Consensus Acceleration in Multiagent Systems with the Chebyshev Semi-Iterative Method

R. L. G. Cavalcante
Fraunhofer Institute for Telecommunications,
Heinrich Hertz Institute /
Technische Universität Berlin
renato.cavalcante@hhi.fraunhofer.de

A. Rogers, N. R. Jennings
University of Southampton
School of Electronics and Computer Science
{acr,nrj}@ecs.soton.ac.uk

ABSTRACT

We consider the fundamental problem of reaching consensus in multiagent systems. To date, the consensus problem has been typically solved with decentralized algorithms based on graph Laplacians. However, the convergence of these algorithms is often too slow for many important multiagent applications, and thus they are increasingly being combined with acceleration methods. Unfortunately, state-of-the-art acceleration techniques require parameters that can be optimally selected only if complete information about the network topology is available, which is rarely the case in practice. We address this limitation by deriving two novel acceleration methods that can deliver good performance even if little information about the network is available. The first is based on the Chebyshev semi-iterative method and maximizes the worst-case convergence speed given that only rough bounds on the extremal eigenvalues of the network matrix are available. It can be applied to systems where agents use unreliable communication links, and its computational complexity is similar to those of simple Laplacian-based methods. This algorithm requires synchronization among agents, so we also propose an asynchronous version that approximates the output of the synchronous algorithm. Mathematical analysis and numerical simulations show that the convergence speed of the proposed acceleration methods decrease gracefully in scenarios where the sole use of Laplacian-based methods is known to be impractical.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Theory

Keywords

Decentralized control, collective dynamics, consensus

1. INTRODUCTION

Reaching agreement (or consensus) between physically distributed agents is one of the fundamental requirements of many multiagent applications including target localization [1], distributed coordination of large swarms of robots [2], distributed control of modular robotic actuators [3], and others [4]. Typically, such approaches require the agents to update a local estimate of an environmental or control parameter, by iteratively communicating with a few local neighbors, such that the estimates of all agents converge to the same value. For example, in [3] a modular robotic setting is described in which agents controlling decentralized modular actuators must reach consensus on the height of their actuators in order to ensure that a platform is kept level. Each agent can only infer (indirectly) the height of its neighbors, so only local control laws can be used, and the agents must reliably converge to a consensus height. Likewise, in one of the steps of the algorithm in [1], agents with individual estimates of the location of a target, iteratively exchange and update these estimates, with the intent that the estimates of all the agents converge to that which would have been reached had they been able to report their initial estimate to a center which could fuse them by taking their average.

To date, consensus problems of the type described above, have typically been solved with classic decentralized iterative algorithms based on graph Laplacians [2–5] and other related techniques that differ in the choice of the network matrices [6, 7]. In these consensus algorithms, every agent produces a sequence of estimates using a simple two-step approach that can be briefly described as follows. First, agents exchange estimates locally with their neighbors. Then each agent updates its current estimate by taking a weighted average of all estimates to which it has access, and the process repeats. As described above, the intent is that the estimates of all the agents converge to that which would have been reached had the average of the agents' initial estimates been taken. Unfortunately, however, it has been recently shown that the convergence of these classic iterative algorithms is often too slow in applications where agents have to agree on initial estimates that have a strong correlation with the agents' positions [2]. Typical scenarios in which this occurs are sensor networks and robotic swarms because the phenomena being measured are often functions of the agents' positions. In more detail, when the initial estimates are spatially correlated, the number of iterations required by Laplacian-based methods to compute the average consensus value with good accuracy can grow proportionally with the square of the network diameter [2]. This fact renders such methods impractical in large scale multiagent systems with sparse communication.

The convergence of these Laplacian-based algorithms can be greatly improved with acceleration techniques that filter the output [8–10]. In particular, efficient two-tap filters have been proposed for systems where agents communicate both synchronously [10] and asynchronously [8]. However, these algorithms typically have a free parameter that has to be chosen by the agents. Such heuristic choices of parameters can be avoided with the optimal polynomial

Cite as: Consensus Acceleration in Multiagent Systems with the Chebyshev Semi-Iterative Method, R. L. G. Cavalcante, A. Rogers, and N. R. Jennings, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. XXX-XXX.

filtering approach proposed in [9]. Unfortunately, this approach requires precise knowledge of the mean value of the network matrix, which again is unlikely to be available in many multiagent systems. In addition, this method is only stable in systems where the communication links are fairly reliable.

Thus, to address the above shortcomings and to make Laplacian-based methods practical in the multiagent scenarios described earlier, we propose low-complexity acceleration methods based on digital filters that require little information about the network topology and are robust against unreliable communication links. In more detail, the main contributions of this study are as follows:

• We derive a novel acceleration method named *synchronous semi-iterative consensus*. This algorithm filters the output of classic consensus algorithms with a polynomial filter that is optimal in the sense of maximizing the worst-case mean convergence speed when the network topology is unknown. Unlike recent acceleration techniques [9], the proposed algorithm only requires rough upper and lower bounds on the extremal eigenvalues of the network matrix (first order statistics is not necessary), and it is amenable to an efficient recursive implementation. Compared to other state-of-the-art acceleration techniques, such as those in [8, 10], our synchronous semi-iterative consensus algorithm has better convergence properties in many practical scenarios, the same communication requirements, and roughly the same computational complexity.

• To handle scenarios where synchronization among agents is not possible, we further extend our approach to devise an asynchronous algorithm, named *asynchronous semi-iterative consensus*, that approximates the output of the proposed synchronous algorithm. This asynchronous algorithm has a strong connection with those in [8,10], but it does not require heuristics for parameter tuning in real applications where the network topology is largely unknown. All parameters of our algorithm are readily obtained from rough upper and lower bounds of the extremal eigenvalues of the unknown network matrix.

The paper is divided as follows. Sect. 2 reviews classic consensus algorithms based on graph Laplacians and other similar approaches. Sect. 3 shows the two novel acceleration schemes. Numerical simulations in Sect. 4 evaluate the proposed methods in scenarios where Laplacian-based methods are impractical.

2. PROBLEM STATEMENT

We start by briefly introducing our notation. In particular, vectors are written in lower-case, bold typeface, and matrices are written in upper-case, bold typeface. Unless otherwise stated, vectors are assumed to be column vectors. For every vector $\boldsymbol{v} \in \mathbb{R}^N$, we define the norm of $\boldsymbol{v}$ by $\|\boldsymbol{v}\| := \sqrt{\boldsymbol{v}^T\boldsymbol{v}}$, where $(\cdot)^T$ denotes the transpose operation. The vector of ones is denoted by $\mathbf{1}$, and its dimension is clear from the specific context. The element of the kth row and the jth column of a matrix $\boldsymbol{X} \in \mathbb{R}^{M\times N}$ is denoted by $[\boldsymbol{X}]_{kj}$. The eigenvalues of a symmetric matrix $\boldsymbol{X} \in \mathbb{R}^{N\times N}$ are denoted by $\lambda_1(\boldsymbol{X}), \ldots, \lambda_N(\boldsymbol{X})$. By $\boldsymbol{D} := \mathrm{diag}(\lambda_1, \ldots, \lambda_N)$, we denote a diagonal matrix $\boldsymbol{D} \in \mathbb{R}^{N\times N}$ having $\lambda_1, \ldots, \lambda_N$ as the entries on its main diagonal.

