Distributed Weight Balancing over Digraphs

Apostolos I. Rikos, *Member, IEEE*, Themistoklis Charalambous, *Member, IEEE,* and

Christoforos N. Hadjicostis, *Senior Member, IEEE*

Abstract

A weighted digraph is balanced if, for each node, the sum of the weights of the edges outgoing from that node is equal to the sum of the weights of the edges incoming to that node. Weight-balanced digraphs play a key role in a number of applications, including cooperative control, distributed optimization, and distributed averaging. We propose distributed algorithms that operate over static topologies, for solving the weight balancing problem when the weights are either nonnegative real numbers or when they are restricted to be nonnegative integers. For the case of real weights, the proposed algorithm is shown to admit geometric convergence rate. For the case of integer weights, the proposed algorithm is shown to converge after a finite number of iterations that we explicitly bound. We also provide examples to illustrate the operation, performance, and potential advantages of the proposed algorithms.

Index Terms

Weight balancing, distributed algorithms, finite time convergence, geometric convergence, digraphs.

I. INTRODUCTION

A distributed system or network can be viewed as a set of subsystems (nodes) that can share information via connection links (edges), which form a generally directed communication topology (digraph). The digraph that describes the underlying communication topology typically proves to be of vital importance for the effectiveness of distributed strategies in performing various tasks [3]–[5]. A weighted digraph is a digraph in which each edge is associated with a real or integer value called the edge weight. A weighted digraph is *weight-balanced* or *balanced* if, for each of its nodes, the sum of the weights of the edges outgoing from the node is equal to the sum of the weights of the edges incoming to the node.

Weight-balanced digraphs find numerous applications in distributed adaptive control or synchronization in complex networks. Examples of applications where balance plays a key role include network adaptation strategies based on the use of continuous second order models [6] and distributed adaptive strategies to tune the coupling weights of a network based on local information of node dynamics [7]. Furthermore, every weight-balanced digraph is also cut-balanced [8] and hence, under the cut-balance

Apostolos I. Rikos is with the Department of Electrical and Computer Engineering at the University of Cyprus, Nicosia, Cyprus. E-mail: arikos01@ucy.ac.cy.

Themistoklis Charalambous is with the Automatic Control Lab, Electrical Engineering Department and ACCESS Linnaeus Center, Royal Institute of Technology (KTH), Stockholm, Sweden. Corresponding author's address: Osquldas väg 10, 100-44 Stockholm, Sweden. E-mail: themisc@kth.se.

Christoforos N. Hadjicostis is with the Department of Electrical and Computer Engineering at the University of Cyprus, Nicosia, Cyprus, and also with the Department of Electrical and Computer Engineering at the University of Illinois, Urbana-Champaign, IL, USA. E-mail: chadjic@ucy.ac.cy.

Parts of the results for balancing with real weights have been presented in [1]. The present version of the paper includes extensions to handle the continuous-time case, time delays and asynchronicity (not addressed in [1]). Parts of the results for balancing with integer weights appear in [2]. The present version of the paper proposes a faster variant of the algorithm for balancing with integer weights and provides a new proof for its convergence (only a sketch – for the other algorithm – was provided in [2]).

condition and without any further assumptions, it is proven that the values of nodes v_i and v_j converge to the same limit if they belong to the same connected component of the "unbounded interactions graph". Weight balancing can also be associated with the Matrix Balancing Problem in network optimization, with numerous applications, such as, predicting the distribution matrix of telephone traffic [9]. In this case, however, we have a distributed way to balance the matrix. Weight balance is also closely related to weights that form a doubly stochastic matrix, which find applications in multicomponent systems (such as sensor networks) where one is interested in distributively averaging measurements at each component. In particular, the distributed average consensus problem has received significant attention from the computer science community [10] and the control community [11] due to its applicability to diverse areas, including multi-agent systems, modeling of flocking behavior [3], cooperative control [12], and estimation and tracking [13]. Averaging in such settings follows a linear iteration, where each node repeatedly updates its value to be a linear combination of its own value and the values of its neighboring nodes weighted according to the edge weights. Asymptotic average consensus is guaranteed (i.e., the nodes asymptotically reach consensus to the average of their initial values) if the weights used in the linear iteration form a doubly stochastic matrix [14].

Balancing a weighted digraph can be accomplished via a variety of centralized algorithms. In this paper, we address the problem of designing *distributed* iterative algorithms that operate over static topologies and allow a networked system to distributively obtain a set of weights that make the underlying digraph weight-balanced. Recently, a few works have appeared dealing with the problem of balancing a strongly connected digraph, for both real- and integer-weight balancing. This paper presents two algorithms that can be used in distributed networks with directed interconnection topologies (digraphs); the first algorithm addresses balancing with nonnegative real weights and the second one addresses weight balancing with nonnegative integer weights. We discuss both in turn.

Regarding real-weight balancing, the two main works present in the literature are [15] and [16]. In [15] a distributed algorithm is proposed, but while its convergence is asymptotic, no convergence bounds are provided. On the other hand, in [16] a distributed algorithm is introduced, in which each agent is assumed to distinguish the information coming from the other agents according to the identifier of the sender. Also, a global stopping time is set at which the iterations stop to form the balancing. A fast algorithm that does not require global information and user identification has been missing. This work aims to fill this gap. Specifically, we propose a linear distributed algorithm that guarantees geometric convergence rate. The simplicity of the algorithm has allowed its extension to asynchronous operation, which is a valuable contribution given that in reality there are inevitable delays in the exchange of information) as well as its continuous-time analog, that guarantees average consensus without the need to obtain a doubly stochastic matrix.

Regarding the integer weight balancing, in [17] the authors introduce a distributed algorithm, where each node increases only the value of the outgoing edge with the smallest value leaving the other outgoing edges intact, which negatively affects the convergence speed. In addition, there is no bound on the convergence of this algorithm. Herein, we propose a faster finite-time distributed algorithm for which we obtain explicit bounds on the number of iterations required for the algorithm to complete (our bound can also be applied to the *imbalance-correcting* algorithm in [17], which established convergence in a finite number of iterations, but did not obtain an explicit bound).

Note that, the nodes can also easily obtain, in a distributed manner, weights that form a doubly stochastic matrix if this

is desirable, either in parallel when the algorithm has asymptotic convergence (e.g., [1], [15]), or after the iterations have converged in finite-time algorithms (e.g., [2], [17]) or asymptotic algorithms for which a stopping time has been set (e.g., [16]).

This paper is organized as follows. In Section II we present some basic notions and notation needed for our development, as well as the problem formulation. In Section III we present the distributed algorithm which achieves balance with real weights and is shown to admit geometric convergence rate. In Section IV we introduce the distributed algorithm which achieves balance with integer weights after a finite number of iterations. In Section V we present simulation results and comparisons. Finally, in Section VI we conclude with a brief summary and remarks about future work.

