Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 67 (2007) 33–46
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Distributed average consensus with least-mean-square deviation

Lin Xiao[a,*], Stephen Boyd[b], Seung-Jean Kim[b]

[a]Center for the Mathematics of Information, California Institute of Technology, Pasadena, CA 91125-9300, USA
[b]Department of Electrical Engineering, Stanford University, Stanford, CA 94305-9510, USA

Received 27 May 2005; accepted 29 August 2006
Available online 27 October 2006

Abstract

We consider a stochastic model for distributed average consensus, which arises in applications such as load balancing for parallel processors, distributed coordination of mobile autonomous agents, and network synchronization. In this model, each node updates its local variable with a weighted average of its neighbors' values, and each new value is corrupted by an additive noise with zero mean. The quality of consensus can be measured by the total mean-square deviation of the individual variables from their average, which converges to a steady-state value. We consider the problem of finding the (symmetric) edge weights that result in the least mean-square deviation in steady state. We show that this problem can be cast as a convex optimization problem, so the global solution can be found efficiently. We describe some computational methods for solving this problem, and compare the weights and the mean-square deviations obtained by this method and several other weight design methods.

Keywords: Distributed average consensus; Least-mean-square; Convex optimization; Edge-transitive graphs

1. Introduction

1.1. Asymptotic average consensus

Average consensus is an important problem in algorithm design for distributed computing. Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be an undirected connected graph with node set $\mathcal{N} = \{1, \ldots, n\}$ and edge set $\mathcal{E}$, where each edge $\{i, j\} \in \mathcal{E}$ is an unordered pair of distinct nodes. Let $x_i(0)$ be a real scalar assigned to node i at time $t = 0$. The (distributed) average consensus problem is to compute the average $(1/n)\sum_{i=1}^{n} x_i(0)$ at every node, via local communication and computation on the graph. Thus, node i carries out its update, at each step, based on its local state and communication with its neighbors $N_i = \{j | \{i, j\} \in \mathcal{E}\}$.

Distributed average consensus has been extensively studied in computer science, for example in distributed agreement and synchronization problems (see, e.g., [17]). It is a central topic for load balancing (with divisible tasks) in parallel computers (see, e.g., [3,9,35]). More recently, it has also found applications in distributed coordination of mobile autonomous agents (e.g., [13,20,22,26]), and distributed data fusion in sensor networks (e.g., [28,29,34]).

There are several simple methods for distributed average consensus. For example, each node can store a table of all initial node values known at that time. At each step each pair of neighbors exchange tables of initial values (or just the entries the other node doesn't have), and update their tables. In this simple flooding algorithm, all nodes know all initial values in a number of steps equal to the diameter of the graph, at which point each can compute the average (or any other function of the initial values). Recently, Moallemi and Van Roy [18] have developed an iterative algorithm for average consensus based on *consensus propagation*.

In this paper, we focus on a particular class of iterative algorithms for average consensus, widely used in the applications cited above. Each node updates itself by adding a weighted sum of the local discrepancies, i.e., the differences between neighboring node values and its own

$$x_i(t+1) = x_i(t) + \sum_{j \in \mathcal{N}_i} W_{ij}(x_j(t) - x_i(t)),$$
$$i = 1, \ldots, n, \quad t = 0, 1, \ldots . \tag{1}$$

* Corresponding author.
E-mail addresses: lin.xiao@microsoft.com (L. Xiao), boyd@stanford.edu (S. Boyd), sjkim@stanford.edu (S.-J. Kim).

0743-7315/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2006.08.010

Here, W_{ij} is a weight associated with the edge $\{i, j\}$. These weights are algorithm parameters. Since we associate weights with undirected edges, we have $W_{ij} = W_{ji}$. (It is also possible to consider nonsymmetric weights, associated with ordered pairs of nodes.) The state at each node in the iteration (1) consists of a single real number, which overwrites the initial value. The algorithm is time-independent, i.e., does not depend on t. The algorithm is meant to compute the average asymptotically.

Setting $W_{ij} = 0$ for $j \notin \mathcal{N}_i$ and $W_{ii} = 1 - \sum_{j\in\mathcal{N}_i} W_{ij}$, the iterative method can be expressed as the simple linear iteration

$$x(t+1) = Wx(t), \quad t = 0, 1, \ldots,$$

with initial condition $x(0) = (x_1(0), \ldots, x_n(0))$. By construction, the weight matrix W satisfies

$$W = W^T, \quad W\mathbf{1} = \mathbf{1}, \quad W \in \mathcal{S}, \tag{2}$$

where $\mathbf{1}$ denotes the vector of all ones, and $\mathcal{S}$ denotes the matrices with sparsity patterns compatible with the graph:

$$\mathcal{S} = \{W \in \mathbf{R}^{n\times n} | W_{ij} = 0 \text{ if } i \neq j \text{ and } \{i, j\} \notin \mathcal{E}\}.$$

Conversely, any matrix W that satisfies these conditions can be associated with a choice of weight parameters in the iterative algorithm.

To achieve (asymptotic) average consensus no matter what the initial node values are, we must have

$$\lim_{t\to\infty} x(t) = \lim_{t\to\infty} W^t x(0) = (1/n)\mathbf{1}\mathbf{1}^T x(0)$$

for all $x(0)$. The (rank one) matrix on the right is the averaging matrix: $(1/n)\mathbf{1}\mathbf{1}^T z$ is the vector all of whose components are the average of the entries of z. We will use this matrix often, so we will denote it as

$$J = (1/n)\mathbf{1}\mathbf{1}^T.$$

The condition that we have asymptotic average consensus is

$$\lim_{t\to\infty} W^t = (1/n)\mathbf{1}\mathbf{1}^T = J.$$

Assuming that W satisfies the properties (2), this condition is equivalent to

$$\|W - J\| < 1, \tag{3}$$

where the norm is the spectral or maximum singular value norm. The norm $\|W - J\|$ gives a measure of the worst-case, asymptotic rate of convergence to consensus. Indeed, the Euclidean deviation of the node values from their average is guaranteed to be reduced by the factor $\|W - J\|$ at each step

$$\|x(t+1) - Jx(0)\| \leqslant \|W - J\|\|x(t) - Jx(0)\|,$$

(The vector norm here is the Euclidean norm, $\|u\| = (u^T u)^{1/2}$.)

Weights that satisfy the basic constraints (2), as well as the convergence condition (3), always exist. For example, we can take

$$W_{ij} = \begin{cases} 1/(d+1) & i \neq j, \ \{i, j\} \in \mathcal{E}, \\ 1 - d_i/(d+1) & i = j, \\ 0 & i \neq j, \ \{i, j\} \notin \mathcal{E}, \end{cases}$$

where d_i is the degree of node i, and $d = \max_i d_i$ is the degree of the graph. These are called the *max-degree* weights. If the graph is not bipartite, we can replace $d + 1$ in the expressions above with d. Another simple set of weights that always yield asymptotic average consensus are the *Metropolis–Hastings* weights,

$$W_{ij} = \begin{cases} 1/(\max\{d_i, d_j\}+1) & i \neq j, \ \{i, j\} \in \mathcal{E}, \\ 1 - \sum_{j\in\mathcal{N}_i} 1/(\max\{d_i, d_j\}+1) & i = j, \\ 0 & i \neq j, \ \{i, j\} \notin \mathcal{E}. \end{cases} \tag{4}$$

(See, e.g., [28,33].)

Many variations of the model (1) have also been studied. These include problems where the weights are not symmetric, problems where final agreement is achieved, but not necessarily to the average (e.g., [13,20,26]), and problems where the final node values have a specified non-uniform distribution (e.g., [11,27,33]). Convergence conditions have also been established for distributed consensus on dynamically changing graphs (e.g., [13,20,26,34]) and with asynchronous communication and computation ([2]; see also the early work in [31,32]). Other work gives bounds on the convergence factor $\|W - J\|$ for a particular choice of weights, in terms of various geometric quantities such as conductance (e.g., [30]). When W_{ij} are nonnegative, the model (1) corresponds to a symmetric Markov chain on the graph, and $\|W - J\|$ is the second largest eigenvalue magnitude (SLEM) of the Markov chain, which is a measure of mixing time (see, e.g., [6,8,10]).

In [33], we formulated the *fastest distributed linear averaging* (FDLA) problem: choose the weights to obtain fastest convergence, i.e., to minimize the asymptotic convergence factor $\|W - J\|$. We showed that (for symmetric weights) this FDLA problem is convex, and hence can be solved globally and efficiently. In this paper we study a similar optimal weight design problem, based on a stochastic extension of the simple averaging model (1).

1.2. Average consensus with additive noise

We now consider an extension of the averaging iteration (1), with a noise added at each node, at each step

$$x_i(t+1) = x_i(t) + \sum_{j\in\mathcal{N}_i} W_{ij}(x_j(t) - x_i(t)) + v_i(t), \quad i = 1, \ldots, n, \ \ t = 0, 1, \ldots . \tag{5}$$

Here, $v_i(t)$, $i = 1, \ldots, n$, $t = 0, 1, \ldots$ are independent random variables, identically distributed, with zero mean and unit variance. We can write this in vector form as

$$x(t+1) = Wx(t) + v(t),$$

where $v(t) = (v_1(t), \ldots, v_n(t))$. In the sequel, we will assume that W satisfies the conditions required for asymptotic average consensus without the noises, i.e., that the basic constraints (2) and the convergence condition (3) hold.

With the additive noise terms, the sequence of node values $x(t)$ becomes a stochastic process. The expected value of $x(t)$ satisfies $\mathbf{E}x(t+1) = W\mathbf{E}x(t)$, so it propagates exactly like the node values without the noise term. In particular, each component of the expected value converges to the average of the initial node values.

But the node values do not converge to the average of the initial node values in any useful sense. To see this, let $a(t) = (1/n)\mathbf{1}^T x(t)$ denote the average of the node values. Thus, $a(0)$ is the average of $x_i(0)$, and we have

$$a(t+1) = a(t) + (1/n)\mathbf{1}^T v(t),$$

using $\mathbf{1}^T W = \mathbf{1}^T$. The second term $(1/n)\mathbf{1}^T v(t)$ is a sequence of independent, zero mean, unit variance random variables. Therefore, the average $a(t)$ undergoes a random walk, starting from the initial average value $a(0) = (1/n)\mathbf{1}^T x(0)$. In particular, we have

$$\mathbf{E}a(t) = a(0), \quad \mathbf{E}(a(t) - \mathbf{E}a(t))^2 = t.$$

This shows that the additive noises induce a (zero mean) error in the average of node values, which has variance that increases linearly with time, independent of the particular weight matrix used. In particular, we do not have average consensus (except in the mean), for any choice of W.