We now turn to the problem formulation. In this study we assume that the multiagent system forms a network represented by a connected undirected graph $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$, where $\mathcal{N} = \{1, \ldots, N\}$ is the set of agents, $\mathcal{E} \subset \{\{k, j\} \mid k, j \in \mathcal{N}\}$ is the edge set, and the edge $\{k, j\} \in \mathcal{E}$ is an unordered pair of agents. For convenience, here we assume that $\{k, k\} \in \mathcal{E}$. Initially, at time $i = 0$, each agent k reports a value $x_k[0] \in \mathbb{R}$, and we are interested in iterative algorithms that produce, in every agent $k \in \mathcal{N}$, sequences $\{x_k[i]\}$ converging to $x_{\mathrm{av}} := 1/N \sum_{k\in\mathcal{N}} x_k[0]$, the average of the initial values reported by the agents.

To be truly decentralized, the algorithms of interest should respect the network topology, i.e., at time instant $i \in \mathbb{N}$, each agent k should exchange information only with its neighbors $\mathcal{N}_k := \{j \in \mathcal{N} \mid \{j, k\} \in \mathcal{E}\}$. In particular, classic algorithms having this desired feature take the form:

$$x_k[i+1] = \sum_{j\in\mathcal{N}_k} [\boldsymbol{W}[i]]_{kj} x_j[i],\ k \in \mathcal{N},\ i \in \mathbb{N}, \tag{1}$$

or, more compactly,

$$\boldsymbol{x}[i+1] = \boldsymbol{W}[i]\boldsymbol{x}[i],\ i \in \mathbb{N}, \tag{2}$$

where $\boldsymbol{x}[i] := [x_1[i]\ \ldots\ x_N[i]]^T \in \mathbb{R}^N$, $\boldsymbol{W}[i] \in \mathbb{R}^{N\times N}$ is a properly selected sequence of (symmetric) matrices, $[\boldsymbol{W}[i]]_{kj}$ is the weight associated with the edge $\{k, j\}$ at time i, and $[\boldsymbol{W}[i]]_{kj} = 0$ if $\{i, j\} \notin \mathcal{E}$. To reach consensus, agents can compute the weights $[\boldsymbol{W}[i]]_{kj}$ in many different ways according to the desired characteristics of the system. In particular, if the network is deterministic, agents only know the local topology, and links are reliable, agents can use simple Laplacian-based methods to compute locally the weights [2, 4]. When links are unreliable, weights can be computed with the method in [5]. In systems where the network topology is known before deployment and links are deterministic, the approach in [6] can be used to compute a fixed matrix $\boldsymbol{W} = \boldsymbol{W}[i]$ that gives better convergence than simple heuristics based on graph Laplacians. In systems where agents operate asynchronously and do not know their neighbors, gossip consensus algorithms [7] can be used to determine the weights.

Hereafter, we do not use a specific method to compute the weights $[\boldsymbol{W}[i]]_{kj}$, and, for maximum generality, we only assume that the matrices $\boldsymbol{W}[i]$ satisfy the following properties:

ASSUMPTION 1. *(Properties of $\boldsymbol{W}[i]$:)*
1. The matrices $\boldsymbol{W}[i]$ ($i \in \mathbb{N}$) in (2) are i.i.d. random[1] matrices with $[\boldsymbol{W}[i]]_{jk} = 0$ if $\{j, k\} \notin \mathcal{G}$.
2. Each matrix $\boldsymbol{W}[i]$ is symmetric and satisfies $\boldsymbol{W}[i]\mathbf{1} = \mathbf{1}$.
3. $\|E[\boldsymbol{W}[i]^T\boldsymbol{W}[i]] - 1/N\mathbf{1}\mathbf{1}^T\|_2 < 1$ (and $\|\overline{\boldsymbol{W}} - 1/N\mathbf{1}\mathbf{1}^T\|_2 < 1$, where $\overline{\boldsymbol{W}} := E[\boldsymbol{W}[i]]$ denotes the mean of $\boldsymbol{W}[i]$).

The above properties are sufficient conditions to guarantee that $\boldsymbol{x}[i]$ in (2) converges to $x_{\mathrm{av}}\mathbf{1} \in \mathbb{R}^N$ in both the mean sense and the mean square sense [7], i.e., $\lim_{i\to\infty} E[\boldsymbol{x}[i]] = x_{\mathrm{av}}\mathbf{1}$ and $\lim_{i\to\infty} E[\|\boldsymbol{x}[i] - x_{\mathrm{av}}\mathbf{1}\|^2] = 0$. Unfortunately, irrespective of the method being used for the computation of the weights $[\boldsymbol{W}[i]]_{kj}$, when agents only have local information about the network topology, consensus algorithms solely based on the iteration in (2) are typically slow. In particular, when the initial values reported by agents have a strong correlation with their locations, Laplacian-based methods have been shown to be impractical in large multiagent systems because the convergence speed scales badly with the network diameter [2]. To address this serious drawback of consensus algorithms based on (2), we develop an acceleration technique that improves the convergence of $\boldsymbol{x}[i]$ in the mean sense. Before deriving the proposed method, we first review convergence properties of (2).

By the i.i.d. assumption of the symmetric matrices $\boldsymbol{W}[i]$ ($i \in \mathbb{N}$), we have that $E[\boldsymbol{W}[i]]$ is a time-invariant symmetric matrix ($E[\boldsymbol{W}[i]] = \overline{\boldsymbol{W}}$ for all $i \in \mathbb{N}$). Let the eigenvalue decomposition of $\overline{\boldsymbol{W}}$ be given by $\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T = \overline{\boldsymbol{W}}$, where $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_N$ are

[1] In this study, we use the same notation for random variables and their realizations. The interpretation that should be applied is clear from the context.

the columns of $\boldsymbol{Q}$, and $\boldsymbol{\Lambda} := \operatorname{diag}\left(\lambda_1(\overline{\boldsymbol{W}}), \ldots, \lambda_N(\overline{\boldsymbol{W}})\right)$ with eigenvalues arranged in non-increasing order of magnitude: $|\lambda_1(\overline{\boldsymbol{W}})| \geq \ldots \geq |\lambda_N(\overline{\boldsymbol{W}})|$. Note that, also from Assumption 1, we have that $\boldsymbol{q}_1 = (1/\sqrt{N})\mathbf{1}$ and that $1 = \lambda_1(\overline{\boldsymbol{W}}) > |\lambda_j(\overline{\boldsymbol{W}})|$ for $j = 2, \ldots, N$. With these definitions, we deduce:

$$\begin{aligned} E[\boldsymbol{x}[i]] &= (\overline{\boldsymbol{W}})^i \boldsymbol{x}[0] = \boldsymbol{Q}\boldsymbol{\Lambda}^i \boldsymbol{Q}^T \boldsymbol{x}[0] \\ &= [(1/\sqrt{N})\mathbf{1}\ \boldsymbol{q}_2\ \ldots \boldsymbol{q}_N] \\ &\quad \cdot \operatorname{diag}(1, (\lambda_2(\overline{\boldsymbol{W}}))^i, \ldots, (\lambda_N(\overline{\boldsymbol{W}}))^i) \\ &\quad \cdot [(1/\sqrt{N})\mathbf{1}\ \boldsymbol{q}_2\ \ldots\ \boldsymbol{q}_N]^T \boldsymbol{x}[0]. \end{aligned} \tag{3}$$

Therefore, by $|\lambda_j(\overline{\boldsymbol{W}})| < 1$ for $j = 2, \ldots, N$, we conclude that $\lim_{i\to\infty} E[\boldsymbol{x}[i]] = (1/N)\mathbf{1}\mathbf{1}^T \boldsymbol{x}[0] = x_{\text{av}}\mathbf{1}$ and that the slowest mode of convergence of (3) is given by $\lambda_2(\overline{\boldsymbol{W}})$ (by taking powers, the eigenvalues $\lambda_j(\overline{\boldsymbol{W}})$ ($j = 3, \ldots, N$) do not decay slower to zero than $\lambda_2(\overline{\boldsymbol{W}})$). Thus, the iteration in (2) can be particularly slow if $\lambda_2(\overline{\boldsymbol{W}})$ is close to one and the vector $\boldsymbol{x}[0]$ has a nonzero projection onto the subspace spanned by the eigenvector corresponding to $\lambda_2(\overline{\boldsymbol{W}})$.

