# Distributed Algorithms for Average Consensus of Input Data With Fast Convergence

Kan Xie, Qianqian Cai, Zhaorong Zhang, and Minyue Fu, *Fellow, IEEE*

***Abstract*—This paper proposes fast convergent distributed algorithms for weighted average consensus of input data. For acyclic graphs, we give an algorithm that converges to the exact weighted average consensus in a finite number of iterations, equal to the graph diameter. For loopy (cyclic) graphs, we offer two remedies. In the first one, we give another distributed algorithm to enable our average consensus algorithm applicable to a loopy graph by converting it into a spanning tree. In the second one, we consider a slightly modified average consensus problem whose optimal solution approximates the consensus solution with arbitrary precision, and give a modified average consensus algorithm with guaranteed exponential convergence to the optimal solution. The proposed average consensus algorithms enjoy low complexities, robustness to transmission adversaries, and asynchronous implementation. Our algorithms are conceptually different from the popular graph Laplacian approach, and converge much faster than the latter approach.**

***Index Terms*—Average consensus, distributed algorithms, distributed estimation, networked control.**

## I. Introduction

DISTRIBUTED consensus has been one of the dominant research problems in networked systems. Applications of distributed consensus range from statistical learning to sensor networks, distributed optimization, and computer science [1]–[17]. This topic has attracted great attention in distributed estimation, control of multiagent systems, and so on. For example, Xu *et al.* [5] devised a distributed algorithm to estimate the relative interagent states for a multiagent system; and Chen and Shi [6] designed a distributed procedure for the consensus of a multiagent system in the frequency domain. Distributed consensus also plays a crucial role in wider applications, including distributed sensing and fusion [18], distributed optimization [19]–[22], and machine learning [23], [24].

Among many kinds of distributed consensus problems, the so-called *average consensus* has been studied mostly extensively [1]–[13]. Most of the average consensus algorithms are iterative algorithms based on the *graph Laplacian* approach (see [1]) which performs *distributed averaging* at each iteration, which amounts to a linear dynamic system with the transition matrix which is a stochastic matrix with the averaging weights. This approach enjoys several benefits, including its simplicity and weak requirement on the network topology. However, average consensus is reached only asymptotically, and its exponential convergence rate is significantly influenced by the network topology. A nonlinear average consensus algorithm is given in [14] and [15] using a nonlinear weighted update protocol, the convergence is faster but still asymptotic.

Many attempts have been made on finite-time algorithms for distributed average consensus. In [7]–[10] and [16], it is shown that finite-time average consensus can be achieved by applying a sequence of time-varying stochastic matrices, a scheme also known as *definitive consensus*. However, how to design these stochastic matrices and how long the sequence must be may not be simple questions to answer in general. Finite-time average consensus can also be achieved using continuous-time protocols [11]–[13], but they require continuous (infinite) information exchange.

In this paper, we propose fast convergent distributed algorithms for weighted average consensus. For acyclic graphs (i.e., tree graphs), we provide an algorithm that converges to the exact weighted average consensus in a finite number of iterations, much faster than previously known finite-time algorithms. The number of iterations required for convergence is the diameter of the graph, i.e., the largest number of hops between two nodes. For most large-scale networks in practice, the graph diameter $d$ is relatively small compared to the network size $n$. For example, the well-known *small-world networks* [25] have $d \propto \log(n)$. Moreover, the so-called *scale-free networks* [26], [27] have $d \propto \log\log(n)$. Note that small-world networks and scale-free networks are commonly used to model social networks, computer networks, communication networks, transportation networks, and biological networks [25]–[27]. Thus, our algorithm is very efficient

Manuscript received January 29, 2019; revised March 19, 2019; accepted April 27, 2019. This work was supported by the National Natural Science Foundation of China under Grant 61633014, Grant U1701264, and Grant 61803101. This paper was recommended by Associate Editor C.-C. Lim. *(Corresponding author: Qianqian Cai.)*

K. Xie and Q. Cai are with the School of Automation, Guangdong University of Technology, Guangzhou 510006, China, and also with the Guangdong Key Laboratory of Intelligent Decision and Cooperative Control, Guangdong University of Technology, Guangzhou 510006, China (e-mail: kanxiegdut@gmail.com; qianqian.cai@outlook.com).

Z. Zhang is with the School of Electrical Engineering and Computer Science, University of Newcastle, Callaghan, NSW 2308, Australia (e-mail: zhaorong.zhang@uon.edu.au).

M. Fu is with the School of Electrical Engineering and Computer Science, University of Newcastle, Callaghan, NSW 2308, Australia, also with the School of Automation, Guangdong University of Technology, Guangzhou 510006, China, and also with the Guangdong Key Laboratory of Intelligent Decision and Cooperative Control, Guangdong University of Technology, Guangzhou 510006, China (e-mail: minyue.fu@newcastle.edu.au).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2019.2914385

for acyclic graphs. Indeed, we show that our algorithm is the *fastest* in the sense that no other algorithm can achieve average consensus with fewer iterations than ours. Our algorithm is inspired by the well-known belief propagation algorithm which finds wide applications in many fields (see [28]–[30]).

For our average consensus algorithm to be applied to loopy graphs, two remedies are offered. First, we give another distributed algorithm to convert a loopy graph to a spanning tree by removing loop-forming edges. This algorithm is also very fast, completing only in $d+1$ iterations. Combining this algorithm with the proposed average consensus algorithm, we achieve a fully distributed average consensus algorithm for loopy graphs with finite-time convergence. Second, we consider a slightly modified average consensus problem whose optimal solution approximates the consensus solution with arbitrary precision by choosing a scaling parameter. We derive a modified average consensus algorithm and show that it has guaranteed exponential convergence to the optimal solution, when applied to a loopy graph directly.

We also show that the proposed distributed consensus algorithms enjoy certain robustness against transmission loss and delay and can be implemented asynchronously. The algorithms have very low complexities, namely, each node's computation, transmission, and storage complexity per iteration is only proportional to the number of neighboring nodes.

The remaining of this paper is organized as follows. Section II formulates the weighted average consensus problem. Section III gives the first proposed algorithm for average consensus over acyclic graphs. Section IV gives a distributed algorithm for loop removal. Section V presents a number of interesting properties of Algorithm 1. Section VI gives the modified average consensus algorithm for loopy graphs. Section VII gives examples. Section VIII concludes this paper.

## II. Problem Description

Consider an undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with a set of nodes $\mathcal{V} = \{1, 2, \ldots, n\}$ and a set of edges $\mathcal{E} = \{(i, j) : i, j \in \mathcal{V}\}$. We say that a graph is *undirected* if information can propagate in two directions between all neighboring nodes. A graph is said to be *acyclic* if it is *connected* and it has no loops, that is, it is a *tree graph*. Denote by $\mathcal{N}_i$ the set of neighboring nodes of node $i$, and by $|\mathcal{N}_i|$ the cardinality of $\mathcal{N}_i$.

*Assume $|\mathcal{N}_i| \ll n$:* Let each node $i \in \mathcal{V}$ be associated with the *input data* consisting of a vector $y_i$ of same dimension and a weight $w_i > 0$. Denote by $\bar{x}$ the weighted average of these $y_i$, i.e.,

$$\bar{x} = \frac{\sum_{i\in\mathcal{V}} w_i y_i}{\sum_{i\in\mathcal{V}} w_i}. \tag{1}$$

The problem of *distributed average consensus* is to provide an iterative algorithm that runs on every node $i \in \mathcal{V}$ to produce an estimate of $\bar{x}$ such that after a number of iterations, the estimate by each node will converge to $\bar{x}$. When $w_i = 1$ for all nodes, weighted average consensus becomes the standard (unweighted) average consensus.

It is clear that certain *constraints* need to be imposed on the algorithm's complexities of communication, computation, and storage to call it *distributed*. In this paper, these include the following.

1) *Local Information Exchange:* Each node $i$ is allowed to exchange information with each $j \in \mathcal{N}_i$ only once per iteration.
2) *Local Computation:* Each node $i$'s computational load is limited to be at most $O(|\mathcal{N}_i|)$ per iteration.
3) *Local Storage:* Each node $i$'s storage is limited to be at most $O(|\mathcal{N}_i|)$ over all iterations.

*Lemma 1:* Under the constraint on local information exchange, a connected undirected graph $\mathcal{G}$ needs a minimum of $d$ iterations to achieve average consensus for general sets of $\{y_i\}$ and $\{w_i\}$, where $d$ is the diameter of $\mathcal{G}$.

*Proof:* Take nodes $i$ and $j$ to be $d$ hops away from each other. The information at node $i$ must be propagated to node $j$ for it to compute $\bar{x}$. By the constraint on local information exchange, this process takes at least $d$ iterations. ■

**Algorithm 1** Distributed Algorithm for Average Consensus

**Inputs:** For each node $i$, the input data are $w_i$ and $y_i$.

**Outputs:** For each node $i$ and iteration $k$, the output is the estimate $\hat{x}_i(k)$ of $\bar{x}_i$.

**Initialization:** For each node $i$, do: For each $j \in \mathcal{N}_i$, set $x_{i\to j}(0) = y_i$, $s_{i\to j}(0) = w_i$ and transmit them to node $j$.

**Main loop:** At iteration $k = 1, 2, \cdots$, for each node $i$, compute

$$\tilde{s}_i(k) = w_i + \sum_{j\in\mathcal{N}_i} s_{j\to i}(k-1) \tag{2}$$

$$\tilde{x}_i(k) = w_i y_i + \sum_{j\in\mathcal{N}_i} s_{j\to i}(k-1) x_{j\to i}(k-1) \tag{3}$$

$$\hat{x}_i(k) = \frac{\tilde{x}_i(k)}{\tilde{s}_i(k)}, \tag{4}$$

then for each $j \in \mathcal{N}_i$, compute

$$s_{i\to j}(k) = \tilde{s}_i(k) - s_{j\to i}(k-1) \tag{5}$$

$$x_{i\to j}(k) = \frac{\tilde{x}_i(k) - s_{j\to i}(k-1) x_{j\to i}(k-1)}{\tilde{s}_i(k) - s_{j\to i}(k-1)} \tag{6}$$

and transmit them to node $j$.

## III. Distributed Algorithm for Average Consensus Over Acyclic Graphs

In this section, we present the first proposed distributed algorithm for average consensus, give its main property on acyclic graphs, and analyze its complexities.

