Gossip Algorithms: Design, Analysis and Applications

Stephen Boyd Arpita Ghosh Balaji Prabhakar Devavrat Shah *
Information Systems Laboratory, Stanford University
Stanford, CA 94305-9510
{boyd, arpitag, balaji, devavrat}@stanford.edu

Abstract— **Motivated by applications to sensor, peer-to-peer and ad hoc networks, we study distributed asynchronous algorithms, also known as *gossip* algorithms, for computation and information exchange in an arbitrarily connected network of nodes. Nodes in such networks operate under limited computational, communication and energy resources. These constraints naturally give rise to "gossip" algorithms: schemes which distribute the computational burden and in which a node communicates with a randomly chosen neighbor.**

We analyze the averaging problem under the gossip constraint for arbitrary network, and find that the averaging time of a gossip algorithm depends on the second largest eigenvalue of a doubly stochastic matrix characterizing the algorithm. Using recent results of Boyd, Diaconis and Xiao (2003), we show that minimizing this quantity to design the fastest averaging algorithm on the network is a semi-definite program(SDP). In general, SDPs cannot be solved distributedly; however, exploiting problem structure, we propose a subgradient method that distributedly solves the optimization problem over the network.

The relation of averaging time to the second largest eigenvalue naturally relates it to the mixing time of a random walk with transition probabilities that are derived from the gossip algorithm. We use this connection to study the performance of gossip algorithm on two popular networks: Wireless Sensor Networks, which are modeled as Geometric Random Graphs, and the Internet graph under the so-called Preferential Connectivity Model.

I. Introduction

The advent of sensor, wireless ad hoc and peer-to-peer networks has necessitated the design of asynchronous, distributed and fault-tolerant computation and information exchange algorithms. This is mainly because such networks are constrained by the following operational characteristics: (i) they may not have a centralized entity for facilitating computation, communication and time-synchronization, (ii) the network topology may not be completely known to the nodes of the network, (iii) nodes may join or leave the network (even expire), so that the network topology itself may change, and (iv) in the case of sensor networks, the computational power and energy resources may be very limited. These constraints motivate the design of simple asynchronous decentralized algorithms for computation where each node exchanges information with only a few of its immediate neighbors in a time instance (or, a round). The goal in this setting is to design algorithms so that the desired computation and communication is done as quickly and efficiently as possible.

We study the problem of averaging as an instance of the distributed computation problem. A toy example to explain the motivation for the averaging problem is sensing temperature of some small region of space by a network of sensors. For example, in Figure 1, sensors are deployed to measure the temperature T of a source. Sensor i, $i = 1, \ldots, 4$ measures $T_i = T + \eta_i$, where the η_i are IID, zero mean Gaussian sensor noise variables. The unbiased, minimum mean squared error (MMSE) estimate is the average $\hat{T} = \frac{\sum_i T_i}{4}$. Thus, to combat

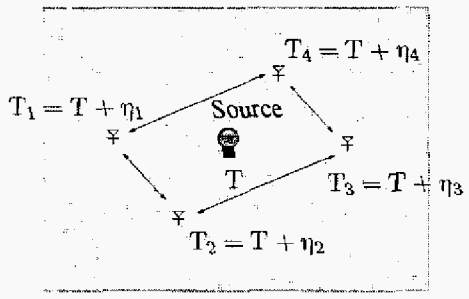

Fig. 1. Sensor nodes deployed to measure ambient temperature.

minor fluctuations in the ambient temperature and the noise in sensor readings, the nodes need to average their readings.

*Author names appear in alphabetical order.

This work is supported in part by a Stanford Graduate Fellowship, and by C2S2, the MARCO Focus Center for Circuit and System Solution, under MARCO contract 2003-CT-888.

0-7803-8968-9/05/$20.00 (C)2005 IEEE

Distributed averaging arises in many applications such as coordination of autonomous agents, estimation and distributed data fusion on ad-hoc networks, and decentralized optimization. [1] Fast distributed averaging algorithms are also important in other contexts; see Kempe et al [KDG03], for example. For an extensive body of related work, see [KK02],[KKD01], [HHL88], [GvRB01], [KEW02], [MFHH02], [vR00], [EGHK99], [IEGH02], [KSSV00a], [SMK+01], [RFH+01].

This paper undertakes an in-depth study of the design and analysis of gossip algorithms for averaging in an *arbitrarily connected* network of nodes. (By gossip algorithm, we mean specifically an algorithm in which each node communicates with no more than one neighbour in each time slot.) Thus, given a graph G, we determine the averaging time, T_{ave}, which is the time taken for the value at each node to be close to the average value (a more precise definition is given later). We find that the averaging time depends on the second largest eigenvalue of a doubly stochastic matrix characterizing the averaging algorithm: the smaller this eigenvalue, the faster the averaging algorithm. The *fastest* averaging algorithm is obtained by minimizing this eigenvalue over the set of allowed gossip algorithms on the graph. This minimization is shown to be a semi-definite program, which is a convex problem, and therefore can be solved efficiently to obtain the global optimum.

The averaging time, T_{ave}, is closely related to the mixing time, T_{mix}, of the random walk defined by the matrix that characterizes the algorithm. This means we can study also averaging algorithms by studying the mixing time of the corresponding random walk on the graph. The recent work of Boyd et al [BDX03] shows that the ratio of the mixing times of the natural random walk to the fastest-mixing random walk can grow without bound as the number of nodes increases; correspondingly, therefore, the optimal averaging algorithm can perform arbitrarily better than the one based on the natural random walk. Thus, computing the optimal averaging algorithm is important: however, this involves solving a semi-definite program, which requires a knowledge of the complete topology. Surprisingly, we find that we can exploit the problem structure to devise a distributed subgradient method to solve the semidefinite program and obtain a near-optimal averaging algorithm.

Finally, we study the performance of gossip algorithms on two network graphs which are very important in practice: Geometric Random Graphs which are used to model wireless sensor networks, and the Internet graph under the preferential connectivity model. We find that for geometric random graphs, the averaging time of the natural is the same order as the optimal averaging algorithm, which, as remarked earlier, need not be the case in a general graph.

We shall state our main results after setting out some notation and definitions in the next section.

A. Problem Formulation and Definitions

Consider a connected graph $G = (V, E)$, where the vertex set V contains n nodes and E is the edge set. The i^{th} component of the vector $x(0) = [x_1(0), ..., x_n(0)]^T$ represents the initial value at node i. Let $x_{\text{ave}} = \frac{\sum_i x_i(0)}{n}$ be the average of the entries of $x(0)$ and the goal is to compute x_{ave} in a distributed and asynchronous manner.

- **Asynchronous time model:** Each node has a clock which ticks at the times of a rate 1 Poisson process. Thus, the inter-tick times at each node are rate 1 exponentials, independent across nodes and over time. Equivalently, this corresponds to a single clock ticking according to a rate n Poisson process at times $Z_k, k \geq 1$, where $\{Z_{k+1} - Z_k\}$ are IID exponentials of rate n. Let $I_k \in \{1, ..., n\}$ denote the node whose clock ticked at time Z_k. Clearly, the I_k are IID variables distributed uniformly over $\{1, \ldots, n\}$. We discretize time according to clock ticks since these are the only times at which the value of $x(\cdot)$ changes. Therefore, the interval $[Z_k, Z_{k+1})$ denotes the k^{th} time-slot and, on average, there are n clock ticks per unit of absolute time. Lemma 1 states a precise translation of clock ticks into absolute time.