3. THE ACCELERATION ALGORITHM

In this section, we derive our novel algorithms and compare them with existing methods. We start by revisiting polynomial filters.

3.1 Polynomial Filtering

In our proposed method, we improve the convergence of (2) (in the mean sense) by using polynomial filters. The idea is similar to that proposed in [9], but the size of the filters that we use increase with the number of iterations. Later in this section we show that this is amenable to implementations with very low computational complexity and memory requirements. In addition, our method is optimal in a well defined sense (c.f. (7)) even if little information about $\boldsymbol{W}[i]$ in (2) is available.

In more detail, each agent k improves its local estimate of x_{av} by filtering $x_k[i]$ obtained with (1):

$$y_k[i] = \sum_{n=0}^{i} \gamma[i,n] x_k[n],\ k \in \mathcal{N}, \tag{4}$$

where $\gamma[i,n] \in \mathbb{R}$ ($i \in \mathbb{N},\ n = 0, \ldots, i$) are *scalars to be designed* (common to all agents), and $y_k[i]$ is the improved estimate of x_{av} at time i in agent k. Stacking $y_1[i], \ldots, y_N[i]$ in a vector $\boldsymbol{y}[i] := [y_1[i]\ \ldots\ y_N[i]]^T$, we can rewrite (4) equivalently as

$$\boldsymbol{y}[i] = \sum_{n=0}^{i} \gamma[i,n] \boldsymbol{x}[n]. \tag{5}$$

Combining (3) and (5) and using Assumption 1, we can compute the mean value of $\boldsymbol{y}[i]$:

$$\begin{aligned} E[\boldsymbol{y}[i]] &= \sum_{n=0}^{i} \gamma[i,n] (\overline{\boldsymbol{W}})^n \boldsymbol{x}[0] \\ &= \boldsymbol{Q} \operatorname{diag}\left(p_i(1), p_i(\lambda_2(\overline{\boldsymbol{W}})) \ldots, p_i(\lambda_N(\overline{\boldsymbol{W}}))\right) \boldsymbol{Q}^T \boldsymbol{x}[0], \\ &= [(1/\sqrt{N})\mathbf{1}\ \boldsymbol{q}_2\ \ldots \boldsymbol{q}_N] \\ &\quad \cdot \operatorname{diag}\left(p_i(1), p_i(\lambda_2(\overline{\boldsymbol{W}})), \ldots, p_i(\lambda_N(\overline{\boldsymbol{W}}))\right) \\ &\quad \cdot [(1/\sqrt{N})\mathbf{1}\ \boldsymbol{q}_2\ \ldots\ \boldsymbol{q}_N]^T \boldsymbol{x}[0], \end{aligned} \tag{6}$$

where $p_i(x)$ is the polynomial $p_i(x) := \sum_{n=0}^{i} \gamma[i,n] x^n$ at time i. Now we need to choose a polynomial p_i that makes (6) a potentially better estimate of $x_{\text{av}}\mathbf{1}$ than (3).

3.2 The Synchronous Consensus Algorithm

By comparing (3) with (6), the slowest mode of convergence of $E[\boldsymbol{y}[i]]$ to $x_{\text{av}}\mathbf{1}$ is faster than that of $E[\boldsymbol{x}[i]]$ if the polynomials p_i satisfy the following properties: (see also [9], which, unlike the proposed method, use filters of short length.)

P1) $p_i(1) = 1$ and

P2) $\max_{j\in\{2,\ldots,N\}} |p_i(\lambda_j(\overline{\boldsymbol{W}}))| < |\lambda_2(\overline{\boldsymbol{W}})|^i$.

Therefore, at each time i, we conclude that we should find polynomials such that $p_i(1) = 1$ and that $|p_i(\lambda_j(\overline{\boldsymbol{W}}))|$ is as close to zero as possible for all $j \in \{2, \ldots, N\}$ and all $i \in \mathbb{N}$. Unfortunately, finding an ideal polynomial having roots at $\lambda_2(\overline{\boldsymbol{W}})$, ..., $\lambda_N(\overline{\boldsymbol{W}})$ (for $i \geq N-1$) would require global information about the network in every agent. To avoid this unrealistic requirement, we assume that the eigenvalues $\lambda_2(\overline{\boldsymbol{W}})$, ..., $\lambda_N(\overline{\boldsymbol{W}})$ belong to the interval $[\alpha, \beta]$, but their exact values are unknown. (Assumption 1 guarantees $-1 < \alpha, \beta < 1$, and the bounds can be obtained from typical application scenarios; see Sect. 4.) With this assumption, a reasonable choice for p_i is the normalized polynomial $p_i(1) = 1$ of degree i least deviating from zero on the interval $[\alpha, \beta]$ (see also [11], [12, Sect. 10.1.5]). We can expect that such a polynomial would satisfy properties P1) and P2) above without knowledge of $\lambda_j(\overline{\boldsymbol{W}})$ ($j = 2, \ldots, N$). More formally, at time i we use the polynomial:

$$p_i^\star \in \arg\min_{p\in\mathcal{S}_i} \{ \max_{\alpha\leq x\leq\beta} |p(x)| \}, \tag{7}$$

where $\mathcal{S}_i$ is the set of polynomials of degree i normalized to satisfy $p_i(1) = 1$. The polynomial in (7), which has been typically used to accelerate the convergence of iterative methods solving systems of linear equations, is unique and given by [11], [12, Sect. 10.1.5]:

$$p_i^\star(x) = \frac{c_i\left(-1 + 2\dfrac{x-\alpha}{\beta-\alpha}\right)}{c_i(\mu)},$$

where

$$\mu := 1 + 2\frac{1-\beta}{\beta-\alpha} \tag{8}$$

and c_i is the Chebyshev polynomial of degree i

$$c_i(x) = \begin{cases} \cos(i\cos^{-1} x), & |x| \leq 1,\ i \in \mathbb{N}, \\ \cosh(i\ \cosh^{-1} x), & |x| > 1,\ i \in \mathbb{N}. \end{cases}$$

Chebyshev polynomials can be generated with the recursion $c_{m+1}(x) = 2xc_m(x) - c_{m-1}(x)$ ($c_0(x) = 1$ and $c_1(x) = x$), so, similarly to the original Chebyshev acceleration algorithm [11], [12, Sect. 10.1.5], we can equivalently compute $E[\boldsymbol{y}[i]]$ (with the polynomial (7)) in the recursive form:

$$\begin{aligned} &E[\boldsymbol{y}[i+1]] \\ &= \omega_{i+1}[(1-\kappa)\boldsymbol{I} + \kappa\overline{\boldsymbol{W}}]E[\boldsymbol{y}[i]] + (1-\omega_{i+1})E[\boldsymbol{y}[i-1]], \end{aligned} \tag{9}$$

where $E[\boldsymbol{y}[1]] = [(1-\kappa)\boldsymbol{I} + \kappa\overline{\boldsymbol{W}}]\boldsymbol{x}[0]$, $\boldsymbol{y}[0] = \boldsymbol{x}[0]$, $\kappa := 2/(2-\alpha-\beta)$, and

$$\omega_{i+1} = \frac{1}{1 - \dfrac{\omega_i}{4\mu^2}},\ i \geq 2,\ \omega_1 = 1,\ \omega_2 = \frac{2\mu^2}{2\mu^2 - 1}. \tag{10}$$