### A. Distributed Algorithm

Let $x_{i\to j}(k)$ denote the information passed from node $i$ to node $j$ at time $k$, which represents a *scaled estimate* of the average $\bar{x}$ known to node $i$ without using information from node $j$. Also denote by $s_{i\to j}(k)$ the *scale* passed from node $i$ to node $j$ at time $k$, which represents the weighted number of nodes used to compute $x_{i\to j}(k)$. We also define two temporary internal variables in node $i$, $\tilde{s}_i(k)$ and $\tilde{x}_i(k)$. Algorithm 1 is the proposed distributed algorithm for weighted average consensus.

*Theorem 1:* Suppose $\mathcal{G}$ is undirected and acyclic with diameter $d$. Then

$$\tilde{s}_i(k) = w_i + \sum_{m\in\mathcal{V}_i(k)} w_m \tag{7}$$

$$\tilde{x}_i(k) = w_i y_i + \sum_{m\in\mathcal{V}_i(k)} w_m x_m \tag{8}$$

for $k = 1, 2, \ldots, d$, where $\mathcal{V}_i(k)$ is the set of nodes in $\mathcal{G}$ at most $k$ hops away from node $i$ (not including node $i$). Consequently, average consensus is achieved by Algorithm 1 after $d$ iterations, i.e.,

$$\hat{x}_i(k) = \bar{x} \quad \forall\ k \geq d,\ \ i \in \mathcal{V}. \tag{9}$$

*Proof:* For any edge $(i, j) \in \mathcal{E}$, we analyze the convergence of $s_{i\to j}(k)$ and $x_{i\to j}(k)$. To do so, we build two disjoint subgraphs $\mathcal{G}_i$ (containing node $i$) and $\mathcal{G}_j$ (containing node $j$) of $\mathcal{G}$ by removing $(i, j)$. Notice that $\mathcal{G}_i$ and $\mathcal{G}_j$ are disjoint due to the acyclic nature of $\mathcal{G}$. Also, it is clear that $\mathcal{G}$ is formed by merging $\mathcal{G}_i$, $\mathcal{G}_j$ and edge $(i, j)$. Denote by $\mathcal{W}_i$ (resp. $\mathcal{W}_j$) the set of nodes in $\mathcal{G}_i$ (resp. $\mathcal{G}_j$). We see from (2)–(6) that $s_{i\to j}(k)$ and $x_{i\to j}(k)$ are constructed using the information in $\mathcal{G}_i$ only because $s_{j\to i}(k-1)$ and $x_{j\to i}(k-1)$ (representing the information flow from node $j$ to node $i$) get removed in each iteration. That is, (2)–(5) can be rewritten as

$$s_{i\to j}(k) = w_i + \sum_{m\in\mathcal{N}_i\backslash\{j\}} s_{m\to i}(k-1)$$

$$x_{i\to j}(k) = w_i y_i + \sum_{m\in\mathcal{N}_i\backslash\{j\}} s_{m\to i}(k-1) x_{m\to i}(k-1)$$

which shows that the information $(s_{j\to i}(k-1), x_{j\to i}(k-1))$ is not used in computing $(s_{i\to j}(k), x_{j\to i}(k))$. Similarly, $s_{j\to i}(k)$ and $x_{j\to i}(k)$ are constructed using the information in $\mathcal{G}_j$ only.

Recall that at Initialization ($k = 0$), $s_{i\to j}(0) = w_i$ is set. Consider the case of $k = 1$. From (2) and (5), we see that $s_{i\to j}(1)$ contains all $s_{m\to i}(0)$ for all the neighboring nodes $m$, except node $j$. Following the definition of $\mathcal{V}_i(k)$, we have $\mathcal{V}_i(1) = \mathcal{N}_i$. It follows that:

$$s_{i\to j}(1) = w_i + \sum_{m\in\mathcal{N}_i\backslash\{j\}} s_{m\to i}(0) = w_i + \sum_{m\in\mathcal{V}_i(1)\backslash\mathcal{W}_j} w_m.$$

Similarly, for $k = 2$, we have

$$s_{i\to j}(2) = w_i + \sum_{m\in\mathcal{N}_i\backslash\{j\}} \left( w_m + \sum_{t\in\mathcal{V}_m(1)\backslash\mathcal{W}_i} w_t \right).$$

The set of $m$ with $m \in \mathcal{N}_i\backslash\{j\}$ contains all the nodes one hop away from node $i$, except those in $\mathcal{G}_j$, and the set of $t$ with $t \in \mathcal{V}_m(1)\backslash\mathcal{W}_i$, $m \in \mathcal{N}_i\backslash\{j\}$ contains all the nodes that are two hops away from node $i$, except those nodes in $\mathcal{G}_j$. Hence

$$s_{i\to j}(2) = w_i + \sum_{m\in\mathcal{V}_i(2)\backslash\mathcal{W}_j} w_m.$$

That is, the weights of all the nodes 2 hops away from node $i$, except those in $\mathcal{G}_j$, will be included in $s_{i\to j}(2)$. Repeating the above process, we get, for any $k$

$$s_{i\to j}(k) = w_i + \sum_{m\in\mathcal{V}_i(k)\backslash\mathcal{W}_j} w_m.$$

Next, consider $\tilde{s}_i(k)$ and $\tilde{x}_i(k)$. Notice from (5) that $\tilde{s}_i(k) = s_{i\to j}(k) + s_{j\to i}(k-1)$. Also notice that node $j$ is 1 hop away from node $i$, meaning that all the nodes in $\mathcal{G}_j$ which are $k-1$ hops away from node $j$ are $k$ hops away from node $i$, when considering $\mathcal{G}$. That is

$$\big(\mathcal{V}_i(k)\backslash\mathcal{W}_j\big) \cup \big(\mathcal{V}_j(k-1)\backslash\mathcal{W}_i\big) \cup \{j\} = \mathcal{V}_i(k).$$

Then (7) follows because:

$$\begin{aligned}\tilde{s}_i(k) &= \left( w_i + \sum_{m\in\mathcal{V}_i(k)\backslash\mathcal{W}_j} w_m \right) + \left( w_j + \sum_{m\in\mathcal{V}_j(k-1)\backslash\mathcal{W}_i} w_m \right) \\ &= w_i + \sum_{m\in\mathcal{V}_i(k)} w_m.\end{aligned}$$

Equation (8) is shown in the same way. Note that $\tilde{x}_i(d)$ is the weighted sum of all the nodes in $\mathcal{G}$ and $\tilde{s}_i(d)$ is the sum of their weights, resulting in $\hat{x}_i(d) = \bar{x}$ for each node $i$.

It remains to check that all the updates will end after iteration $d$. Consider any path $p$ in $\mathcal{G}$ going through nodes $i$ and $j$. This path is split into three segments: 1) path $p_i$ in $\mathcal{G}_i$; 2) edge $(i, j)$; and 3) path $p_j$ in $\mathcal{G}_j$. Denote the maximum path length of $p_i$ by $d_i$, and the maximum path length of $p_j$ by $d_j$. Since the set $\mathcal{V}_i(k)\backslash\mathcal{W}_j$ contains all the nodes in $\mathcal{G}_i$ that are at most $k$ hops away from node $i$ (excluding node $i$ itself), it is clear that $\mathcal{V}_i(k)\backslash\mathcal{W}_j = \mathcal{V}_i(d_i)\backslash\mathcal{W}_j$ for all $k \geq d_i$. It follows that $s_{i\to j}(k)$ and $x_{i\to j}(k)$ must converge after $d_i$ iterations. Similarly, $s_{j\to i}(k)$ and $x_{j\to i}(k)$ will converge after $d_j$ iterations. Because the maximum path length of $p$ is $d$, we get $d_i + d_j + 1 \leq d$. In particular, $d \geq d_i$ and $d - 1 \geq d_j$. Hence, at iteration $d$, all the terms $s_{i\to j}(d), x_{i\to j}(d), s_{j\to i}(d-1), x_{j\to i}(d-1)$ must have converged. From (5) and (6), the above implies that $\tilde{s}_i(d)$ and $\tilde{x}_i(d)$ must have converged. ■

*Remark 1:* Although Theorem 1 states that average consensus is achieved at every node after $d$ iterations, most nodes can achieve so earlier than $d$ iterations in reality. Indeed, following the definition of $\mathcal{V}_i(k)$ and (7) and (8) in Theorem 1, we see that convergence at each node $i$ is achieved at iteration $k_i$ as soon as $\mathcal{V}_i(k_i) = \mathcal{V}$, and this is so when $k_i$ equals to the number of hops needed to reach any node in $\mathcal{V}$ from node $i$. It is clear that $k_i \leq d$ in general and $k_i < d$ for most $i$.

### *B. Complexity Analysis*

It is straightforward to verify that Algorithm 1 satisfies the aforementioned constraints 1)–3). In fact, in iteration $k$, node $i$ uses $(s_{j\to i}(k-1), x_{j\to i}(k-1))$ received from each neighboring node $j$ only once, then transmits back the computed $(s_{i\to j}(k), x_{i\to j}(k))$. The computational complexity of (2)–(4) is $O(|\mathcal{N}_i|)$ for each node $i$. Finally, each node $i$ needs only to store $(s_{i\to j}(k), x_{i\to j}(k))$ for each $j \in \mathcal{N}_i$ [other than $\hat{x}_i(k)$ and the temporary variables $\tilde{s}_i(k)$ and $\tilde{x}_i(k)$], thus its complexity is also $O(|\mathcal{N}_i|)$. Combining the above with Theorem 1, we see that the complexity of Algorithm 1 for an acyclic graph is $O(d|\mathcal{N}_i|)$ for each node $i$. The complexity of Algorithm 1 depends on the graph diameter $d$. But as explained in Section I, $d$ is typically small compared to the network size for most large-scale networks in practice.

## IV. Distributed Algorithm for Loop Removal

One main restriction of Algorithm 1 is that it is applicable to acyclic graphs only. When applied to loopy graphs, approximate average consensus will be produced. In order to properly apply Algorithm 1 to loopy graphs, it is necessary to remove certain edges in $\mathcal{G}$ to make it loop free. That is, our problem is to construct a *spanning tree*, a tree graph which is a connected subgraph of $\mathcal{G}$ containing all the nodes of $\mathcal{G}$. Fortunately, this is a well-known classical problem in graph theory. The standard (and most popular) algorithms are depth-first search (DFS) and breath-first search (BFS) [31], [32].

