

We consider a connected ad hoc network with K nodes, where the set of nodes is denoted by $\mathcal{K}$, i.e., $|\mathcal{K}| = K$. We denote $\mathcal{N}_k$ as the set of neighbors of node k, i.e., the nodes that are linked to node k (node k excluded), and $|\mathcal{N}_k|$ is referred to as the degree of node k. We define $\mathbf{I}$ as the identity matrix, and $\mathbf{1}$ as a vector with all entries equal to one (dimensions should be clear from the context).

The adjacency matrix $\mathbf{A} = [a_{kq}]_{K\times K}$ of the network graph is defined as

$$a_{kq} = a_{qk} = \begin{cases} 1 & \text{if } q \in \mathcal{N}_k \\ 0 & \text{otherwise .} \end{cases} \tag{1}$$

Let $\mathbf{W} = [w_{kq}]_{K\times K}$ denote the weighted adjacency matrix, where w_{kq} is a non-negative weight assigned to the link between node k and node q (by definition, $w_{kq} = 0$ if $k \notin \mathcal{N}_q$). If there are no link weights defined, we set $\mathbf{W} = \mathbf{A}$.

The Laplacian matrix $\mathbf{L} = [l_{kq}]_{K\times K}$ is defined as

$$l_{kq} = l_{qk} = \begin{cases} \sum_{j\in\mathcal{N}_k} w_{kj} & \text{if } k = q \\ -w_{kq} & \text{if } q \in \mathcal{N}_k \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The Laplacian matrix $\mathbf{L}$ is always positive semidefinite, and it has a zero eigenvalue $\lambda_1 = 0$ (eigenvalues are sorted in increasing order of magnitude, i.e., $0 = \lambda_1 \le \lambda_2 \le \ldots \le \lambda_K$) with corresponding eigenvector $\frac{1}{\sqrt{K}}\mathbf{1}$. In case of connected graphs, this zero eigenvalue is unique.

2.2. Consensus averaging (CA)

The CA algorithm computes the network-wide average or sum over quantities that are distributed over multiple nodes. To this end, assume that a node $k \in \mathcal{K}$ has access to a numeric value $x_k^{(0)}$, then the goal of the CA algorithm is to compute $\overline{x} = \frac{1}{K}\mathbf{1}^T\mathbf{x}^{(0)}$ (where $\mathbf{x}^{(0)}$ has $x_k^{(0)}$ as its k-th entry). A common approach is to let neighboring nodes exchange intermediate estimates and combine these with their own local estimate by computing a weighted sum, i.e., all nodes $k \in \mathcal{K}$ simultaneously compute $x_k^{(i+1)} = g_{kk}x_k^{(i)} + \sum_{q\in\mathcal{N}_k} g_{kq}x_q^{(i)}$. This can be compactly written in the network-wide equation

$$\mathbf{x}^{(i+1)} = \mathbf{G}\cdot\mathbf{x}^{(i)} \tag{3}$$

with $\mathbf{G} = [g_{kq}]_{K\times K}$ (where $g_{kq} = 0$ if $q \notin \mathcal{N}_k$). This iterative update is repeated until $x_k^{(\infty)} = \overline{x}$, $\forall\, k \in \mathcal{K}$, i.e., the consensus state is reached (see [3] for necessary and sufficient conditions on the weights g_{kq} to achieve this). A popular choice for the weight matrix $\mathbf{G}$ is the Laplacian weighting rule, i.e.,

$$\mathbf{G} = \mathbf{I} - \sigma\mathbf{L} \,. \tag{4}$$

It can be shown that the use of this matrix in (3) will yield convergence to the consensus state if σ takes a value between $0 < \sigma < \frac{2}{\lambda_K}$ [3]. In this case, $\mathbf{G}$ has a unique maximum eigenvalue equal to one, with corresponding eigenvector $\frac{1}{\sqrt{K}}\mathbf{1}$. Since the absolute value of all other eigenvalues is smaller than 1, it holds that $\lim_{i\to\infty}\mathbf{G}^i = \frac{1}{K}\mathbf{1}\mathbf{1}^T$. It is noted that the maximum-degree rule [1] is a special case of (4) where $\mathbf{W} = \mathbf{A}$ such that each link is weighted equally, and $\sigma = 1/K$, which guarantees that $0 < \sigma < \frac{2}{\lambda_K}$ is satisfied.