Unfortunately, unless $\boldsymbol{W}[i]$ is a constant matrix, the recursion in (9) cannot be implemented in a multiagent system because $\overline{\boldsymbol{W}}$ is

not available. Therefore, we replace expectations by sample values, and we obtain the following algorithm, which can be implemented in multiagent systems because the iteration in (1) (or, equivalently, (2)) can be readily implemented (using local computation of the resulting matrix-vector multiplications):

ALGORITHM 1. *(Synchronous Semi-Iterative Consensus Algorithm)*

$$\boldsymbol{z}[i+1] = \omega_{i+1}\left[(1-\kappa)\boldsymbol{I} + \kappa\boldsymbol{W}[i]\right]\boldsymbol{z}[i] + (1-\omega_{i+1})\boldsymbol{z}[i-1], \tag{11}$$

where $\boldsymbol{z}[1] = [(1-\kappa)\boldsymbol{I} + \kappa\boldsymbol{W}[0]]\boldsymbol{x}[0]$ *and* $\boldsymbol{z}[0] = \boldsymbol{x}[0]$.

The proposition below shows that some convergence properties of the original Chebyshev algorithm are retained, even though we replaced expectations by sample values.

PROPOSITION 1. *(Properties of the Synchronous Semi-Iterative Consensus Algorithm)*

Assume the conditions in Assumption 1. Then, the algorithm in (11) satisfies the following:

a) The algorithm is average preserving, i.e., $(1/N)\mathbf{1}^T\boldsymbol{z}[i] = (1/N)\mathbf{1}^T\boldsymbol{x}[0] = x_{\rm av}$ *for every* $i \in \mathbb{N}$.

b) In the mean sense, the convergence of $\boldsymbol{z}[i]$ *is identical to that of* $\boldsymbol{y}[i]$ *in (5) with* $\gamma[i,n]$ *chosen as the coefficients of the optimal polynomial in the sense of (7). In other words,* $E[\boldsymbol{z}[i]] = E[\boldsymbol{y}[i]]$. *Furthermore, if* α *and* β *are such that* $-1 < \alpha \leq \min_{j\in\{2,\ldots,N\}}\lambda_j(\overline{\boldsymbol{W}}) \leq \max_{j\in\{2,\ldots,N\}}\lambda_j(\overline{\boldsymbol{W}}) \leq \beta < 1$, *then*

$$\|E[\boldsymbol{z}[i]] - x_{\rm av}\mathbf{1}\| \leq \frac{\|\boldsymbol{x}[0]\|}{c_i(\mu)}, \tag{12}$$

which shows that $\lim_{i\to\infty} E[\boldsymbol{z}[i]] = x_{\rm av}\mathbf{1}$ *because* $\lim_{i\to\infty} c_i(\mu) = \infty$.

PROOF. In the following, for notational convenience, we define: $\boldsymbol{M}[i] := [(1-\kappa)\boldsymbol{I} + \kappa\boldsymbol{W}[i]]$ and $\overline{\boldsymbol{M}} := [(1-\kappa)\boldsymbol{I} + \kappa\overline{\boldsymbol{W}}]$.

(a) By $\mathbf{1}^T\boldsymbol{W}[i] = \mathbf{1}^T$, we can check that

$$\mathbf{1}^T\boldsymbol{M}[i] = [(1-\kappa)\mathbf{1}^T\boldsymbol{I} + \kappa\mathbf{1}^T\boldsymbol{W}[i]] = \mathbf{1}^T,$$

and the result follows by induction. More precisely, note that $\mathbf{1}^T\boldsymbol{z}[0] = \mathbf{1}^T\boldsymbol{x}[0] = Nx_{\rm av}$ and that $\mathbf{1}^T\boldsymbol{z}[1] = \mathbf{1}^T\boldsymbol{M}[0]\boldsymbol{x}[0] = N\ x_{\rm av}$. Now, by assuming $\mathbf{1}^T\boldsymbol{z}[i] = Nx_{\rm av}$ and $\mathbf{1}^T\boldsymbol{z}[i-1] = Nx_{\rm av}$, we obtain

$$\begin{aligned}\mathbf{1}^T\boldsymbol{z}[i+1] &= \omega_{i+1}\mathbf{1}^T\boldsymbol{M}[i]\boldsymbol{z}[i] + (1-\omega_{i+1})\mathbf{1}^T\boldsymbol{z}[i-1]\\ &= \omega_{i+1}Nx_{\rm av} + (1-\omega_{i+1})Nx_{\rm av} = Nx_{\rm av},\end{aligned}$$

which concludes the proof of (a).

(b) The proof can be informally shown as follows. From the i.i.d. assumption of the matrices $\boldsymbol{W}[i]$, we have that $\boldsymbol{z}[i]$ is independent of $\boldsymbol{W}[i]$, and thus $\boldsymbol{z}[i]$ is also independent of $\boldsymbol{M}[i]$. Now, apply the expectation operator in both sides of (11) to obtain

$$E[\boldsymbol{z}[i+1]] = \omega_{i+1}\overline{\boldsymbol{M}}E[\boldsymbol{z}[i]] + (1-\omega_{i+1})E[\boldsymbol{z}[i-1]] \tag{13}$$

where $E[z[1]] = \overline{\boldsymbol{M}}\boldsymbol{x}[0]$ and $E[\boldsymbol{z}[0]] = \boldsymbol{x}[0]$, and we conclude that $E[\boldsymbol{z}[i]] = E[\boldsymbol{y}[i]]$ for every $i \in \mathbb{N}$ (see (9)), and thus (13) is equivalent to (6) with $\boldsymbol{y}[i]$ replaced by $\boldsymbol{z}[i]$ and with p_i being the optimal polynomial in (7). Subtract $x_{\rm av}\mathbf{1} = (1/N)\mathbf{1}\mathbf{1}^T\boldsymbol{x}[0]$ from both sides of (6) and use the facts that $|p_i^\star(\lambda)| \leq 1/|c_i(\mu)|$ ($\alpha \leq \lambda \leq \beta$) and that $\|\boldsymbol{A}\boldsymbol{b}\|_2 \leq \|\boldsymbol{A}\|_2\,\|\boldsymbol{b}\|_2$ for any matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$ of compatible sizes [12] to obtain (12). □

Intuitively, Proposition 1 shows that our algorithm (with properly selected parameters α and β) is guaranteed to converge in the mean sense, and the convergence in the mean is typically faster than that of the original scheme in (2). Unfortunately, unless the matrix $\boldsymbol{W}[i]$ is a constant matrix, the results in Proposition 1 are not enough to guarantee stability. In particular, Proposition 1 does not guarantee mean-square convergence, but this problem is also present in existing acceleration techniques that consider time-varying network matrices [8, 9]. However, in Sect. 4 we show that our method is robust in many practical scenarios. In addition, note that Proposition 1(a) holds even if the algorithm diverges (which could be the case when the parameter α is overestimated), so it can be useful to devise hybrid schemes with stronger convergence guarantees, but such methods are not investigated here.