First, we explain how DFS works. We assume that the graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ is connected. Start from a randomly chosen root node, say node $i$. Mark this node as a visited node, declare this as the progressing node and proceed to the following iteration. For the progressing node $i$, if there is no more unvisited edge from node $i$, declare the parent node of $i$ as the progressing node and proceed to the next iteration; otherwise, take any unvisited edge $(i,\ j)$. If $j$ was visited before, then the edge $(i,\ j)$ (called *back edge*) should be removed; otherwise, mark node $j$ as a visited node and the progressing node and proceed to the next iteration. The iterative process stops when the progressing node returns to the root node.

Many variants of DFS are available, under the name of "distributed DFS" where the term "distributed" means that each node in the resulting spanning tree will know their parent node and child nodes (see [33] for a summary of them). Efficient algorithms also exist for finding a minimum weight diameter spanning tree of a weighted graph [34], [35].

The standard DFS or BFS has two disadvantages. First, the searching of loop-forming edges is done sequentially, thus the complexity of $O(|\mathcal{V}| + |\mathcal{E}|)$ is usually too high for large-scale networks. Second, it is not clear how to select a root node in a distributed way (in our sense).

We will give a new DFS algorithm to address these issues.

### A. Distributed DFS Algorithm

As in a standard DFS algorithm, we start with a root node in $\tilde{\mathcal{G}}$ (how to choose it will be discussed later) and send out a token from it. Instead of searching for loop-forming edges in a sequential way (as in the standard DFS), we allow searching to be done on *multiple* branches simultaneously to reduce the iteration number considerably. More specifically, a token is transmitted to all the neighboring nodes which then relay it to other neighboring nodes and so on. Each node, when receiving a token, checks if it is looped back from its earlier transmission, and if so, marks the incoming edge as a loop-forming edge. The details are in Algorithm 2.

We first address the issue of how to choose a root node in a distributed fashion. It turns out that, with the assumption that every node has a unique numerical ID (which is a quite reasonable assumption for most networks), this problem can be solved using a max consensus algorithm [17]. For a connected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with diameter $d$, the max consensus algorithm finds the maximum node value in $d$ iterations, and it obeys the three aforementioned constraints for distributed algorithms. Denoting by $z_i$ the node value for every node $i \in \mathcal{V}$, the max consensus algorithm is given below:

1) each node $i$ initializes $z_i(0) = z_i$ and transmits it to its neighbors;
2) for iterations $k = 1, 2, \ldots$, each node $i$ updates $z_i(k) = \max_{j\in\mathcal{N}_i} z_j(k-1)$ and transmits it to its neighbors.

Then, max consensus is reached in $d$ steps, i.e., $z_i(d) = \max_{j\in\mathcal{V}} z_j$, provided that $\mathcal{G}$ is connected. The details of the max consensus algorithm can be found in [17].

**Algorithm 2** Distributed DFS Algorithm for Spanning Tree

**Input**: The given graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$.
**Output**: The spanning tree with modified (marked) $\mathcal{E}$.
**Initialization**: Select a root node in $\mathcal{G}$ (by running the max-consensus algorithm on node ID numbers for $d$ iterations and setting the node with maximum ID number as the root node). Mark the root node as "visited" and all other nodes as "unvisited." Transmit a token from the root node to each of its neighboring nodes.
**Iterations** $k = 1, 2, \ldots$: For each node $i$, do the following:
1) If it does not receive the token, do nothing;
2) If it is "unvisited" and it receives the token from only *one* neighbor, then mark the node as "visited", and relay the token to all other neighboring nodes without the "removal" mark, except the incoming edge (i.e., the edge where the token came from);
3) If it is "unvisited" and it receives the token from *multiple* neighbors, then mark the node as "visited", leave one (any one) incoming edge alone and mark all other the incoming edges as "removal", and then relay the token to all other neighboring nodes without the "removal" mark, except the remaining incoming edge;
4) If it is already "visited" and it receives the token, mark the incoming edge as "removal" and do not relay the token further.

The promised distributed DFS algorithm is given in Algorithm 2. It is assumed that $\mathcal{G}$ is connected.

*Theorem 2:* Suppose $\mathcal{G}$ has diameter $d$ and has a unique numerical ID for each node. Then, Algorithm 2 gives a unique root node and it converges in at most $d+1$ iterations, and the graph $\mathcal{G}$ after removing all the edges marked with "removal," is a spanning tree of $\mathcal{G}$.

*Proof:* The uniqueness of the root node comes from the uniqueness of the maximum node label. Denote this node by $r$. Recall our earlier definition of $\mathcal{V}_r(k)$, which is the set of nodes in $\mathcal{G}$ at most $k$ hops away from $r$. Also denote by $\mathcal{L}_r(k)$ the set of nodes in the $k$th level of the resulting spanning tree. In particular, $\mathcal{L}_r(0) = \{r\}$, i.e., the chosen root node is the root node of the resulting spanning tree. We now focus on rest of the algorithm.

Initially, none of the edges has a "removal" mark, and all the nodes (except the root node) is "unvisited" (meaning that it has not received a token before). We show below that $\mathcal{L}_r(k)$ is indeed the set of the nodes which are visited for the first time in the $k$th iteration.

Indeed, at iteration 1, the token is passed on from node $r$ to all the nodes in $\mathcal{V}_r(1)$, and these nodes naturally form the first level of the spanning tree, i.e., they are in $\mathcal{L}_r(1)$.

At iteration $k \geq 2$, each node which receives a token, unless visited earlier, relays the token to all of its unmarked neighbors excluding those where the token just came from [see steps 2) and 3)]. This means that if a node receives a token at iteration $k$ for the first time, it must be in $\mathcal{V}_r(k)$ and not in $\mathcal{L}_r(k-1)$. These nodes naturally form $\mathcal{L}_r(k)$.

To see how the loop-forming edges are removed, we note that if a node receives the token multiple times (whether in a single iteration or in several iterations), it means that the root node has multiple paths to this node. Therefore, all of these incoming edges, except one (any one), should be removed without loosing its connectivity to the root node. This is done in steps 3) and 4). The survived incoming edge corresponds to the earliest visit [steps 2) and 3)], meaning that only the shortest path (not necessarily unique though) to the root node is kept.

By the definition of graph diameter, it takes at most $d$ iterations for all the nodes to be visited. It then takes another iteration to remove the remaining loop-forming edges (if any). After $d+1$ iterations, the token ceases to exist and each node has one and only one incoming edge that is linked to the root node, hence the remaining graph is a spanning tree. ■

*Remark 2:* We commented earlier the works of [34] and [35] which give efficient distributed algorithms for finding a minimum weight diameter spanning tree for a weighted graph. Our algorithm differs from that in [34] and [35] in the sense that ours requires only $d+1$ iterations, whereas [34] and [35] focus on the weights of a graph and the overall efficiency of the algorithm.

### B. Complexity Comparison With Standard DFS Algorithm

Recall that the standard DFS algorithm has complexity of $O(|\mathcal{V}| + |\mathcal{E}|)$ [31]. In contrast, the proposed Algorithm 2 has very low complexity for each node because during each of the $d$ iterations, each node relays the token only to its neighbors where were not visited before. In contrast, the above complexity is per node and thus is not contradictory to the total complexity of $O(|\mathcal{V}|+|\mathcal{E}|)$ for the standard DFS algorithm. That is, we have distributed the total complexity to individual nodes.

The use of multiple tokens in our algorithm significantly reduces the number of iterations (down to $d+1$). In contrast, the standard DFS algorithm requires $|\mathcal{V}|$ iterations because every node needs to be visited sequentially via a single token.

## V. Algorithm Properties and Modifications

In this section, we present a number of properties and modifications of the proposed algorithm for average consensus.

### A. Robustness to Transmission Loss and Delay

Transmission loss and delay are unavoidable in networks. The proposed Algorithm 1 is resilient to these errors. This is because each node uses the most recent information received from its neighbors to update its own information. Delayed arrival of information from neighbors can only delay the update and the convergence of the algorithm, without changing the final consensus. Similarly, if a packet loss happens, it is sufficient that retransmission occurs and the information arrives eventually, which is implemented by most communications protocols, e.g., transmission control protocol (TCP), via acknowledgment (ACK). This type of packet loss becomes transmission delay, thus will not alter the eventual consensus.

More specifically, for any edge $e$ in $\mathcal{G}$, denote its transmission time by a positive integer $T(e) : T(e) = 1$ (unit of time) for no delay; $T(e) > 1$ if there is $T(e)$ units of time delay. For a path $p$ in $\mathcal{G}$, its *path transmission time* $T(p)$ is defined to be the time it takes to relay a packet along this path, by adding up the transmission times of the edges. The following result holds.

*Corollary 1:* Suppose there exists (*maximum path transmission time*) $T_{\max}$ such that $T(p) \leq T_{\max}$ for every path $p$ in $\mathcal{G}$. Then, under the conditions of Theorem 1, Algorithm 1 achieves average consensus within $T$ iterations, i.e., $\hat{x}_i(k) = \bar{x}$ for all $k \geq T_{\max}$, $i \in \mathcal{V}$.

*Proof:* Consider any node $i$. Since $\mathcal{G}$ is acyclic, all the nodes in $\mathcal{G}$ can be organized as a tree, with node $i$ in the root. Note from (7) and (8) in Theorem 1 that if there is no delay and loss, then at each iteration $k$, the weights and state variables that are $k$ hops away from node $i$ are added to $\tilde{s}_i(k)$ and $\tilde{x}_i(k)$. It is clear that if there is $T(e) > 1$ for some edge $e = (i_1, i_2)$ along some path $p$ originated from node $i$ (assuming that node $i_1$ is closer to node $i$ than node $i_2$), then edge $e$ can be replaced with a path of $(i_1, j_1), (j_1, j_2), \ldots, (j_{(T(e)-1)}, i_2)$ without delay in each edge in the path, where the extra nodes $j_1, j_2, \ldots, j_{(T(e)-1)}$ are fictitiously added with the corresponding weights $w_{j_1} = \ldots w_{j_{(T(e)-1)}} = 0$ and any state variables. Following Theorem 1, we see that the weights in all the nodes along the part of the path $p$ starting from node $i_2$ will take $T(e) - 1$ extra iterations to reach node $i_1$. Because the maximum path transmission time is $T$, we conclude that after $T_{\max}$ iterations, $\hat{x}_i(k) = \bar{x}$ for all $k \geq T_{\max}$, $i \in \mathcal{V}$. Note that the fictitious weights and variables do not affect the result. ■

### B. Asynchronous Implementation

Asynchronous implementation of an algorithm refers to the idea that each node executes its own calculations without a common (or global) clock. This feature is very important in practice due to the fact a precise common clock for all the nodes is very expensive to maintain.