- **Synchronous time model:** In the synchronous time model, time is assumed to be slotted commonly across nodes. In each time slot, each node contacts one of its neighbors independently and (not necessarily uniformly) at random. Note that in this model all nodes communicate simultaneously, in contrast to the asynchronous model where only one node communicates at a given time. On the other hand, in both models each node contacts only one other node at a time.

  This paper uses the asynchronous time model whereas previous work, notably that of [KSSV00b],

[1] The theoretical framework developed in this paper is not merely restricted to averaging algorithms. It easily extends to the computation of other functions which can be computed via pair-wise operations: e.g., the maximum, minimum or product functions. It can also be extended for analyzing information exchange algorithms, although this extension is not as direct. For concreteness and for stating our results as precisely as possible, we shall consider averaging algorithms in the rest of the paper.

[KDG03], considers the synchronous time model. The qualitative and quantitative conclusions are unaffected by the type of model; we choose the asynchronous time model for convenience.

- **Algorithm** $\mathcal{A}(P)$: We consider a class of algorithms, denoted by $\mathcal{A}$. An algorithm in this class is characterized by an $n \times n$ matrix $P = [P_{ij}]$ of non-negative entries with the condition that $P_{ij} > 0$ only if $(i,j) \in E$. For technical reasons, we assume that P is a stochastic matrix with its largest eigenvalue equal to 1 and all the remaining $n-1$ eigenvalues are strictly less than 1 in magnitude. (Such a matrix can always be found if the underlying graph G is connected and non-bipartite. We will assume that the network graph G satisfies these conditions for the remainder of the paper.) The algorithm associated with P, denoted by $\mathcal{A}(P)$, is described as follows:
  In the k^{th} time-slot, let node i's clock tick and let it contact some neighboring node j with probability P_{ij}. At this time both nodes set their values equal to the average of their current values. Formally, let $x(k)$ denote the vector of values at the end of the time-slot k. Then,

$$x(k) \quad = \quad W(k)x(k-1), \tag{1}$$

  where with probability $\frac{1}{n}P_{ij}$ ($\frac{1}{n}$ is the probability that the i^{th} node's clock ticked and P_{ij} is the chance that it contacted node j) the random matrix $W(k)$ is

$$W_{ij} \quad = \quad I - \frac{(e_i - e_j)(e_i - e_j)^T}{2}, \tag{2}$$

  where $e_i = [0 \cdots \ 0 \ 1 \ 0 \cdots 0]^T$ is an $n \times 1$ unit vector with the i^{th} component equal to 1.

- **Quantity of Interest:** Our interest is in determining the time (number of clock ticks) it takes for $x(k)$ to converge to $x_{\text{ave}}\mathbf{1}$, where $\mathbf{1}$ is the vector of all ones.
  Definition 1: For any $0 < \epsilon < 1$, the $\epsilon-$averaging time of an algorithm $\mathcal{A}(P)$ is denoted by $T_{\text{ave}}(\epsilon, P)$ and equals

$$\sup_{x(0)} \inf \left\{ k : \Pr\left(\frac{\|x(k) - x_{\text{ave}}\mathbf{1}\|}{\|x(0)\|} \geq \epsilon \right) \leq \epsilon \right\}, \tag{3}$$

  where $\|v\|$ denotes the l_2 norm of the vector v.

Thus the ϵ-averaging time is the smallest number of clock ticks it takes for $x(\cdot)$ to get within ϵ of $x_{\text{ave}}\mathbf{1}$ with high probability, regardless of the initial value $x(0)$.

The following lemma relates the number of clock ticks to absolute time.

Lemma 1: For any $k \geq 1$, $E[Z_k] = k/n$. Further, for any $\delta > 0$,

$$\Pr\left(\left| Z_k - \frac{k}{n} \right| \geq \frac{\delta k}{n} \right) \quad \leq \quad 2\exp\left(-\frac{\delta^2 k}{2} \right). \tag{4}$$

Proof: By definition, $E[Z_k] = \sum_{j=1}^{k} E[Z_j - Z_{j-1}] = \sum_{j=1}^{k} 1/n = k/n$. Equation (4) follows directly from Cramer's Theorem (see [DZ99], pp. 30 & 35). ■

As a consequence of the Lemma 1, for $k \geq n$,

$$Z_k = \frac{k}{n}\left(1 + \sqrt{\frac{2\log n}{n}} \right)$$

with high probability (*i.e.*probability at least $1 - 1/n^2$). In this paper, all the results about ϵ-averaging times are at least n. Hence, dividing the quantities measured in terms of the number of clock ticks by n gives the corresponding quantities when measured in absolute time (for an example, see Corollary 2).

B. Previous Results

A general lower bound for any graph G and any averaging algorithm was obtained in [KSSV00a] in the synchronous setting. Their result is:

Theorem 1: For any gossip algorithm on any graph G and for $0 < \epsilon < 0.5$, the ϵ-averaging time (in synchronous steps) is lower bounded by $\Omega(\log n)$.

For a complete graph and a synchronous averaging algorithm, [KDG03] obtain the following result.

Theorem 2: For a complete graph, there exists a gossip algorithm such that the $1/n$-averaging time of the algorithm is $O(\log n)$.

The problem of (synchronous) fast distributed averaging on an arbitrary graph without the gossip constraint is studied in [XB03]; here, $W(t) = W$ for all t; *i.e.*, the system is completely deterministic. Distributed averaging has also been studied in the context of distributed load balancing ([RSW98]), where an analysis based on Markov chains is used to obtain bounds on the time required to achieve averaging (upto the integer constraint) upto a certain accuracy. However, each iteration is governed either by a constant stochastic matrix, or a fixed sequence of matchings is considered. Some other results on distributed averaging can be found in [BS03], [Mur03], [LBF04], [OSM04], [JLS03].

Not much is known about good randomized gossip algorithms for averaging on arbitrary graphs. The algorithm of [KDG03] is quite dependent on the fact that the underlying graph is a complete graph, and the general result of [KSSV00a] is a non-constructive lower bound.

C. Our Results

In this paper, we design and characterize the performance of averaging algorithms for arbitrary graphs. Our main result is the following theorem, which we shall later (in Section IV) apply to specific types of graphs that are of interest in applications.

Theorem 3: The averaging time, $T_{\text{ave}}(\epsilon, P)$, of the algorithm $\mathcal{A}(P)$ is bounded as follows:

$$T_{\text{ave}}(\epsilon, P) \leq \frac{3\log\epsilon^{-1}}{\log\lambda_2(W)^{-1}}, \text{ and} \quad (5)$$

$$T_{\text{ave}}(\epsilon, P) \geq \frac{0.5\log\epsilon^{-1}}{\log\lambda_2(W)^{-1}}, \quad (6)$$

where

$$W \triangleq I - \frac{1}{2n}D + \frac{P+P^T}{2n}, \quad (7)$$

and D is the diagonal matrix with entries

$$D_i = \sum_{j=1}^{n}[P_{ij} + P_{ji}].$$

Theorem 3 is proved in Section II.

In Section III we show that the problem of finding the fastest averaging algorithm can be formulated as a semidefinite program (SDP). In general, it is not possible to solve a semidefinite program in a distributed fashion. However, we exploit the structure of the problem to propose a completely distributed algorithm that solves the optimization problem on the network, based on a subgradient method. The description of the algorithm and proof of convergence are found in Section III-A.

Section IV relates averaging time of an algorithm on a graph G with the mixing time of an associated random walk on G, and uses this result to study applications of our results in the context of two networks of practical interest: wireless networks, and the Internet.

II. Proof of Theorem 3

We prove bounds (5) and (6) in Lemmas 2 and 3 on the number of discrete times (or equivalently clock ticks) required to get within ϵ of $x_{\text{ave}}\mathbf{1}$ (analogous to (5) and (6)).