There is, however, a more useful measure of consensus for the sequence $x(t)$. We define $z(t)$ to be the vector of deviations of the components of $x(t)$ from their average. This can be expressed in component form as $z_i(t) = x_i(t) - a(t)$, or as

$$z(t) = x(t) - Jx(t) = (I - J)x(t).$$

We define the (total) mean-square deviation as

$$\delta(t) = \mathbf{E}\sum_{i=1}^{n}(x_i(t) - a(t))^2 = \mathbf{E}\|(I - J)x(t)\|^2.$$

This is a measure of relative deviation of the node values from their average, and can also be expressed as

$$\delta(t) = \frac{1}{n}\mathbf{E}\sum_{i<j}(x_i(t) - x_j(t))^2,$$

i.e., it is proportional to the average pairwise expected deviation among the node values (the exact average need a factor $2/(n(n-1))$ instead of $1/n$). This shows that the mean-square deviation $\delta(t)$ can be interpreted as a measure of how far the components of $x(t)$ are from consensus.

We will show that (assuming W satisfies (2) and (3)), the mean-square deviation $\delta(t)$ converges to a finite (steady-state) value as $t \to \infty$, which we denote δ_{ss}:

$$\delta_{\mathrm{ss}} = \lim_{t\to\infty}\delta(t).$$

This steady-state mean-square deviation is a function of the weights W, so we will denote it as $\delta_{\mathrm{ss}}(W)$. The steady-state mean-square deviation $\delta_{\mathrm{ss}}(W)$ is a measure of how well the weight matrix W is able to enforce consensus, despite the additive errors introduced at each node at each step.

1.3. Least-mean-square consensus problem

In this paper we study the following problem: given the graph, find edge weights that yield the smallest steady-state mean-square deviation. This can be posed as the following optimization problem:

$$\begin{array}{ll}\text{minimize} & \delta_{\mathrm{ss}}(W)\\ \text{subject to} & W = W^T, \quad W\mathbf{1} = \mathbf{1},\\ & \|W - J\| < 1, \ W \in \mathcal{S},\end{array} \tag{6}$$

with variable $W \in \mathbf{R}^{n\times n}$. We call the problem (6) the *least-mean-square consensus* (LMSC) problem.

For future use, we describe an alternative formulation of the LMSC problem that is parametrized by the edge weights, instead of the weight matrix W. We enumerate the edges $\{i, j\} \in \mathcal{E}$ by integers $k = 1, \ldots, m$, where $m = |\mathcal{E}|$. We write $k \sim \{i, j\}$ if the edge $\{i, j\}$ is labeled k. We assign an arbitrary direction or orientation for each edge. Now suppose $k \sim \{i, j\}$, with the edge direction being from i to j. We associate with this edge the vector a_{ij} in $\mathbf{R}^n$ with ith element $+1$, jth element -1, and all other elements zero. We can then write the weight matrix as

$$W = I - \sum_{\{i,j\}\in\mathcal{E}} W_{ij}a_{ij}a_{ij}^T = I - \sum_{k=1}^{m} w_k a_k a_k^T, \tag{7}$$

where w_k denotes the weight on the kth edge. (Note that terms $a_k a_k^T$ are independent of the orientation assigned to the edges. Indeed, these matrices have only 4 nonzero elements, with 1 at two locations on the diagonal, i, i and j, j, and -1 at two locations off the diagonal, i, j and j, i.)

It can be verified that the parametrization (7) of W automatically satisfies the basic constraints (2), and that conversely, any W that satisfies the basic constraints (2) can be expressed in the form (7). Thus, we can express the LMSC problem (6) as

$$\begin{array}{ll}\text{minimize} & \delta_{\mathrm{ss}}\left(I - \sum_{k=1}^{m} w_k a_k a_k^T\right)\\ \text{subject to} & \left\|I - J - \sum_{k=1}^{m} w_k a_k a_k^T\right\| < 1,\end{array} \tag{8}$$

with variable $w \in \mathbf{R}^m$. In this formulation the only constraint is the convergence condition $\|W - J\| < 1$.

1.4. Applications

The model of average consensus with additive noises (5) and the LMSC problem (6) arise naturally in many practical applications. Here, we briefly discuss its role in load balancing, coordination of autonomous agents, and network synchronization.

In the literature of load balancing, most work has focused on the *static* model (1), which is called a *diffusion scheme* because it can be viewed as a discretized diffusion equation (Poisson equation) on the graph [3]. Nevertheless, the stochastic version (5) is often more relevant in practice, in particular,

for *dynamic* load balancing problems where a random amount of (divisible) tasks are generated during the load balancing process. In fact, one of the first models for a diffusion scheme proposed in [9] is of this kind

$$q_i(t+1) = q_i(t) + \sum_{j\in\mathcal{N}_i} W_{ij}(q_j(t) - q_i(t)) - c + u_i(t).$$

Here, $q_i(t)$ is the amount of (divisible) tasks waiting to be processed at node i at time t (the queue length), c is the constant number of tasks that every processor can complete in unit time, and $u_i(t)$ is a nonnegative random variable that accounts for new tasks generated at time t for processor i. The quantity $W_{ij}(q_j(t) - q_i(t))$ is the amount of tasks transferred from processor j to i (a negative number means transferring in the opposite direction). As discussed in [9], the most interesting case is when $\mathbf{E}u_i(t) = c$, and this is precisely the model (5) with the substitutions $v_i(t) = u_i(t) - c$ and $x_i(t) = q_i(t) - q_i(0)$. (Instead of adding the constraint $q_i(t) \geqslant 0$, we assume the initial queue lengths $q_i(0)$ are large so that $q_i(t)$ remain nonnegative with very high probability.)

In dynamic load balancing problems, it is desirable to keep the mean-square deviation as small as possible, i.e., to distribute the loads most evenly in a stochastic sense. This is precisely the LMSC problem (6), which (to our knowledge) has not been addressed before.

For distributed coordination of mobile autonomous agents, the variable $x_i(t)$ can represent the position or velocity of each individual agent (e.g., in the context of [13,22]). The additive noises $v_i(t)$ in (5) can model random variations, e.g., caused by disturbances on the dynamics of each local agent. Here, the LMSC problem (6) is to obtain the best coordination in steady-state by optimizing the edge weights.

Another possible application of the LMSC problem is *drift-free* clock synchronization in distributed systems (e.g., [25]). Here, $x_i(t)$ represents the reading of a local relative clock (with the constant rate deducted), corrupted by random noise $v_i(t)$. Each node of the network adjusts its local clock via the diffusion scheme (5). The LMSC problem (6) amounts to finding the optimal edge weights that give the smallest (mean-square) synchronization error.

1.5. Outline

In Section 2, we derive several explicit expressions for the steady-state mean-square deviation $\delta_{\mathrm{ss}}(W)$, and show that the LMSC problem is a convex optimization problem. In Section 3 we discuss computational methods for solving the LMSC problem, and explain how to exploit problem structure such as sparsity in computing the gradient and Hessian of δ_{ss}. In Section 4, we consider a special case of the LMSC problem where all edge weights are taken to be equal, and illustrate its application to edge-transitive graphs. In Section 5, we present some numerical examples of the LMSC problem, and compare the resulting mean-square deviation with those given by other weight design methods, including the FDLA weights in [33].

2. Steady-state mean-square deviation

In this section we give a detailed analysis of the steady-state mean-square deviation δ_{ss}, including several useful and interesting formulas for it. We start with

$$x(t+1) = Wx(t) + v(t),$$

where W satisfies the basic constraints (2) and the convergence condition (3), and $v_i(t)$ are independent, identically distributed random variables with zero mean and unit variance. The deviation vector $z(t)$, defined as $z(t) = (I - J)\,x(t)$, satisfies $\mathbf{1}^T z(t) = 0$, and the recursion

$$z(t+1) = (W - J)z(t) + (I - J)v(t). \tag{9}$$

Therefore, we have

$$\mathbf{E}z(t) = (W - J)^t \mathbf{E}z(0) = (W - J)^t (I - J)x(0),$$

which converges to zero as $t \to \infty$, since $\|W - J\| < 1$.

Let $\Sigma(t) = \mathbf{E}z(t)z(t)^T$ be the second moment matrix of the deviation vector. The total mean-square deviation can be expressed in terms of $\Sigma(t)$ as

$$\delta(t) = \mathbf{E}\|z(t)\|^2 = \mathbf{Tr}\,\Sigma(t).$$

By forming the outer products of both sides of Eq. (9), we have

$$\begin{aligned} z(t+1)z(t+1)^T = {} & (W - J)z(t)z(t)^T(W - J) \\ & + (I - J)v(t)v(t)^T(I - J) \\ & + (W - J)z(t)v(t)^T(I - J) \\ & + (I - J)v(t)z(t)^T(W - J). \end{aligned}$$

Taking the expectation on both sides, and noticing that $v(t)$ has zero mean and is independent of $z(t)$, we obtain a difference equation for the deviation second moment matrix,

$$\begin{aligned} \Sigma(t+1) &= (W - J)\Sigma(t)(W - J) + (I - J)I(I - J) \\ &= (W - J)\Sigma(t)(W - J) + I - J. \end{aligned} \tag{10}$$

(The second equality holds since $(I - J)$ is a projection matrix.) The initial condition is

$$\Sigma(0) = \mathbf{E}z(0)z(0)^T = (I - J)x(0)x(0)^T(I - J).$$

Since $\|W - J\| < 1$, the difference equation (10) is a stable linear recursion. It follows that the recursion converges to a steady-state value $\Sigma_{\mathrm{ss}} = \lim_{t\to\infty} \Sigma(t)$, that is independent of $\Sigma(0)$ (and therefore $x(0)$), which satisfies the discrete-time Lyapunov equation

$$\Sigma_{\mathrm{ss}} = (W - J)\Sigma_{\mathrm{ss}}(W - J) + (I - J). \tag{11}$$

We can express Σ_{ss} as

$$\begin{aligned} \Sigma_{\mathrm{ss}} &= \sum_{t=0}^{\infty} (W - J)^t (I - J)(W - J)^t \\ &= (I - J) + \sum_{t=1}^{\infty} \left(W^2 - J\right)^t \end{aligned}$$

$$= \sum_{t=0}^{\infty} \left(W^2 - J\right)^t - J$$
$$= \left(I + J - W^2\right)^{-1} - J.$$

In several steps here we use the conditions (2) and (3), which ensure the existence of the inverse in the last line. We also note, for future use, that $I + J - W^2$ is positive definite, since it can be expressed as $I + J - W^2 = (\Sigma_{ss} + J)^{-1}$.