Although Algorithm 1 is written as a synchronous algorithm (i.e., every node uses the same time index $k$ and updates at the same time), this is purely for convenience. The implementation can be completely asynchronous without degradation to the consensus result. To allow asynchronous implementation, we modify Algorithm 1 as follows.

Replace the common iteration number $k$ with a local counter $k_i$ for each node $i$ and initialize it to zero. According to its own local clock time zero, each node $i$ transmits the initial packet $(s_{i\to j}(k_i), x_{i\to j}(k_i), k_i)$ (with $k_i = 0$) to every node $j \in \mathcal{N}_i$ and then increment $k_i$. Then, each node $i$ simply waits till it receives all the packets $(s_{j\to i}(k_i - 1), x_{j\to i}(k_i - 1), k_i - 1)$

from all the neighboring nodes $j \in \mathcal{N}_i$, then computes $(s_{i\to m}(k_i), x_{i\to m}(k_i))$ as in (2)–(6) for every neighboring node $m \in \mathcal{N}_i$ and transmits the new packet $(s_{i\to m}(k_i), x_{i\to m}(k_i), k_i)$ to node $m$, then increment $k_i$. This process is repeated until $k_i = d$.

It is easy to see that if all the nodes have a synchronized clock, the modified algorithm coincides with Algorithm 1. When the nodes are not synchronized, packets do not arrive at a node simultaneously. But each node $i$ waits till all the information of $s_{j\to i}(k_i - 1)$ and $x_{j\to i}(k_i - 1)$ from all the neighboring nodes $j$ arrives before computing $s_{i\to m}(k_i)$ and $x_{i\to m}(k_i)$. This ensures the correctness of the updating. Hence, it is clear that the modified algorithm achieves the same result when all $k_i$ reach $d$, which is guaranteed to happen when all the local clocks have a positive minimum rate.

### C. Regional Average Consensus

For a large-scale network, it is often important and sufficient for each node to compute the average consensus over a subnetwork around itself. This can be called *regional average consensus*. We can define the size of a region as follows: a region of size $\delta$ for node $i$, denoted by $\mathcal{G}_i^\delta$, is the subnetwork with all the nodes at most $\delta$ hops away from node $i$.

Similar to the study in the previous section, we see from (7) and (8) in Theorem 1 that Algorithm 1 can be used for this purpose by stopping it after $\delta$ iterations. This will yield $\hat{x}_i(\delta)$ which is the weighted average consensus over $\mathcal{G}_i^\delta$.

An application of the above property is that instead of using the actual diameter $d$ in Algorithm 1, one may replace it with an estimate of it, or an average diameter, etc., which will result in an approximation of $\bar{x}$ for each node.

Moreover, if the asynchronous implementation discussed earlier is used, the algorithm becomes robust against time delay and asynchronous local clocks in the sense that correct regional average consensus is guaranteed by each node $i$ stops updating when its local counter $k_i$ reaches $\delta$.

## VI. Distributed Average Consensus for Loopy Graphs

In this section, we modify the distributed average consensus algorithm, Algorithm 1, to make it suitable for direct application to loopy graphs. This is done by relaxing the strict requirement that all the nodes must reach exact consensus. More precisely, for a given undirected and connected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ as before (but cyclic), we consider the following optimization problem:

$$\min_x \sum_{i\in\mathcal{V}} w_i\|x_i - y_i\|^2 + \gamma \sum_{(i,j)\in\mathcal{E}} \|x_i - x_j\|^2 \tag{10}$$

where $w_i$ and $y_i$ are as given before, $x = \text{col}\{x_1, x_2, \ldots, x_n\}$ is the decision vector and $\gamma > 0$ is a large weighting (or penalty) parameter to be tuned. It is intuitive to see that as $\gamma \to \infty$, all $x_i$ will become the same due to the connectedness of $\mathcal{G}$, and the solution to (10) will become the solution to

$$\bar{x} = \arg\min_{x_0} \sum_{i\in\mathcal{V}} w_i\|x_0 - y_i\|^2$$

which can be easily verified to be identical to (1). The relaxation mentioned above is to solve (10) instead of (1) for a sufficiently large $\gamma$.

Before presenting a distributed algorithm for solving (10), we give its centralized optimal solution below. Denote $y = \text{col}\{y_1, y_2, \ldots, y_n\}$, $W = \text{diag}\{w_1, w_2, \ldots, w_n\}$, and define the $n \times n$ Laplacian matrix $L = [\ell_{ij}]$ as

$$\ell_{ij} = \begin{cases} |\mathcal{N}_i|, & i = j \\ -1, & (i,j) \in \mathcal{E} \\ 0, & \text{otherwise.} \end{cases}$$

*Lemma 2:* The optimal solution to (10) is given by

$$x^\star = (\gamma L + W)^{-1} W y. \tag{11}$$

*Proof:* It is easy to verify that the objective function in (10) can be rewritten as $(x-y)^T W(x-y) + x^T(\gamma L)x$. Differentiating it with respect to $x$ and setting it to zero gives

$$W(x - y) + \gamma L x = 0.$$

Solving it gives the solution (11). This solution is optimal because $L$ is symmetric and positive semi-definite, ensuring that $\gamma L + W$ is symmetric and positive definite, which in turn ensuring that the objective function is strictly convex. ■

The modified Algorithm 1 is presented in Algorithm 3. For notational convenience, we denote $f_\gamma(w) = \gamma w/(\gamma + w)$.

**Algorithm 3** Modified Algorithm for Average Consensus

**Inputs:** For each node $i$, the input data are $w_i$, $y_i$ and $\gamma$.

**Outputs:** For each node $i$ and iteration $k$, the output is the estimate $\hat{x}_i(k)$ of $x_i^\star$.

**Initialization:** For each node $i$, do: For each $j \in \mathcal{N}_i$, set $x_{i\to j}(0) = y_i$, $s_{i\to j}(0) = f_\gamma(w_i)$ and transmit them to node $j$.

**Main loop:** At iteration $k = 1, 2, \cdots$, for each node $i$, compute

$$\tilde{s}_i(k) = w_i + \sum_{j\in\mathcal{N}_i} s_{j\to i}(k-1) \tag{12}$$

$$\tilde{x}_i(k) = w_i y_i + \sum_{j\in\mathcal{N}_i} s_{j\to i}(k-1)x_{j\to i}(k-1) \tag{13}$$

$$\hat{x}_i(k) = \frac{\tilde{x}_i(k)}{\tilde{s}_i(k)}, \tag{14}$$

then for each $j \in \mathcal{N}_i$, compute

$$s_{i\to j}(k) = f_\gamma(\tilde{s}_i(k) - s_{j\to i}(k-1)) \tag{15}$$

$$x_{i\to j}(k) = \frac{\tilde{x}_i(k) - s_{j\to i}(k-1)x_{j\to i}(k-1)}{\tilde{s}_i(k) - s_{j\to i}(k-1)} \tag{16}$$

and transmit them to node $j$.

*Theorem 3:* Suppose that the graph $\mathcal{G}$ is undirected, connected, and acyclic with diameter $d$. Then, running Algorithm 3 yields that

$$\hat{x}(k) = x^\star \quad \forall k \geq d \tag{17}$$

where $\hat{x}(k) = \text{col}\{\hat{x}_1(k), \hat{x}_2(k), \ldots, \hat{x}_n(k)\}$ and $x^\star$ is in (11).

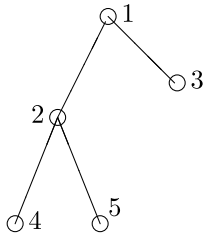



Fig. 1. Illustration graph for Example 1.

*Example 1:* To help visualize the proof, we first discuss the simple graph in Fig. 1 with $d = 2$. For this graph, we have

$$[\gamma L + W \mid Wy] = \left[\begin{array}{ccccc|c} 2\gamma + w_1 & -\gamma & -\gamma & 0 & 0 & w_1y_1 \\ \hline -\gamma & 3\gamma + w_2 & 0 & -\gamma & -\gamma & w_2y_2 \\ \hline -\gamma & 0 & \gamma + w_3 & 0 & 0 & w_3y_3 \\ \hline 0 & -\gamma & 0 & \gamma + w_4 & 0 & w_4y_4 \\ \hline 0 & -\gamma & 0 & 0 & \gamma + w_5 & w_5y_5 \end{array}\right]. \tag{18}$$

Solving $x_1$ in $(\gamma L + W)x = Wy$ can be done by applying Gauss elimination step by step on $[\gamma L+W \mid Wy]$ above. Define $g(w) = \gamma/(\gamma + w)$.

*Step 0.1 ($k = 1$):* Adding a scaled version of row 3 to row 1 with scaling value of $g(w_3)$ will eliminate the (1, 3)-entry of (18). Similarly, adding a scaled version of row 4 to row 2 with scaling value of $g(w_4)$ will eliminate the (2, 4)-entry of (18), and adding a scaled version of row 5 to row 2 with scaling value of $g(w_5)$ will eliminate the (2, 5)-entry of (18). This results in

$$\left[\begin{array}{ccccc|c} \gamma + w_1 + f_\gamma(w_3) & -\gamma & 0 & 0 & 0 & w_1y_1 + f_\gamma(w_3)y_3 \\ \hline -\gamma & \gamma + w_2 + f_\gamma(w_4) + f_\gamma(w_5) & & 0 & 0 & w_2y_2 + f_\gamma(w_4)y_4 + f_\gamma(w_5)y_5 \\ \hline -\gamma & 0 & \gamma + w_3 & 0 & 0 & w_3y_3 \\ \hline 0 & -\gamma & 0 & \gamma + w_4 & 0 & w_4y_4 \\ \hline 0 & -\gamma & 0 & 0 & \gamma + w_5 & w_5y_5 \end{array}\right]. \tag{19}$$

We see that the Gauss eliminations above effectively do the following: add $s_{3\to1}(0)$ and $s_{3\to1}(0)x_{3\to1}(0)$ to the (1, 1)-entry and (1, 6)-entry in (19), respectively; and add $s_{4\to2}(0) + s_{5\to2}(0)$ and $s_{4\to2}(0)x_{4\to2}(0) + s_{5\to2}(0)x_{5\to2}(0)$ to the (2, 2)-entry and (2, 6)-entry, respectively. Note in (19) that nodes 1 and 2 are now uncoupled from nodes 3, 4, and 5. Also, because node 3 is a leaf node, we see from (12)–(16) that $(s_{3\to1}(k), x_{3\to1}(k)) = (s_{3\to1}(0), x_{3\to1}(0))$ for all $k \geq 1$.