A. Upper Bound

Lemma 2: For algorithm $\mathcal{A}(P)$, for any initial vector $x(0)$, for $k \geq K^*(\epsilon)$,

$$\Pr\left(\frac{\|x(k) - x_{\text{ave}}\mathbf{1}\|}{\|x(0)\|} \geq \epsilon\right) \leq \epsilon,$$

where

$$K^*(\epsilon) \triangleq \frac{3\log\epsilon^{-1}}{\log\lambda_2(W)^{-1}}, \text{ and} \quad (8)$$

Proof: Recall that under algorithm $\mathcal{A}(P)$, from (1) and (2),

$$x(k+1) = W(k+1)x(k), \quad (9)$$

where with probability $\frac{1}{n}P_{ij}$ the random matrix $W(k)$ is

$$W_{ij} = I - \frac{(e_i - e_j)(e_i - e_j)^T}{2}. \quad (10)$$

First note that $W(k)$ are doubly stochastic matrices for all (i,j). For doubly stochastic matrices, the vector $\frac{1}{n}\mathbf{1}$ is the eigenvector corresponding to the largest eigenvalue 1. With this observation, and with our assumptions on P, it can be shown that $x(k) \to x_{\text{ave}}\mathbf{1}$. Our interest is in finding out how fast it converges. In particular, we would like to obtain bounds on the error random vector $y(k)$,

$$y(k) = x(k) - x_{\text{ave}}\mathbf{1}. \quad (11)$$

Note that, $y(k) \perp \mathbf{1}$ since $y(k)^T\mathbf{1} = 0$.

Consider the evolution of $y(\cdot)$:

$$\begin{aligned} y(k+1) &= x(k+1) - x_{\text{ave}}\mathbf{1} \\ &\overset{(a)}{=} W(k+1)x(k) - x_{\text{ave}}W(k)\mathbf{1} \\ &= W(k+1)\left(x(k) - x_{\text{ave}}\mathbf{1}\right) \\ &= W(k+1)y(k). \end{aligned} \quad (12)$$

Here (a) follows from the fact that $\mathbf{1}$ is an eigenvector for all $W(k+1)$. Thus $y(\cdot)$ evolves according to the same linear system as $x(\cdot)$.

To obtain probabilistic bounds on $y(k)$, we will first compute the second moment of $y(k)$ and then apply Markov's inequality as below.

Computing W:

Let,

$$\begin{aligned} W &\triangleq E[W(0)] = E[W(k)] \\ &= \frac{1}{n}\sum_{i,j} P_{ij}W_{ij} \end{aligned} \quad (13)$$

Then, the entries of W are as follows:

1) for $i \neq j$, $W_{ij} = \frac{P_{ij}+P_{ji}}{2n}$, and
2) $W_{ii} = 1 - \frac{[\sum_{j=1}^{n}(P_{ij}+P_{ji})]-2P_{ii}}{2n}$.

This yields the W in (7), that is

$$W = I - \frac{1}{2n}D + \frac{P+P^T}{2n}, \quad (14)$$

where D is the diagonal matrix with entries

$$D_i = \left(\sum_{j=1}^{n}[P_{ij}+P_{ji}]\right).$$

Note that, if $P = P^T$, then P is doubly stochastic. This implies that $D_i = 2$, which in turn implies that $W = I(1-1/n) + P/n$.

Computing Second Moment $E[y(k)^T y(k)]$:

For each k, $W(k) = W_{ij}$ with probability $\frac{P_{ij}}{n}$, so that

$$W(k)^T W(k) = \left(I - \frac{(e_i-e_j)(e_i-e_j)^T}{2}\right)^2 \quad (15)$$
$$= \left(I - \frac{(e_i-e_j)(e_i-e_j)^T}{2}\right) \quad (16)$$
$$= W(k). \quad (17)$$

Since this is true for each instance of the random matrix W,

$$E[W(0)^T W(0)] = E[W(0)] = W. \quad (18)$$

Now, from (12),

$$\begin{aligned} &E[y(k+1)^T y(k+1)] \\ &= E[y(k)^T W(k+1)^T W(k+1) y(k)] \\ &= E[y(k)^T E[W(k+1)^T W(k+1)|y(k)] y(k)] \\ &= E[y(k)^T W y(k)], \end{aligned} \quad (19)$$

using (18), and the fact that the $W(k+1)$ are IID (independent of $y(k)$).

The matrix W is symmetric[2] positive-semidefinite (since $W = W^T W$) and hence it has non-negative real eigenvalues.

As stated earlier, $y(k) \perp \mathbf{1}$, which is the eigenvector corresponding to the largest eigenvalue $\lambda_1 = 1$ of W. So, from the variational characterization of the second eigenvalue, we have

$$y(k)^T W y(k) \leq \lambda_2(W) y(k)^T y(k). \quad (20)$$

From (18) and (20),

$$E[y(k+1)^T y(k+1)] \leq \lambda_2(W) E[y(k)^T y(k)]. \quad (21)$$

[2]The symmetry of W does not depend on P being symmetric.

Recursive application of (21) yields

$$E[y(k)^T y(k)] \leq \lambda_2(W)^k y(0)^T y(0). \quad (22)$$

Now,

$$\begin{aligned} y(0)^T y(0) &= x(0)^T x(0) - n x_{\text{ave}}^2 \\ &\leq x(0)^T x(0). \end{aligned} \quad (23)$$

Application of Markov's Inequality:

From (22), (23) and an application of Markov's inequality, we have

$$\begin{aligned} \Pr\left(\frac{\|x(k) - x_{\text{ave}}\mathbf{1}\|}{\|x(0)\|} \geq \epsilon\right) &= \Pr\left(\frac{y(k)^T y(k)}{x(0)^T x(0)} \geq \epsilon^2\right) \\ &\leq \epsilon^{-2}\frac{E[y(k)^T y(k)]}{x(0)^T x(0)} \\ &= \epsilon^{-2}\lambda_2(W)^k. \end{aligned} \quad (24)$$

From (24), it follows that for $k \geq K(\epsilon) \triangleq \frac{3\log\epsilon^{-1}}{\log\lambda_2(W)^{-1}}$,

$$\Pr\left(\frac{\|x(k) - x_{\text{ave}}\mathbf{1}\|}{\|x(0)\|} \geq \epsilon\right) \leq \epsilon. \quad (25)$$

This proves the Lemma, and gives us an upper bound on the ϵ-averaging time. ■

B. Lower Bound

Lemma 3: For algorithm $\mathcal{A}(P)$, there exists an initial vector $x(0)$, such that for $k < K_*(\epsilon)$,

$$\Pr\left(\frac{\|x(k) - x_{\text{ave}}\mathbf{1}\|}{\|x(0)\|} \geq \epsilon\right) > \epsilon,$$

where

$$K_*(\epsilon) \triangleq \frac{0.5\log\epsilon^{-1}}{\log\lambda_2(W)^{-1}}. \quad (26)$$

Proof:

From (12) and (18), we obtain

$$E[y(k)] = W^k y(0). \quad (27)$$

We have shown that W is a symmetric positive-semidefinite doubly stochastic matrix. W has (non-negative real) eigenvalues

$$1 = \lambda_1(W) \geq \lambda_2(W) \geq \ldots \geq \lambda_n(W) \geq 0,$$

with corresponding orthonormal eigenvectors $\frac{1}{\sqrt{n}}\mathbf{1}$, v_2, $v_3, \ldots, v_n$. Choose

$$x(0) = \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{n}}\mathbf{1} + v_2\right) \Rightarrow y(0) = \frac{1}{\sqrt{2}}v_2.$$

For this choice of $x(0)$, $\|x(0)\| = 1$. Now from (27),

$$E[y(k)] = \frac{1}{\sqrt{2}}\lambda_2^k(W)v_2. \quad (28)$$

For this particular choice of $x(0)$, we will lower bound the $\epsilon-$averaging time by lower bounding $E[\|y(k)\|^2]$ and using Lemma 4 as stated below.