A fast convergence of the CA algorithm can be of crucial importance, especially so when it is used as a subroutine or in a nested iteration in other distributed algorithms [6, 8, 9]. In the next subsection, we explain how the convergence speed of the CA algorithm can be significantly improved by taking the network topology into account.

3. CENTRALIZED OPTIMIZATION OF THE LAPLACIAN WEIGHTING RULE

3.1. Optimization of σ

Since the CA algorithm relies on a simple power iteration (3), its convergence speed directly depends on the ratio of the largest and one-but-largest eigenvalue of $\mathbf{G}$. To define convergence more formally, we use the so-called asymptotic convergence factor [3]

$$\rho = \sup_{\mathbf{x}\neq\overline{x}\mathbf{1}} \lim_{i\to\infty} \left(\frac{\|\mathbf{x}^{(i)} - \overline{x}\mathbf{1}\|}{\|\mathbf{x}^{(0)} - \overline{x}\mathbf{1}\|}\right)^{\frac{1}{i}} \tag{5}$$

where a smaller ρ corresponds to a faster convergence.

If $\mathbf{G}$ satisfies the necessary and sufficient conditions to achieve consensus, it can be shown that ρ is equal to the second largest eigenvalue (in absolute value) of $\mathbf{G}$ [3]. Therefore, if $\mathbf{G}$ satisfies (4) with $0 < \sigma < \frac{2}{\lambda_K}$, then

$$\rho = \max(|1 - \sigma\lambda_2|, |1 - \sigma\lambda_K|) \tag{6}$$

where λ_2 is the smallest non-zero eigenvalue of $\mathbf{L}$, also referred to as the algebraic connectivity of the network graph [7], and where λ_K is the largest eigenvalue of $\mathbf{L}$. The σ that minimizes ρ can then easily be found to be [3]

$$\sigma^* = \frac{2}{\lambda_K + \lambda_2} \,. \tag{7}$$

When substituting this in (6), we find that the minimal asymptotic convergence factor is

$$\rho^* = \frac{\lambda_K - \lambda_2}{\lambda_K + \lambda_2} \,. \tag{8}$$

3.2. Optimization of the link weights

The link weights that are used in (2) to define $\mathbf{L}$ provide us with additional degrees of freedom to optimize ρ, i.e., by tuning these weights we can manipulate the eigenvalues of $\mathbf{L}$. To this end, we first derive a centralized projected subgradient algorithm. In Section 4, we show how this algorithm can be implemented in a distributed fashion.

The value ρ^* depends on the weights w_{kq} and so can be treated as an objective function $\rho^*(\mathbf{W})$ which is then minimized with respect to the optimization variables in $\mathbf{W}$. From (8), the partial derivative of ρ^* with respect to the weight w_{kq} is found to be

$$\frac{\partial \rho^*}{\partial w_{kq}} = \frac{2}{(\lambda_K + \lambda_2)^2} \left(\lambda_2 \frac{\partial \lambda_K}{\partial w_{kq}} - \lambda_K \frac{\partial \lambda_2}{\partial w_{kq}} \right) . \quad (9)$$