*Step 0.2 ($k = 2$):* Adding a scaled version of row 2 to row 1 with scaling value of $g(w_2 + f_\gamma(w_4) + f_\gamma(w_5))$ will eliminate the (1, 2)-entry of (19), giving the following equation for $x_1$:

$$\begin{aligned} &\big(w_1 + f_\gamma(w_3) + f_\gamma\big(w_2 + f_\gamma(w_4) + f_\gamma(w_5)\big)x_1 \\ &= w_1y_1 + f_\gamma(w_3)y_3 \\ &\quad + g\big(w_2 + f_\gamma(w_4) + f_\gamma(w_5)\big)\big(w_2y_2 + f_\gamma(w_4)y_4 + f_\gamma(w_5)y_5\big). \end{aligned}$$

We see from (12)–(16) that this is the same as sending $s_{2\to1}(1) = f_\gamma(w_2 + f_\gamma(w_4) + f_\gamma(w_5))$ and $x_{2\to1}(1) = (w_2y_2 + f_\gamma(w_4)y_4 + f_\gamma(w_5)y_5)/(w_2 + f_\gamma(w_4) + f_\gamma(w_5))$ to node 1 to give

$$\begin{aligned} \tilde{s}_1(2) &= w_1 + s_{3\to1}(1) + s_{2\to1}(1) \\ &= w_1 + f_\gamma(w_3) + f_\gamma\big(w_2 + f_\gamma(w_4) + f_\gamma(w_5)\big) \\ \tilde{x}_1(2) &= w_1y_1 + s_{3\to1}(1)x_{3\to1}(1) + s_{2\to1}(1)x_{2\to1}(1) \\ &= w_1y_1 + f_\gamma(w_3)y_3 + g\big(w_2 + f_\gamma(w_4) + f_\gamma(w_5)\big) \\ &\quad \times \big(w_2y_2 + f_\gamma(w_4)y_4 + f_\gamma(w_5)y_5\big). \end{aligned}$$

This confirms that $\hat{x}_1(2)$ in (14) coincides with $x_1^\star$. Similar to step 0.2, we see that $(s_{2\to1}(k), x_{2\to1}(k)) = (s_{2\to1}(1), x_{2\to1}(1))$. Therefore, $\hat{x}_1(k) = \hat{x}_1(2)$ for all $k > 2$.

Now, we proceed to prove Theorem 3.

*Proof:* We focus on solving (11) for a general acyclic graph $\mathcal{G}$ with diameter $d$, i.e., solving $(\gamma L+W)x = Wy$. We take any node and consider the tree representation of $\mathcal{G}$ with this node as the root. Because the ordering of the nodes does not affect the solution of (10), we assume, without loss of generality, that the root node is labeled as node 1. Thus, it suffices to show that $\hat{x}_1(k) = x_1^\star$ for all $k \geq d$, where $x_1^\star$ is the first element of $x^\star$.

Denote by $t$ the depth of the tree graph, i.e., the longest path from a leaf node to the root node (node 1), it is clear that $t \leq d$. Without loss of generality, assume that the nodes are listed in the following order: all the leaf nodes are at the bottom of the list; once all the leaf nodes are removed, the leaf nodes of the remaining graph are at the bottom of the remaining list; and so on. As before, we study the steps of Gauss elimination for computing $x_1^\star$.

*Step 1 ($k = 1$):* Let node $i$ be any leaf node and node $j$ be its (unique) neighboring node. Adding a scaled version of row $i$ to row $j$ with the scaling parameter of $g(w_i)$ will eliminate the $(j, i)$-entry of the matrix $[\gamma L+W|Wy]$. As verified before, this operation is equivalent to adding $s_{i\to j}(0)$ to the $(j, j)$-entry and adding $s_{i\to j}(0)x_{i\to j}(0)$ to the $(j, (n+1))$-entry of $[\gamma L+W|Wy]$. Similar to step 0.1 above, we have $s_{i\to j}(k) = s_{i\to j}(0)$ and $s_{i\to j}(k)x_{i\to j}(k) = s_{i\to j}(0)x_{i\to j}(0)$ for all $k > 0$, due to the fact that $i$ is a leaf node.

*Step 2 ($k = 2$):* Consider the reduced graph with all the leafs removed and the remaining matrix of $[\gamma L+W|Wy]$. Denote by $n_1$ the remaining number of nodes. Let node $i$ be any new leaf (which is the neighbor of an old leaf) and let $j$ be the (unique) neighbor of $i$. From step 1, we see that the $(i,i)$-entry contains a $\gamma$ term and $w_i + \sum_{v\in\mathcal{N}_i\setminus j} s_{v\to i}(k-1)$, which is the same as $\tilde{s}_i(k) - s_{j\to i}(k-1)$. Similarly, the $(i,(n_1+1))$-entry contains $w_iy_i + \sum_{v\in\mathcal{N}_i\setminus j} s_{v\to i}(k-1)x_{v\to i}(k-1)$, which is the same as $\tilde{x}_i(k) - x_{j\to i}(k-1) = (\tilde{s}_i(k) - s_{j\to i}(k-1))x_{i\to j}(k)$. We can treat $\tilde{s}_i(k) - s_{j\to i}(k-1)$ as the new $w_i$ and treat $(\tilde{s}_i(k) - s_{j\to i}(k-1))x_{i\to j}(k)$ as the new $w_iy_i$ and apply the Gauss elimination in step 1 again.

*Step $k$ ($k \geq t$):* The above process can be repeated until only the root node remains. Similar to the graph in Fig. 1, it is tedious but straightforward to verify that the resulting $x_1^\star$ is indeed given by $\hat{x}_1(t)$ in (17). It is similar to steps 0.1 and 0.2 that $\hat{x}_1(k) = \hat{x}_1(t)$ for all $k > t$. Since $t \leq d$, the above means that $\hat{x}_1(k) = \hat{x}_1(d)$ for all $k > d$. Since the root node is arbitrarily chosen, we conclude that $\hat{x}(k) = \hat{x}(d) = x^\star$ for all $k > d$. ■

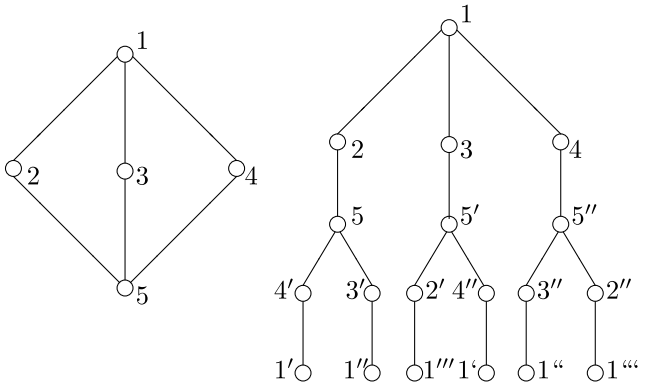



Fig. 2. Left: Loopy graph. Right: Unwrapped tree around node 1 with four layers ($t = 4$).

Next, we show that Algorithm 3 indeed applies to loopy graphs directly and still enjoys the convergence of $\hat{x}_i(k) \to x_i^\star$ as $k \to \infty$, for all $i \in \mathcal{V}$. First, we point out an obvious fact of (15) that $s_{i\to j}(k) < \gamma$ because $f_\gamma(w) = \gamma w/(\gamma + w) < \gamma$, i.e., $s_{i\to j}(k)$ is uniformly bounded.

To study the convergence of Algorithm 3 for loopy graphs, we construct an *unwrapped tree* (also known as *computation tree*) with *depth* $t > 0$ for a loopy graph $\mathcal{G}$ [29]. Take an arbitrary node, say node 1, to be the root and then iterate the following procedure $t$ times:

1) find all leaves of the tree (start with the root);
2) for each leaf, find all the nodes in the loopy graph that neighbor this leaf node, except its parent node in the tree, and add all these node as the children to this leaf node.

The variables and weights for each node in the unwrapped tree are copied from the corresponding nodes in the loopy graph. Taking a different node as root node will generate a different unwrapped tree. Fig. 2 shows the unwrapped tree rooted at node 1. Note that the nodes $1', 1,'' 1','' 1$‘, 1“, 1“‘ all carry the same values $y_1$ and $w_1$.

List the nodes in the unwrapped tree in *breadth first* order, by starting from the root node, followed by the first layer, then the second layer, etc. Denote the unwrapped quantities using $\breve{}$ and the unwrapped tree as $\breve{\mathcal{G}}_t$. Then, the unwrapped quantities and the original ones are related through a matrix $O$ [29]

$$\breve{y} = Oy; \quad \breve{w} = Ow. \tag{20}$$

Each row of $O$ contains only 1 and the rest are all 0.

Consider the optimization problem (10) for $\breve{\mathcal{G}}_t$ (with unwrapped quantities). By Lemma 2, its solution is given by

$$\breve{x}^\star = (\gamma\breve{L} + \breve{W})^{-1}\breve{W}\breve{y} \tag{21}$$

where $\breve{L}$ is the Laplacian matrix for $\breve{\mathcal{G}}_t$.

We introduce two *key* lemmas. Lemma 3 shows how the solution (21) for $\breve{\mathcal{G}}_t$ is related to the execution of Algorithm 3 on $\mathcal{G}$. Lemma 4, derived by following [29], reveals the relationships between the quantities in $\mathcal{G}$ and those in $\breve{\mathcal{G}}_t$.