By Jensen's inequality and (28),

$$\begin{aligned} E[\sum_{i=1}^{n} y_i(k)^2] &\geq \sum_{i=1}^{n} E^2[y_i(k)] \\ &= E[y(k)]^T E[y(k)] \\ &= \frac{1}{2}\lambda_2^{2k}(W)v_2^T v_2 \\ &= \frac{1}{2}\lambda_2^{2k}(W). \end{aligned} \quad (29)$$

Lemma 4: Let X be a random variable such that $0 \leq X \leq B$. Then, for any $0 < \epsilon < B$,

$$\Pr(X \geq \epsilon) \geq \frac{E[X] - \epsilon}{B - \epsilon}.$$

Proof:

$$\begin{aligned} E[X] &\leq \epsilon \Pr(X < \epsilon) + B \Pr(X \geq \epsilon) \\ &= \Pr(X \geq \epsilon)(B - \epsilon) + \epsilon. \end{aligned}$$

Rearranging terms gives us the lemma. ■

From (28), $\|y(k)\|^2 \leq \|y(0)\|^2 \leq 1/2$. Hence Lemma (4) and (29) imply that for $k < K_*(\epsilon)$

$$\Pr\left(\|y(k)\| \geq \epsilon\right) > \epsilon. \quad (30)$$

This completes the proof of Lemma 3.

■

The following corollaries are immediate.

Corollary 1: For large n and symmetric P, $T_{\text{ave}}(\epsilon, P)$ is bounded as follows:

$$T_{\text{ave}}(\epsilon, P) \leq \frac{3n \log \epsilon^{-1}}{(1 - \lambda_2(P))} \triangleq T^*(\epsilon, P) \quad (31)$$

$$T_{\text{ave}}(\epsilon, P) \geq \frac{0.5n \log \epsilon^{-1}}{(1 - \lambda_2(P))} \triangleq T_*(\epsilon, P). \quad (32)$$

Proof: By definition, $\lambda_2(W) = \left(1 - \frac{1}{n}(1 - \lambda_2(P))\right)$. For large n, $\frac{1}{n}(1 - \lambda_2(P))$ is very small, and hence

$$\log\left(1 - \frac{1}{n}(1 - \lambda_2(P))\right) \approx -\frac{1}{n}(1 - \lambda_2(P)).$$

This along with Theorem 3 completes the proof. ■

Corollary 2: For a symmetric P, the absolute time, $Z_{T^*(\epsilon,P)}$, it takes for $T^*(\epsilon, P)$ clock ticks to happen is given by

$$Z_{T^*(\epsilon,P)} = \frac{T^*(\epsilon, P)}{n}\left(1 \pm \frac{2}{\sqrt{n}}\right), \quad (33)$$

with probability at least $1 - 2\epsilon$.

Proof: For $\delta = \frac{\sqrt{2(1-\lambda_2(P))}}{\sqrt{3n}}$ and $k = T^*(\epsilon, P)$ and using (31), the right hand side of (4) evaluates to

$$2\exp\left(-\frac{2(1 - \lambda_2(P))}{3n} * \frac{3n \log \epsilon^{-1}}{2(1 - \lambda_2(P))}\right) = 2\epsilon.$$

Since $-1 \leq \lambda_2(P) \leq 1$ for a non-negative doubly stochastic symmetric matrix P, $\delta = \frac{2}{\sqrt{n}}$ is larger than the above choice of δ. This completes the proof. ■

III. Optimal Averaging Algorithm

From Theorem 5, we see that the averaging time is a monotonically increasing function of the second largest eigenvalue of $W = \sum_{i,j=1}^{n} \frac{1}{n} P_{ij} W_{ij}$. Thus, finding the fastest averaging algorithm corresponds to finding P such that $\lambda_2(W)$ is the smallest, while satisfying constraints on P. Thus, we have the optimization problem

$$\begin{array}{ll} \text{minimize} & \lambda_2(W) \\ \text{subject to} & W = \sum_{i,j=1}^{n} \frac{1}{n} P_{ij} W_{ij} \\ & P \geq [0], \ P_{ij} = 0 \text{ if } \{i,j\} \notin E, \\ & \sum_j P_{ij} = 1, \ \forall i. \end{array} \quad (34)$$

The objective function, which is the second largest eigenvalue of a doubly stochastic matrix, is a convex function on the set of symmetric matrices, and therefore we have a convex optimization problem. This problem can be reformulated as the following semidefinite program (SDP):

$$\begin{array}{ll} \text{minimize} & s \\ \text{subject to} & W - \mathbf{1}\mathbf{1}^T/n \preceq sI, \\ & W = \sum_{i,j=1}^{n} \frac{1}{n} P_{ij} W_{ij} \\ & P \geq [0], \ P_{ij} = 0 \text{ if } \{i,j\} \notin E, \\ & \sum_j P_{ij} = 1, \ \forall i. \end{array} \quad (35)$$

For general background on SDPs, eigenvalue optimization, and associated interior-point methods for solving these problems, see, for example, [BV03], [WSV00], [LO96], [Ove92], and references therein. Interior point methods can be used to solve problems with a thousand edges or so; subgradient methods can be used to solve the problem for larger graphs with upto a hundred thousand edges. The disadvantage of a subgradient method compared to a primal-dual interior point method is that the algorithm is relatively slow (in terms of number of iterations), and has no simple stopping criterion that can guarantee a certain level of suboptimality.

Thus, given a graph topology, we can solve the semidefinite program (35) to find the P^* for the fastest averaging algorithm.

A. Distributed Optimization

Finding the fastest averaging algorithm is a convex optimization problem, and can therefore be solved efficiently to obtain the optimal distribution P^*. Unfortunately, a P^* computed via a centralized computation is not very useful in our setting. It is natural to ask if in this setting, the optimization (like the averaging itself), can also be performed distributedly; *i.e.*, is it possible for the nodes on the graph, possessing only local information, and with only local communication, to compute the probabilities P_{ij} that lead to the fastest averaging algorithm?

In this section, we outline a completely distributed algorithm based on an *approximate subgradient method* which converges to a neighborhood of the optimal P^*. The algorithm uses distributed averaging to compute a subgradient; the accuracy to which the averaging is performed determines the size of the neighbourhood. The greater the accuracy, the smaller the neighbourhood, *i.e.*, the better the approximation to the optimal P^*. The exact relation between the accuracy of the distributed averaging and the size of the neighbourhood is stated in Theorem 4 at the end of this section. First we start with some notation.

Notation: It will be easier to analyze the subgradient method if we collect the entries of the matrix P_{ij} into a vector, which we will call p. Since there is no symmetry requirement on the matrix P, the vector p will need to have entries corresponding to P_{ij} as well as P_{ji} (this corresponds to replacing each edge in the *undirected* graph G by two directed edges, one in each direction).

The vector p corresponds to the matrix P as follows. Let the total number of (non self-loop) edges in G be m. Assign numbers to the edges (i,j) from 1 through m. If $i<j$ then $p_l = P_{ij}$, where l is the number assigned to (the undirected) edge (i,j) (which we will denote by $l \sim (i,j)$); if $i>j$ then $p_{-l} = P_{ij}$. (Recall that we are not considering self-loop edges.)