II. NOTATION AND PRELIMINARIES

We adapt some basic notions and notation from [1] and [2] that are needed for our development. The sets of real, integer and natural numbers are denoted by $\mathbb{R}, \mathbb{Z}$ and $\mathbb{N}$, respectively; the positive part of $\mathbb{R}$ or $\mathbb{Z}$ is denoted by the subscript $+$ (e.g. $\mathbb{R}_+$). The symbol $\mathbb{N}_0$ denotes the set of nonnegative integers. Vectors are denoted by small letters whereas matrices are denoted by capital letters; A^{-1} denotes the inverse of matrix A. With I we denote the identity matrix (of appropriate dimensions), whereas with $\mathbf{1}$ we denote a column vector (of appropriate dimension) whose elements are all equal to one. A matrix with nonnegative elements, called nonnegative matrix, is denoted by $A \geq 0$, and a matrix with positive elements, called positive matrix, is denoted by $A > 0$. Notation $\sigma(A)$ denotes the spectrum of matrix A, $\lambda_i(A)$ denotes the i^{th} eigenvalue of matrix A, and $\rho(A)$ denotes its spectral radius. Notation $\mathrm{diag}(x_i)$ is used to denote the diagonal matrix with elements in the finite set $\{x_1, x_2, \dots, x_i, \dots\}$ on the leading diagonal and zeros elsewhere. We also denote by $e_j^{\mathrm{T}} = [0, \dots, 0, 1_{j^{th}}, 0, \dots, 0] \in \mathbb{R}^{1\times n}$ a row vector, where the single "1" entry is at the j^{th} position.

A distributed system whose components can exchange information via (possibly directed) interconnection links, can be captured by a digraph (directed graph). A weighted digraph of order n ($n \geq 2$), is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, where $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$ is the set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} - \{(v_j, v_j) \mid v_j \in \mathcal{V}\}$ is the set of edges, and $\mathcal{W} = [w_{ji}] \in \mathbb{R}_+^{n\times n}$ is a weighted $n \times n$ adjacency matrix where w_{ji} are nonnegative values. A directed edge from node v_i to node v_j is denoted by $m_{ji} \triangleq (v_j, v_i) \in \mathcal{E}$, which denotes that node v_j can receive information from node v_i. A directed edge $m_{ji} \in \mathcal{E}$ if and only if $w_{ji} > 0$. Note that the definition of $\mathcal{G}$ excludes self-edges (though self-weights need to be added if we want to consider bistochastic digraphs [17]). The graph is undirected if and only if $m_{ji} \in \mathcal{E}$ implies $m_{ij} \in \mathcal{E}$ and $w_{ji} = w_{ij}$. Note that a digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ can be thought of as a weighted digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ by defining the weighted adjacency matrix $\mathcal{W}$ with $w_{ji} = 1$ if $m_{ji} \in \mathcal{E}$, and $w_{ji} = 0$ otherwise.

A digraph is called *strongly connected* if for each pair of vertices $v_j, v_i \in \mathcal{V}$, $v_j \neq v_i$, there exists a directed *path* from v_i to v_j i.e., we can find a sequence of vertices $v_i \equiv v_{l_0}, v_{l_1}, \dots, v_{l_t} \equiv v_j$ such that $(v_{l_{\tau+1}}, v_{l_\tau}) \in \mathcal{E}$ for $\tau = 0, 1, \dots, t-1$. All nodes that can transmit information to node v_j directly are said to be in-neighbors of node v_j and belong to the set $\mathcal{N}_j^- = \{v_i \in V \mid m_{ji} \in \mathcal{E}\}$. The cardinality of $\mathcal{N}_j^-$, is called the *in-degree* of j and is denoted by $\mathcal{D}_j^-$. The nodes that receive information from node j comprise its out-neighbors and are denoted by $\mathcal{N}_j^+ = \{v_l \in \mathcal{V} \mid m_{lj} \in \mathcal{E}\}$. The cardinality of $\mathcal{N}_j^+$ is called the *out-degree* of v_j and is denoted by $\mathcal{D}_j^+$.

Definition 1. *Given a weighted digraph $\mathcal{G}(\mathcal{V},\mathcal{E},\mathcal{W})$ of order n, the total in-weight of node v_j is denoted by $\mathcal{S}_j^-$ and is defined as $\mathcal{S}_j^- = \sum_{v_i\in\mathcal{N}_j^-} w_{ji}$, whereas the total out-weight of node v_j is denoted by $\mathcal{S}_j^+$ and is defined as $\mathcal{S}_j^+ = \sum_{v_l\in\mathcal{N}_j^+} w_{lj}$.*

Definition 2. *Given a weighted digraph $\mathcal{G}(\mathcal{V},\mathcal{E},\mathcal{W})$ of order n, the weight imbalance of node v_j is denoted as x_j and is defined by $x_j = \mathcal{S}_j^- - \mathcal{S}_j^+$.*

Definition 3. *Given a weighted digraph $\mathcal{G}(\mathcal{V},\mathcal{E},\mathcal{W})$ of order n, the total imbalance of digraph $\mathcal{G}$ is denoted as $\varepsilon = \sum_{j=1}^n |x_j|$.*

Definition 4. *A weighted digraph $\mathcal{G}(\mathcal{V},\mathcal{E},\mathcal{W})$ is called weight-balanced if its total imbalance is 0, i.e., $\varepsilon \triangleq \sum_{j=1}^n |x_j| = 0$.*

Problem formulation. Given a strongly connected digraph $\mathcal{G}(\mathcal{V},\mathcal{E})$, we want to develop distributed algorithms that allow the nodes to iteratively adjust the weights on their outgoing edges so that they eventually obtain a weight matrix $\mathcal{W} = [w_{ji}]$ that satisfies the following:

1) $w_{ji} > 0$ for each edge $(v_j, v_i) \in \mathcal{E}$;
2) $w_{ji} = 0$ if $(v_j, v_i) \notin \mathcal{E}$;
3) $\mathcal{S}_j^+ = \mathcal{S}_j^-$ for every $v_j \in \mathcal{V}$.

The algorithm is iterative, we will use k to denote the iteration. For example, $\mathcal{S}_j^+[k]$ will denote the value of the out-weight of node v_j at time instant k, $k \in \mathbb{N}_0$.

III. DISTRIBUTED REAL BALANCING

In this section we allow weights to take nonnegative real values, and develop a distributed iterative algorithm that asymptotically reaches weight balancing. The algorithm achieves weight-balance as long as the underlying digraph is strongly connected (or is a collection of strongly connected digraphs). The main difference of this approach with other approaches, such as [16], [18]–[20] is that the corresponding matrix, which is neither row nor column stochastic, is still converging to a balanced situation. The rate of convergence of the algorithm is geometric and depends exclusively on the structure of the given digraph and some constant parameters chosen by the nodes, in a way that we precisely characterize.

Each node observes (but cannot set) the weights of its incoming edges and determines the weights on its outgoing edges. It is worth pointing out that each node maintains equal weights on all of its outgoing edges, which makes implementation particularly easy in settings where broadcasting is possible. Given a strongly connected digraph $\mathcal{G}(\mathcal{V},\mathcal{E})$, the distributed algorithm has each node v_j initialize the weights of all of its outgoing edges to unity, i.e., $w_{lj}[0] = 1$, $\forall v_l \in \mathcal{N}_j^+$. Then, each node v_j enters an iterative stage where at iteration k it performs the following steps:

1) It computes its weight imbalance defined by

$$x_j[k] \triangleq \mathcal{S}_j^-[k] - \mathcal{S}_j^+[k]. \tag{1}$$

2) If $x_j[k]$ is positive (resp. negative), all the weights of its outgoing edges are increased (resp. decreased) by an equal amount and proportionally to $x_j[k]$.

We discuss why the above distributed algorithm asymptotically leads to weights that balance the digraph (and also characterize its rate of convergence) after we describe the algorithm in more detail. Note that 2) above implies that w_{lj} for $v_l \in \mathcal{N}_j^+$ will

always have the same (nonnegative) value.