It can be shown that [7, 10]

$$\frac{\partial \lambda_2}{\partial w_{kq}} = (f_k - f_q)^2 \quad (10)$$

where f_k and f_q denote the k-th and q-th entry of the (normalized) λ_2-eigenvector of $\mathbf{L}$. This eigenvector is often referred to as the Fiedler (eigen)vector [7, 11], and it is denoted here as $\mathbf{f}$. Similarly, if we denote the principal eigenvector of $\mathbf{L}$ (corresponding to λ_K) as $\mathbf{p}$, then

$$\frac{\partial \lambda_K}{\partial w_{kq}} = (p_k - p_q)^2 . \quad (11)$$

Define the $K \times K$ matrices $\mathbf{P} = \mathbf{p} \cdot \mathbf{1}^T$ and $\mathbf{F} = \mathbf{f} \cdot \mathbf{1}^T$, then the (sub)gradient of $\rho^*(\mathbf{W})$ is equal to

$$\nabla = \frac{2}{(\lambda_K + \lambda_2)^2} \left(\lambda_2 \left(\mathbf{P} - \mathbf{P}^T \right)^{\odot 2} - \lambda_K \left(\mathbf{F} - \mathbf{F}^T \right)^{\odot 2} \right) \quad (12)$$

where the operator $(\mathbf{X})^{\odot 2}$ takes the square of each entry of $\mathbf{X}$. The matrix $\mathbf{W}$ can then be optimized based on a projected (sub)gradient method, i.e.,

$$\mathbf{W}^{(i+1)} = \left(\mathbf{W}^{(i)} - \mu \, \nabla^{(i)} \right)_+ \odot \mathbf{A} \quad (13)$$

where μ is a user-defined stepsize, $\nabla^{(i)}$ denotes the (sub)gradient (12) evaluated in $\mathbf{W}^{(i)}$, $\odot$ denotes an elementwise multiplication (Hadamard product), and the operator $(\mathbf{X})_+$ sets all negative entries of $\mathbf{X}$ to zero. It is noted that the result of the (sub)gradient update is indeed projected onto the feasible set, i.e., the set of non-negative link weights. If we initialize (13) with $\mathbf{W}^{(0)} = \mathbf{A}$, then each update will yield a new Laplacian matrix $\mathbf{L}^{(i)}$, and hence a new set of Laplacian weights in the matrix $\mathbf{G}^{(i)}$ with improved convergence properties (see also Section 5). In Section 4, we will demonstrate that the particular form of (12) allows for a convenient distributed implementation.

Remark I: It is observed that links between nodes with a large difference between their respective entries in $\mathbf{f}$ will receive a larger weight[2] when applying (13). It can be shown that these links typically correspond to 'bridge links' between densely-connected node clusters in the network [7] (the Fiedler vector is also often used to reveal these node clusters). As these bridge links are the bottle necks in the information dissemination over the network, they should indeed receive a larger weight in the averaging procedure to achieve efficient information exchange between the different node clusters [7]. The Fiedler vector is also often used as a heuristic to manipulate the network topology, and hence to optimize the information dissemination [7, 10].

4. TA DISTRIBUTED ADAPTATION OF THE LAPLACIAN WEIGHTS

4.1. Distributed computation of λ_2, λ_K, $\mathbf{f}$ and $\mathbf{p}$

If all nodes would have access to λ_2 and λ_K, then σ^* could be computed at each node with (7), and the local Laplacian weights g_{kq}, $\forall\, k, q \in \mathcal{K}$ could be updated accordingly with (4). Furthermore, if each node $k \in \mathcal{K}$ would also have access to f_k, p_k, f_q, and p_q, $\forall\, q \in \mathcal{N}_k$, then the nodes could also update their corresponding link weights based on (13).