We will also introduce the notation $\mathbf{p_i}$ corresponding to the non-zero entries in the ith row of P (we do this to make concise the constraint that the sum of elements in each row should be 1). That is, we define for $1 \le i \le n$,

$$\mathbf{p_i} = [P_{ij}; (i,j) \in \mathcal{E}]. \tag{36}$$

Define $n \times n$ matrices E_l, $l \sim (i,j)$ as follows: $E_{l_{ij}} = E_{l_{ji}} = +1$, $E_{l_{ii}} = E_{l_{jj}} = -1$, and all other entries of E_l are zero. Then, we have that

$$E_l = 2(W_{ij} - I).$$

Finally, denote the degree of node i by m_i.

1) Subgradient method: We will describe the subgradient method for the optimization problem restated in terms of the variable p. We can state (35) in terms of the variables $p = [p_{-m}, \ldots p_{-1}, p_1, \ldots p_m]$ as follows:

$$\begin{array}{ll} \text{minimize} & \lambda_2(I + \frac{1}{2n}\sum_{l=1}^{m} p_l E_l + p_{-l}E_{-l}) \\ \text{subject to} & 1^T \mathbf{p_i} \le 1, \quad \forall i \\ & p_l \ge 0, \quad 1 \le |l| \le m, \end{array} \tag{37}$$

where $\mathbf{p_i}$ is as defined in (36).

We will use the subgradient method to solve this problem distributedly. The use of the subgradient method to solve eigenvalue problems is well-known; see for example [BDX03], [OW93], [Lew96], [Lew99] for material on non-smooth analysis of spectral function, and [Cla90], [HUL93], [BL00] for more general background on non-smooth optimization.

Recall that a *subgradient* of λ_2 at W is a symmetric matrix G that satisfies the inequality

$$\begin{aligned} \lambda_2(\tilde{W}) &\ge \lambda_2(W) + \langle G, \tilde{W} - W \rangle \\ &= \lambda_2(W) + \mathbf{Tr}\, G(\tilde{W} - W) \end{aligned}$$

for any feasible, *i.e.*, symmetric stochastic matrix $\tilde{W}$. Let u be a unit eigenvector associated with $\lambda_2(W)$, then the matrix $G = uu^T$ is a subgradient of $\lambda_2(W)$ (see, for example, [BDX03]).

Using

$$W(p) = I + \frac{1}{2n}\Big(\sum_{l=1}^{m}(p_l E_l + p_{-l}E_{-l})\Big) = I + \frac{1}{2n}\Big(\sum_{|l|=1}^{m} p_l E_l\Big),$$

in terms of the probability vector p, we obtain

$$\lambda_2(W(\tilde{p})) \ge \lambda_2(W(p)) + \sum_{|l|=1}^{m} (v^T(\frac{1}{2n}E_l)v)(\tilde{p}_l - p_l), \tag{38}$$

so that the subgradient $g(p)$ is given by

$$g(p) = \frac{1}{2n}\left(u^T E_{-m} u, \ldots, u^T E_m u\right), \tag{39}$$

with components

$$g_l(p) = \frac{1}{2n} u^T E_l u = -\frac{1}{2n}(u_i - u_j)^2, \quad l \sim (i,j),$$

where $|l| = 1, \ldots, m$.

Observe that if each node i knows its own component u_i of the unit eigenvector, then this subgradient can be computed locally, using only local information.

The following is the projected subgradient method for (40):

- **Initialization:** Initialize p to some feasible vector, for example, p corresponding to the natural random walk. Set $k := 1$.
- **Repeat** for $k \geq 1$,
  - **Subgradient step.** Compute a subgradient $g^{(k)}$ at p, and set
  $$p := p - \nu_k g^{(k)}$$
  - **Projection onto feasible set.** At each node i, project $\mathbf{p_i}$ obtained from the subgradient step onto $\mathbf{1}^T q \leq 1, q \succeq 0$. This is achieved as follows:
    1) If $\sum_{j=1}^{m_i} \max\{0, \mathbf{p_{ij}}\} \leq \mathbf{1}$, then set $\mathbf{p_i} = \max\{0, \mathbf{p_i}\}$, stop.
    2) If not, then use bisection to find $x \geq 0$ such that $\sum_{j=1}^{m_i} \max\{0, \mathbf{p}_{ij} - x\} = 1$; set $\mathbf{p_i} = \max\{0, \mathbf{p}_{ij} - x\}$, stop.

In this algorithm, Step 1 moves p in the direction of the subgradient with stepsize ν_k; we will discuss the stepsizes a little later in this section. Step 2 projects the vector p onto the feasible set. Since the constraints at each node are separable, the variables $\mathbf{p_i}$ corresponding to nodes i are projected onto the feasible set separately.

The projection method is derived from the optimality conditions of the projection problem

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^{m_i} (q_j - \mathbf{p}_{ij})^2 \\ \text{subject to} & \mathbf{1}^T q \leq 1, \quad q \succeq 0 \end{array} \tag{40}$$

as shown.

Introduce Lagrange multipliers $\lambda \in \mathbf{R}_+^m$ for the inequality $q \succeq 0$, and ν for $\mathbf{1}^T q - 1 \leq 0$. The KKT conditions for optimal primal and dual variables q^*, λ^*, ν^* are

$$\begin{aligned} & q^* \succeq 0, \quad \mathbf{1}^T q^* \leq 1 \\ & \lambda^* \succeq 0, \quad \nu^* \geq 0 \\ & \nu^*(\mathbf{1}^T q^* - 1) = 0, \quad \lambda_j^* q_j^* = 0, \ j = 1, \ldots, m_i, \\ & 2(q_j^* - \mathbf{p}_{ij}) + \nu^* - \lambda_j^* = 0, \ j = 1, \ldots, m_i. \end{aligned}$$

Eliminating the slack variables λ_j, we get the equivalent optimality conditions

$$q^* \succeq 0, \quad \mathbf{1}^T q^* \leq 1, \tag{41}$$
$$\nu^* \geq 0, \quad \nu^*(\mathbf{1}^T q^* - 1) = 0, \tag{42}$$
$$q_j^*(2(q_j^* - \mathbf{p}_{ij}) + \nu^*) = 0, \ j = 1, \ldots, m_i, \tag{43}$$
$$2(q_j^* - \mathbf{p}_{ij}) + \nu^* \geq 0, \ j = 1, \ldots, m_i. \tag{44}$$

If $\nu^* < 2\mathbf{p}_{ij}$, then from the last condition, necessarily $q_i^* > 0$. From (43), this gives us $q_j^* = \mathbf{p}_{ij} - \nu^*/2$. If on the other hand $\nu^* \geq 2\mathbf{p}_{ij}$, then $\nu^* \geq 2\mathbf{p}_{ij} - 2q_j^*$ as well since $q_j^* \geq 0$, and so to satisfy (43), we must have $q_j^* = 0$. Combining these gives us that

$$q_j^* = \max\{0, \mathbf{p}_{ij} - \frac{\nu^*}{2}\}. \tag{45}$$

The q_j^* must satisfy $\mathbf{1}^T q^* \leq 1$, *i.e.*, $\sum \max\{0, q_j - \nu^*/2\} \leq 1$. However, we must also satisfy the complementary slackness condition $\nu^*(\mathbf{1}^T q^* - 1) = 0$. These two conditions combined together lead to a unique solution for ν^*, obtained either at $\nu^* = 0$, or at the solution of $\sum \max\{0, q_j - \nu^*/2\} = 1$; from ν^* the q_j^* can be found as described.