2.1. Expressions for steady-state mean-square deviation

Now we can write the steady-state mean-square deviation as an explicit function of W:

$$\delta_{ss}(W) = \mathbf{Tr}\,\Sigma_{ss} = \mathbf{Tr}\left(I + J - W^2\right)^{-1} - 1, \tag{12}$$

which we remind the reader holds assuming W satisfies the average consensus conditions $W = W^T$, $W\mathbf{1} = \mathbf{1}$, and $\|W - J\| < 1$. This expression shows that δ_{ss} is an analytic function of W, since the inverse of a matrix is a rational function of the matrix (by Cramer's formula). In particular, it has continuous derivatives of all orders.

We give another useful variation of the formula (12). We start with the identity

$$I + J - W^2 = (I - J + W)(I + J - W),$$

which can be verified by multiplying out, and noting that $J^2 = J$ and $JW = WJ = J$. Then we use the identity

$$((I - B)(I + B))^{-1} = (1/2)(I + B)^{-1} + (1/2)(I - B)^{-1},$$

with $B = W - J$ to obtain

$$(I + J - W^2)^{-1} = (1/2)(I + W - J)^{-1} + (1/2)(I - W + J)^{-1}.$$

Therefore, we can express $\delta_{ss}(W)$ as

$$\delta_{ss}(W) = (1/2)\mathbf{Tr}(I + J - W)^{-1} + (1/2)\mathbf{Tr}(I - J + W)^{-1} - 1. \tag{13}$$

The condition $\|W - J\| < 1$ is equivalent to $-I \prec W - J \prec I$, where $\prec$ denotes (strict) matrix inequality. These inequalities can be expressed as

$$I + J - W \succ 0, \quad I - J + W \succ 0.$$

This shows that the two matrices inverted in expression (13) are positive definite. We can therefore conclude that δ_{ss} is a convex function of W, since the trace of the inverse of a positive definite symmetric matrix is a convex function of the matrix [7, Exercise 3.57]. This, in turn, shows that the LMSC problem (6), and its formulation in terms of the edge weights (8), are convex optimization problems.

Finally, we give an expression for δ_{ss} in terms of the eigenvalues of W. From (13), and using the fact that the trace of a matrix is the sum of its eigenvalues, we have

$$\delta_{ss}(W) = (1/2)\sum_{i=1}^{n} \frac{1}{\lambda_i(I + J - W)} + (1/2)\sum_{i=1}^{n} \frac{1}{\lambda_i(I - J + W)} - 1,$$

where $\lambda_i(\cdot)$ denotes the ith largest eigenvalue of a symmetric matrix. Since $W\mathbf{1} = \mathbf{1}$ (which corresponds to the eigenvalue $\lambda_1(W) = 1$), the eigenvalues of $I - J + W$ are one, together with $1 + \lambda_2(W), \ldots, 1 + \lambda_n(W)$. A similar analysis shows that the eigenvalues of $I + J - W$ are one, together with $1 - \lambda_2(W), \ldots, 1 - \lambda_n(W)$. Therefore, we can write

$$\delta_{ss}(W) = (1/2)\sum_{i=2}^{n} \frac{1}{1 - \lambda_i(W)} + (1/2)\sum_{i=2}^{n} \frac{1}{1 + \lambda_i(W)} = \sum_{i=2}^{n} \frac{1}{1 - \lambda_i(W)^2}. \tag{14}$$

This simple formula has a nice interpretation. To achieve asymptotic average consensus, the weight matrix W is required to have $\lambda_1(W) = 1$, with the other eigenvalues strictly between -1 and 1 (since $\|W - J\| < 1$). It is the eigenvalues $\lambda_2(W), \ldots, \lambda_n(W)$ that determine the dynamics of the average consensus process. The asymptotic convergence factor is given by

$$\|W - J\| = \max\{\lambda_2(W), -\lambda_n(W)\}$$

and so is determined entirely by the largest (in magnitude) eigenvalues (excluding $\lambda_1(W) = 1$). The formula (14) shows that the steady-state mean-square deviation is also a function of the eigenvalues (excluding $\lambda_1(W) = 1$), but one that depends on all of them, not just the largest and smallest. The function $1/(1 - \lambda^2)$ can be considered a barrier function for the interval $(-1, 1)$ (i.e., a smooth convex function that grows without bound as the boundary is approached). The steady-state mean-square deviation δ_{ss} is thus a barrier function for the constraint that $\lambda_2(W), \ldots, \lambda_n(W)$ must lie in the interval $(-1, 1)$. In other words, δ_{ss} grows without bound as W approaches the boundary of the convergence constraint $\|W - J\| < 1$.

2.2. Some bounds on steady-state mean-square deviation

Our expression for δ_{ss} can be related to a bound obtained in [9]. If the covariance matrix of the additive noise $v(t)$ is given by $\sigma^2 I$, then it is easy to show that

$$\delta_{ss}(W) = \sum_{i=2}^{n} \frac{\sigma^2}{1 - \lambda_i(W)^2}.$$

The upper bound on δ_{ss} in [9] is

$$\delta_{ss}(W) \leqslant \frac{(n-1)\sigma^2}{1 - \|W - J\|^2}$$

which is a direct consequence of the fact $|\lambda_i(W)| \leqslant \|W - J\|$ for $i = 2, \ldots, n$.

We can give a similar bound, based on both the spectral norm $\|W - J\|$ (which is $\max\{\lambda_2(W), -\lambda_n(W)\}$), and the Frobenius norm $\|W - J\|_F$,

$$\|W - J\|_F^2 = \sum_{i,j=1}^{n} (W - J)_{ij}^2 = \sum_{i=2}^{n} \lambda_i(W)^2.$$

For $|u| \leqslant a < 1$, we have

$$1 + u^2 \leqslant \frac{1}{1 - u^2} \leqslant 1 + \frac{1}{1 - a^2} u^2.$$

Using these inequalities with $a = \|W - J\|$ and $u = \lambda_i$, for $i = 2, \ldots, n$, we obtain

$$n - 1 + \sum_{i=2}^{n} \lambda_i(W)^2 \leqslant \delta_{\mathrm{ss}}(W) = \sum_{i=2}^{n} \frac{1}{1 - \lambda_i(W)^2}$$

and

$$\delta_{\mathrm{ss}}(W) \leqslant n - 1 + \frac{1}{1 - \|W - J\|^2} \sum_{i=2}^{n} \lambda_i(W)^2.$$

Thus we have

$$n - 1 + \|W - J\|_F^2 \leqslant \delta_{\mathrm{ss}}(W) \leqslant n - 1 + \frac{\|W - J\|_F^2}{1 - \|W - J\|^2}.$$

2.3. Computing the steady-state mean-square deviation

In this section we describe methods that can be used to compute δ_{ss} for a fixed W (the weight matrix) or w (the vector of edge weights). One straightforward method is to compute all the eigenvalues of W, which allows us to check the convergence condition $\|W - J\| < 1$, as well as evaluate $\delta_{\mathrm{ss}}(W)$ using (14). If we exploit no structure in W (other than, of course, symmetry), the computational cost of this approach is $O(n^3)$.

We can also use the formula (12). We first form the matrix $I + J - W^2$, and then carry out Cholesky factorization of it (which serves to verify $\|W - J\| < 1$):

$$U^T U = I + J - W^2,$$

where U is upper triangular. We then form the inverse U^{-1}, and evaluate $\delta_{\mathrm{ss}}(W)$ as

$$\begin{aligned}\delta_{\mathrm{ss}}(W) &= \mathbf{Tr}(I + J - W^2)^{-1} - 1 \\ &= \mathbf{Tr}\, U^{-1} U^{-T} - 1 \\ &= \|U^{-1}\|_F^2 - 1 \\ &= \sum_{i,j} (U^{-1})_{ij}^2 - 1.\end{aligned}$$

Ignoring all structure in W (other than symmetry) this method is also $O(n^3)$.

The last method we describe is based on the formula (13), and has the advantage that it can be modified to exploit sparsity of the graph. We first describe the basic method, which does not exploit sparsity of W. We first carry out Cholesky factorizations,

$$F = I - J + W = U^T U, \quad G = I + J - W = \tilde{U}^T \tilde{U}, \quad (15)$$

which also serves to verify that F and G are positive definite, which is equivalent to $\|W - J\| < 1$. We then compute the inverses of these Cholesky factors, and compute their Frobenius norms:

$$\begin{aligned}\mathbf{Tr}\, F^{-1} &= \mathbf{Tr}\, U^{-1} U^{-T} = \|U^{-1}\|_F^2 = \sum_{i,j} (U^{-1})_{ij}^2, \\ \mathbf{Tr}\, G^{-1} &= \mathbf{Tr}\, \tilde{U}^{-1} \tilde{U}^{-T} = \|\tilde{U}^{-1}\|_F^2 = \sum_{i,j} (\tilde{U}^{-1})_{ij}^2.\end{aligned}$$

Finally, we have $\delta_{\mathrm{ss}}(W) = (1/2)\,\mathbf{Tr}\, F^{-1} + (1/2)\,\mathbf{Tr}\, G^{-1} - 1$. If we exploit no structure in W, the computational cost of this approach is $O(n^3)$, the same as the two methods already described.

This last method, however, can be adapted to exploit sparsity of W, and therefore can handle larger graphs. Assuming that the graph (and therefore W) is sparse, both F and G have the form of a rank one matrix plus a sparse matrix. Using the Sherman–Morrison–Woodbury formula, we can compute $\mathbf{Tr}\, F^{-1}$ and $\mathbf{Tr}\, G^{-1}$ efficiently. We start with

$$\begin{aligned}F^{-1} &= (I + W - (1/n)\mathbf{1}\mathbf{1}^T)^{-1} \\ &= (I + W)^{-1} - \frac{1}{n(1 - (1/n)\mathbf{1}^T (I + W)^{-1}\mathbf{1})} \\ &\quad \times (I + W)^{-1} \mathbf{1}\mathbf{1}^T (I + W)^{-1}. \quad (16)\end{aligned}$$

Taking the trace we obtain

$$\begin{aligned}\mathbf{Tr}\, F^{-1} &= \mathbf{Tr}(I + W)^{-1} - \frac{1}{n - \mathbf{1}^T (I + W)^{-1}\mathbf{1}} \|(I + W)^{-1}\mathbf{1}\|^2 \\ &= \sum_{i,j} (U^{-1})_{ij}^2 - \frac{1}{n - \|U^{-T}\mathbf{1}\|^2} \|U^{-1} U^{-T}\mathbf{1}\|^2,\end{aligned}$$

where U is the Cholesky factor, after re-ordering, of $I + W$ (which is sparse and positive definite):

$$P U^T U P^T = I + W.$$

(Here P is the permutation matrix chosen to reduce the number of nonzeros elements of U.) Let N denote the number of nonzero elements in U. The effort of forming the inverse U^{-1} will be dominated by the n back-substitutions, i.e., computing $U^{-1}e_1, \ldots, U^{-1}e_n$, which has cost $O(nN)$. (That is, we can ignore the cost of the Cholesky factorization, as well as computing the quantity on the right-hand side of the last equation above.) Thus the cost of computing $\mathbf{Tr}\, F^{-1}$ is $O(nN)$. The matrix U^{-1} is never needed all at once; we can compute its columns one by one, and accumulate the sum of the squares of the entries of the columns to obtain the Frobenius norm. Thus, the storage requirement of this method is $O(N)$, not $O(nN)$.