The nodes can learn their respective entries in $\mathbf{f}$ and $\mathbf{p}$ by performing power iterations (PIs) based on $\mathbf{L}$, which is implicitely encoded in the network, hence allowing for an efficient distributed implementation [7]. For example, consider the PI

$$\mathbf{y}^{(t+1)} = \frac{1}{r^{(t)}} \mathbf{L} \cdot \mathbf{y}^{(t)} \quad (14)$$

where $r^{(t)}$ is an estimate of $\overline{r}^{(t)} = \frac{\|\mathbf{L}\mathbf{y}^{(t)}\|}{\|\mathbf{y}^{(t)}\|}$, which is used to avoid that $\lim_{t\to\infty} \|\mathbf{y}^{(t)}\| = 0$ or $\lim_{t\to\infty} \|\mathbf{y}^{(t)}\| = \infty$. This PI is known to converge to a principal eigenvector $\mathbf{p}$ (assuming $\mathbf{p}^T \mathbf{y}^{(0)} \neq 0$) and can be easily implemented in a distributed fashion since $y_k^{(t+1)}$ only relies on $y_k^{(t)}$ and $y_q^{(t)}, \forall q \in \mathcal{N}_k$. The estimation and tracking of $\overline{r}^{(t)}$ can be taken care of by a distributed algorithm that runs in parallel with (14). For example, $\overline{r}^{(t)}$ is estimated by using gossip in [12] and by using diffusion adaptation in [11]. Note that $\lim_{t\to\infty} \overline{r}^{(t)} = \lambda_K$, hence λ_K can be extracted from the estimate $r^{(t)}$. In the sequel, we refer to (14) as the $\mathbf{p}$-algorithm.

The computation of $\mathbf{f}$ and λ_2 can rely on similar PI-based principles, although it is slightly more elaborate since $\mathbf{f}$ does not correspond to an extreme eigenvalue [11]. Consider the matrix

$$\mathbf{V} = \mathbf{L} \left(\mathbf{I} - \frac{1}{\alpha} \mathbf{L} \right)^N \quad (15)$$

where $\alpha > \lambda_K$ and where N is a positive integer. It can be shown that, if N is sufficiently large, $\mathbf{f}$ is the principal eigenvector of $\mathbf{V}$, which is then computed by an in-network PI similar to (14). We refer to [11] for more details on

[2] This can be seen from the combination of (10), (12) and (13).

this distributed computation of $\mathbf{f}$, which we refer to as the $\mathbf{f}$-algorithm.

It is noted that the $\mathbf{p}$- and $\mathbf{f}$-algorithms in [7, 11] estimate $\mathbf{p}$ and $\mathbf{f}$ up to an unknown scaling. For example, the $\mathbf{f}$-algorithm in [11] generates a converging series $\{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(t)}\}$, where $\lim_{t\to\infty} \mathbf{y}^{(t)} = \overline{\mathbf{f}} = \beta\mathbf{f}$ with $\beta \neq 0$. However, since (13) requires normalized eigenvectors, an additional distributed normalization procedure is required. To this end, we apply the following CA-based iteration[3]

$$\mathbf{n}^{(t+1)} = \mathbf{G}\left(\mathbf{n}^{(t)} + \Delta^{(t)}\right) \tag{16}$$

where

$$\Delta^{(t)} = \left(\mathbf{y}^{(t)}\right)^{\odot 2} - \left(\mathbf{y}^{(t-1)}\right)^{\odot 2} \tag{17}$$

which is initiated with $\mathbf{n}^{(0)} = \mathbf{0}$ and $\mathbf{y}^{(0)} = \mathbf{0}$. Using $\lim_{t\to\infty} \mathbf{G}^t = \frac{1}{K}\mathbf{1}\mathbf{1}^T$, $\lim_{t\to\infty} \mathbf{y}^{(t)} = \overline{\mathbf{f}}$, and the fact that the sequence $\{\|\Delta^{(t)}\|\}_{t\in\mathbb{N}}$ is square-summable[4], it can be shown (details omitted) that (16) will converge to $\lim_{t\to\infty} \mathbf{n}^{(t)} = \frac{1}{K}\|\overline{\mathbf{f}}\|^2\mathbf{1}$, i.e., a node $k \in \mathcal{K}$ can compute

$$f_k = \frac{\overline{f}_k}{\sqrt{n_k}} \tag{18}$$

where we have ignored[5] the scaling factor with K. A similar normalization procedure is applied in the $\mathbf{p}$-algorithm, yielding a normalized $\mathbf{p}$ with the same norm as $\mathbf{f}$.