Algorithm 1 Balancing with real weights

Input: A strongly connected digraph $\mathcal{G}(\mathcal{V},\mathcal{E})$ with $n=|\mathcal{V}|$ nodes and $m=|\mathcal{E}|$ edges (and no self-loops).
Initialization: Each node $v_j \in \mathcal{V}$
1) Sets $w_{lj}[0]=1$, $\forall v_l \in \mathcal{N}_j^+$.
2) Chooses a proportionality gain $\beta_j \in (0,1)$.
Iteration: For $k=0,1,2,\ldots$, each node $v_j \in \mathcal{V}$ updates the weights of each of its outgoing edges w_{lj}, $\forall v_l \in \mathcal{N}_j^+$, as

$$w_{lj}[k+1] = w_{lj}[k] + \beta_j \left(\frac{\mathcal{S}_j^-[k]}{\mathcal{D}_j^+} - w_{lj}[k] \right) \tag{2}$$

The intuition behind the proposed algorithm is that we compare $\mathcal{S}_j^-[k]$ with $\mathcal{S}_j^+[k] = \mathcal{D}_j^+ w_{lj}[k]$. If $\mathcal{S}_j^+[k] > \mathcal{S}_j^-[k]$ (resp. $\mathcal{S}_j^+[k] < \mathcal{S}_j^-[k]$), then the algorithm reduces (resp. increases) the weights on the outgoing edges according to a proportionality gain $\beta_j \in (0,1)$.

Remark 1. *If node v_j sets $\beta_j = 1$, then from Eq. (2), it is clear that the weights on its outgoing links are set to be equal and their sum at time $k+1$ matches the sum of the weights in its incoming links at time k. Note, however, that if $\beta_j = 1$, $\forall v_j \in \mathcal{V}$, then the weighted adjacency matrix P (defined below) is not necessarily primitive and hence the algorithm does not necessarily converge to weights that form a weight-balanced digraph [21].*

Proposition 1. *If digraph $\mathcal{G}(\mathcal{V},\mathcal{E})$ is strongly connected,* Algorithm 1 *asymptotically reaches a steady state weight matrix W^* that forms a weight-balanced digraph, with geometric convergence rate equal to $R_\infty(P) = -\ln \delta(P)$, where*

$$P_{ji} \triangleq \begin{cases} 1-\beta_j, & \text{if } i=j, \\ \beta_j/\mathcal{D}_j^+, & \text{if } v_i \in \mathcal{N}_j^-, \end{cases} \tag{3}$$

and $\delta(P) \triangleq \max\{|\lambda| : \lambda \in \sigma(P)), \lambda \neq 1\} \equiv \rho(P - \frac{\mathbf{v}\mathbf{u}^T}{\mathbf{u}^T\mathbf{v}})$, where $\mathbf{v}$ and $\mathbf{u}$ are the unique right and left eigenvectors of P at eigenvalue 1, respectively.

Proof: First, we observe from equation (2) that all the outgoing edges have the same weight, i.e., $w_{hj} = w_{lj}$, $\forall v_h, v_l \in \mathcal{N}_j^+$ (because they are equal at initialization and they are updated in the same fashion). Hence, from hereafter, we will denote the weight on any outgoing edge of node v_j, at iteration k, as $w_j[k]$. In order to study the system with update formula (2) for each node in the digraph, we define $w[k] = (w_1[k] \;\; w_2[k] \;\; \ldots \;\; w_n[k])^T$ with $w_j[k] = w_{lj}[k]$ $(v_l \in \mathcal{N}_j^+)$. We can write the evolution of the weights in matrix form as

$$w[k+1] = Pw[k], w[0] = w_0 \;, \tag{4}$$

where $w_0 = \mathbf{1}$. It should be clear from the above update equation that the weights remain nonnegative during the execution of the algorithm.

Matrix P can be written as $P = I - B + BD^{-1}A$, where I is the identity matrix, $B = \mathrm{diag}(\beta_j)$, $D = \mathrm{diag}(\mathcal{D}_j^+)$ and A is the adjacency matrix with $a_{ji} = 1$ if $m_{ji} \in \mathcal{E}$, and $a_{ji} = 0$ otherwise. Since $\sigma(D^{-1}A) = \sigma(AD^{-1})$, then $\rho(D^{-1}A) = \rho(AD^{-1})$. In addition, $\rho(AD^{-1}) = 1$ because matrix AD^{-1} is column stochastic. As a result, $\rho(D^{-1}A) = 1$.

Also, note that $\bar{P} \triangleq I - B + AD^{-1}B$ is column stochastic and therefore $\rho(\bar{P}) = 1$. Furthermore,

$$\rho(\bar{P}) = \rho(\bar{P}B^{-1}DD^{-1}B) = \rho(D^{-1}B\bar{P}B^{-1}D)$$
$$= \rho(I - B + BD^{-1}A) = \rho(P) = 1.$$

Since P is a nonnegative matrix, we can ask whether P is primitive,[1] i.e., whether $P^h > 0$ for some $h \geq 1$. Since the digraph is strongly connected, and for $0 < \beta_j < 1$, $\forall v_j \in \mathcal{V}$, all the main diagonal entries are positive, we easily conclude that $h \leq n-1$ [22, *Lemma 8.5.5*] and P is primitive. Hence, there is no other eigenvalue with modulus equal to the spectral radius. A sufficient condition for primitivity is that a nonnegative irreducible matrix A has at least one positive diagonal element [23, Example 8.3.3], which means that some β_j can also be set at unity (as long as at least one is strictly smaller than unity). Hence, iteration (4) has a geometric convergence rate $R_\infty(P) = -\ln \delta(P)$, where $\delta(P) \triangleq \max\{|\lambda| : \lambda \in \sigma(P)), \lambda \neq 1\}$; by the Perron-Frobenius theorem, this is equal to $\rho(P - \frac{\mathbf{v}\mathbf{u}^T}{\mathbf{u}^T\mathbf{v}})$, where $\mathbf{v}$ and $\mathbf{u}$ are the right and left eigenvectors of P at eigenvalue 1, respectively. ■

Example 1. *In this illustrative example (borrowed from [24]), we demonstrate the validity of the proposed algorithm in the network depicted in Figure 1. In our plots we are typically concerned with the total imbalance in Definition 3. If weight balance is achieved, then* $\varepsilon[k] = 0$ *and* $x_j[k] = 0$, $\forall v_j \in \mathcal{V}$.

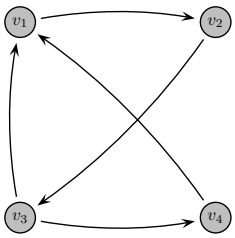

Fig. 1. A simple digraph which is weight-balanceable.

Figure 2 plots the total imbalance of Algorithm 1 *when* $\beta_j = 0.1$ *or* 0.5 *or* 0.9 *for all* $v_j \in \mathcal{V}$*, as it evolves during the execution of the algorithm. These plots agree with the claims in Proposition 1 and validate that the algorithm converges to a weight-balanced digraph with geometric convergence rate. In this particular example, with the choice of* $\beta_j = 0.1$ *or* 0.5 *or* 0.9*, the rate discussed in the proof of Proposition 1 is given by* $R_\infty\big(P(\beta_j)\big) = 0.1204, 0.5180$ *and* 0.2524*, respectively.*

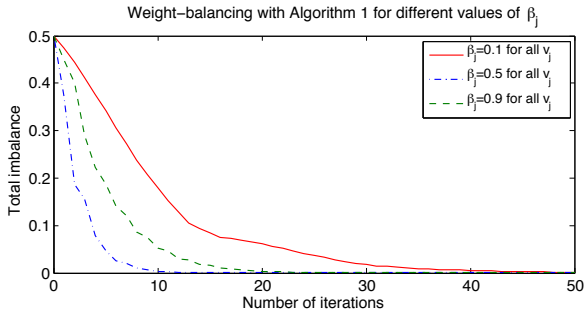

Fig. 2. Total imbalance for weight-balancing algorithm with real weights (*Algorithm 1*) for the digraph depicted in Figure 1.