A similar method can be used to compute $\mathbf{Tr}\, G^{-1}$, although there is a subtlety involved, since $I-W$ is singular (its nullspace is the line generated by $\mathbf{1}$). To use the Sherman–Morrison–Woodbury formula, we need to express G as the sum of a nonsingular sparse matrix and a low rank matrix. One way to do this is

$$\begin{aligned} G &= I - W + e_1 e_1^T + (1/n)\mathbf{1}\mathbf{1}^T - e_1 e_1^T \\ &= I - W + e_1 e_1^T + V D V^T, \end{aligned}$$

where e_1 is the vector with its first entry being one and all other entries being zero, and

$$V = [\,\mathbf{1}\ e_1\,] \in \mathbf{R}^{n\times 2}, \quad D = \begin{bmatrix} 1/n & 0 \\ 0 & -1 \end{bmatrix}.$$

The matrix $I - W + e_1 e_1^T$ is sparse and positive definite, since $I - W$ is positive semidefinite, and the second term $e_1 e_1^T$ is nonzero on the nullspace of $I - W$ (i.e., the line generated by $\mathbf{1}$). Now we use the Sherman–Morrison–Woodbury formula to obtain

$$\begin{aligned} G^{-1} = (I - W + e_1 e_1^T)^{-1} - (I - W + e_1 e_1^T)^{-1} \\ \times V C V^T (I - W + e_1 e_1^T)^{-1}, \end{aligned} \tag{17}$$

where

$$C = (I + V^T (I - W + e_1 e_1^T)^{-1} V)^{-1} \in \mathbf{R}^{2\times 2}.$$

Now we can find $\mathbf{Tr}\, G^{-1}$ efficiently as follows. Let U be the Cholesky factor, after re-ordering, of $I - W + e_1 e_1$:

$$P U^T U P^T = I - W + e_1 e_1^T.$$

Then we have

$$\begin{aligned} \mathbf{Tr}\, G^{-1} &= \mathbf{Tr}\left(I - W + e_1 e_1^T\right)^{-1} - \mathbf{Tr}\left(I - W + e_1 e_1^T\right)^{-1} \\ &\quad \times V C V^T \left(I - W + e_1 e_1^T\right)^{-1} \\ &= \sum_{i,j} \left(U^{-1}\right)_{ij}^2 - \mathbf{Tr}\left(C\left(V^T P U^{-1} U^{-T} P^T\right)\right. \\ &\quad \left. \times \left(P U^{-1} U^{-T} P^T V\right)\right). \end{aligned}$$

The second term in the last expression, which is the trace of a 2×2 matrix, looks complicated but is easily computed. Indeed, the $n\times 2$ matrix $U^{-1}U^{-T}P^T V$ is nothing more than $(I - W + e_1 e_1)^{-1}[\mathbf{1} e_1]$, which can be found (as the formula suggests) by a back and a forward substitution. Evaluating the first term, the sum of the squares of the elements of U^{-1}, can be done by back substitutions of $e_1, \ldots, e_n$. Thus the computational cost of computing $\mathbf{Tr}\, G^{-1}$ is $O(nN)$, the same as computing $\mathbf{Tr}\, F^{-1}$.

All together, the total flop count of this method is $O(nN)$. When N is on the order of n, this gives an $O(n^2)$ algorithm, one order faster than the methods described above (that do not exploit sparsity), which are $O(n^3)$. The storage requirement in $O(N)$.

2.4. Derivation via spectral functions

In this section we show how convexity of $\delta_{\mathrm{ss}}(W)$, with the expression (14), can be derived using the theory of *convex spectral functions* [4, Section 5.2]. For $y \in \mathbf{R}^n$, we write $[y]$ as the vector with its components rearranged into nonincreasing order; i.e., $[y]_i$ is the ith largest component of y. A function $g : \mathbf{R}^n \to \mathbf{R}$ is called *symmetric* if $g(y) = g([y])$ for all vectors $y \in \mathbf{R}^n$. In other words, a symmetric function is invariant under permutation of its arguments. Let g be a symmetric function and $\lambda(\cdot)$ denote the vector of eigenvalues of a symmetric matrix, arranged in nonincreasing order. The composite function $g \circ \lambda$ is called a *spectral function*. It is easily shown that a spectral function is *orthogonally invariant*; i.e.,

$$(g \circ \lambda)(Q W Q^T) = (g \circ \lambda)(W)$$

for any orthogonal Q and any symmetric matrix W in $\mathbf{R}^{n\times n}$.

A spectral function $g\circ\lambda$ is closed and convex if and only if the corresponding symmetric function g is closed and convex (see, e.g., [14] and [4, Section 5.2]). Examples of convex spectral functions include the trace, largest eigenvalue, and the sum of the k largest eigenvalues, for any symmetric matrix; and the trace of the inverse, and log determinant of the inverse, for any positive definite matrix. More examples and details can be found in, e.g., [14,23].

From the expression (14), we see that the function $\delta_{\mathrm{ss}}(W)$ is a spectral function, associated with the symmetric function

$$g(y) = \begin{cases} \displaystyle\sum_{i=2}^{n} \frac{1}{1 - [y]_i^2} & \text{if } [y]_{i=2}^n \in (-1, 1)^{n-1}, \\ +\infty & \text{otherwise.} \end{cases}$$

Since g is closed and convex, we conclude that the spectral function δ_{ss} is also closed and convex. Furthermore, δ_{ss} is twice continuously differentiable because the above symmetric function g is twice continuously differentiable at $[y]$. We can derive the gradient and Hessian of δ_{ss} following the general formulas for spectral functions, as given in [4, Section 5.2, 15]. In this paper, however, we derive simple expressions for the gradient and Hessian by directly applying the chain rule; see Section 3.

3. Solving the LMSC problem

In this section we describe computational methods for solving the LMSC problem (6). We will focus on the formulation (8), with edge weights as variables. We have already noted that the steady-state mean-square deviation δ_{ss} is a barrier function for the convergence condition $\|W - J\| < 1$, which can therefore be neglected in the optimization problem (8), provided we interpret δ_{ss} as ∞ when the convergence condition does not hold. In other words, we must solve the unconstrained problem

$$\text{minimize} \quad f(w) = \delta_{\mathrm{ss}}\left(I - \sum_{k=1}^{m} w_k a_k a_k^T\right), \tag{18}$$

with variable $w \in \mathbf{R}^m$, where we interpret $f(w)$ as ∞ whenever $\|I - \sum_{k=1}^m w_k a_k a_k^T - J\| \geqslant 1$.

This is a smooth unconstrained convex optimization problem, and so can be solved by many standard methods, such as gradient descent method, quasi-Newton method, conjugate gradient method, or Newton's method. These methods have well known advantages and disadvantages in speed of convergence, computational cost per iteration, and storage requirements; see, e.g., [16,1,7,21]. These algorithms must be initialized with a point, such as the Metropolis–Hastings weight (4), that satisfies $f(w) < \infty$. At each step of these algorithms, we need to compute the gradient $\nabla f(w)$, and for Newton's method, the Hessian $\nabla^2 f(w)$ as well. In the next few sections we derive expressions for the gradient and Hessian, and describe methods that can be used to compute them.

3.1. Gradient

We start with the formula (13), with $W = I - \sum_{k=1}^m w_k a_k a_k^T$,

$$f(w) = (1/2)\,\mathbf{Tr}\, F(w)^{-1} + (1/2)\,\mathbf{Tr}\, G(w)^{-1} - 1,$$

where

$$F(w) = 2I - \sum_{k=1}^m w_k a_k a_k^T - J, \quad G(w) = \sum_{k=1}^m w_k a_k a_k^T + J.$$

Suppose that weight w_k corresponds to edge $\{i, j\}$, i.e., $k \sim \{i, j\}$. Then we have

$$\begin{aligned}
\frac{\partial f}{\partial w_k} &= -(1/2)\,\mathbf{Tr}\left(F^{-1}\frac{\partial F}{\partial w_k}F^{-1}\right) - (1/2)\,\mathbf{Tr}\left(G^{-1}\frac{\partial G}{\partial w_k}G^{-1}\right)\\
&= (1/2)\,\mathbf{Tr}\left(F^{-1}a_k a_k^T F^{-1}\right) - (1/2)\,\mathbf{Tr}\left(G^{-1}a_k a_k^T G^{-1}\right)\\
&= (1/2)\|F^{-1}a_k\|^2 - (1/2)\|G^{-1}a_k\|^2\\
&= (1/2)\left\|\left(F^{-1}\right)_{:,i} - \left(F^{-1}\right)_{:,j}\right\|^2 - (1/2)\left\|\left(G^{-1}\right)_{:,i}\right.\\
&\quad \left. - \left(G^{-1}\right)_{:,j}\right\|^2, \qquad (19)
\end{aligned}$$

where $(F^{-1})_{:,i}$ denotes the ith column of F^{-1} (and similarly for G). In the first line, we use the fact that if a symmetric matrix X depends on a parameter t, then

$$\frac{\partial X^{-1}}{\partial t} = -\left(X^{-1}\frac{\partial X}{\partial t}X^{-1}\right).$$

The formula (19) gives us the optimality conditions for the problem (18): a weight vector $w^\star$ is optimal if and only if $F(w^\star) \succ 0$, $G(w^\star) \succ 0$, and, for all $\{i, j\} \in \mathcal{E}$, we have

$$\begin{aligned}
&\left\|\left(F(w^\star)^{-1}\right)_{:,i} - \left(F(w^\star)^{-1}\right)_{:,j}\right\|\\
&\quad = \left\|\left(G(w^\star)^{-1}\right)_{:,i} - \left(G(w^\star)^{-1}\right)_{:,j}\right\|.
\end{aligned}$$

The formula (19) also gives us a simple method for computing the gradient $\nabla f(w)$. We first compute F^{-1} and G^{-1}. Then for each $k = 1, \ldots, m$, we compute $\partial f/\partial w_k$, using the last line of (19). For each k, this involves subtracting two columns of F^{-1}, and finding the norm squared of the difference, and the same for G^{-1}, which has a cost $O(n)$, so this step has a total cost $O(mn)$. Assuming no structure is exploited in forming the inverses, the total cost is $O(n^3 + mn)$, which is the same as $O(n^3)$, since $m \leqslant n(n-1)/2$. If W is sparse, we can compute F^{-1} and G^{-1} efficiently using the method described in Section 2.3 based on the formulas (16) and (17).