4.2. Algorithm outline

Based on the distributed computation of λ_2, λ_K, $\mathbf{f}$ and $\mathbf{p}$, as explained in Section 4.1, we can formulate a distributed algorithm for adapting the Laplacian weights. In the sequel, the operators $F(\mathbf{f})$ and $P(\mathbf{p})$ perform a *single* iteration of the $\mathbf{f}$- and $\mathbf{p}$-algorithm, respectively (returning a new estimate of $\mathbf{f}$, λ_2, $\mathbf{p}$, and λ_K as an output). The operator $N\left(\mathbf{n}^{(t)}, \mathbf{y}^{(t+1)}, \mathbf{y}^{(t)}\right)$ performs a *single* iteration of (16). With this notation, we can describe the TA distributed updating procedure of the Laplacian weights as follows:

1. Initialize $\mathbf{W} \leftarrow \mathbf{A}$, $t \leftarrow 0$
2. Initialize $\overline{\mathbf{f}}^{(0)} \leftarrow \mathbf{1}$, $\overline{\mathbf{p}}^{(0)} \leftarrow \mathbf{1}$, $\mathbf{n}_f^{(0)} \leftarrow \mathbf{1}$ and $\mathbf{n}_p^{(0)} \leftarrow \mathbf{1}$.
3. Compute $\mathbf{L}$ according to (2).
4. Repeat until convergence of $\overline{\mathbf{f}}$, $\overline{\mathbf{p}}$, $\mathbf{n}_f$, and $\mathbf{n}_p$:
   - $[\overline{\mathbf{f}}^{(t+1)}, \lambda_2^{(t+1)}] \leftarrow F\left(\overline{\mathbf{f}}^{(t)}\right)$
   - $[\overline{\mathbf{p}}^{(t+1)}, \lambda_K^{(t+1)}] \leftarrow P\left(\overline{\mathbf{p}}^{(t)}\right)$
   - $\mathbf{n}_f^{(t+1)} \leftarrow N\left(\mathbf{n}_f^{(t)}, \overline{\mathbf{f}}^{(t+1)}, \overline{\mathbf{f}}^{(t)}\right)$
   - $\mathbf{n}_p^{(t+1)} \leftarrow N\left(\mathbf{n}_p^{(t)}, \overline{\mathbf{p}}^{(t+1)}, \overline{\mathbf{p}}^{(t)}\right)$
   - $t \leftarrow t+1$
5. Set $\sigma \leftarrow \frac{2}{\lambda_2^{(t)} + \lambda_K^{(t)}}$ and compute $\mathbf{G}$ according to (4).
6. $\forall\, k \in \mathcal{K}$ compute f_k and p_k based on their corresponding entries in $\overline{\mathbf{f}}^{(t)}$, $\overline{\mathbf{p}}^{(t)}$, $\mathbf{n}_f^{(t)}$ and $\mathbf{n}_p^{(t)}$, similar to (18).
7. Update $\mathbf{W}$ according to (13).
8. Return to step 3.

This algorithm can run in parallel with the CA algorithm, and both can even run completely independently and at different paces (there is no need to jointly synchronize them).

Remark II: A simplified TA distributed algorithm can be obtained by removing steps 6 and 7, as well as the computation of $\mathbf{n}_p$ and $\mathbf{n}_f$. In this case, the algorithm only optimizes σ, while using the same weights for all links, i.e., $\mathbf{W} = \mathbf{A}$. This simplification significantly improves the convergence of the Laplacian weights, at the cost of a smaller increase in CA convergence rate.

Remark III: If the matrix $\mathbf{G}$, as computed in step 5, is also used in (16), this creates a form of feedback which may destabilize the algorithm if σ temporarily becomes larger than $\frac{2}{\lambda_K}$ (e.g., due to inaccuracies or sudden changes in the topology). Therefore, to allow the algorithm to correct itself in such situations, it may be better to use a TU weighting rule in (16), e.g., with Metropolis weights [1].