3.3 The Asynchronous Consensus Algorithm

A potential limitation of Algorithm 1 is that agents should know the time instant i to compute ω_i. In some systems, such as those with agents communicating via network gossiping [7], knowing precisely the time index may not be possible. This fact renders Algorithm 1 impractical, and, to address this limitation, we propose an asynchronous semi-iterative consensus algorithm that is based on the following observation:

FACT 1. *[11] [12, p. 517] (On the convergence of* ω_i*)*

Let ω_i *be as in (10) and* $-1 < \alpha < \beta < 1$. *Then, for* $i > 1$, ω_i *satisfies the following properties: i)* $1 < \omega_i < 2$, *ii)* ω_i *is strictly decreasing, and iii)*

$$\lim_{i\to\infty}\omega_i = \frac{2}{1+\sqrt{1-1/\mu^2}} =: \omega_\infty. \tag{14}$$

Since ω_i is convergent (and the asymptotic convergence is usually fast), we can try to approximate the output of Algorithm 1 by fixing ω_i to ω_∞, the limit in (14). The resulting algorithm is formally presented below.

ALGORITHM 2. *(Asynchronous semi-iterative consensus algorithm)*

$$\boldsymbol{z}[i+1] = \omega_\infty\left[(1-\kappa)\boldsymbol{I} + \kappa\boldsymbol{W}[i]\right]\boldsymbol{z}[i] + (1-\omega_\infty)\boldsymbol{z}[i-1], \tag{15}$$

where $\boldsymbol{z}[1] = [(1-\kappa)\boldsymbol{I} + \kappa\boldsymbol{W}[0]]\boldsymbol{x}[0]$ *and* $\boldsymbol{z}[0] = \boldsymbol{x}[0]$.

It is not difficult to show that that Algorithm 2 is also average preserving and converges in the mean sense to $x_{\rm av}\mathbf{1}$.

3.4 Relation with Existing Methods

We now compare the proposed algorithms against the original consensus algorithm with and without state-of-the-art acceleration methods. We start by rewriting (11) in the equivalent form:

$$\begin{aligned}z_k[i+1] = \sum_{j\in\mathcal{N}_k}\omega_{i+1}\left(1-\kappa+\kappa[\boldsymbol{W}[i]]_{kj}\right)z_j[i]\\ + (1-\omega_{i+1})z_k[i-1], \quad k\in\mathcal{N},\end{aligned} \tag{16}$$

where $z_k[1] = \sum_{j\in\mathcal{N}_k}(1-\kappa+\kappa[\boldsymbol{W}[i]]_{kj})x_j[0]$ and $z_k[0] = x_k[0]$. (Note: Algorithm 2 is obtained by fixing ω_i in (16).)

From the above, we see that we need to keep two scalars in the memory of each agent, instead of one as in the original consensus algorithm in (1). In addition, in terms of local computation complexity per agent, Algorithm 1 is slightly more complex than the original consensus algorithm because (16) requires fewer additional sums and multiplications per iteration as compared to (1). However, the slightly higher computational complexity and memory requirements of the proposed method can be ignored because, in a real-world implementation with wireless links, agents spend most energy and time with the information exchange rather

than computation [13]. If agents implement either (1) or (16), they communicate with exactly the same neighbors and exchange the same amount of information per iteration. However, to reach the desired solution $x_{\rm av}$ within a prescribed precision, the iteration in (16) typically requires fewer iterations than the scheme in (1). As a result, the proposed methods can lead to great time and energy savings (because less information must be exchanged).

From the equivalence between $E[\boldsymbol{z}[i]]$ and $E[\boldsymbol{y}[i]]$ in (9) (or (6)), which is proved in Proposition 1(b), Algorithm 1 is applying a long polynomial filter that is optimal in a well defined sense and that uses all estimates $\boldsymbol{z}[0], \boldsymbol{z}[1], \ldots$, even though its implementation only requires $\boldsymbol{z}[i]$ and $\boldsymbol{z}[i-1]$. This is in stark contrast with the implementation of the two algorithms in [9], both of which typically use short filters of fixed length and keep in the memory of each agent more than two samples of the sequence of estimates of $x_{\rm av}$. One of the algorithms in [9] uses filters based on an intuitive approach that lacks an optimality criterion. The other approach in [9] uses filters optimal in a well defined sense, but it requires precise knowledge of the eigenvalues of $\overline{\boldsymbol{W}}$, which is an information not required by our approaches.

We can also relate our approaches with the two-register algorithms in [8, 10]. Assume that $\alpha = -\beta$ with $\beta > 0$, in which case $\kappa = 1$. Therefore, (16) reduces to:

$$z_k[i+1] = \sum_{j \in \mathcal{N}_k} \omega_{i+1} [\boldsymbol{W}[i]]_{kj} z_j[i] + (1-\omega_{i+1}) z_k[i-1], \quad k \in \mathcal{N}, \quad (17)$$

where $z_k[0]$ and $z_k[1]$ are as in (16). As a result, the approaches in [8, Eq. (9)] and [10, Eq. (28)] (this last by also fixing the matrix $\boldsymbol{W}[i]$) are recovered if we fix ω_i to any number in the interval $(1, 2)$. Fixing ω_i was also the approach used to derive Algorithm 2, which is an algorithm that gives strong arguments to use $\omega_i = \omega_\infty$ and not arbitrary values within the range $(1, 2)$ (provided that the upper bound β is available). Note that we can also argue that fixing ω_∞ to an arbitrary value within the range $(1, 2)$ is equivalent to making an arbitrary choice of β. More precisely, given $\omega_\infty \in (1, 2)$ and $\alpha = -\beta$ (which was used to derive (17)), we can use (14) to calculate the value of β that results in such a choice of ω_∞. In the next section we show that the choice $\alpha = -\beta$ can be too pessimistic in many cases, and, as a result, the acceleration capabilities of the algorithm can be reduced.

A less strict reader could also argue that Algorithm 2 in its full generality is also equivalent to the algorithms in [8, Eq. (9)] and [10, Eq. (28)] with $\boldsymbol{W}[i]$ replaced by $(1-\kappa)\boldsymbol{I} + \kappa \boldsymbol{W}[i]$. In such a case, these existing algorithms are accelerating the consensus algorithm $\boldsymbol{x}[i+1] = ((1-\kappa)\boldsymbol{I} + \kappa \boldsymbol{W}[i])\boldsymbol{x}[i]$ and not the iteration in (2). Algorithm 2 shows how to choose κ (and ω_∞) when bounds on the eigenvalues are available.

4. EMPIRICAL EVALUATION

We now show the superiority of our proposed algorithms over existing methods, and we also evaluate the stability of the proposed algorithms in practical scenarios.In particular, as in [2], we place N agents uniformly at random in a square of unit area, and then we connect agents within distance $d = \sqrt{\log(N)/N}$ from each other (unless otherwise stated), which is a distance that guarantees that the resulting graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is connected with high probability [14]. We discard graphs not fully connected. In all simulations, each agent k initially reports values $x_k[0] = 50\sqrt{2} \cdot \|[\mathcal{X}_k \; \mathcal{Y}_k]^T\| + n_k$, where n_k is a sample of a Gaussian random variable with mean zero and unit variance,[2] and $\mathcal{X}_k$ and $\mathcal{Y}_k$ are the Euclidean spatial coordinates of agent k in the unit grid. (Note that agents start with values strongly correlated with their positions as is common in the multiagent systems described earlier [2].)