2) Decentralization: Now consider the issue of decentralization. Observe that in the above algorithm, g can be computed locally at each node if u, the unit eigenvector corresponding to $\lambda_2(W)$, is known; more precisely, if each node i is aware of its own component of u and that of its immediate neighbours. The projection step can be carried out exactly at each node using local information alone.

The rest of the section proceeds as follows: first we will discuss approximate distributed computation of the eigenvector u of W, and then show that the subgradient method converges to a certain neighborhood of the optimal value in spite of the error incurred during the distributed computation of u at each iteration.

The problem of distributedly computing the top-k eigenvectors of a matrix on a graph is discussed in [KM04]; a distributed implementation of and error analysis for orthogonal iterations is described. By distributed computation of an eigenvector u of a matrix W, we mean that each node i is aware of the i^{th} row of W, and can only communicate with its immediate neighbours; given these constraints, the distributed computation ensures that each node holds its value u_i in the unit eigenvector u.

Since the matrix W is symmetric and stochastic (it is a convex combination of symmetric stochastic matrices), we know that the first eigenvector is $\mathbf{1}$. Therefore orthogonal iterations takes a particularly simple form (in particular, we do not need any Cholesky factorization type of computations at the nodes). We describe orthogonal iterations for this problem below:

- **DecentralOI**: Initialize the process with some randomly chosen vector v_0; for $k \geq 1$, repeat
  - Set $v_k = W v_{k-1}$
  - (Orthogonalize) $v_k = v_k - (\sum_{i=1}^n \frac{1}{n} v_{k_i})\mathbf{1}$
  - (Scale to unit norm) $v_k = v_k / \|v_k\|$

Here, the multiplication by W is distributed, since W respects the graph structure, *i.e.*, $W_{ij} \neq 0$ only if (i, j) is

an edge. So entry i of v_k can be found using only values of v_{k-1} corresponding to neighbours of node i, *i.e.*, the computation is distributed. The orthogonalize and scale steps can be carried out distributedly using the gossip algorithm outlined in this paper, or just by distributed averaging as described in [XB03] and used in [KM04]. Observe that the very matrix W can be used for the distributed averaging step, since it is also a probability matrix. We state the following result (applied to our special case) from [KM04], which basically states that it is possible to compute the eigenvector upto an arbitrary accuracy:

Lemma 5: If DecentralOI is run for $\Omega\left(t\tau_{\text{mix}}\log(16/\epsilon)\right)$ iterations, producing orthogonal vector u, then

$$\|u - u_r\| \leq O\left(\left(\frac{\lambda_3}{\lambda_2}\right)^t n\right) + 3\epsilon^{4t}, \qquad (46)$$

where $\|u - u_r\|$ is the L_2 distance between u and the eigenspace of λ_2; u_r is the vector in the eigenspace achieving this distance.

It is therefore clear that an approximate eigenvector, and therefore an approximate subgradient can be computed distributedly.

3) Convergence analysis: It now remains to show that the subgradient method converges despite approximation errors in computation of the eigenvector, which spill over into computation of the subgradient. To show this, we will use a result from [Kiw04] on the convergence of *approximate* subgradient methods.

Given an optimization problem with objective function f and feasible set S, the approximate subgradient method generates a sequence $\{x^k\}_{k=1}^{\infty} \subset S$ such that

$$x^{k+1} = P_S(x^k - \nu_k g^k),\ g^k \in \partial_{\epsilon_k} f_S(x^k), \qquad (47)$$

where P_S is a projection onto the feasible set, $\nu_k > 0$ is a stepsize, and

$$\partial_{\epsilon_k} f_S(x^k) = \{g : f_S(x) \geq f_S(x^k) + \langle g, x - x^k\rangle - \epsilon_k \ \ \forall x\} \qquad (48)$$

is the ϵ_k subdifferential of the objective function f_S at x^k.

Let $\gamma_k = (1/2)|g_k|^2\nu_k$, and $\delta_k = \gamma_k + \epsilon_k$. Then we have the following theorem from [Kiw04],

Lemma 6: If $\sum \nu_k = \infty$, then

$$\liminf_k f(x^k) \leq f^* + \delta,$$

where $\delta = \limsup \delta_k$, and f^* is the optimal value of the objective function.

Consider the k-th iteration of the subgradient method, with current iterate $p(k)$, and let $\sqrt{\epsilon}$ be the error in the (approximate) eigenvector u corresponding to $\lambda_2(W(p(k)))$. (By error in the eigenvector, we mean the L_2 distance between u and the (actual) eigenspace corresponding to λ_2). Again, denote by u_r the vector in the eigenspace minimizing the distance to u, and denote the exact subgradient computed from u_r by g_r.

We have $\|u - u_r\|^2 \leq \epsilon$. First we find ϵ_k in terms of ϵ as follows:

$$\begin{aligned}\lambda_2(W(p)) &\geq \lambda_2(W(p(k))) + \langle g_r, p - p(k)\rangle \\ &= \lambda_2(W(p(k))) + \langle g, p - p(k)\rangle \\ &\quad - \langle g - g_r, p - p(k)\rangle\end{aligned}$$

This implies,

$$\epsilon_k = \sup_p \langle g - g_r, p - p(k)\rangle = c\|g - g_r\|^2,$$

where c is a scaling constant.

Next, we will find $\|g - g_r\|^2$ in terms of ϵ as follows:

$$\begin{aligned}\|u - u_r\|^2 \leq \epsilon &\Rightarrow \sum_{i=1}^{n}(u_i - u_{r_i})^2 \leq \epsilon \\ &\Rightarrow (u_i - u_{r_i})^2 \leq \epsilon,\ \ 1 \leq i \leq n.\end{aligned}$$

Now, the l^{th} component of $g - g_r$ is

$$\begin{aligned}(g - g_r)_l &= \frac{1}{2n}\left((u_i - u_j)^2 - (u_{r_i} - u_{r_j})^2\right) \\ &= \frac{1}{2n}\left((u_i - u_{r_i}) - (u_j - u_{r_j})\right) \times \\ &\quad \left((u_i - u_j) + (u_{r_i} - u_{r_j})\right).\end{aligned}$$

Combining the facts that $|u_i - u_{r_i}| \leq \sqrt{\epsilon},\ \ \forall i$; and (since $\|u\| = 1$) $|u_i - u_j| \leq \sqrt{2},\ \ \forall i, j$; we get the following

$$(g - g_r)_l^2 \leq \frac{1}{4n^2}(2\sqrt{\epsilon})^2(2\sqrt{2})^2 = 8\epsilon/n^2.$$

Summing over all m edges gives us $\|g - g_r\|^2 \leq 8m\epsilon/n^2$, *i.e.*, $\epsilon_k \leq 8cm\epsilon/n^2 \leq 8c\epsilon$ since $m \leq n^2$ for all graphs.

Now choose $\nu_k = 1/k$. From (39), it can be seen that $\|g_k\|^2$ is bounded above by $\sqrt{m}/n$, and so γ_k in Theorem 6 converges to 0. Therefore if in each iteration i, the eigenvector is computed to within an error of ϵ_i, and $\epsilon = \liminf \epsilon_i$, we have the following result:

Theorem 4: The distributed subgradient method described above converges to a distribution p for which $\lambda_2(W(p))$ is within $8cm\epsilon/n^2 (\leq 8c\epsilon)$ of the globally optimal value $\lambda_2(W(p^*))$.

IV. Applications

In this section, we briefly discuss applications of our results in the context of wireless ad-hoc networks and the Internet. We examine how the performance of averaging algorithms scales with the size (in terms of the number of nodes) of the network.

Before we study this, we need the following result, relating the averaging time of an algorithm $\mathcal{A}(P)$ and the mixing time of the Markov chain on G that evolves according to $W = W(P)$. (Since W is a positive-semidefinite doubly stochastic matrix, the Markov chain with transition matrix W has uniform equilibrium distribution.)