4.3. Limitations of the distributed algorithm

The distributed algorithm described in Section 4.2 mimics the centralized optimization algorithm described in Section 3. However, simulations have indicated that the distributed algorithm often breaks down when either $\lambda_K \approx \lambda_{K-1}$ or $\lambda_2 \approx \lambda_3$, i.e., when the algorithm has reached a point where $\mathbf{p}$ or $\mathbf{f}$ get close to non-uniqueness. This is because the algorithm then continuously needs to switch between two eigenvectors which are almost orthogonal to each other. This quasi-orthogonality results in convergence problems within the PIs, which significantly affects the overall algorithm. Therefore, the proposed distributed algorithm (in its current form) can only optimize ρ up to this breakdown point. However, despite this limitation, the algorithm still significantly improves the convergence speed of the CA algorithm (see Section 5).

5. SIMULATIONS

We have performed Monte-Carlo (MC) simulations of the CA algorithm in networks consisting of $K = 32$ nodes. Each network was generated as 4 random subnetworks with 8 nodes

[3] Note that we can either use TA Laplacian weights or a fixed TU weighting rule for $\mathbf{G}$ (see also Remark III in Section 4.2).

[4] This means that $\sum_{t=0}^{\infty} \|\Delta^{(t)}\|^2 < \infty$, which follows from the fact that the sequence $\{\mathbf{y}^{(t)}\}_{t\in\mathbb{N}}$ is generated by a PI, which always produces square-summable errors.

[5] If the same procedure is used to normalize $\mathbf{p}$, it will hold that $\|\mathbf{f}\| = \|\mathbf{p}\| = \sqrt{K}$, such that the gradient update (12)-(13) is still correct (the factor $\sqrt{K}$ is then incorporated in the stepsize μ).

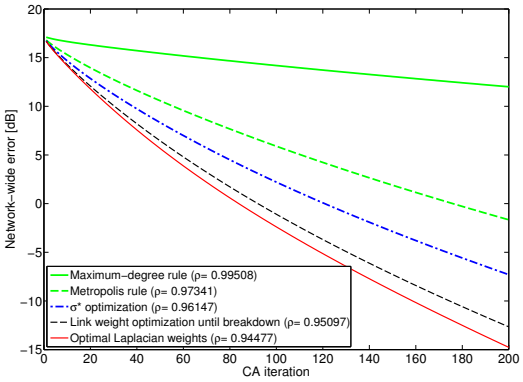

Fig. 1. Convergence speed of the CA algorithm when using different weighting rules (averaged over 200 MC runs) .

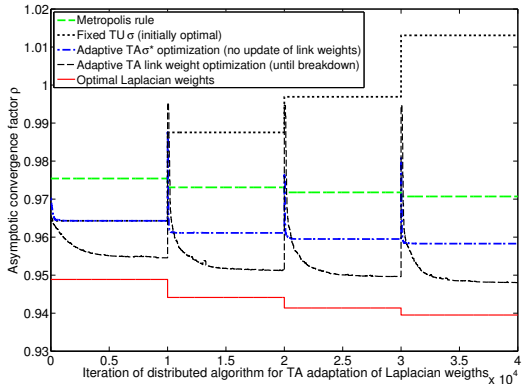

Fig. 2. Evolution of the CA asymptotic convergence factor during topology changes (averaged over 200 MC runs).

(in which each node has 3 neighbors on average), which were then interconnected by 8 additional random links. Assuming the nodes have to compute the average $\overline{x}$ of the entries in $\mathbf{x}^{(0)}$, then the network-wide error (in dB) at iteration i of the CA algorithm is defined as $10\log_{10}\left(\|\mathbf{x}^{(i)}-\overline{x}\mathbf{1}\|\right)$. The stepsize μ was manually tuned and set to $\mu = 0.3$.