*Lemma 3:* The optimal quantity $\breve{x}_1^\star$ [the first element of (21)] for $\breve{\mathcal{G}}_t$ coincides with $\hat{x}_1(t)$ [the result of (14) for node 1 after running Algorithm 3 on $\mathcal{G}$ for $t$ iterations], i.e.,

$$\hat{x}_1(t) = \breve{x}_1^\star. \tag{22}$$

*Proof:* We analyze the execution of Algorithm 3 on $\mathcal{G}$ for node 1 only. Similar to the proof of Theorem 3, after one iteration, the computed $\hat{x}_1(1)$ is effectively the optimal solution to (10) with node 1 and its neighboring nodes. Similarly, for a general $t > 1$, the computed $\hat{x}_1(t)$ by Algorithm 3 on $\mathcal{G}$ is effectively the optimal solution to (10) with node 1 and all the nodes within $t$ hops away from it, and if any node is visited multiple times, an unwrapped version of $(y_i, w_i)$ is used. Hence, the computed $\hat{x}_1(t)$ on $\mathcal{G}$ coincides with the optimal solution (21) for $\breve{\mathcal{G}}_t$. ■

*Lemma 4:* The following equations hold for the depth-$t$ unwrapped graph $\breve{\mathcal{G}}_t$ of $\mathcal{G}$:

$$OW = \breve{W}O \tag{23}$$

$$OL = \breve{L}O + E \tag{24}$$

$$\breve{x}_1^\star = x_1^\star + e_1^T(\gamma\breve{L} + \breve{W})^{-1}Ex^\star. \tag{25}$$

In the above, $e_1$ is the column vector with 1 in the first element and zero everywhere else, and $E$ is an error matrix with zero in all rows except for the last $L_t$ rows, where $L_t$ is the number of its depth-$t$ leaf nodes in $\breve{\mathcal{G}}_t$.

*Proof:* Using $W = \text{diag}\{w_i\}$ and $\breve{W} = \text{diag}\{\breve{w}_i\}$, it is easy to check that $OW = \breve{w} = \breve{W}O$, hence (23) holds. The matrix equation (24) is directly verified for all the rows except that the last rows of $\breve{L}O$ correspond to the leaf nodes, which miss their connected nodes in the original graph, as pointed out in [29], which results in the error matrix $E$ with zero in all rows except for the last $L_t$ rows. It remains to check (25). We have from (11)

$$(\gamma L + W)x^\star = Wy \tag{26}$$

$$(\gamma\breve{L} + \breve{W})\breve{x}^\star = \breve{W}\breve{y}. \tag{27}$$

Multiplying $O$ to the left of (26) and applying (23) and (24) yields

$$\begin{aligned}
&O(\gamma L + W)x^\star = OWy \\
&=> \ (\gamma\breve{L} + \breve{W})Ox^\star + \gamma Ex^\star = \breve{W}Oy \\
&=> \ (\gamma\breve{L} + \breve{W})Ox^\star = \breve{W}\breve{y} - \gamma Ex^\star \\
&=> \ Ox^\star = (\gamma\breve{L} + \breve{W})^{-1}(\breve{W}\breve{y} - \gamma Ex^\star) \\
&= \breve{x}^\star - (\gamma\breve{L} + \breve{W})^{-1}\gamma Ex^\star \\
&=> e_1^T Ox^\star = e_1^T\breve{x}^\star - e_1^T(\gamma\breve{L} + \breve{W})^{-1}\gamma Ex^\star.
\end{aligned}$$

Note that the last equation above is the same as (25) because $e_1^T Ox^\star = x_1^\star$ and $e_1^T\breve{x}^\star = \breve{x}_1^\star$. ■

Define the error

$$z_1^\star = \breve{x}_1^\star - x_1^\star \tag{28}$$

and denote $r = \gamma\breve{W}^{-1}Ex^\star$. Note that $\breve{W}$ is diagonal, hence only the last $L$ elements of $r$ are nonzero, corresponding to the depth-$t$ leaf nodes. Then, (25) can be rewritten as

$$z_1^\star = e_1^T(\gamma\breve{L} + \breve{W})^{-1}\breve{W}r. \tag{29}$$

Next, we give the crucial result that $z_1^\star \to 0$ exponentially fast as the depth $t \to \infty$.

*Lemma 5:* The following convergence property holds:

$$\|z_1^\star\| \le \eta^{t-1} r_{\max} \tag{30}$$

where $0 < \eta < 1$ is given by

$$\eta = \max_{i\in\mathcal{V}} \frac{(|\mathcal{N}_i|-1)\gamma}{w_i + (|\mathcal{N}_i|-1)\gamma}. \tag{31}$$

*Proof:* From (29), we understand that as far as node 1 is concerned, $z_1^\star$ corresponds to the optimal solution to (10) for $\breve{\mathcal{G}}_t$ with variables $\breve{y} = r$. In particular, the variables are zero for all the nodes in $\breve{\mathcal{G}}_t$ except for the depth-$t$ leaf nodes. Moreover, from $r = \gamma \breve{W}^{-1} E x^\star$, we see that these variables are bounded with $\|r_i\| \le r_{\max}$ for all $i \in \breve{\mathcal{V}}$ with some constant $r_{\max}$ (not growing as the depth $t$ increases). In fact, $x^\star$ only depends on $\mathcal{G}$, $\breve{W}^{-1}$ contains diagonal $w_i^{-1}$ only, and the nonzero elements of $E$ are also bounded due to (24). Notice that $E = OL - \breve{L}O$, the elements in $L$ and $\breve{L}$ depend only on $\mathcal{G}$, and $O$ contains only 0 and 1.

Because $\breve{\mathcal{G}}_t$ is a tree graph, from Theorem 3, running Algorithm 3 on $\breve{\mathcal{G}}_t$ with the variables $\breve{y} = r$ for $t$ iterations will give $\hat{\breve{x}}_1(t) = z_1^\star$. To avoid overusing of notation, we will drop $\breve{}$ in the rest of the proof. That is, we apply Algorithm 3 on a tree graph $\mathcal{G}_t$ with depth $t$ with variables $y_i$ being zero everywhere except for the depth-$t$ leaf nodes for which $\|y_i\| \le r_{\max}$. Note from (15) and $f_\gamma(w) = \gamma w/(\gamma+w) \le \gamma$, we know $s_{i\to j}(k) \le \gamma$ for all $i, j, k$.

We start from any depth-$t$ leaf node $i$. Denote its connecting depth-$(t-1)$ node as node $j$. Then, $\|x_{i\to j}(0)\| \le r_{\max}$.

Next, consider any depth-$(t-1)$ node $i$ and its depth-$(t-2)$ connecting node $j$. There are two cases. *Case 1: $i$* is not connected to any depth-$t$ node. In this case, $x_{i\to j}(1) = 0$ because the components of $r$ are nonzero only for the depth-$t$ nodes. *Case 2: $i$* is connected to some depth-$t$ nodes. From (16), we get

$$\begin{aligned}\|x_{i\to j}(1)\| &= \frac{\|w_i y_i + \sum_{v\in\mathcal{N}_i\setminus j} s_{v\to i}(0) x_{v\to i}(0)\|}{w_i + \sum_{v\in\mathcal{N}_i\setminus j} s_{v\to i}(0)} \\ &\le \frac{\sum_{v\in\mathcal{N}_i\setminus j} s_{v\to i}(0)}{w_i + \sum_{v\in\mathcal{N}_i\setminus j} s_{v\to i}(0)} r_{\max} \le \eta r_{\max} \end{aligned} \tag{32}$$

where $\eta$ is in (31). In both cases, we have $\|x_{i\to j}(1)\| \le \eta r_{\max}$.

Continue the above to depth-$(t-2)$ nodes and so on until reaching depth-1 nodes. We obtain, for any node $i$ connected to the root node, $\|x_{i\to 1}(t-1)\| \le \eta^{t-1} r_{\max}$.

Using (12)–(14) on the root node yields

$$\begin{aligned}\|\hat{x}_1(t)\| &= \frac{\|w_1 y_1 + \sum_{j\in\mathcal{N}_1} s_{j\to 1}(t-1) x_{j\to 1}(t-1)\|}{w_1 + \sum_{j\in\mathcal{N}_1} s_{j\to 1}(t-1)} \\ &\le \eta^{t-1} r_{\max}.\end{aligned}$$

Finally, we invoke Theorem 3 on the unwrapped tree $\breve{\mathcal{G}}_t$ to confirm that the above $\hat{x}_1(t)$ is indeed the same as the $z_1^\star$ in (29). The proof is complete. ∎

We are finally ready to present our main result on the convergence of Algorithm 3.

*Theorem 4:* Suppose the graph $\mathcal{G}$ is undirected and connected. Then, running Algorithm 3 for $k \ge 1$ iterations yields that

$$\|\hat{x}_i(k) - x_i^\star\| \le \eta^{k-1} r_{\max} \quad \forall\ i \in \mathcal{V},\ k = 0, 1, 2, \dots \tag{33}$$

where $0 < \eta < 1$ is given by (31) and $r_{\max}$ is a constant independent of $k$.

*Proof:* Taking any node $i \in \mathcal{V}$ as the root node and from the unwrapped tree $\breve{\mathcal{G}}_k$ around node $i$. Using Lemma 5 by treating node $i$ as node 1, we get that $\|z_i^\star\| = \|\breve{x}_i^\star - x_i^\star\| \le \eta^{k-1} r_{\max}$. Invoking Theorem 3 on the tree $\breve{\mathcal{G}}_k$, we get $\breve{x}_i^\star = \hat{x}_i(k)$ for the original graph $\mathcal{G}$, thus (33) holds. ∎

*Remark 3:* Theorem 4 shows that $\hat{x}(k)$ in (14) in Algorithm 3 converges to the optimal solution $x^\star$ at the decay rate of at least $\eta$. In the first glance, because $\eta \to 1$ as $\gamma \to \infty$ and that $\gamma$ needs to be sufficiently large to ensure that (10) approximates (1) well, one may expect the decay rate (convergence rate) to be very slow (i.e., $\eta$ is close to 1). But we note that the decay rate of $\eta$ is actually quite loose, due to the use of several over bounds in the proof of Lemma 5 for getting a simple decay rate expression. In particular, the bounding steps of (32) for $\eta$ are very loose due to the fact that many $x_{v\to i}(0)$ are actually zero for many nodes $v$ because they are not connected to the depth-$t$ nodes. The actual decay rate is typically much faster, as we will show in Section VII-C.