3.2. Hessian

From the gradient formula above, we can derive the Hessian of f as

$$\begin{aligned}
\frac{\partial^2 f}{\partial w_l \partial w_k} &= \frac{\partial}{\partial w_l}\left((1/2)\,\mathbf{Tr}\left(F^{-1}a_k a_k^T F^{-1}\right)\right.\\
&\quad \left. -(1/2)\,\mathbf{Tr}\left(G^{-1}a_k a_k^T G^{-1}\right)\right)\\
&= +(1/2)\,\mathbf{Tr}\left(\frac{\partial F^{-1}}{\partial w_l}a_k a_k^T F^{-1} + F^{-1}a_k a_k^T \frac{\partial F^{-1}}{\partial w_l}\right)\\
&\quad -(1/2)\,\mathbf{Tr}\left(\frac{\partial G^{-1}}{\partial w_l}a_k a_k^T G^{-1} + G^{-1}a_k a_k^T \frac{\partial G^{-1}}{\partial w_l}\right)\\
&= +(1/2)\,\mathbf{Tr}\left(F^{-1}a_l a_l^T F^{-1}a_k a_k^T F^{-1}\right.\\
&\quad \left. +F^{-1}a_k a_k^T F^{-1}a_l a_l^T F^{-1}\right)\\
&\quad +(1/2)\,\mathbf{Tr}\left(G^{-1}a_l a_l^T G^{-1}a_k a_k^T G^{-1}\right.\\
&\quad \left. +G^{-1}a_k a_k^T G^{-1}a_l a_l^T G^{-1}\right)\\
&= (a_k F^{-1}a_l)(a_k^T F^{-2}a_l) + (a_k G^{-1}a_l)\left(a_k^T G^{-2}a_l\right).
\end{aligned}$$

In the last line, we use the formula $\mathbf{Tr}\, Aab^T = b^T Aa$ for $a, b \in \mathbf{R}^n$ and $A = A^T \in \mathbf{R}^{n\times n}$. Suppose that weight w_l corresponds to edge $\{p, q\}$, i.e., $l \sim \{p, q\}$. Then we have

$$\begin{aligned}
\frac{\partial^2 f}{\partial w_l \partial w_k} &= \alpha\left((F^{-1})_{:,i} - (F^{-1})_{:,j}\right)^T\left(\left(F^{-1}\right)_{:,p} - \left(F^{-1}\right)_{:,q}\right)\\
&\quad +\beta\left(\left(G^{-1}\right)_{:,i} - \left(G^{-1}\right)_{:,j}\right)^T\\
&\quad \times\left(\left(G^{-1}\right)_{:,p} - \left(G^{-1}\right)_{:,q}\right),
\end{aligned}$$

where

$$\begin{aligned}
\alpha &= a_k F^{-1}a_l\\
&= \left(\left(F^{-1}\right)_{i,p} - \left(F^{-1}\right)_{i,q} - \left(F^{-1}\right)_{j,p} + \left(F^{-1}\right)_{j,q}\right),\\
\beta &= a_k G^{-1}a_l\\
&= \left(\left(G^{-1}\right)_{i,p} - \left(G^{-1}\right)_{i,q} - \left(G^{-1}\right)_{j,p} + \left(G^{-1}\right)_{j,q}\right).
\end{aligned}$$

Once the matrices F^{-1} and G^{-1} are formed, for each $k, l = 1, \ldots, m$, we can compute $\partial^2 f/\partial w_l \partial w_k$ using the last formula above, which has a cost $O(n)$. The total cost of forming the Hessian is $O(n^3 + m^2 n)$, which is the same as $O(m^2 n)$, irrespective of the sparsity of the graph (and hence W). The computational cost per Newton step is $O(m^2 n + m^3)$, which is the same as $O(m^3)$ (the Hessian is fully dense even when the graph is sparse).

4. LMSC with constant edge weight

In this section we consider a special case of the LMSC problem, where all edge weights are taken to be equal. This special case is interesting on its own, and in some cases, the optimal solution of the more general LMSC problem can be shown to occur when all edge weights are equal.

When the edge weights are equal we have $w_k = \alpha$, so

$$W = I - \alpha \sum_{k=1}^{m} a_k a_k^T = I - \alpha L,$$

where L is the *Laplacian* matrix of the graph, defined as

$$L_{ij} = \begin{cases} -1 & \{i, j\} \in \mathcal{E}, \\ d_i & i = j, \\ 0 & \text{otherwise}, \end{cases} \tag{20}$$

where d_i is the degree of node i. The Laplacian matrix is positive semidefinite, and since we assume the graph is connected, it has a single eigenvalue $\lambda_n(L) = 0$, with associated eigenvector $\mathbf{1}$. We have

$$\lambda_i(W) = 1 - \alpha \lambda_{n-i+1}(L), \quad i = 1, \ldots, n,$$

so the convergence condition $\|W - J\| < 1$ is equivalent to $0 < \alpha < 2/\lambda_1(L)$.

The steady-state mean-square deviation is

$$\begin{aligned} \delta_{\mathrm{ss}}(I - \alpha L) &= \sum_{i=1}^{n-1} \frac{1}{1 - (1 - \alpha \lambda_i(L))^2} \\ &= \sum_{i=1}^{n-1} \frac{1}{\lambda_i(L)\alpha} \frac{1}{2 - \lambda_i(L)\alpha}. \end{aligned}$$

The LMSC problem reduces to

$$\text{minimize} \quad \sum_{i=1}^{n-1} \frac{1}{\lambda_i(L)\alpha} \frac{1}{2 - \lambda_i(L)\alpha}, \tag{21}$$

with scalar variable α, and the implicit constraint $0 < \alpha < 2/\lambda_1(L)$. The optimality condition is simply $\partial \delta_{\mathrm{ss}}/\partial \alpha = 0$, which is equivalent to

$$\sum_{i=1}^{n-1} \frac{1}{\lambda_i(L)} \frac{1 - \lambda_i(L)\alpha}{(2 - \lambda_i(L)\alpha)^2} = 0. \tag{22}$$

The left-hand side is monotone decreasing in α, so a simple bisection can be used to find the optimal weight α. A Newton method can be used to obtain very fast final convergence.

From (22) we can conclude that the optimal edge weight $\alpha^\star$ satisfies $\alpha^\star \geqslant 1/\lambda_1(L)$. To see this, we note that the left-hand side of (22) is nonnegative when $\alpha = 1/\lambda_1(L)$ and is $-\infty$ when $\alpha = 2/\lambda_1(L)$. Thus we have

$$\frac{1}{\lambda_1(L)} \leqslant \alpha^\star < \frac{2}{\lambda_1(L)}. \tag{23}$$

So we can always estimate $\alpha^\star$ within a factor of two, e.g., with $\alpha = 1/\lambda_1(L)$.

4.1. LMSC problem on edge-transitive graphs

For graphs with large symmetry groups, we can exploit symmetry in the LMSC problem to develop far more efficient computational methods. In particular, we show that for *edge-transitive* graphs, it suffices to consider constant edge weight in the (general) LMSC problem.

An *automorphism* of a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is a permutation π of $\mathcal{N}$ such that $\{i, j\} \in \mathcal{E}$ if and only if $\{\pi(i), \pi(j)\} \in \mathcal{E}$. A graph is edge-transitive if given any pair of edges there is an automorphism which transforms one into the other. For example, rings and hypercubes are edge-transitive.

For edge-transitive graphs, we can assume without loss of generality that the optimal solution to the LMSC problem is a constant weight on all edges. To see this, let $w^\star$ be any optimal weight vector, not necessarily constant on all edges. Let $\pi(w^\star)$ denote the vector whose elements are rearranged by the permutation π. If π is an automorphism of the graph, then $\pi(w^\star)$ is also feasible. Let $\overline{w}$ denote the average of such vectors induced by all automorphisms of the graph. Then $\overline{w}$ is also feasible (because each $\pi(w)$ is feasible and the feasible set is convex), and moreover, using convexity of δ_{ss}, we have $\delta_{\mathrm{ss}}(\overline{w}) \leqslant \delta_{\mathrm{ss}}(w^\star)$. It follows that $\overline{w}$ is optimal. By construction, $\overline{w}$ is also invariant under the automorphisms. For edge-transitive graphs, this implies that $\overline{w}$ is a constant vector, i.e., its components are equal. (See [7, Exercise 4.4].) More discussion of exploiting symmetry in convex optimization problems can be found in [5,12,24].

4.2. Edge-transitive examples

In this section, we consider several examples of graphs that are edge-transitive. The optimal weights are therefore constant, with value α (say) on each edge.

4.2.1. Rings

For rings with n nodes, the Laplacian matrix is circulant, and has eigenvalues

$$2\left(1 - \cos\frac{2k\pi}{n}\right), \quad k = 0, \ldots, n-1.$$

Therefore we have

$$\delta_{\mathrm{ss}} = \sum_{k=1}^{n-1} \frac{1}{1 - \left(1 - 2\left(1 - \cos\frac{2k\Pi}{n}\right)\alpha\right)^2}.$$

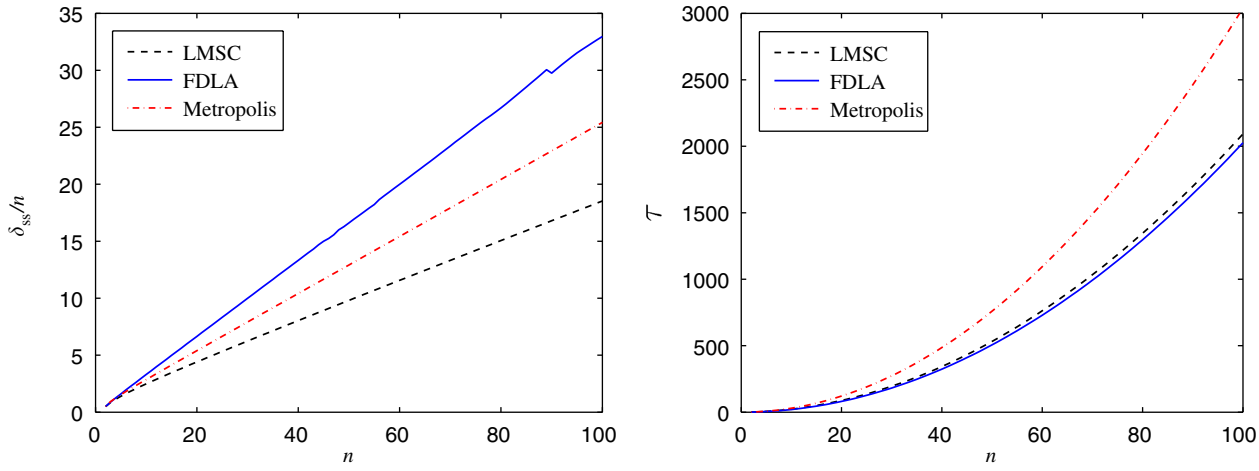

Fig. 1. Average mean-square deviation δ_{ss}/n (left) and convergence time τ (right) for paths with n nodes.