[1] A nonnegative matrix is said to be primitive if it is irreducible and has only one eigenvalue of maximum modulus [22].

Remark 2. *The continuous-time analog of Equation* (2) *is given by*

$$\frac{dw_{lj}(t)}{dt} = \beta_j \left(\frac{S_j^-(t)}{\mathcal{D}_j^+} - w_{lj}(t) \right), \quad \beta_j > 0. \tag{5}$$

which can be written in matrix form as

$$\dot{w}(t) = -Qw(t), \quad w(0) = w_0 \ , \tag{6}$$

where $Q = I - P$ and $w_0 = \mathbb{1}$. Since $\rho(P) = 1$, rank$(Q) = n - 1$ and all nontrivial eigenvalues of Q have positive real parts, since all eigenvalues are located in a Geršhgorin disk in the closed right-hand plane that touches the imaginary axis at zero. As a result, consensus is asymptotically reached for all initial states. Note that, in continuous-time multi-agent systems, a weight-balanced digraph implies that average-consensus is asymptotically reached [4], [25]. In practice, the weight-balancing algorithm will be used for a finite time, long enough to establish a small error (and what is of interest in terms of performance is the associated error or deviation after this finite time).

Asynchronous Operation of *Algorithm 1*

In practice, there will always be a signaling delay associated with transmitting information regarding the weights. Consequently, a realistic analysis of the weight-balancing algorithm must consider heterogeneous time-varying delays. We postulate an asynchronous operation of *Algorithm 1*, where each agent might update the weights of its outgoing edges based on bounded heterogeneous time-varying delayed information regarding the weights on its incoming edges.

We use the integer $\tau_{ji}[k] \geq 0$ to represent the delay of a message sent from node v_i to node v_j at time instant k. We require that $0 \leq \tau_{ji}[k] \leq \bar{\tau}_{ji} \leq \bar{\tau}$ for all $k \geq 0$ for some finite $\bar{\tau} = \max\{\bar{\tau}_{ji}\}$, $\bar{\tau} \in \mathbb{Z}_+$. We make the reasonable assumption that $\tau_{jj}[k] = 0$, $\forall v_j \in \mathcal{V}$, at all time instances k (i.e., the own value of a node is always available without delay). With this notation at hand, equation (2) becomes

$$w_{lj}[k+1] = (1 - \beta_j) w_{lj}[k] + \frac{\beta_j}{\mathcal{D}_j^+} \sum_{v_i \in \mathcal{N}_j^-} w_{ji}\big[k - \tau_{ji}[k]\big], \tag{7}$$

Again, we note that all the w_{lj} have the same value, which we will denote by w_j. Let $x_{lj}[k] = w_{lj}[k]/w_j^*$, where w_j^* is the j-th element of the positive vector w^* that satisfies $\lim_{k\to\infty} P^k w[0] = w^*$ for the matrix defined in (3). Hence, equation (7) can be written as

$$x_{lj}[k+1] = (1 - \beta_j) x_{lj}[k] + \frac{\beta_j}{\mathcal{D}_j^+} \sum_{v_i \in \mathcal{N}_j^-} \frac{x_{ji}\big[k - \tau_{ji}[k]\big] w_i^*}{w_j^*}. \tag{8}$$

By letting

$$\alpha_{ji} = \begin{cases} \frac{\beta_j w_i^*}{\mathcal{D}_j^+ w_j^*} \ , & \text{if } i \neq j, \\ 1 - \beta_j \ , & \text{otherwise,} \end{cases}$$

equation (8) becomes

$$x_{lj}[k+1] = \sum_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} \alpha_{ji} x_{ji}\big[k - \tau_{ji}[k]\big], \tag{9}$$

or

$$x_j[k+1] = \sum_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} \alpha_{ji} x_i\big[k - \tau_{ji}[k]\big]. \tag{10}$$

Furthermore, notice that $\alpha_{ji} \geq 0$ and

$$\sum_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} \alpha_{ji} = 1 - \beta_j + \sum_{v_i \in \mathcal{N}_j^-} \frac{\beta_j w_i^*}{\mathcal{D}_j^+ w_j^*} = 1,$$

which stems from the fact that $Pw^* = w^*$. As a result, according to [26, *Proposition 3.2, pp. 515*], and since the iteration (7) is initialized with positive values, then there exists a positive value c such that $\lim_{k\to\infty} w[k] = cw^*$ and convergence takes place at the rate of a geometric progression.

IV. DISTRIBUTED INTEGER BALANCING

In this section we present a distributed algorithm (Algorithm 2) in which the nodes iteratively adjust the positive integer weights of their outgoing edges, such that the digraph becomes weight-balanced after a finite number of iterations. As in Section III, we assume that each node observes but cannot set the weights of its incoming edges, and based on these weights it adjusts the weights on its outgoing edges. It is required that the weights on the outgoing edges of each node can be adjusted *differently* if necessary. (Note that this requirement is not present in Section III (nor in [15]) where each node sets equal weights to all of its outgoing edges; when restricting ourselves to integer weights, however, this requirement becomes necessary for balancing to be possible, see, for example, [17]).

Given a strongly connected digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Algorithm 2 has each node $v_j \in \mathcal{V}$ initialize the weights of all of its outgoing edges to n (or some constant greater than unity[2]). Then, it enters an iterative stage where each node v_j performs the following steps (a formal description of Algorithm 2 appears later):

1) The node computes its weight imbalance.
2) If the node has positive imbalance, it increases the integer weights of its outgoing edges so that it becomes weight-balanced (assuming no further changes by its in-neighbors on its incoming edges). Specifically, the outgoing edges are assigned, if possible, equal integer weights; otherwise, if this is not possible, they are assigned integer weights such that the maximum difference among them is equal to unity. This means that some of the outgoing edges of each node might get larger weights (by unity) than others, and we assume that each node selects *a priori* a fixed (possibly randomly selected) ordering of its out-neighbors that determines the precedence with which outgoing edges get higher weight.[3]

[2]It will become evident from our analysis that this constant does not have to be the same for all nodes (and its value does not affect the termination of the algorithm), but for simplicity we take it to be n in this analysis. Note that if edge weights were initialized to 1, the execution of Algorithm 2 is identical to the execution of the algorithm in [2] because nodes with negative imbalance never take any action.

[3]The exact ordering is not critical and, in fact, other strategies are possible as long as they keep some balance among weights. For example, the algorithm proposed in [17], by having each node with positive imbalance increase the weight of the outgoing edge with minimum weight, also imposes some sort of balance among the weights on its outgoing edges.

3) If the node has negative imbalance, it decreases (if possible) the integer weights of its outgoing edges so that i) they have value greater or equal to 1, and ii) its weight imbalance becomes equal to -1 (assuming no further changes by its in-neighbors on its incoming edges). Specifically, the outgoing edges are assigned, if possible, equal integer weights; otherwise, if this is not possible, they are assigned integer weights (greater or equal to 1) such that the maximum difference among them is equal to unity (again, we assume each node determines which of its outgoing edges get higher weight based on some *a priori* fixed ordering of its out-neighbors).