We consider cases where agents exchange information not only with reliable communication links, but also with unreliable communication links because practical algorithms should be robust against link failures (a common occurrence in wireless links owing to the presence of jammers, obstacles, etc.). Therefore, for each scenario, the following network matrices $\boldsymbol{W}[i]$ are used:

• (*Reliable links.*) For simulations using reliable links, we set the network matrix to $\boldsymbol{W}[i] = \boldsymbol{I} - \epsilon \boldsymbol{L}$ ($i \in \mathbb{N}$), where $\boldsymbol{L}$ is the Laplacian matrix of the graph $\mathcal{G}$ and $\epsilon > 0$ is a properly selected scalar that guarantees that $\boldsymbol{W}[i]$ satisfies the conditions in Assumption 1.[3] In particular, in this study we set $\epsilon = 0.05$ because it guaranteed the conditions in Assumption 1 in all our simulations. For a graph $\mathrm{G} = (\mathcal{N}, \mathcal{E})$, the Laplacian matrix $\boldsymbol{L} \in \mathbb{R}^{N \times N}$ is given by $\boldsymbol{L} := \boldsymbol{D} - \boldsymbol{A}$, where $\boldsymbol{D} := \mathrm{diag}(|\mathcal{N}_1| - 1, \ldots, |\mathcal{N}_N| - 1)$ is the degree matrix ($|\cdot|$ denotes the cardinality of a set) and $\boldsymbol{A}$ is the adjacency matrix [2, 4] $[\boldsymbol{A}]_{kj} = \begin{cases} 1, & \text{if } \{k,j\} \in \mathcal{E} \text{ and } k \neq j \\ 0, & \text{otherwise.} \end{cases}$

• (*Unreliable links.*) For this scenario, we use the model of unreliable links proposed in [5]. In more detail, we start with a connected graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ obtained as described above. At each time instant i, we copy all edges from $\mathcal{E}$ to a new edge set $\mathcal{E}[i]$, and then we remove with probability p each edge $\{k, j\}$ ($k \neq j$) from $\mathcal{E}[i]$. The edge set $\mathcal{E}[i]$ defines a new graph $\mathcal{G}[i] = (\mathcal{N}, \mathcal{E}[i])$, and we use $\boldsymbol{W}[i] = \boldsymbol{I} - \epsilon \boldsymbol{L}[i]$, where $\boldsymbol{L}[i]$ is the Laplacian matrix associated with $\mathcal{G}[i] = (\mathcal{N}, \mathcal{E}[i])$. The physical interpretation of this model is that communication links, each of which corresponds to an edge in $\mathcal{E}$, can fail with probability p. As in the case with reliable links, we chose the value $\epsilon = 0.05$.

All acceleration methods in this section use the matrices $\boldsymbol{W}[i]$ described above. For convenience, we use the acronyms SSCA for the synchronous semi-iterative consensus algorithm and ASCA for the asynchronous semi-iterative consensus algorithm. The proposed acceleration schemes are compared with Laplacian-based methods without acceleration [2, 5] and with the following acceleration techniques:

• *The two-register eigenvalue shaping filter (ESF) in [10] (which is also the algorithm in [8] when the matrices $\boldsymbol{W}[i]$ are designed for consensus via network gossiping).* We showed in the discussion after (17) that this algorithm is equivalent to Algorithm 2 with $\alpha = -\beta$ and $\omega_\infty \in (1, 2)$. Our results are useful to help with the choice of ω_∞ when an upper bound on the second largest eigenvalue of the network matrix is available.

• *The optimal short-length polynomial filtering in [9] (algorithm denominated "polynomial" in the figures).* This algorithm uses more information than that available to the proposed schemes and the ESF acceleration method.

As in [8, 9], the performance metric of interest is the (absolute) squared error $\|\boldsymbol{o}[i] - x_{\rm av}\mathbf{1}\|^2$, where $\boldsymbol{o}[i] \in \mathbb{R}^N$ is the output of a given consensus algorithm at time i (e.g., $\boldsymbol{o}[i] = \boldsymbol{z}[i]$ in the case of our proposed acceleration schemes or $\boldsymbol{o}[i] = \boldsymbol{x}[i]$ in the case of the original consensus algorithm in (2)). In simulations where the network matrix is deterministic, all algorithms are guaranteed

[2] The samples n_k are different in different runs of the simulation

[3] The matrices $\boldsymbol{W}[i]$ are fixed and deterministic in this scenario, so we can simply ignore the expectation operator in Assumption 1.

to converge if the eigenvalues of the network matrix (except for the eigenvalue 1) fall into the interval $[\alpha, \beta]$. Therefore, we plot the average squared error performance of the algorithms in such scenarios. However, unless otherwise stated, we show the (sample) 99th-percentile squared error performance when bounds on the eigenvalues are not exactly known. By plotting percentile curves, we identify algorithms that consistently provide good performance, and we also give less emphasis to rare situations where acceleration algorithms may not converge (see also the discussion after Proposition 1). Sample average and percentile curves are calculated from the results of 1,000 simulations, each of which use different initial conditions.

4.1 Networks with Reliable Links

Having described the basic setup of the simulations, we now study the performance of the algorithms in specific settings. We begin with an ideal scenario where the topology is fixed, links are reliable, and the minimum and second largest eigenvalues of the network matrix are precisely known. This scenario is useful to show the maximum acceleration gain that can be achieved with the algorithms because the minimum and second largest eigenvalue are simple to compute. The network under consideration is depicted in Fig. 1. For simplicity, denote $\boldsymbol{W} := \boldsymbol{W}[i]$ (because in this scenario the network matrices are fixed and deterministic), $\lambda_{\max} := \max_{j\in\{2,\dots,N\}} \lambda_j(\boldsymbol{W})$ ($\lambda_{\max} > 0$ in all our simulations), and $\lambda_{\min} = \min_{j\in\{2,\dots,N\}} \lambda_j(\boldsymbol{W})$. We use the following parameters for the proposed acceleration schemes: SSCA ($\alpha = \lambda_{\min},\ \beta = \lambda_{\max}$) and ASCA ($\alpha = \lambda_{\min},\ \beta = \lambda_{\max}$). The parameter c in [10, Eq. (28)] is set to $c = 1 - \omega_\infty$, where ω_∞ is computed according to (14) with $\beta = -\alpha = \lambda_{\max}$ (see the discussion after (17) for the justification of this choice). Note that the ESF algorithm is basically the ASCA algorithm with $\beta = -\alpha$. We use filters of length eight for the method in [9] (this value is also used in [9]). Fig. 2 shows the performance of the algorithms.

As thoroughly discussed in [2], Laplacian-based algorithms have poor performance if the initial values reported by agents have a strong correlation with their positions, and this fact is indeed observed in Fig. 2. However, dramatic performance gains can be obtained by combining Laplacian-based methods with acceleration techniques. In particular, the SSCA and ASCA algorithms are able to provide in every agent values extremely close to x_{av} in very few iterations (as compared to the network size). The performance of the asynchronous algorithm ASCA closely follows that of its synchronous counterpart, the SSCA algorithm, which is optimal in a well defined sense. This result is not surprising because the asymptotic convergence of ω_i is fast. The performance advantage of the ASCA and SSCA algorithms over the ESF algorithm is explained by the fact that the former two algorithms use information about the lower bound on the eigenvalues of the network matrix. In contrast, the ESF algorithm, a particular case of the proposed method (see the discussion in Sect. 3.4), uses only information about the second largest eigenvalue (in magnitude), and the minimum eigenvalue is largely underestimated with the conservative lower bound $\alpha = -\beta$, which adversely affects the performance. The polynomial filtering scheme in [9], which requires precise knowledge of $\boldsymbol{W}$ has performance comparable to the ESF algorithm. However, note that the scheme in [9] requires more information about the network matrix $\boldsymbol{W}$ and has higher computation complexity than all other acceleration schemes. Its performance is inferior to that of the SSCA and ASCA algorithms because the proposed acceleration schemes are based on filters with increasing length.