## VII. Illustrative Examples

In this section, we give three examples, one for illustrating the distributed DFS algorithm, one for demonstrating the distributed average consensus algorithm (Algorithm 1), and one for demonstrating the modified distributed average consensus algorithm (Algorithm 3).

### A. Distributed DFS Algorithm

Consider the graph $\mathcal{G}$ in Fig. 3(a) with 13 distinct labeled nodes. We run Algorithm 2. Applying the max-consensus algorithm on $\mathcal{G}$ (taking the negated labels) yields node 1 as the root node. The token is transmitted from node 1 to nodes 2–4. In iteration 1, node 2 sends the token to nodes 3, 5, and 6 and node 3 sends the token to nodes 2 and 7. In iteration 3, nodes 2 and 3 both mark edge (2, 3) as "removal"; node 5 sends the token to node 8; node 6 sends the token to nodes 7 and 9; and node 7 sends the token to nodes 6 and 9. In iteration 3, nodes 6 and 7 mark edge (6, 7) as "removal"; node 8 sends the token to nodes 9–11; node 9 marks edge (7, 9) as "removal" and sends the token to nodes 8, 12, and 13. In iteration 4, nodes 8 and 9 mark edge (8, 9) as "removal"; node 10 sends the token to node 11; node 11 sends the token to nodes 10 and 12; and node 12 sends the token to node 11. In iteration 5, edges (10, 11) and (11, 12) are marked as "removal." The obtained spanning tree is shown in Fig. 3(b), and the diameter $d = 6$.

In contrast, the standard DFS algorithm can result in a very unbalanced spanning tree, depending on the traversing sequence. Fig. 3(c) shows a possible outcome, with $d = 10$.

### B. Distributed Average Consensus Using Algorithms 1 and 2

Consider the example as in Fig. 3(b) with $y_i = i$ and $w_i = 1$. We have $\sum_{i=1}^{13} y_i = 91$ and $\bar{x} = 91/13$. Simulation shows that after $d = 6$ iterations, all the $\tilde{x}_i(k)$ converge to 91 as expected. On average, each node requires about $3 \sim 4$ flops (additions

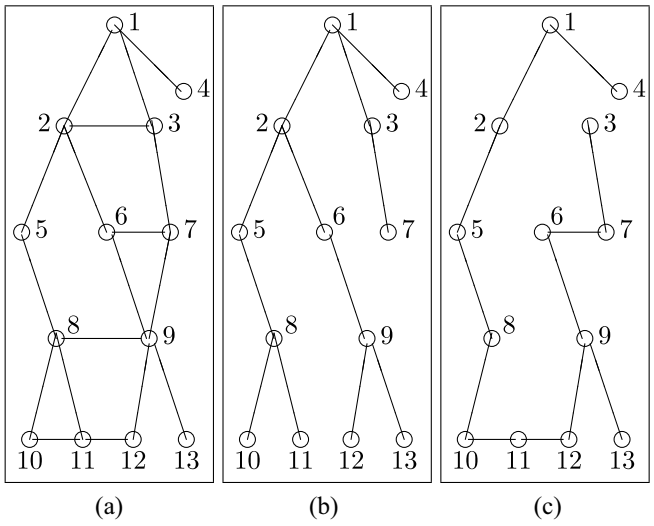



Fig. 3. Demonstration of distributed DFS algorithm. (a) Original graph. (b) Distributed DFS. (c) Traditional DFS.

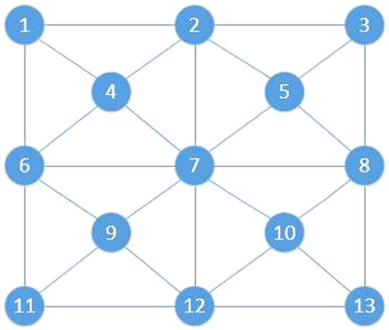



Fig. 4. 13-node loopy graph.

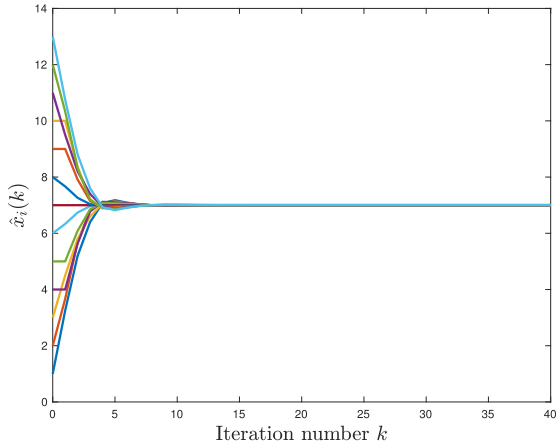



Fig. 5. Convergence of the proposed Algorithm 3.

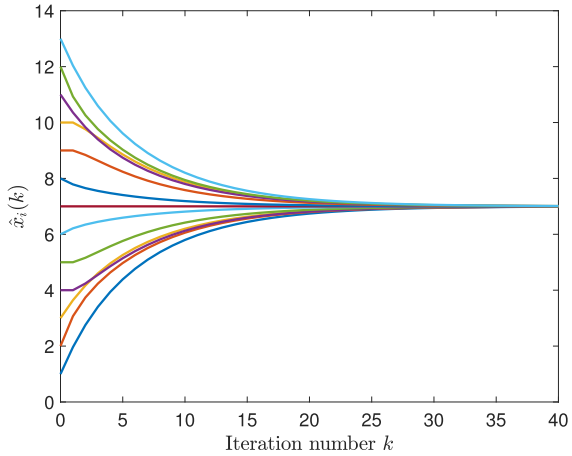



Fig. 6. Convergence of the Laplacian-based algorithm [1].

and multiplications) per iteration. Many $x_{i\to j}(k)$ converge very early.

We now suppose that: 1) one unit of delay occurs for the transmission of $(x_{1\to 2}(3), s_{1\to 2}(3))$ and 2) one packet loss occurs for $(x_{3\to 7}(4), s_{3\to 7}(4))$.

It turns out that $(x_{1\to 2}(k)$ [thus $(s_{1\to 2}(k)]$ converges after $k \geq 2$, so the delay has no effect at all. But $(x_{3\to 7}(k), s_{3\to 7}(k))$ converges after $k \geq 5$. Hence, the packet loss at $k = 4$ will mean that this information is not available at $k = 4$. However, at $k = 5$, $(x_{3\to 7}(k), s_{3\to 7}(k))$ will be available.

### C. Distributed Average Consensus for Loopy Graphs

To demonstrate the performance of Algorithm 3 on loopy graphs, we apply it to the 13-node graph in Fig. 4, with $w_i = 1$, $y_i = i$, $i = 1, 2, \ldots, 13$, and $\gamma = 1000$. The weighted average consensus (1) is verified to be $\bar{x} = 7$. The result of Algorithm 3 is shown in Fig. 5. The convergence rate is checked to be much faster than $\eta$ in (31) which is very close to 1.

For comparison purposes, we first consider the Laplacian matrix-based average consensus algorithm in [1]

$$\hat{x}(k+1) = \hat{x}(k) - \alpha L\hat{x}(k), \quad \hat{x}(0) = y \tag{34}$$

with the optimal choice of $\alpha = 1/\lambda_{\max}(L)$ (inverse of the maximum eigenvalue of $L$) for fastest convergence [1]. The convergence of $\hat{x}(k)$ to $\bar{x} = 7$ is shown in Fig. 6. We can see clearly from Figs. 5 and 6 that Algorithm 3 convergences much faster than the Laplacian matrix-based algorithm.

We also consider the distributed algorithm in [36] for solving linear equations. This algorithm is generalized from a Laplacian matrix-based consensus approach by applying orthogonal projection. To apply the algorithm in [36], we rewrite (11) as a linear equation

$$(\gamma L + W)x^\star = Wy. \tag{35}$$

Defining $\mathbf{x}_i(k)$ as the estimate of $x^\star$ by node $i$ at iteration $k$, the distributed algorithm in [36] deploys the following distributed iteration:

$$\mathbf{x}_i(k+1) = \mathbf{x}_i(k) - \frac{1}{|\mathcal{N}_i|}P_i\left(|\mathcal{N}_i|\mathbf{x}_i(k) - \sum_{j\in\mathcal{N}_i}\mathbf{x}_j(k)\right) \tag{36}$$

with the initial condition of $\mathbf{x}_i(0)$ to be any solution for the $i$th row of (35). In particular, we take $\mathbf{x}_i(0) = w_i y_i/(\gamma \ell_{ii} + w_i)e_i$ with $e_i$ being the vector with $i$th component equal to 1 and all other components equal to 0. The result is plotted in Fig. 7 for $\gamma = 10$. We see that the distributed algorithm in [36] is not well suited for solving the approximated average consensus problem because it converges too slowly.

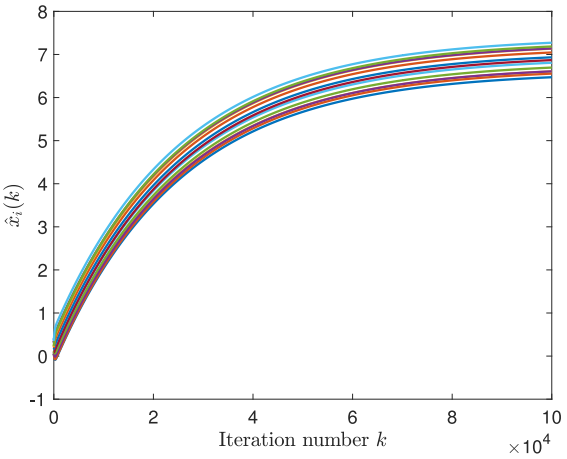



Fig. 7. Convergence of the distributed algorithm for linear equations [36].