Recall that the mixing time is defined as follows:

Definition 2 (Mixing Time): For a Markov chain with symmetric transition matrix W, let $\Delta_i(t) = \frac{1}{2}\sum_{j=1}^{n} |W_{ij}^t - \frac{1}{n}|$. Then, the ϵ-mixing time is defined as

$$T_{\text{mix}}(\epsilon) = \sup_i \inf\{t : \Delta_i(t') \leq \epsilon, \forall\ t' \geq t\}. \quad (49)$$

We have the following relation between mixing times and averaging times, the proof of which can be found in [BGPS04].

Theorem 5: For a symmetric matrix P, the ϵ-averaging time (in terms of absolute time) of the gossip algorithm $\mathcal{A}(P)$ is related to the mixing time of the Markov chain with transition matrix P as

$$T_{\text{ave}}(\epsilon, P) = \Theta\left(\log n + T_{\text{mix}}(\epsilon)\right).$$

Figure 2 is a pictorial description of Theorem 5. The x-axis denotes mixing time and the y-axis denotes averaging time. The scale on the axis is in order notation. As shown in the figure, for P such that $T_{\text{mix}}(P) = o(\log n)$, $T_{\text{ave}}\left(\frac{1}{n}, P\right) = \Theta(\log n)$; for P such that $T_{\text{mix}}(P) = \Omega(\log n)$, $T_{\text{ave}}\left(\frac{1}{n}, P\right) = \Theta(T_{\text{mix}})$. Thus, knowing mixing property of random walk essentially characterizes the averaging time in the order sense.

A. Wireless Network

The Geometric Random Graph, introduced by Gupta and Kumar [GK00], has been used successfully to model ad-hoc wireless networks. A d-dimensional Geometric Random Graph on n nodes, modeling wireless ad-hoc networks of n nodes with wireless transmission radius r, is denoted as $G^d(n, r)$, and is obtained as follows: place n nodes on a d dimensional unit cube uniformly at random and connect any two nodes that are within distance r of each other. An example of a two dimensional graph, $G^2(n, r)$ is shown in the Figure3.

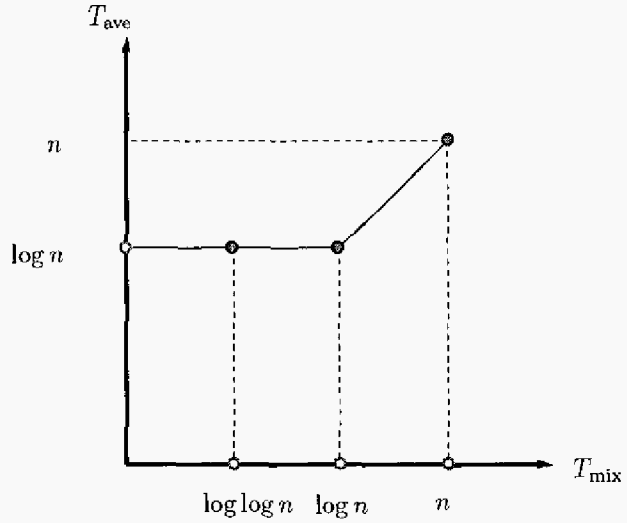

Fig. 2. Graphical interpretation of Theorem 5.

The following is a well-known result about the connectivity of $G^d(n, r)$ (for a proof, see [GK00], [GMPS04], [Pen03]):

Lemma 7: For $nr^d \geq 2\log n$, the $G(n, r)$ is connected with probability at least $1 - 1/n^2$.

Theorem 6: On the Geometric Random Graph, $G^d(n, r)$, the absolute $1/n^\alpha$-averaging time, $\alpha > 0$, of the optimal averaging algorithm is $\Theta\left(\frac{\log n}{r^2}\right)$.

Proof: In [BGPS05], the authors show that for $\epsilon = 1/n^\alpha, \alpha > 0$ the ϵ-mixing times for the fastest-mixing random walk on the geometric random graph $G^d(n, r)$ is of order $\Theta(\frac{\log n}{r^2})$. Therefore, using this and the results of Corollaries 1 and 2, we have the theorem. ■

Thus, in wireless sensor networks with a small radius of communication, distributed computing is necessarily slow, since the fastest averaging algorithm is itself slow. However, consider the natural averaging algorithm, based on the natural random walk, which can be described as follows: each node, when it becomes active, chooses one of its neighbors uniformly at random and averages its value with the chosen neighbor. As noted before, in general, the performance of such an algorithm can be far worse than the optimal algorithm. Interestingly, in the case of $G^d(n, r)$, the performances of the natural averaging algorithm and the optimal averaging algorithm are comparable (i.e. they have averaging times of the same order). We state the following Theorem, which is obtained exactly the same way as Theorem 6, using a result on T_{mix} for the natural random walk from [BGPS05]:

Theorem 7: On the Geometric Random Graph,

$G^d(n,r)$, the absolute $1/n^\alpha$-averaging time, $\alpha > 0$, of the natural averaging algorithm is of the same order as the optimal averaging algorithm, *i.e.*, $\Theta\left(\frac{\log n}{r^2}\right)$.

Implication. In a wireless sensor network, Theorem 6 suggests that for a small radius of transmission, even the fastest averaging algorithm converges slowly; however, the good news is that the natural averaging algorithm, based only on local information, scales just as well as the fastest averaging algorithm. Thus, at least in the order sense, it is not necessary to optimize for the fastest averaging algorithm in a wireless sensor network.

B. Internet

The Preferential Connectivity (PC) model [MPS03] is one of the popular models for the Internet. In [MPS03], it is shown that the Internet is an expander under the preferential connectivity model. This means that there exists a positive constant $\delta > 0$ (independent of the size of the graph), such that for the transition matrix corresponding to the natural random walk, call it P,

$$\delta \leq (1 - \lambda_{\max}(P)) \leq 1, \tag{50}$$

where $\lambda_{\max}(P)$ is the second largest eigenvalue of P in magnitude, *i.e.*, the spectral gap is bounded away from zero by a constant. Let P^* be the transition matrix corresponding to the fastest mixing random walk on the Internet graph under the PC model. The random walk corresponding to P^* must mix at least as fast as the natural one, and therefore,

$$\delta \leq (1 - \lambda_{\max}(P^*)) \leq 1. \tag{51}$$

It is easy to argue that there exists an optimal P^* that is symmetric (given any optimal P_0, the matrix $1/2(P_0 + P_0^T)$ is symmetric, and leads to the same $E[W]$ as P_0). Therefore, from (50), (51), Theorem 3 and Corollary 2, we obtain the following Theorem.

Theorem 8: Under the PC model, the optimal averaging algorithm on the Internet has an absolute ϵ-averaging time $T_{\text{ave}}(\epsilon) = \Theta\left(\log \epsilon^{-1}\right)$.

Implication. The absolute time for distributed computation on the Internet is independent of the size of the network, and depends only on the desired accuracy of the computation[3]. One implication is that exchanging information on Internet via peer-to-peer network built on top of it is extremely fast!

[3]Althought the asymmetry of the P matrix for the natural random walk on the Internet prevents us from exactly quantifying the averaging time, we believe that averaging will be fast even under the natural random walk, since the spectral gap for this random walk is bounded away from 1 by a constant.