We now study the stability of our proposed schemes when upper and lower bounds of the eigenvalues of the network matrix are imprecisely estimated. For visual clarity, we use the network in Fig. 1 in all runs of the simulation, and we plot only results obtained with the original Laplacian-based algorithm (for reference) and the following versions of the SSCA algorithm:[4] SSCA-under ($\beta = 0.9\lambda_{\max}, \alpha = \lambda_{\min}$) and SSCA-over ($\beta = \lambda_{\max}, \alpha = \lambda_{\min} + 0.05$). Note that SSCA-under underestimates the upper bound, whereas SSCA-over overestimates the lower bound. Fig. 3 shows the performance of the algorithms.

From Fig. 3 it is clear that, for the proposed algorithms, underestimating the upper bound is not so problematic as overestimating the lower bound. In contrast, neither convergence nor boundedness of $\boldsymbol{z}[i]$ is guaranteed if α is overestimated because $|p_i^\star(x)|$ grows fast outside $[\alpha, 1]$. This last fact explains the divergence of the SSCA-over algorithm.

In the simulations above, we have assumed exact knowledge of $\lambda_{\max}$ and $\lambda_{\min}$, which is rarely the case in practice. Therefore,

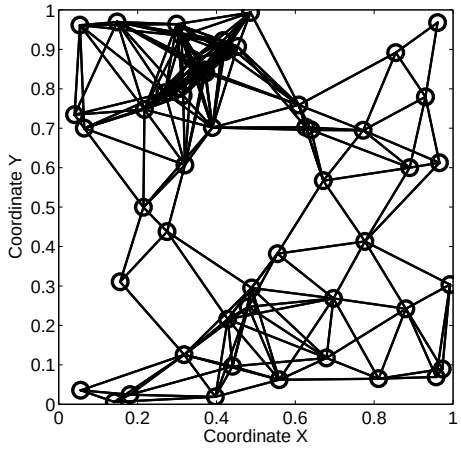

Figure 1: Network with 50 agents distributed uniformly at random in a square with unit area. Agents are represented by circles, and lines indicate communication links.

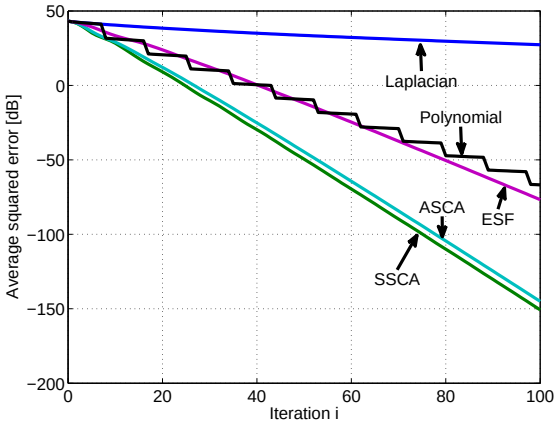

Figure 2: Transient performance of the algorithms. Every run of the simulation uses the topology in Fig 1.

[4] We omit the performance curves of the asynchronous algorithms because they are similar to those of the corresponding synchronous algorithms.

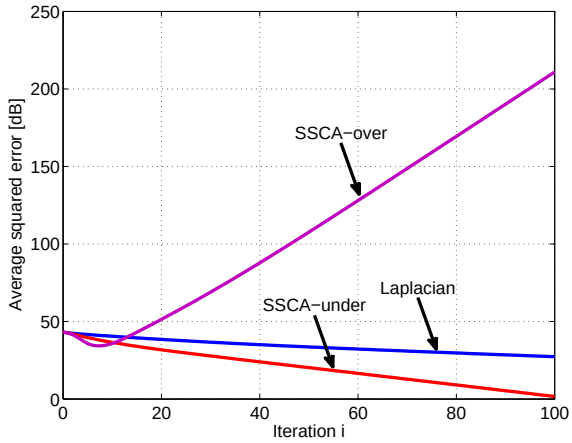

Figure 3: Stability of the algorithm with wrongly estimated bounds. Every run of the simulation uses the topology in Fig. 1.

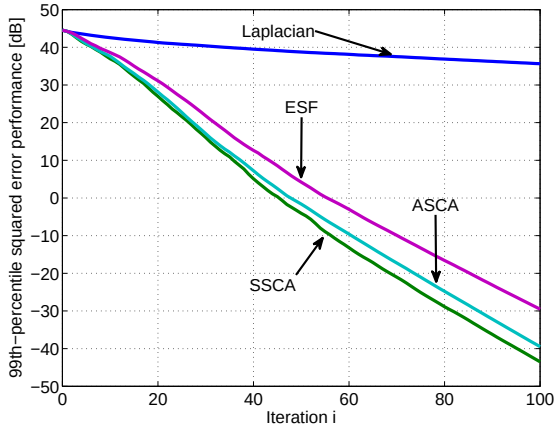

Figure 4: Transient performance of the algorithms. Network matrices $\boldsymbol{W}[i]$ are fixed and deterministic, but they change in every run of the simulation.

to evaluate the algorithm in more practical settings, we consider a scenario where the topology changes at every run of the simulation. In such a case, α and β should be set to appropriate values based on likely bounds on the eigenvalues. Given that the proposed algorithms are robust against underestimated upper bounds, we can set β to a value expected to be greater than $|\lambda_2(\boldsymbol{W}[i])|$ with fairly high probability. However, we should take a conservative approach to choosing α because, as discussed above, overestimated values can render the proposed algorithms unstable. By simulating 100,000 different networks with the geometric approach described above, $|\lambda_2(\boldsymbol{W}[i])| \leq 0.994$ occurred in less than 1.32% of the simulations, so we choose $\beta = 0.994$ for both the SSCA and ASCA algorithms because we do not need to be overly conservative on the choice of β. As for the parameter α, we use $\alpha = -0.5$ because eigenvalues less than -0.5 have not been observed in our simulations. Therefore, we can expect that the proposed algorithms using $\alpha = -0.5$ converge with high probability. Fig. 4 shows the 99th-percentile squared error performance of the algorithms obtained by randomizing the network (and also the initial values reported by the agents) at each run of the simulation. In this figure, we once again set the parameter c in [10, Eq. (28)] to $c = 1-\omega_\infty$, where ω_∞ was computed by using $0.994 = \beta = -\alpha$. We do not show the results of the polynomial filtering algorithm in [9] because, as in Fig. 2, its performance is worse than that obtained with other acceleration methods. In addition, it requires precise information about the network matrix in every run of the simulation, a very strong assumption in many multiagent systems.

With the settings in Fig. 4, the performance of Laplacian-based consensus methods is also poor, and all acceleration methods can greatly improve the convergence. The ASCA and SSCA algorithms were stable in all runs of our simulations, which is not surprising given the conservative choice of α. The ESF algorithm is basically the ASCA algorithm with an unduly underestimated parameter α, and this fact explains the worse performance of the ESF algorithm as compared to the SSCA and ASCA algorithms.

4.2 Networks with Unreliable Links

To study the stability of the acceleration algorithms with time-varying matrices $\boldsymbol{W}[i]$, we consider in Fig. 5 a scenario similar to that in Fig. 2, but with the difference that the communication links fail with probability $p = 0.2$ at each iteration (see the discussion in the beginning of this section). The parameters of all algorithms

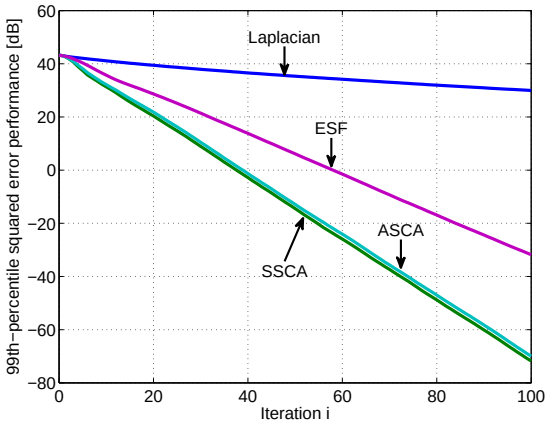

Figure 5: Transient performance of the algorithms. The network topology is the one in Fig. 1, but communication links fail with probability $p = 0.2$ at each iteration of the algorithms.

are the same as those in Fig. 2. We do not use exact bounds on the eigenvalues of $\overline{\boldsymbol{W}}$ to choose α and β because we want to illustrate a situation where the topology is supposed to be fixed and known, but the communication links are subject to failures that cannot be predicted (a common scenario in wireless networks). We omit once again the performance of the polynomial filtering approach in [9] because it did not converge in most runs of our simulations.