For n even, $\lambda_1(L) = 4$ so by (23) we have $1/4 \leqslant \alpha^\star < 1/2$. For n odd, $\lambda_1(L) = 2(1 + \cos(\pi/n))$, so we have

$$\frac{1}{2(1+\cos(\pi/n))} \leqslant \alpha^\star < \frac{1}{1+\cos(\pi/n)}.$$

4.2.2. Meshes

Consider a two-dimensional mesh, with n nodes in each direction, with wraparounds at the edges. This mesh graph is the Cartesian products of two n-node rings (see, e.g., [19]). The Laplacian is the Kroneker product of two circulant matrices, and has eigenvalues

$$4\left(1-\cos\frac{(k+j)\pi}{n}\cos\frac{(k-j)\pi}{n}\right), \quad k, j = 0, \ldots, n-1.$$

Therefore,

$$\delta_{ss} = -1 + \sum_{k,j=0}^{n-1} \frac{1}{1-\left(1-4\left(1-\cos\frac{(k+j)\pi}{n}\cos\frac{(k-j)\pi}{n}\right)\alpha\right)^2}.$$

Again we can bound the optimal solution $\alpha^\star$ by Eq. (23). For example, when n is even, we have $\lambda_1(L) = 8$, so $1/8 \leqslant \alpha^\star < 1/4$.

4.2.3. Stars

The star graph with n nodes consists of one center node and $n-1$ peripheral nodes connected to the center. The Laplacian matrix has three distinct eigenvalues: 0, n, and 1. The eigenvalue 1 has multiplicity $n-2$. We have

$$\delta_{ss} = \frac{1}{2n\alpha - n^2\alpha^2} + \frac{n-2}{2\alpha - \alpha^2}.$$

The optimality condition (22) boils down to

$$\frac{1-n\alpha^\star}{n(2-n\alpha^\star)^2} + (n-2)\frac{1-\alpha^\star}{(2-\alpha^\star)^2} = 0.$$

This leads to a cubic equation for $\alpha^\star$, which gives an analytical (but complicated) expression for $\alpha^\star$. In any case, the bounds (23) give $1/n \leqslant \alpha^\star < 2/n$.

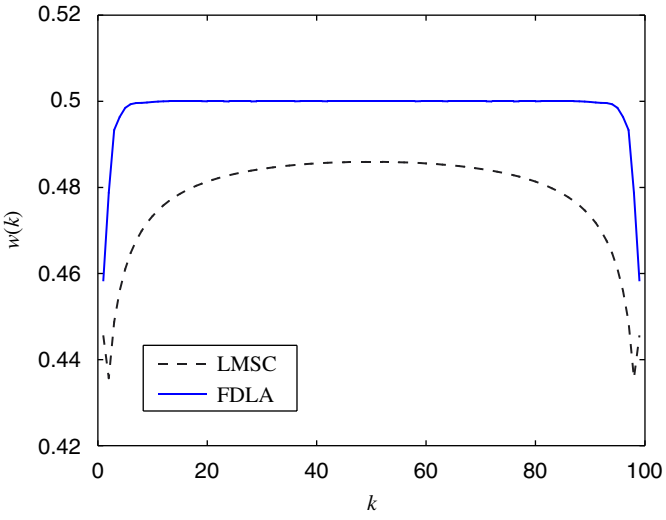

Fig. 2. LMSC and FDLA optimal edge weights on a path with $n = 100$ nodes.

4.2.4. Hypercubes

For a d-dimensional hypercube, there are 2^d vertices, each labeled with a binary word with length d. Two vertices are connected by an edge if their words differ in exactly one component. The Laplacian matrix has eigenvalues $2k$, $k = 0, 1, \ldots, d$, each with multiplicity $\binom{d}{k}$ (e.g., [19]). Substituting these eigenvalues into (21), we find that

$$\delta_{ss} = \sum_{k=1}^{d} \binom{d}{k} \frac{1}{4k\alpha - 4k^2\alpha^2},$$

with domain $0 < \alpha < 1/d$. The bounds (23) give $1/(2d) \leqslant \alpha^\star < 1/d$.

According to the numerical results in Section 5, we conjecture that the optimal solution is $\alpha^\star = 1/(d+1)$, but we have not been able to prove this yet. The value $\alpha = 1/(d+1)$ is also the solution for the FDLA problem studied in [33] (see also [9,19,24]).

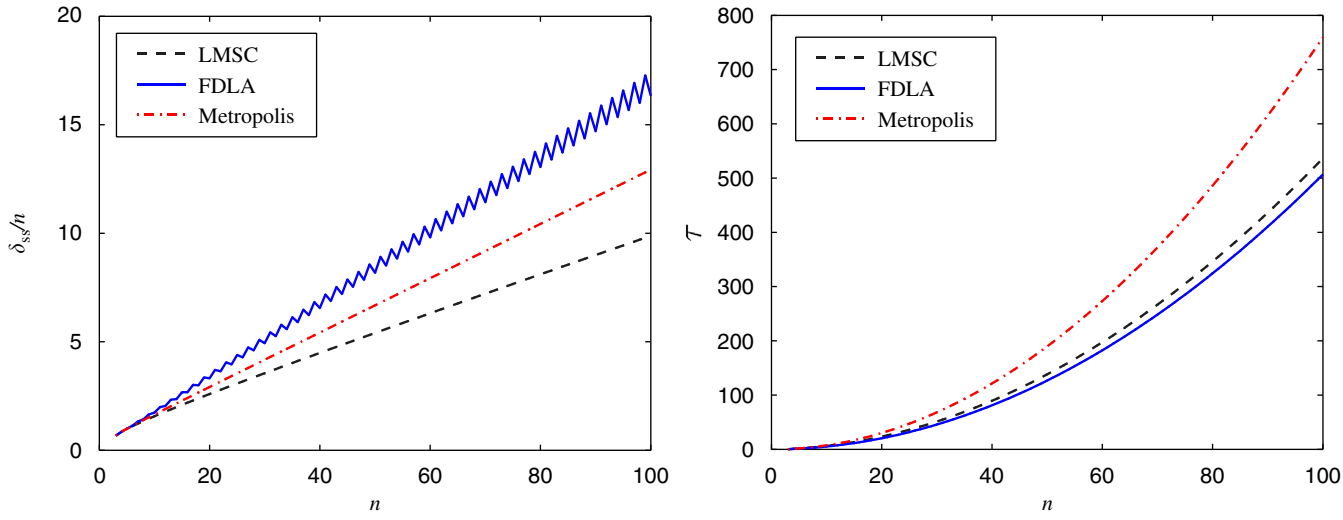

Fig. 3. Average mean-square deviation δ_{ss}/n (left) and convergence time τ (right) for rings with n nodes.

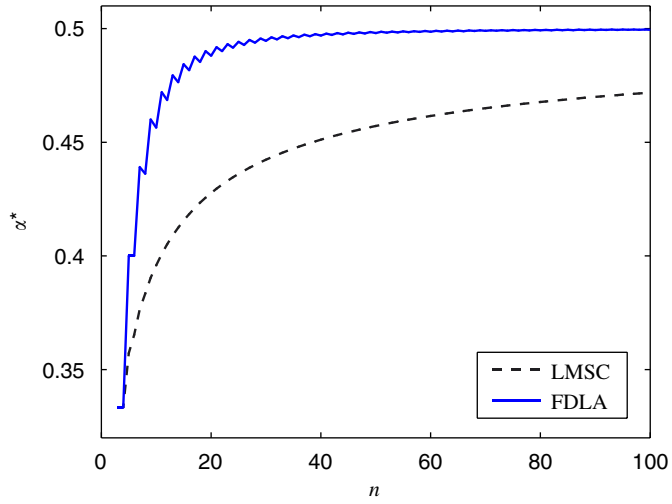

Fig. 4. LMSC and FDLA edge weight for rings with n varying from 3 to 100.

5. Examples

In this section, we give some numerical examples of the LMSC problem (6), and compare the solutions obtained with the Metropolis weights (4) and weights that yield fastest asymptotic convergence, i.e., a solution of

$$\begin{array}{ll}\text{minimize} & \|W - J\| \\ \text{subject to} & W \in \mathcal{S}, \quad W = W^T, \ W\mathbf{1} = \mathbf{1}.\end{array} \tag{24}$$

This FDLA problem need not have a unique solution, so we simply use *an* optimal solution. (See [33] for details of the FDLA problem.)

For each example, we consider a family of graphs that vary in the number of nodes or edges. For each graph instance, we report both the average mean-square deviation δ_{ss}/n, which gives the asymptotic mean-square deviation per node. We also report the asymptotic convergence time, defined as

$$\tau = \frac{1}{\log(1/\|W - J\|)}.$$

This gives the asymptotic number of steps for the error $\|x(t) - Jx(0)\|$ to decrease by a factor e, in the absence of noise. The FDLA weights minimize the convergence time τ.

5.1. Paths

Fig. 1 shows δ_{ss}/n and τ for paths with a number of nodes ranging from 2 to 100. We see that the LMSC weights achieve much smaller average mean-square deviation than the FDLA weights, and the Metropolis weights (in this case a constant weight $1/3$ on all edges) have a mean-square deviation in between. In terms of the convergence time, however, there is not much difference between the LMSC weights and FDLA weights, with the Metropolis weights giving much slower convergence. Fig. 2 shows the distribution of both the LMSC weights and FDLA weights on a path with 100 nodes (and 99 edges), where the edges are labeled $k = 1, \ldots, 99$ on the horizontal axis. The Metropolis weights, which are not shown, are $1/3$ on all edges.

5.2. Rings

Fig. 3 shows δ_{ss}/n and τ for rings with a number of nodes ranging from 3 to 100. The relative comparison of different weights are similar to those on paths, but the average mean-square deviation and convergence time are much smaller for the rings. Since rings are edge-transitive, the optimal weights for both the LMSC and FDLA problems are constant on all edges. Fig. 4 shows the optimal weights on rings with number of nodes from 3 to 100. In general, the LMSC weights are smaller than the FDLA weights, i.e., the LMSC weights have larger self weights at the nodes.