For simplicity, we assume that during the execution of the distributed algorithm, the nodes update the weights on their outgoing edges in a synchronous manner, but it should be evident from our proof that the algorithm can also be extended to asynchronous settings where, during each iteration, a node is selected (randomly or otherwise) to update the weights on its outgoing edges based on its imbalance at that point.[4]

Algorithm 2 Balancing with integer weights

Input: A strongly connected digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges.
Initialization: Set $k = 0$; each node $v_j \in V$ sets its outgoing edge weights as

$$w_{lj}[0] = \begin{cases} 0, & \text{if } v_l \notin \mathcal{N}_j^+, \\ n, & \text{if } v_l \in \mathcal{N}_j^+. \end{cases}$$

Node v_j also orders its out-neighbors in the set $\mathcal{N}_j^+$ in some random (but fixed) order.
Iteration: For $k = 0, 1, 2, \ldots$, each node $v_j \in \mathcal{V}$ does the following:

1) It computes its weight imbalance $x_j[k] = \mathcal{S}_j^-[k] - \mathcal{S}_j^+[k]$.
2) If $x_j[k] > 0$, it sets the values of the weights on its outgoing edges as $w_{lj}[k+1] = \left\lfloor \frac{\mathcal{S}_j^-[k]}{\mathcal{D}_j^+} \right\rfloor$, $\forall v_l \in \mathcal{N}_j^+$. Then, it chooses the first $\mathcal{S}_j^-[k] - \mathcal{D}_j^+ \left\lfloor \frac{\mathcal{S}_j^-[k]}{\mathcal{D}_j^+} \right\rfloor$ of its outgoing edges (according to the ordering of its out-neighbors chosen during initialization), and increases their value by 1 so that $|w_{lj} - w_{hj}| \leq 1, \forall v_l, v_h \in \mathcal{N}_j^+$.
3) If $x_j[k] < -1$, it does the following:
   (i) If $\left\lfloor \frac{\mathcal{S}_j^-[k]}{\mathcal{D}_j^+} \right\rfloor \geq 1$, then node v_j sets the values of the weights on its outgoing edges as $w_{lj}[k+1] = \left\lfloor \frac{\mathcal{S}_j^-[k]}{\mathcal{D}_j^+} \right\rfloor$, $\forall v_l \in \mathcal{N}_j^+$. Then, it chooses the first $1 + \mathcal{S}_j^-[k] - \mathcal{D}_j^+ \left\lfloor \frac{\mathcal{S}_j^-[k]}{\mathcal{D}_j^+} \right\rfloor$ of its outgoing edges (according to the ordering of its out-neighbors chosen during initialization), and increases their weight by 1 so that $|w_{lj} - w_{hj}| \leq 1, \forall v_l, v_h \in \mathcal{N}_j^+$.
   (ii) If $\left\lfloor \frac{\mathcal{S}_j^-[k]}{\mathcal{D}_j^+} \right\rfloor = 0$, then node v_j sets the values of the weights on its outgoing edges as $w_{lj}[k+1] = 1$.

Example 2. Consider the digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in Figure 3, where $\mathcal{V} = \{v_1, v_2, \ldots, v_7\}$, $\mathcal{E} = \{e_1, e_2, \ldots, e_{13}\}$, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The weight on each edge is initialized to $w_{lj}[0] = 7$ for $(v_l, v_j) \in \mathcal{E}$ (otherwise $w_{lj}[0] = 0$).

As a first step, each node computes its weight imbalance $x_j[0] = \mathcal{S}_j^-[0] - \mathcal{S}_j^+[0]$ (the corresponding imbalances are shown in Figure 3). Algorithm 2 requires each node with positive imbalance to increase the value of the weights on its outgoing edges by equal integer amounts (or with maximum difference between them equal to unity), so that the total increase makes the node balanced. This ensures that weights remain strictly positive and $\mathcal{S}_j^+[k+1] = \mathcal{S}_j^-[k]$. In particular, the balance of node v_j will become zero, unless the weights of its incoming edges are changed by its in-neighbors. In this case, the nodes that have positive imbalance (equal to 14 and 7, respectively) are nodes 5 and 6, which distribute their imbalance to their outgoing edges

[4]A key requirement in that case is that, as long as the graph is not balanced, no node is completely excluded from selection.

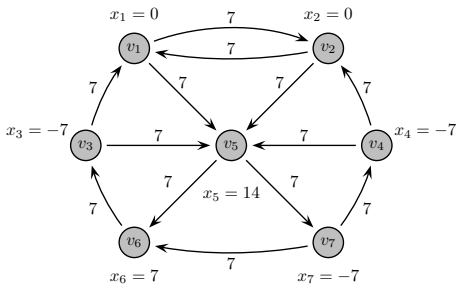

Fig. 3. Weighted digraph with initial weights and initial imbalance for each node.

as shown in Figure 4. For example, node 5 has imbalance $x_5[0] = 14$ and sets $w_{65}[1] = 7 + 7 = 14$ and $w_{75}[1] = 7 + 7 = 14$.

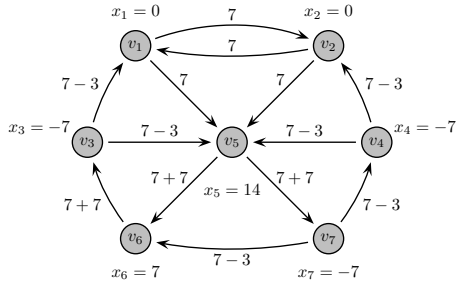

Fig. 4. Distribution of weights to outgoing edges by nodes with positive imbalance.

The distributed algorithm also requires each node with negative imbalance to decrease the value of the weights on its outgoing edges by equal integer amounts (or with maximum difference between them equal to unity), so that i) the weights are at least 1 and ii) the total decrease makes the nodes imbalance equal to -1. This ensures that weights remain strictly positive and $\mathcal{S}_j^+[k+1] = \mathcal{S}_j^-[k] + 1$. The balance of node v_j remains negative at -1, unless the weights of its incoming edges are changed by its in-neighbors. In this case, the nodes that have negative imbalance (all equal to -7) are nodes 3, 4 and 7, which distribute their imbalance to their outgoing edges as shown in Figure 4. For example, node 7 has imbalance $x_7[0] = -7$ and sets $w_{67}[1] = 7 - 3 = 4$ and $w_{47}[1] = 7 - 3 = 4$.

In the next iteration, after the integer weight update on the outgoing edges of each node with positive (or negative) imbalance at $k = 0$, the nodes recalculate their imbalance as $x_j[1] = \mathcal{S}_j^-[1] - \mathcal{S}_j^+[1]$, and the process is repeated. After a finite number of iterations, which we explicitly bound in the next section, we reach the weighted digraph with the integer weights shown in Figure 5. In this example, this occurs after 17 iterations. □

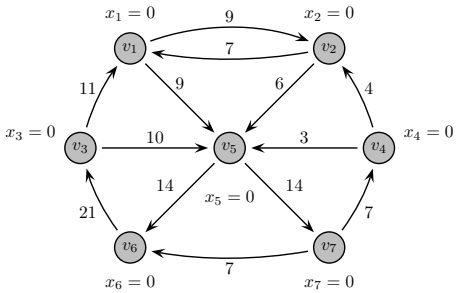

Fig. 5. Weight-balanced digraph after 17 iterations.