We can see in Fig. 5 that, with failing links, the proposed acceleration schemes (the ESF being a particular case) is acceptable because all agents are close to reach consensus on x_{av} in few iterations. In addition, the proposed algorithms converged in all runs of the simulation, which shows the good stability properties of our algorithms, even though Proposition 1 has only proved convergence in the mean sense. The relative performance of the algorithms is similar to that obtained in previous simulations, and the reason is the same as before.

In the last simulation, we study the impact of the network size on the convergence properties of the algorithms. In more detail, in Fig. 6 we show the (sample) median number of iterations that each

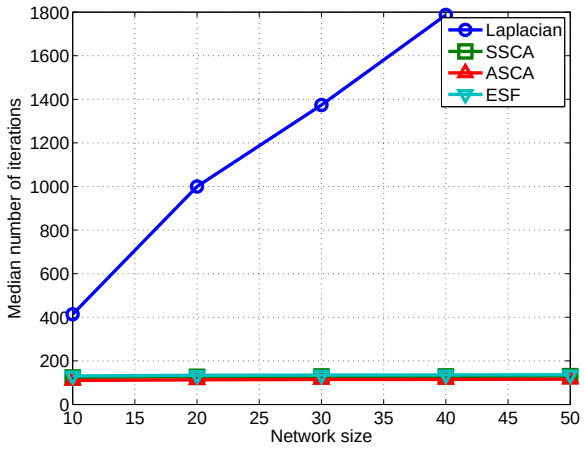

Figure 6: Median number of iterations required to reach the precision $\|\boldsymbol{o}[i] - \boldsymbol{x}[0]\|/\|\boldsymbol{x}[0]\| \leq 0.001$ as a function of the network size. The network topology changes at each run of the simulation, and communication links can fail with probability $p = 0.2$ at each iteration of the algorithms.

algorithm requires to reach the precision $\|\boldsymbol{o}[i] - \boldsymbol{x}[0]\|/\|\boldsymbol{x}[0]\| \leq 0.001$. In this figure, the network topology is randomized in every run of the simulation, and we decrease the connection range between agents to $d = 0.7\sqrt{\log(N)/N}$ to stress further the limitations of Laplacian-based methods and the gains obtained with acceleration algorithms. In addition, links fail with probability $p = 0.2$ at each iteration of the algorithms. If links are reliable, by keeping other conditions the same, the value 0.999 is a good estimate of the second largest eigenvalue of networks with sizes ranging from 10 to 50, so we use $\beta = 0.999$. More precisely, for networks of size 50, the second largest eigenvalue of the network matrix is greater than $\beta = 0.999$ with (empirical) probability less than 2% (with smaller networks, the probability is lower). For the minimum eigenvalue, eigenvalues less than -0.1 have not been observed in the simulations, so we use $\alpha = -0.1$. The parameters α and β have thus been adjusted to accommodate eigenvalues of (reliable) networks of different sizes and topologies. Note that the simulations in Fig. 6 use networks with unreliable links, and we did not try to estimate the eigenvalues of $\overline{\boldsymbol{W}}$ because the probability of failures cannot be usually predicted in real-world applications. The parameter c in [10, Eq. (28)] was once again set to $c = 1 - \omega_\infty$, where ω_∞ is given by (14) with $\beta = -\alpha$.

As proved in [2] and also observed in Fig. 6, Laplacian-based methods scale badly with the network size. However, all acceleration techniques are relatively insensitive to the network size, so they can be good alternatives to Laplacian-based methods in spatial computers. The compared acceleration schemes have similar performance because the precision, although fairly high, can be achieved in few iterations by all acceleration schemes (in previous simulations we can see that differences are usually more pronounced when we show 99th-percentile curves). Therefore, choosing parameters based on expected bounds on the eigenvalues of the network matrix (as proposed in this study) makes simple consensus algorithms practical in applications where approximate averages have to be computed with few iterations.

5. CONCLUSIONS

Laplacian-based methods for consensus have been identified as too slow to be practical in many multiagent applications, especially those involving large-scale systems [2]. However, in this study we have demonstrated that such methods can still be useful in large systems if they are combined with acceleration techniques. In particular, the convergence speed of our two novel algorithms is fast and decreases gracefully with the network size in scenarios where the sole use of Laplacian-based methods are known to be impractical. Our first algorithm requires agents with synchronized clocks, and it is optimal in a well defined sense. The second algorithm is an asynchronous method that is able to provide performance very close to that of the optimal synchronous algorithm. Unlike existing acceleration methods, we have only assumed that rough bounds on the extremal eigenvalues of the network matrix are available (those bounds can be readily obtained by considering typical application scenarios).

6. REFERENCES

[1] R. L. G. Cavalcante, A. Rogers, N. R. Jennings, and I. Yamada, "Distributed multiagent learning with a broadcast adaptive subgradient method," in *Proc. of the 9th Int. Conf. on Autonomous Agents and Multiagent Systems*, 2010, pp. 1039–1046.

[2] N. Elhage and J. Beal, "Laplacian-based consensus on spatial computers," in *Proc. of the 9th Int. Conf. on Autonomous Agents and Multiagent Systems*, May 2010, pp. 907–914.

[3] C.-H. Yu and R. Nagpal, "Sensing-based shape formation on modular multi-robot systems: A theoretical study," in *Proc. of the 7th Int. Conf. on Autonomous Agents and Multiagent Systems*, May 2008, pp. 71–78.

[4] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proc. IEEE*, vol. 95, no. 1, pp. 215–233, Jan. 2007.

[5] S. Kar and J. M. F. Moura, "Sensor networks with random links: Topology design for distributed consensus," *IEEE Trans. Signal Processing*, vol. 56, pp. 3315–3326, July 2008.

[6] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems and Control Letters*, vol. 53, pp. 65–78, Sept. 2004.

[7] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Trans. Inform. Theory*, vol. 52, no. 6, pp. 2508–2530, June 2006.

[8] M. Cao, D. A. Spielman, and E. M. Yeh, "Accelerated gossip algorithms for distributed computation," in *Forty-Fourth Annual Allerton Conference*, Sept. 2006, pp. 952–959.

[9] E. Kokiopoulou and P. Frossard, "Polynomial filtering for fast convergence in distributed consensus," *IEEE Transactions on Signal Processing*, vol. 57, no. 1, pp. 342–354, Jan. 2009.

[10] D. Scherber and H. C. Papadopoulos, "Distributed computation of averages over ad hoc networks," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 4, pp. 776–787, Apr. 2005.

[11] G. H. Golub and R. S. Varga, "Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order Richardson iterative methods," *Numerische Mathematik*, no. 3, pp. 145–156, 1961.

[12] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 3rd ed. Baltimore, MD: The Johns Hopkins Univ. Press, 1996.

[13] G. J. Pottie and W. J. Kaiser, "Wireless integrated network sensors," *Communications of the ACM*, vol. 43, no. 5, pp. 51–58, 2000.

[14] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388–404, Mar. 2000.