5.3. Grids and meshes

Fig. 5 shows δ_{ss}/n^2 and τ for $n \times n$ grids with n ranging from 2 to 10. Similar results for meshes (grids with wrap-arounds at two ends of both directions) are shown in Fig. 6. The $n \times n$ meshes are edge-transitive, so both the LMSC abd

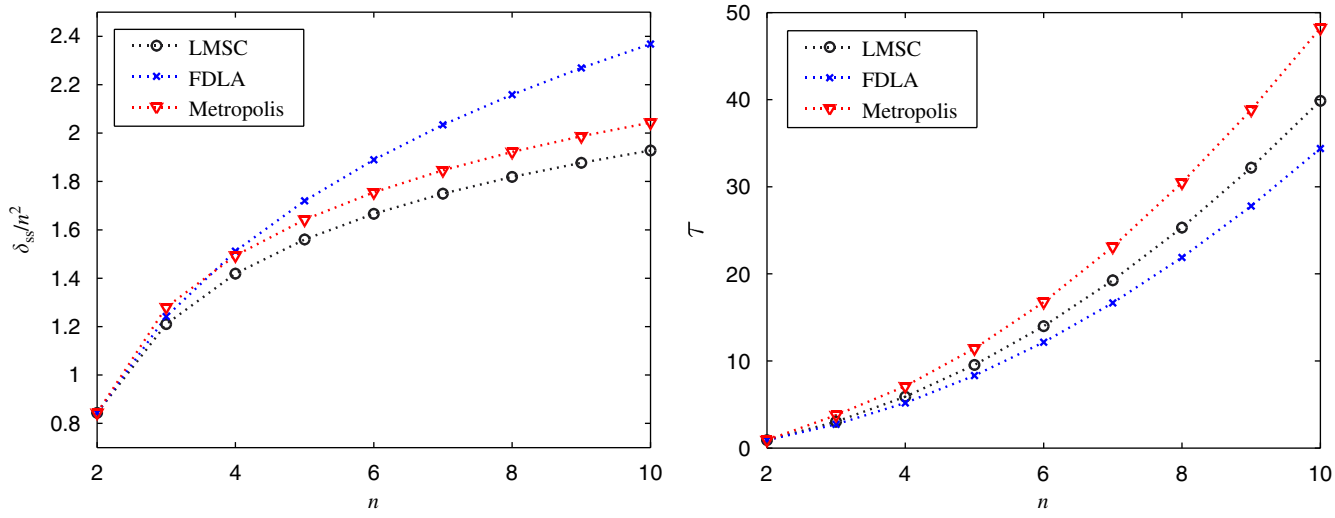

Fig. 5. Average mean-square deviation δ_{ss}/n^2 (left) and convergence time τ (right) of $n \times n$ grids.

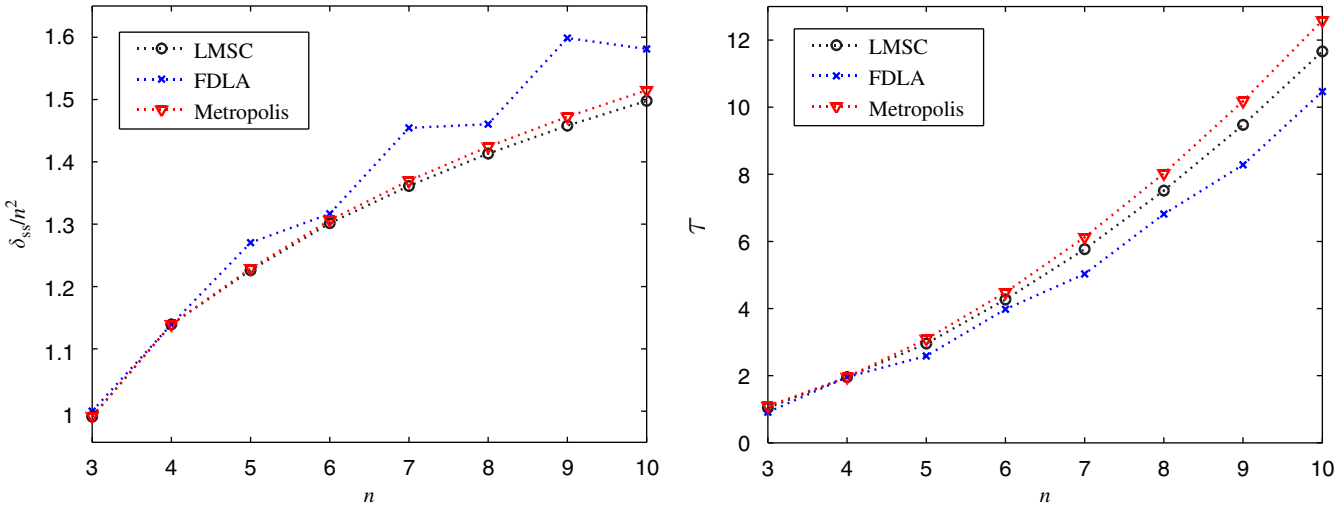

Fig. 6. Average mean-square deviation δ_{ss}/n^2 (left) and convergence time τ (right) of $n \times n$ meshes.

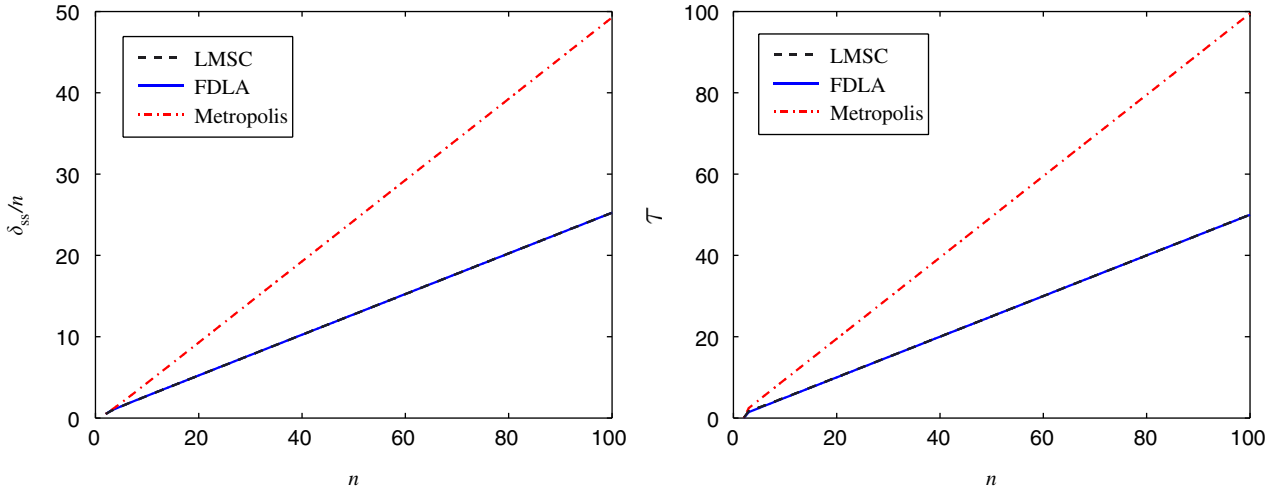

Fig. 7. Average mean-square deviation δ_{ss}/n (left) and convergence time τ (right) of stars with n nodes.

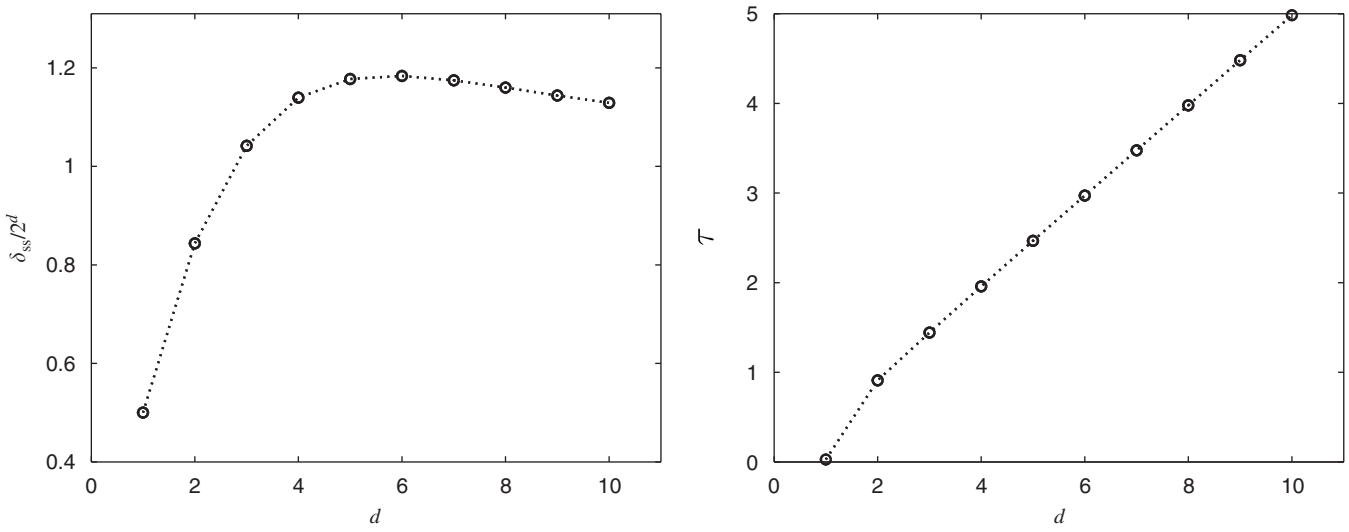

Fig. 8. Average mean-square deviation δ_{ss}/n (left) and convergence time τ (right) of d-dimensional hypercubes with constant edge weight $\alpha^\star = 1/(d+1)$, the solution for both LMSC and fastest convergence.

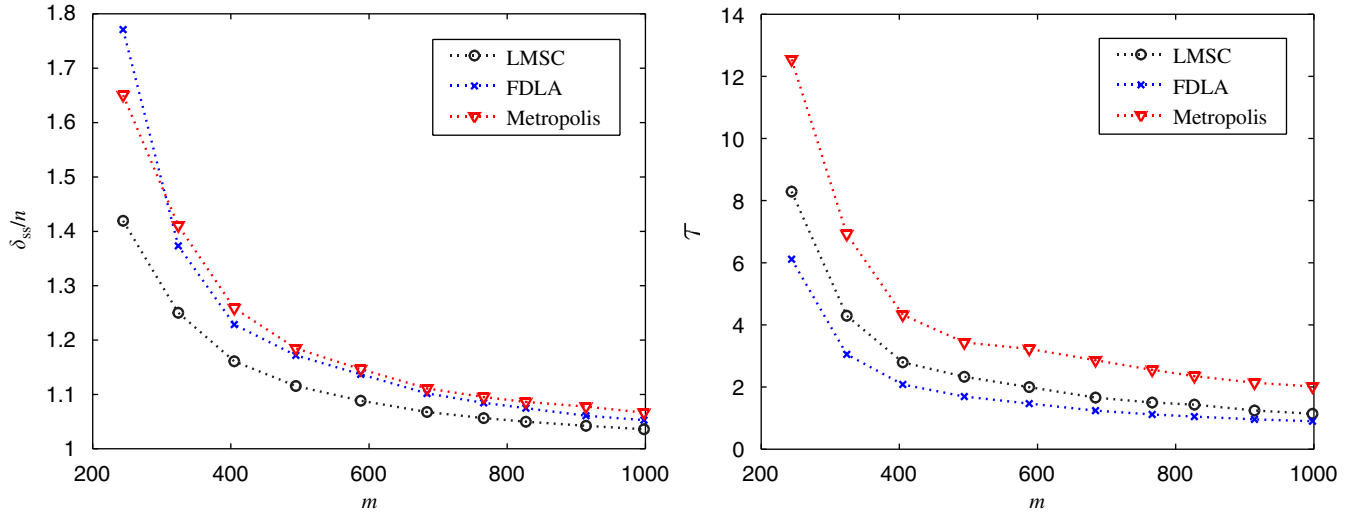

Fig. 9. Average mean-square deviation δ_{ss}/n (left) and convergence time τ (right) of a random family of graphs. The horizontal axis shows the number of edges (the number of nodes n is fixed).