Remark 3. *Note that Algorithm 2 resembles Algorithm 1 in many ways: when a node has positive imbalance, it increases the weights on its outgoing edges; whereas when it has negative imbalance, it decreases the weights on its outgoing edges. The*

restriction to integers, however, creates some "anomalies" that need to be addressed. In particular, one has to worry about the following cases:

(i) When the imbalance is positive but not divisible by the out-degree of the node, it is not possible to increase the weights of the outgoing edges by equal integer amounts. In order to keep the weights on its outgoing edges integer-valued, the node makes the needed adjustments while allowing the weights to differ by at most 1.

(ii) The same applies when the imbalance is negative (in which case the weights have to decrease). An obvious additional constraint in this case is the fact that the weights have to be kept strictly positive.

(iii) An interesting feature of Algorithm 2 is that when the imbalance of a node is negative, the node adjusts the weights on its outgoing edges so that it achieves an imbalance of -1 *(not zero) assuming no changes in their incoming weights. The reasons for this choice become clear in the proof of termination later on, but effectively this choice resembles the requirement we have in Section III for at least one of the* β_j*'s to be smaller than unity (see Remark 1).*

Remark 4. *It is worth pointing out that Algorithm 2 is similar to the weight balancing algorithm presented in [2], but allows nodes to adjust the weights of their outgoing edges regardless of whether they have positive or negative imbalance; this means that Algorithm 2 will typically converge faster than the algorithm presented in [2] (though in the worst case they will take a similar number of iterations to converge).*

Analysis of Execution Time of Algorithm 2

Herein, we calculate an explicit bound on the number of iterations required, *in the worst-case*, for the given digraph to become balanced. Our bound is $O(n^3m^2)$ iterations and, since $m < n^2$, we can also bound the number of iterations as $O(n^7)$. Note that the bound we obtain also applies for the imbalance-correcting algorithm in [17] (note that [17] established convergence in finite time for the imbalance-correcting algorithm, but did not provide a bound on the number of iterations). Thus, the importance of the result in this section is that the number of iterations (for both our proposed algorithm and for the imbalance-correcting algorithm in [17]) is polynomial in the size of the digraph and not exponential.

Let the total imbalance of the digraph at iteration k be

$$\varepsilon[k] = \sum_{j=1}^{n} |x_j[k]| \ ,$$

where $x_j[k] = \mathcal{S}_j^-[k] - \mathcal{S}_j^+[k]$ with $\mathcal{S}_j^-[k] = \sum_{v_i \in \mathcal{N}_j^-} w_{ji}[k]$ and $\mathcal{S}_j^+[k] = \sum_{v_l \in \mathcal{N}_j^+} w_{lj}[k]$. Clearly, the total imbalance is a nonnegative quantity that is zero if and only if the digraph is balanced. Also, since we have $\sum_j x_j[k] = 0$ (because each weight $w_{ji}[k]$ appears twice, once with a positive sign and once with a negative sign), we see that: (i) the total imbalance of the digraph at any given k is an even number, and (ii) if there is a node with positive imbalance, then there is also (at least one) node with negative imbalance. For convenience, in the remainder of this section, we will sometimes refer to nodes with positive (negative) imbalance as *positive* (*negative*) nodes, and to nodes with zero imbalance as *balanced* nodes.

Proposition 2. *During the execution of Algorithm 2, we have*

$$\varepsilon[k+1] \leq \varepsilon[k] \leq \varepsilon[0] \leq n^2(n-2) \ .$$

Proof: Since the digraph is assumed to be strongly connected, each node has at least one incoming edge and at least one outgoing edge. Also, each node has at most $n-1$ incoming edges and at most $n-1$ outgoing edges. Since at initialization all edges have weight n, we have $|x_j[0]| \leq n(n-2)$, which means that $\varepsilon[0] \leq n^2(n-2)$.

To gain some insight, assume (for now) that at iteration k, only node v_j changes the weights on its outgoing edges (with the other nodes *not* making any changes regardless of their imbalance). We have the following three cases to consider:
(i) *Case 1: Node v_j has positive imbalance $x_j[k]$.* In such case, node v_j uniformly[5] increases the weights on its outgoing edges in such a way so that $x_j[k+1] = 0$ (because $\mathcal{S}_j^+[k+1] = \mathcal{S}_j^-[k]$ and $\mathcal{S}_j^-[k+1] = \mathcal{S}_j^-[k]$ —since no other node updates the weights on its outgoing edges). In order to see how the total imbalance changes, we look at

$$\begin{aligned}
&\varepsilon[k+1] - \varepsilon[k] = \textstyle\sum_{l=1}^n |x_l[k+1]| - |x_l[k]| \\
&\overset{(a)}{=} -x_j[k] + \textstyle\sum_{v_l \in \mathcal{N}_j^+} |w_{lj}[k+1] - w_{lj}[k] + x_l[k]| - |x_l[k]| \\
&\overset{(b)}{\leq} -x_j[k] + \textstyle\sum_{v_l \in \mathcal{N}_j^+} |w_{lj}[k+1] - w_{lj}[k]| \\
&\overset{(c)}{=} -x_j[k] + \textstyle\sum_{v_l \in \mathcal{N}_j^+} (w_{lj}[k+1] - w_{lj}[k]) = 0 \, .
\end{aligned}$$

(a) follows because $x_j[k+1] = 0$, $x_j[k]$ is positive, and only nodes in $\mathcal{N}_j^+$ see changes in their weights from incoming edges; in particular, node v_l sees its incoming weight from node v_j increase by $w_{lj}[k+1] - w_{lj}[k]$. (b) stems from the triangle inequality, and (c) from Step 2 of Algorithm 2 and the fact that $w_{lj}[k+1] \geq w_{lj}[k]$ for $v_l \in \mathcal{N}_j^+$. Note that equality holds in Case 1 if *all* nodes in $\mathcal{N}_j^+$ have positive or zero balance.

(ii) *Case 2: Node v_j with negative imbalance $x_j[k] \leq -2$.* In this case, node v_j decreases the weights on its outgoing edges so that $x_j[k+1] = -a_j$ for some integer $a_j \geq 1$. The aim is for a_j to be unity (in Step 3.1 of Algorithm 2) but this may not be possible as the weights on the outgoing edges of v_j are constrained to remain above unity (in Step 3.2 of Algorithm 2). Using similar arguments as in the previous case (but keeping in mind the difference in the signs of the various quantities), we have

$$\begin{aligned}
&\varepsilon[k+1] - \varepsilon[k] = a_j - |x_j[k]| + \textstyle\sum_{l \neq j} |x_l[k+1]| - |x_l[k]| \\
&= a_j + x_j[k] + \textstyle\sum_{v_l \in \mathcal{N}_j^+} |w_{lj}[k+1] - w_{lj}[k] + x_l[k]| - |x_l[k]| \\
&\leq a_j + x_j[k] + \textstyle\sum_{v_l \in \mathcal{N}_j^+} |w_{lj}[k+1] - w_{lj}[k]| \\
&\overset{(d)}{=} a_j + x_j[k] - \textstyle\sum_{v_l \in \mathcal{N}_j^+} (w_{lj}[k+1] - w_{lj}[k]) = 0 \, ,
\end{aligned}$$

where (d) follows from the fact that the total decrease in the weights satisfies

$$\begin{aligned}
\textstyle\sum_{v_l \in \mathcal{N}_j^+} (w_{lj}[k+1] - w_{lj}[k]) &= \mathcal{S}_j^+[k+1] - \mathcal{S}_j^+[k] \\
&= \mathcal{S}_j^-[k] + a_j - \mathcal{S}_j^+[k] \\
&= a_j + x_j[k] \, .
\end{aligned}$$

In Case 2, equality holds if all nodes in $\mathcal{N}_j^+$ have negative or zero balance.