## VIII. Conclusion

A novel distributed average consensus algorithm has been proposed in this paper. For acyclic graphs, our algorithm (Algorithm 1) is shown to converge in only $d$ iterations, where $d$ is the graph diameter. Several nice properties of the algorithm have been presented, including low complexities, robustness to transmission adversaries, and asynchronous implementability. The modified distributed average consensus algorithm (Algorithm 3) can be applied directly to loopy graphs with exponential convergence to a solution that closely approximates the average consensus. Also proposed in this paper is a distributed algorithm for spanning tree (Algorithm 2). This result is of interest on its own, but it allows the proposed average consensus algorithm to be applied to loopy graphs. While both methods (Algorithms 1 and 2 together or Algorithm 3 alone) can be used for loopy graphs, the first method is preferred for the case when the network topology does not change because Algorithm 2 only needs to be executed once and Algorithm 1 has finite convergence property, but the second method tends to adapt well when the network topology changes over time. Note that the approximation error given by the parameter $\gamma$ in Algorithm 3 can be made negligible for most applications. Our proposed average consensus algorithm is conceptually different from the graph Laplacian approach. It is our hope that this may have the potential of leading to new distributed estimation and learning solutions.

## References


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

[2] T. Li, M. Fu, L. Xie, and J.-F. Zhang, "Distributed consensus with limited communication data rate," *IEEE Trans. Autom. Control*, vol. 56, no. 2, pp. 279–292, Feb. 2011.

[3] A. Kashyap, T. Baüsar, and R. Srikant, "Quantized consensus," *Automatica*, vol. 43, pp. 1192–1203, May 2007.

[4] S. Sundaram and C. N. Hadjicostis, "Finite-time distributed consensus in graphs with time-invariant topologies," in *Proc. Amer. Control Conf.*, New York, NY, USA, 2007, pp. 711–716.

[5] Y. Xu, Z.-G. Wu, Y.-J. Pan, C. K. Ahn, and H. Yan, "Consensus of linear multiagent systems with input-based triggering condition," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published.

[6] Y. Chen and Y. Shi, "Distributed consensus of linear multiagent systems: Laplacian spectra-based method," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published.

[7] J. M. Hendrickx, G. Shi, and K. H. Johnson, "Finite-time consensus using stochastic matrices with positive diagonals," *IEEE Trans. Autom. Control*, vol. 60, no. 4, pp. 1070–1073, Apr. 2015.

[8] C.-K. Ko and X. Gao, "On matrix factorization and finite-time average-consensus," in *Proc. IEEE Conf. Decis. Control*, Shanghai, China, 2009, pp. 5798–5803.

[9] A. Y. Kibangou, "Finite-time average consensus based protocol for distributed estimation over AWGN channels," in *Proc. IEEE Conf. Decis. Control*, Orlando, FL, USA, 2011, pp. 5595–5600.

[10] J. M. Hendrickx, R. M. Jungers, A. Olshevsky, and G. Vankeerberghen, "Graph diameter, eigenvalues, and minimum-time consensus," *Automatica*, vol. 50, no. 2, pp. 635–640, 2014.

[11] J. Cortés, "Finite-time convergent gradient flows with applications to network consensus," *Automatica*, vol. 42, no. 11, pp. 1993–2000, 2006.

[12] Q. Hui, W. M. Haddad, and S. P. Bhat, "Finite-time semistability and consensus for nonlinear dynamical networks," *IEEE Trans. Autom. Control*, vol. 53, no. 8, pp. 1887–1900, Sep. 2008.

[13] L. Wang and F. Xiao, "Finite-time consensus problems for networks of dynamic agents," *IEEE Trans. Autom. Control*, vol. 55, no. 4, pp. 950–955, Apr. 2010.

[14] L. Georgopoulos and M. Hasler, "Nonlinear average consensus," in *Proc. Int. Symp. Nonlin. Theory Appl.*, 2009, pp. 10–14.

[15] L. Georgopoulos and M. Hasler, "Early consensus in complex networks under variable graph topology," in *Proc. Eur. Conf. Circuit Theory Design*, Antalya, Turkey, 2009, pp. 575–578.

[16] L. Georgopoulos, "Definitive consensus for distributed data inference," M.S. thesis, School Comput. Commun. Sci., École polytechnique fédérale de Lausanne, Lausanne, Switzerland, 2011.

[17] B. M. Nejad, S. A. Attia, and J. Raisch, "Max-consensus in a max-plus algebraic setting: The case of fixed communication topologies," in *Proc. XXII Int. Symp. Inf. Commun. Autom. Technol.*, 2009, pp. 1–7.

[18] J. Zhou, Y. Zhu, Z. You, and E. Song, "An efficient algorithm for optimal linear estimation fusion in distributed multisensor systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 36, no. 5, pp. 1000–1009, Sep. 2006.

[19] H. Li, Q. Lü, X. Liao, and T. Huang, "Accelerated convergence algorithm for distributed constrained optimization under time-varying general directed graphs," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published.

[20] D. Wang, J. Zhou, Z. Wang, and W. Wang, "Random gradient-free optimization for multiagent systems with communication noises under a time-varying weight balanced digraph," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published.

[21] E. Camponogara and L. B. de Oliveira, "Distributed optimization for model predictive control of linear-dynamic networks," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 39, no. 6, pp. 1331–1338, Nov. 2009.

[22] S. Yang, Q. Liu, and J. Wang, "Distributed optimization based on a multiagent system in the presence of communication delays," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 5, pp. 717–728, May 2017.

[23] C. L. P. Chen and Z. Liu, "Broad learning system: An effective and efficient incremental learning system without the need for deep architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 10–24, Jan. 2018.

[24] Z. Liu and C. L. P. Chen, "Broad learning system: Structural extensions on single-layer and multi-layer neural networks," in *Proc. Int. Conf. Security Pattern Anal. Cyber.*, Dec. 2017, pp. 15–17.

[25] M. Buchanan, *Nexus: Small Worlds and the Groundbreaking Theory of Networks*. New York, NY, USA: Norton, 2003.

[26] J. P. Onnela *et al.*, "Structure and tie strengths in mobile communication networks," *Proc. Nat. Acad. Sci. USA*, vol. 104, no. 18, pp. 7332–7336, 2007.

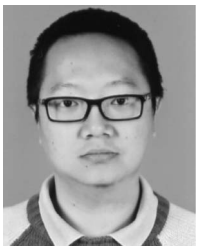
[27] K. Choromanski, M. Matuszak, and J. MieKisz, "Scale-free graph with preferential attachment and evolving internal vertex structure," *J. Stat. Phys.*, vol. 151, no. 6, pp. 1175–1183, 2013.

[28] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*. San Mateo, CA, USA: Morgan Kaufman, 1988.

[29] Y. Weiss and W. T. Freeman, "Correctness of belief propagation in Gaussian graphical models of arbitrary topology," *Neural Comput.*, vol. 13, no. 10, pp. 2173–2200, 2001.

[30] E. P. Vargo, E. J. Bass, and R. Cogill, "Belief propagation for large-variable-domain optimization on factor graphs: An application to decentralized weather-radar coordination," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 2, pp. 460–466, Mar. 2013.

[31] M. T. Goodrich and R. Tamassia, *Algorithm Design: Foundations, Analysis, and Internet Examples*. New York, NY, USA: Wiley, 2001.
[32] S. S. Skiena, *The Algorithm Design Manual*, 2nd ed. London, U.K.: Springer, 2008.
[33] S. A. M. Makki and G. Havas, "Distributed algorithms for depth-first search," *Inf. Process. Lett.*, vol. 60, no. 1, pp. 7–12, 1996.
[34] M. Bui, F. Butelle, and C. Lavault, "A distributed algorithm for the minimum diameter spanning tree problem," *J. Parallel Distrib. Comput.*, vol. 64, pp. 571–577, Jan. 2004.
[35] B. Awerbuch, "Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems," in *Proc. 19th Annu. ACM Symp. Theory Comput.*, 1987, pp. 230–240.
[36] S. Mou, J. Liu, and A. S. Morse, "A distributed algorithm for solving a linear algebraic equation," *IEEE Trans. Autom. Control*, vol. 60, no. 11, pp. 2863–2878, Nov. 2015.



**Kan Xie** was born in Hubei, China. He received the M.S. degree in software engineering from the South China University of Technology, Guangzhou, China, in 2009. He is currently pursuing the Ph.D. degree in intelligent signal and information processing with the Guangdong University of Technology, Guangzhou.

His current research interests include machine learning, non-negative signal processing, blind signal processing, and biomedical signal.

**Qianqian Cai** received the B.S. degree in environmental engineering from the University of Shanghai for Science and Technology, Shanghai, China, in 2011, and the Ph.D. degree in engineering from the University of Newcastle, Callaghan, NSW, Australia, in 2016.

She is currently a Post-Doctoral Researcher with the School of Automation, Guangdong University of Technology, Guangdong, China. Her current research interests include water pollution control engineering, environmental monitoring, automation, and adaptive neural fuzzy inference systems.

**Zhaorong Zhang** received the bachelor of electrical engineering degree from the Nanjing University of Science and Technology, Nanjing, China, in 2016. She is currently pursuing the Ph.D. degree in networked control systems with the University of Newcastle, Callaghan, NSW, Australia.

Her current research interest includes distributed state estimation.

**Minyue Fu** (F'04) received the bachelor's degree in electrical engineering from the University of Science and Technology of China, Hefei, China, in 1982 and the M.S. and Ph.D. degrees in electrical engineering from the University of Wisconsin–Madison, Madison, WI, USA, in 1983 and 1987, respectively.

From 1983 to 1987, he held a teaching assistantship and a research assistantship with the University of Wisconsin–Madison. From 1987 to 1989, he served as an Assistant Professor with the Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI, USA. He joined the Department of Electrical and Computer Engineering, University of Newcastle, Callaghan, NSW, Australia, in 1989, where he is currently a Chair Professor of Electrical Engineering with the School of Electrical Engineering and Computing. He has been a Visiting Associate Professor with the University of Iowa, Iowa City, IA, USA; a Visiting Professor with Nanyang Technological University, Singapore; a Changiang Professor with Shandong University, Jinan, China; a Distinguished Scholar with Zhejiang University, Hangzhou, China; and a Bai-Ren Scholar with the Guangdong University of Technology, Guangzhou, China. His current research interests include networked control systems, multiagent systems, and distributed estimation and control.

Dr. Fu has been an Associate Editor for the IEEE TRANSACTIONS ON AUTOMATIC CONTROL, IEEE TRANSACTIONS ON SIGNAL PROCESSING, *Automatica*, and *Journal of Optimization and Engineering*. He is a fellow of Engineers Australia and the Chinese Association of Automation.