FDLA optimization problems have only one variable, allowing very large scale problems to be solved using the formulation in Section 4.

5.4. Stars

Fig. 7 shows the average mean-square deviation and convergence time for stars with n nodes. The LMSC weight and FDLA weight give almost identical results for large n, which is much better than the Metropolis weight $\alpha = 1/n$. For $n \geqslant 3$, the optimal solution to the FDLA problem is $\alpha^\star_{\mathrm{FDLA}} = 2/(n+1)$, which is an excellent approximation for the LMSC weight. For stars with large n, the minimum mean-square deviation and fastest convergence can be achieved almost simultaneously.

5.5. Hypercubes

Fig. 8 shows the average mean-square deviation and convergence time for d-dimensional hypercubes. Again, these are edge-transitive graphs and there is only one variable in both the LMSC and FDLA problems. The numerical results show that the optimal constant weights for these two problems coincide, which is also obtained by the simple Metropolis methods. So the hypercubes are special graphs that the minimum mean-square deviation and fastest convergence can be achieved simultaneously.

5.6. A random family

We generate a family of graphs, all with 100 nodes, as follows. First we generate a symmetric matrix $R \in \mathbf{R}^{100\times 100}$, whose entries R_{ij}, for $i \leqslant j$, are independent and uniformly distributed on [0, 1]. For each threshold value $c \in [0, 1]$ we construct a graph by placing an edge between vertices i and j for $i \neq j$ if $R_{ij} \leqslant c$. By increasing c from 0 to 1, we obtain a family of graphs. This family is monotone: the graph associated with a larger value of c contains all the edges of the graph as-

sociated with a smaller value of c. We start with a large enough value of c that the resulting graph is connected.

Fig. 9 shows δ_{ss}/n and τ for the graphs obtained for 10 different values of c (in the range $[0.05, 0.2]$). Of course both the average mean-square deviation and convergence time decrease as the number of edges m increases. For this random family of graphs, the Metropolis weights often give smaller mean-square deviation than the FDLA weights.

Acknowledgments

We thank Devavrat Shah for discussions on the average consensus model with additive noises, and thank Anders Rantzer for discussions that helped identify an error in a previous draft.

References

[1] D.P. Bertsekas, Nonlinear of Programming, second ed., Athena Scientific, 1999.
[2] V.D. Blondel, J.M. Hendrickx, A. Olshevsky, J.N. Tsitsiklis, Convergence in multiagent coordination, consensus, and flocking, in: Proceedings of IEEE Conference on Decision and Control, Seville, Spain, 2005, pp. 2996–3000.
[3] J. Boillat, Load balancing and Poisson equation in a graph, Concurrency: Practice and Experience 2 (1990) 289–313.
[4] J.M. Borwein, A.S. Lewis, Convex analysis and nonlinear optimization, theory and examples, Canadian Mathematical Society Books in Mathematics, Springer, New York, 2000.
[5] S. Boyd, P. Diaconis, P.A. Parrilo, L. Xiao, Symmetry analysis of reversible Markov chains, Internet Mathematics 2 (1) (2005) 31–71.
[6] S. Boyd, P. Diaconis, L. Xiao, Fastest mixing Markov chain on a graph, SIAM Rev. problems and techniques section 46 (4) (2004) 667–689.
[7] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, Cambridge, MA, 2004, Available at: ⟨http://www.stanford.edu/~boyd/cvxbook.html⟩.
[8] P. Brémaud, Markov chains, Gibbs fields, Monte Carlo simulation and queues, Texts in Applied Mathematics, Springer, Berlin, Heidelberg, 1999.
[9] G. Cybenko, Load balancing for distributed memory multiprocessors, J. Parallel and Distributed Computing 7 (1989) 279–301.
[10] P. Diaconis, D. Stroock, Geometric bounds for eigenvalues of Markov chains, The Ann. Appl. Probab. 1 (1) (1991) 36–61.
[11] R. Elsässer, B. Monien, R. Preis, Diffusive load balancing schemes on heterogeneous networks, Theory Comput. Systems 35 (2002) 305–320.
[12] K. Gatermann, P.A. Parrilo, Symmetry groups, semidefinite programs, and sums of squares, J. Pure Appl. Algebra 192 (2004) 95–128.
[13] A. Jadbabaie, J. Lin, A.S. Morse, Coordination of groups of mobile autonomous agents using nearest neighbor rules, IEEE Trans. Automat. Control 48 (6) (2003) 988–1001.
[14] A.S. Lewis, Convex analysis on the Hemitian matrices, SIAM J. Optim. 6 (1996) 164–177.
[15] A.S. Lewis, H.S. Sendov, Twice differentiable spectral functions, SIAM J. Matrix Anal. Appl. 23 (2001) 368–386.
[16] D.G. Luenberger, Introduction to Linear and Nonlinear Programming, second ed., Addison-Wesley, Reading, MA, 1984.
[17] N.A. Lynch, Distributed Algorithms, Morgan Kaufmann Publishers, San Francisco, CA, 1996.
[18] C.C. Moallemi, B. Van Roy, Consensus propagation. Draft, 2005.
[19] B. Mohar, Some applications of Laplace eigenvalues of graphs, in: G. Hahn, G. Sabidussi (Eds.), Graph Symmetry: Algebraic Methods and Applications, NATO ASI Series C 497, Kluwer Academic Publishers, Dordrecht, MA, 1997, pp. 225–275.
[20] L. Moreau, Stability of multi-agent systems with time-dependent communication links, IEEE Trans. Automat. Control 50 (2) (2005) 169–182.
[21] J. Nocedal, S.J. Wright, Numerical Optimization, Springer Series in Operations Research, Springer, New York, 1999.
[22] R. Olfati-Saber, R.M. Murray, Consensus problems in networks of agents with switching topology and time-delays, IEEE Trans. Automat. Control 49 (9) (2004) 1520–1533.
[23] M.L. Overton, R.S. Womersley, Optimality conditions and duality theory for minimizing sums of the largest eigenvalues of symmetric matrices, Math. Programming 62 (1993) 321–357.
[24] P.A. Parrilo, L. Xiao, S. Boyd, P. Diaconis, Fastest mixing Markov chain on graphs with symmetries, Draft, 2006.
[25] B. Patt-Shamir, S. Rajsbaum, A theory of clock synchronization, in: Proceedings of the 26th Annual ACM Symposium on Theory of Computing, Montreal, Canada, 1994, pp. 810–819.
[26] W. Ren, R.W. Beard, Consensus seeking in multi-agent systems under dynamically changing interaction topologies, IEEE Trans. Automat. Control 50 (5) (2005) 655–661.
[27] T. Rotaru, H.-H. Nägeli, Dynamic load balancing by diffusion in heterogeneous systems, J. Parallel and Distributed Comput. 64 (4) (2004) 481–497.
[28] D.S. Scherber, H.C. Papadopoulos, Locally constructed algorithms for distributed computations in ad-hoc networks, in: Proceedings of the Third International Symposium on Information Processing in Sensor Networks, Berkeley, CA, April 2004. ACM Press, New York, pp. 11–19.
[29] D.P. Spanos, R. Olfati-Saber, R.M. Murray, Distributed sensor fusion using dynamic consensus, in: Proceedings of the 16th IFAC World Congress, Prague, Czech, 2005.
[30] R. Subramanian, I.D. Scherson, An analysis of diffusive load-balancing, in: Proceedings of the Sixth Annual ACM Symposium on Parallel Algorithms and Architectures, Cape May, NJ, USA, 1994, pp. 220–225.
[31] J.N. Tsitsiklis, Problems in decentralized decision making and computation, Ph.D. Thesis, Massachusetts Institute of Technology, 1984.
[32] J.N. Tsitsiklis, D.P. Bertsekas, M. Athans, Distributed asynchronous deterministic and stochastic gradient optimization algorithms, IEEE Trans. Automat. Control 31 (9) (1986) 803–812.
[33] L. Xiao, S. Boyd, Fast linear iterations for distributed averaging, Systems Control Lett. 53 (2004) 65–78.
[34] L. Xiao, S. Boyd, S. Lall, A scheme for robust distributed sensor fusion based on average consensus, in: Proceedings of the Fourth International Conference on Information Processing in Sensor Networks, Los Angeles, California, USA, 2005, pp. 63–70.
[35] C.-Z. Xu, F. Lau, Load Balancing in Parallel Computers: Theory and Practice, Kluwer Academic Publishers, Dordrecht, MA, 1997.

Lin Xiao is a Researcher at Microsoft Research, Redmond, Washington. He received the Bachelor and Master degrees from Beijing University of Aeronautics and Astronautics in 1994 and 1997, respectively, and the Ph.D. degree from Stanford University in 2004. After a two-year postdoctoral fellowship at California Institute of Technology, he joined Microsoft Research in 2006. His current research interests include convex optimization, distributed computing, signal processing, and machine learning.

Stephen Boyd is the Samsung Professor of Engineering, and Professor of Electrical Engineering in the Information Systems Laboratory at Stanford University. He received the A.B. degree in Mathematics from Harvard University in 1980, and the Ph.D. in Electrical Engineering and Computer Science from the University of California, Berkeley, in 1985, and then joined the faculty at Stanford. His current research focus is on convex optimization applications in control, signal processing, and circuit design.

Seung-Jean Kim received the Ph.D. degree in electrical engineering from Seoul National University, Seoul, Korea. Since 2002, he has been with the Information Systems Laboratory, Department of Electrical Engineering, Stanford University, where he is currently a consulting assistant professor. His current research interests include large-scale numerical and convex optimization with applications in computer-aided design of digital circuits, computational imaging, communications, signal processing, statistical learning, and systems and control theory.