(iii) *Case 3: Node v_j with $x_j[k] = -1$ or $x_j[k] = 0$.* In this case, node v_j does not do anything so we easily conclude that $\varepsilon[k+1] - \varepsilon[k] = 0$.

[5]It is possible that some weights are not increased, but, due to the fixed ordering of the out-neighbors of node v_j (with which weights are increased by one over the value of other weights), we have $w_{lj}[k+1] \geq w_{lj}[k]$ for all $v_l \in \mathcal{N}_j^+$.

Clearly, the arguments in the above three cases establish that $\varepsilon[k+1] \leq \varepsilon[k]$ in an asynchronous setting where, at each iteration k, a single node v_j is selected (randomly or otherwise) to update the weights on its outgoing edges based on its imbalance $x_j[k]$ at that point. In fact, with a little bit of book-keeping, we can extend the above argument in the synchronous setting of Algorithm 2, where nodes adjust the weights on their outgoing edges simultaneously. The key difference is that now the weights on the incoming edges of each node v_j may change: $\mathcal{S}_j^-[k+1]$ is no longer identical to $\mathcal{S}_j^-[k]$ and has to be explicitly accounted for. Letting $\Delta w_{ji} = w_{ji}[k+1] - w_{ji}[k]$, we have

$$\begin{aligned}
&\varepsilon[k+1] - \varepsilon[k] = \textstyle\sum_{j=1}^n |x_j[k+1]| - |x_j[k]| \\
&= \textstyle\sum_{v_j \in \mathcal{V}} |\mathcal{S}_j^-[k+1] - \mathcal{S}_j^+[k+1]| - |x_j[k]| \\
&= \textstyle\sum_{v_j \in \mathcal{V}} |\sum_{v_i \in \mathcal{N}_j^-} \Delta w_{ji} - \sum_{v_l \in \mathcal{N}_j^+} \Delta w_{lj} + x_j[k]| - |x_j[k]|.
\end{aligned}$$

Consider now the following partition of $\mathcal{V}$: set $\mathcal{A} = \{v_j \mid x_j[k] > 0\}$, set $\mathcal{B} = \{v_j \,|\, x_j[k] \leq -2\}$, and set $\mathcal{C} = \{v_j \mid x_j[k] = -1 \text{ or } x_j[k] = 0\}$. Using these partitions,

$$\begin{aligned}
\varepsilon[k+1] - \varepsilon[k] \overset{(e)}{=} &\textstyle\sum_{v_j \in \mathcal{A}} |\sum_{v_i \in \mathcal{N}_j^-} \Delta w_{ji}| - x_j[k] \\
&\overset{(f)}{+} \textstyle\sum_{v_j \in \mathcal{B}} |\sum_{v_i \in \mathcal{N}_j^-} \Delta w_{ji} - a_j| + x_j[k] \\
&\overset{(g)}{+} \textstyle\sum_{v_j \in \mathcal{C}} |\sum_{v_i \in \mathcal{N}_j^-} \Delta w_{ji} + x_j[k]| + x_j[k] \;,
\end{aligned}$$

where (e) follows from Case 1, (f) follows from Case 2, and (g) from the fact that $x_j[k] \leq 0$ for $v_j \in \mathcal{C}$. Using the triangle inequality on each line we have

$$\begin{aligned}
&\varepsilon[k+1] - \varepsilon[k] \\
&\leq \textstyle\sum_{v_j \in \mathcal{A}} \sum_{v_i \in \mathcal{N}_j^-} |\Delta w_{ji}| - x_j[k] \\
&\;+ \textstyle\sum_{v_j \in \mathcal{B}} \sum_{v_i \in \mathcal{N}_j^-} |\Delta w_{ji}| + a_j + x_j[k] \\
&\;+ \textstyle\sum_{v_j \in \mathcal{C}} \sum_{v_i \in \mathcal{N}_j^-} |\Delta w_{ji}| \\
&= \textstyle\sum_{v_j \in \mathcal{A}} \sum_{v_l \in \mathcal{N}_j^+} |\Delta w_{lj}| - x_j[k] \\
&\;+ \textstyle\sum_{v_j \in \mathcal{B}} \sum_{v_l \in \mathcal{N}_j^+} |\Delta w_{lj}| + a_j + x_j[k] \\
&\;+ \textstyle\sum_{v_j \in \mathcal{C}} \sum_{v_l \in \mathcal{N}_j^+} |\Delta w_{lj}| = 0 \;,
\end{aligned}$$

where the key was to re-arrange the summation of the $|\Delta w_{ji}|$ and to take advantage of the inequalities we proved earlier, in Cases 1, 2 and 3. ■

Note that Proposition 2 essentially gives us a way to analyze the execution time of the proposed algorithm (and also the algorithm in [17]). The basic idea is to bound the number of steps K it takes for $\varepsilon[k+K]$ to become strictly smaller than $\varepsilon[k]$. As argued in Proposition 3 below, we have $K \leq m^2$, where m is the number of edges of the digraph; this implies that the execution time of the algorithm can be bounded by

$$\text{Execution Time} \leq m^2 \frac{\varepsilon[0]}{2} \leq [n(n-1)]^2 \frac{n^2(n-2)}{2} \leq n^7 \;.$$

[Note that the number of edges m in a digraph satisfies $m \leq n(n-1)$, the initial total imbalance satisfies $\varepsilon[0] \leq n^2(n-2)$, and each decrease that takes place is by at least two (since $\varepsilon[k]$ is always an even number).]

Proposition 3. *During the execution of* Algorithm 2*, we have*

$$\varepsilon[k+K] < \varepsilon[k], \quad k = 0, 1, 2, ...$$

when $\varepsilon[k] > 0$ *and* $K > m^2$ *(*$m = |\mathcal{E}|$ *is the number of edges in the given digraph* $\mathcal{G}$*).*

Before providing the proof of Proposition 3, we discuss an example of a "difficult" digraph in order to provide intuition about the problem.

Example 3. Consider the digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in Figure 6, where $\mathcal{V} = \{v_1, v_2, \ldots, v_8\}$, $\mathcal{E} = \{e_1, e_2, \ldots, e_{15}\}$, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Edges $\{e_1, e_2, \ldots, e_{15}\}$ are not denoted in the figure to avoid cluttering the diagram. The weight on each edge is $w_{lj}[0] = 1$ for $(v_l, v_j) \in \mathcal{E}$ (otherwise, $w_{lj}[0] = 0$). $\mathcal{G}$ involves of 4 cycles C_1, C_2, C_3, C_4, which comprise of the following edges:

C_1 : $< (v_2, v_1), (v_3, v_2), (v_1, v_3) >$,

C_2 : $< (v_4, v_2), (v_5, v_4), (v_3, v_5), (v_2, v_3) >$,

C_3 : $< (v_6, v_4), (v_7, v_6), (v_5, v_7), (v_4, v_5) >$,

C_4 : $< (v_8, v_6), (v_7, v_8), (v_6, v_7) >$.

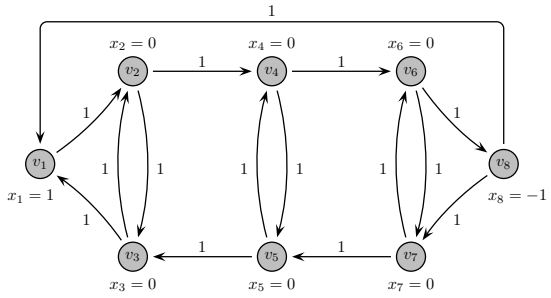

Fig. 6. Weighted digraph with initial weights and initial imbalances.