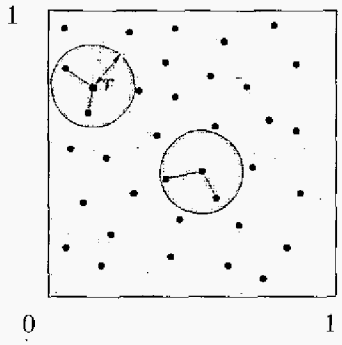

Fig. 3. An example of a Geometric Random Graph in two-dimensions. A node is connected to all other nodes that are within distance r of itself.

V. Conclusion

We presented a framework for the design and analysis of a randomized asynchronous distributed averaging algorithm on an arbitrary connected network. We characterized the performance of the algorithm precisely in the terms of second largest eigenvalue of an appropriate doubly stochastic matrix. This allowed us to find the fastest averaging of this class of algorithms, by establishing the corresponding optimization problem to be convex. We established a tight relation between the averaging time of the algorithm and the mixing time of an associated random walk, and utilized this connection to design fast averaging algorithms for two popular and well-studied networks: Wireless Sensor Networks (modeled as Geometric Random Graphs), and the Internet graph (under the so-called Preferential Connectivity Model). In these models, we find that the natural algorithm is as fast as the optimal algorithm.

In general, solving semidefinite programs in a distributed manner is not possible. However, we utilized the structure of the problem in order to solve the semidefinite program (corresponding to the optimal averaging algorithm) in a distributed fashion using the subgradient method. This allows for self-tuning weights: that is, the network can start out with some arbitrary averaging matrix, say, one derived from the natural random walk, and then locally, without any central coordination, converge to the optimal weights corresponding to the fastest averaging algorithm.

The framework developed in this paper is general and can be utilized for the purpose of design and analysis of distributed algorithms in many other settings.

Acknowledgment

D. Shah thanks Bob Gallager for his suggestions.

REFERENCES

[BDX03] S. Boyd, P. Diaconis, and L. Xiao. Fastest mixing Markov chain on a graph. Submitted to *SIAM Review*, problems and techniques section, February 2003. Available at `www.stanford.edu/~boyd/fmmc.html`.

[BGPS04] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Analysis and optimization of randomized gossip algorithms. In *Proc. 2004 IEEE CDC*, 2004.

[BGPS05] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Mixing times of random walks on geometric random graphs. *Proc. SIAM ANALCO*, 2005.

[BL00] J. M. Borwein and A. S. Lewis. *Convex Analysis and Nonlinear Optimization, Theory and Examples*. Canadian Mathematical Society Books in Mathematics. Springer-Verlag, New York, 2000.

[BS03] R. W. Beard and V. Stepanyan. Synchronization of information in distributed multiple vehicle coordinated control. In *Proceedings of IEEE Conference on Decision and Control*, December 2003.

[BV03] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2003. Available at `http://www.stanford.edu/~boyd/cvxbook.html`.

[Cla90] F. H. Clarke. *Optimization and Nonsmooth Analysis*. SIAM, Philadelphia, 1990.

[DZ99] A. Dembo and O. Zeitouni. *Large Deviations Techniques and Applications*. Springer, 1999.

[EGHK99] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proc. 5th Intl. Conf. on Mobile Computing and Networking*, 1999.

[GK00] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, March 2000.

[GMPS04] A. El Gamal, J. Mammen, B. Prabhakar, and D. Shah. Throughput-delay trade-off in wireless networks. In *Proc. 2004 INFOCOM*, 2004.

[GvRB01] I. Gupta, R. van Renesse, and K. Birman. Scalable fault-tolerant aggregation in large process groups. In *Proc. Conf. on Dependable Systems and Networks*, 2001.

[HHL88] S. Hedetniemi, S. Hedetniemi, and A. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18:319–349, 1988.

[HUL93] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms*. Springer-Verlag, Berlin, 1993.

[IEGH02] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of netowrk density on data aggregation in wireless sensor networks. In *Proc. Intl. Conf. on Distributed Computing Systems*, 2002.

[JLS03] A. Jadbabaie, J. Lin, and A. S.Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, June 2003.

[KDG03] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proc. Conference on Foundations of Computer Science*. IEEE, 2003.

[KEW02] B. Krishnamachari, D. Estrin, and S. Wicker. The impact of data aggregation in wireless sensor networks. In *Intl. Workshop of Distributed Event Based Systems*, 2002.

[Kiw04] K. Kiwiel. Convergence of approximate and incremental subgradient methods for convex optimization. *SIAM Journal on Optimization*, 14(3):807–840, 2004.

[KK02] D. Kempe and J. Kleinberg. Protocols and impossibility results for gossip-based communication mechanisms. In *Proc. 43st IEEE Symp. on Foundations of Computer Science*, 2002.

[KKD01] D. Kempe, J. Kleinberg, and A. Demers. Spatial gossip and resource location protocols. In *Proc. 33rd ACM Symp. on Theory of Computing*, 2001.

[KM04] D. Kempe and Frank McSherry. A decentralized algorithm for spectral analaysis. In *Symposium on Theory of Computing*. ACM, 2004.

[KSSV00a] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Proc. 41st IEEE Symp. on Foundations of Computer Science*, 2000.

[KSSV00b] R. Karp, C. Schindelhauer, S. Shenker, and B. Vcking. Randomized rumor spreading. In *Proc. Symposium on Foundations of Computer Science*. IEEE, 2000.

[LBF04] Z. Lin, M. Brouke, and B. Francis. Local control strategies for groups of mobile autonomous agents. 49(4):622–629, April 2004.

[Lew96] A. S. Lewis. Convex analysis on the Hermitian matrices. *SIAM Journal on Optimization*, 6:164–177, 1996.

[Lew99] A. S. Lewis. Nonsmooth analysis of eigenvalues. *Mathematical Programming*, 84:1–24, 1999.

[LO96] A. S. Lewis and M. L. Overton. Eigenvalue optimization. *Acta Numerica*, 5:149–190, 1996.

[MFHH02] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. In *Proc. 5th Symp.on Operating Systems Design and Implementation*, 2002.

[MPS03] M. Mihail, C. Papadimitriou, and A. Saberi. Internet is an exapnder. In *Proc. Conf. on Foundations of Computer Science*, 2003.

[Mur03] L. Mureau. Leaderless coordination via bidirectional and unidirectional time-dependent communication. In *Proceedings of IEEE Conference on Decision and Control*, December 2003.

[OSM04] R. Olfati-Saber and R. M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 49(9):1520–1533, September 2004.

[Ove92] M. L. Overton. Large-scale optimization of eigenvalues. *SIAM Journal on Optimization*, 2:88–120, 1992.

[OW93] M. L. Overton and R. S. Womersley. Optimality conditions and duality theory for minimizing sums of the largest eigenvalues of symmetric matrices. *Mathematical Programming*, 62:321–357, 1993.

[Pen03] M. Penrose. *Random geometric graphs*. Oxford studies in probability. Oxford University Press, Oxford, 2003.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc.of the ACM SIGCOMM Conference*, 2001.

[RSW98] Y. Rabani, A. Sinclair, and R. Wanka. Local divergence of Markov chains and the analysis of iterative load-balancing schemes. In *Proc. Conference on Foundations of Computer Science*. IEEE, 1998.

[SMK+01] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc.of the ACM SIGCOMM Conference*, 2001.

[vR00] R. van Renesse. Scalable and secure resource location. In *33rd Hawaii Intl. Conf. on System Sciences*, 2000.

[WSV00] H. Wolkowicz, R. Saigal, and L. Vandenberghe. *Handbook of Semidefinite Programming, Theory, Algorithms, and Applications*. Kluwer Academic Publishers, 2000.

[XB03] L. Xiao and S. Boyd. Fast linear iterations for distributed averaging. In *Proc. 2003 Conference on Decision and Control*, December 2003.