Fig. 1 shows the decrease of the network-wide error in the CA algorithm (averaged over 200 MC runs) for several different choices of $\mathbf{G}$, i.e., the maximum-degree rule [1], the Metropolis rule [1], and the TA Laplacian weighting rule with (a) optimized σ (and $\mathbf{W} = \mathbf{A}$), (b) optimized link weights until the breakdown point (as explained in Subsection 4.3), and (c) fully-optimized link weights. It is noted that the CA algorithm here only starts *after* the optimization of the link weights. It is observed that the optimization of σ already provides a significant improvement. Convergence is further improved by also optimizing the link weights based on (13). Even when (13) is stopped at the breakdown point, the CA convergence speed is close to optimal.

In a second experiment, starting from the same initial topology, a random link was added or removed (with equal probability[6]) after every 10000 iterations (now the CA algorithm runs in parallel with the weight adaptation algorithm). In Fig. 2, we show the evolution of the asymptotic convergence factor ρ, when using different strategies for TA updating of the Laplacian weights. As a reference, we also show the ρ corresponding to the Metropolis rule, and to a fixed Laplacian rule (where σ is optimized for the initial topology, and then fixed). It is observed that the latter becomes unstable ($\rho > 1$), which shows the importance of the TA adaptation of the Laplacian weights. It is also observed that the simplified algorithm that only updates σ converges much faster than the algorithm that also optimizes the link weights.

6. CONCLUSIONS

We have proposed a TA distributed algorithm for on-line adaptation of the Laplacian weights to improve the convergence speed of the CA algorithm. The algorithm performs an optimization of a network-wide parameter σ, which is used to construct the Laplacian weights, and further improvements can be obtained by also optimizing the individual link weights. The algorithm is based on an in-network computation of two eigenvectors of the Laplacian matrix. The performance of the algorithm, as well as its limitations have been demonstrated by means of numerical Monte-Carlo simulations.

[6] In general, this should improve the convergence of the CA algorithm, as the network will become less 'clustered'.

7. REFERENCES

[1] A. H. Sayed, "Diffusion adaptation over networks," in *E-Reference Signal Processing*, R. Chellapa and S. Theodoridis, Eds. Elsevier, 2013.

[2] S. Chouvardas, K. Slavakis, and S. Theodoridis, "Adaptive robust distributed learning in diffusion sensor networks," *IEEE Trans. Signal Processing*, vol. 59, no. 10, pp. 4692 –4707, oct. 2011.

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

[4] D. Jakovetic, J.M.F. Moura, and J. Xavier, "Distributed detection over noisy networks: Large deviations analysis," *IEEE Trans. Signal Processing*, vol. 60, no. 8, pp. 4306 –4320, aug. 2012.

[5] 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.

[6] A. Scaglione, R. Pagliari, and H. Krim, "The decentralized estimation of the sample covariance," in *Asilomar Conference on Signals, Systems and Computers*, oct. 2008, pp. 1722 –1726.

[7] 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*, May 2013.

[8] D. Kempe and F. McSherry, "A decentralized algorithm for spectral analysis," *Journal of Computer and System Sciences*, vol. 74, no. 1, pp. 70–83, 2008.

[9] S.V. Macua, P. Belanovic, and S. Zazo, "Consensus-based distributed principal component analysis in wireless sensor networks," in *Int. Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, june 2010, pp. 1 –5.

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

[11] A. Bertrand and M. Moonen, "Distributed computation of the fiedler vector with application to topology inference in ad hoc networks," *Signal Processing*, vol. 93, no. 5, pp. 1106–1117, May 2013.

[12] M. Jelasity, G. Canright, and K. Engo-Monsen, "Asynchronous distributed power iteration with gossip-based normalization," in *Lecture Notes in Computer Science*. 2007, vol. 4641, pp. 514–525, Springer.