Initially, each node computes its weight imbalance $x_j[0] = \mathcal{S}_j^-[0] - \mathcal{S}_j^+[0]$ (this initial imbalance is indicated in Figure 6). The only node that will take action is node v_1, which has positive imbalance equal to 1 (all other nodes have imbalance zero or -1). Note that node v_1 will increase the weight on edge (v_2, v_1), causing an imbalance to node v_2 who will then be forced to take action. Depending on whether v_2 increases the weight on edge (v_3, v_2) or (v_4, v_2), node v_3 or node v_4 will be forced to take action. Thus, there are different possibilities (executions of the algorithm) that depend on the fixed ordering of out-neighbors, and below we consider a particular such execution. [Note that at each iteration, only one node takes action because all other nodes have imbalance zero or -1: initially node v_1, then node v_2, etc.; thus, we use the term "transferring of the imbalance" to indicate the fact that the imbalanced node forces an out-neighbor to be imbalanced.]

- In the first 3 iterations, the imbalance gets transferred to nodes v_2, v_3 and then back to node v_1. Note that the choices we made here were for node v_2 to transfer the imbalance to node v_3, and for node v_3 to transfer the imbalance back to node v_1.
- In the next 7 iterations, the imbalance gets transferred to nodes v_2, v_4, v_5, v_3, v_2, v_3 and back to v_1. Again, at each iteration, only one node has positive imbalance: first node v_1, then node v_2, then v_4, then v_5, then v_3, then v_2, then v_3, and finally v_1. Note that, given the previous choices, the only choice we had here was whether at node v_4 we increase

the weight at edge (v_6, v_4) or edge (v_5, v_4); we assumed the latter.

- In the next 11 iterations, the imbalance gets transferred to nodes v_2, v_4, v_6, v_7, v_5, v_4, v_5, v_3, v_2, v_3, and back to v_1, respectively.
- In the last 4 iterations, the imbalance gets transferred to nodes v_2, v_4, v_6, and v_8 respectively.
- The resulting balanced digraph is shown in Figure 7 and is reached after 25 iterations.

Summarizing, we have that cycle C_1 was crossed four times, cycle C_2 was crossed three times, cycle C_3 was crossed two times, and cycle C_4 was crossed one time. As a result, the number of iterations required, in order for digraph $\mathcal{G}$ to reach weight balance, is $4|C_1| + 3|C_2| + 2|C_3| + |C_4|$, where $|C_i|$ is the length of cycle C_i.

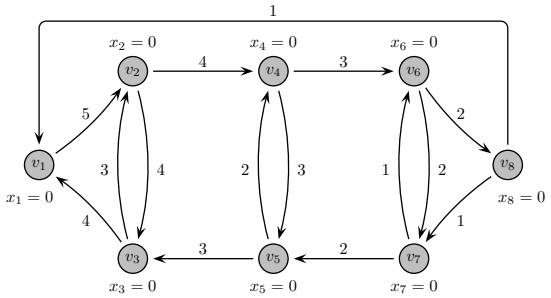

Fig. 7. Weight-balanced digraph after 25 iterations.

We are now ready to proceed with the proof of Proposition 3.

Proof: Since $\varepsilon[k] > 0$, we have at least one node with positive imbalance, say node v_1, and at least one node with negative imbalance, say node v_n.

Note that, at each iteration of Algorithm 2, node v_n (in fact, any node with negative imbalance) will retain its negative imbalance unless at least one of its in-neighbors v_{n_i}, $(v_n, v_{n_i}) \in \mathcal{E}$, has *positive* imbalance and increases the weight on the edge (v_n, v_{n_i}). The reason is that the changes that v_n initiates on the weights on its outgoing edges can only make its imbalance -1; thus, for the imbalance to become zero or positive, it has to be that one or more of its incoming weights are increased. This can only happen if one or more of its in-neighbors have positive imbalance, at which point it follows from the proof of Proposition 2 (strict inequality for Case 1) that the total imbalance will decrease (by at least two).

In order to determine a bound on the number of steps K required for the total imbalance to decrease, we can assume without loss of generality that negative nodes remain negative (because at the moment any negative node becomes balanced or positive, we also have a decrease in the total imbalance). Consider the (worst[6]) case where v_1 has imbalance b for some positive integer b, v_n has imbalance $-b$, and the remaining nodes v_2, v_3, ..., v_{n-1} are all balanced (refer to Figure 8, where v_1 is the node on the far left and v_n is the node on the far right). At the first iteration, node v_1 sends its imbalance at least one of its out-neighbors (by increasing the weight on at least one of its outgoing edges). This out-neighbor (resp. these out-neighbors) of node v_1 does (resp. do) the same at the next iteration, and this process is repeated. If, at any point, node v_n is reached, the overall imbalance will decrease by at least two (i.e., $\varepsilon[k+1] \leq \varepsilon[k] - 2$).

Let us now analyze the number of iterations it takes for node v_n to be reached (i.e., for one of its in-neighbors to become

[6]It will become evident that this is the worst case scenario at the completion of the argument.

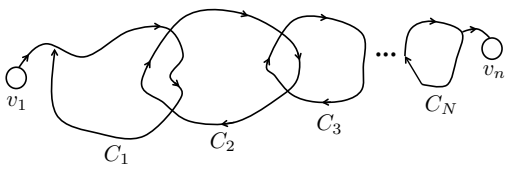

Fig. 8. Transfer of positive imbalance from node v_1 on the left to node v_n on the right.

positive). Observe the following:

(i) At each iteration, the imbalance gets transferred from a node to one or more of that nodes's out-neighbors. If more than one out-neighbors are involved, then the decrease will occur faster; thus, we consider the case when the imbalance gets transferred to only one out-neighbor.

(ii) As the iterations proceed, mark the first time an edge is traversed for the second time, and call the cyclic sequence of edges visited up to this point C_1 (refer to Figure 8). Note that C_1 is a cyclic sequence of edges (not nodes, i.e., a certain node may be visited more than once while traversing cycle C_1). Also note that C_1 has at most $n(n-1)$ edges because that is the total number of edges of the digraph.

(iii) While traversing C_1 for the second time, we will be forced at some point to traverse a new edge (that has not been traversed before (otherwise, the digraph is not strongly connected). The reason is the fact that the weights on the out-going edges from each node cannot differ by more than unity (this is where the notion of approximate balance among the weights on the out-going edges plays are role). Let C_2 denote the set of edges traversed until the time we stop traversing new edges (i.e., we are forced to traverse an edge that we have already visited). Let C_2 be the set of edges that are not in C_1 and have been visited so far.

(iv) We can continue in this fashion (defining C_i as shown in Figure 8) until we reach node v_n. Note that the number of cycles N satisfies $N \leq m$ (where $m = |\mathcal{E}|$ is the number of edges of the given digraph $\mathcal{G}$) because each cycle has at least one edge and the digraph has a total of $n(n-1)$ cycles.

From the above discussion (see also Example 3), we have

$$\#(\text{iterations to reach } v_n) \leq \sum_{i=1}^{N}(N-i+1)|C_i| \leq Nm \ ,$$

since $\sum_{i=1}^{N}|C_i| \leq m$. Finally, each cycle C_i has at least one edge, which means we can have at most m cycles. This allows us to conclude that #(iterations to reach v_n) $\leq m^2$, which completes the proof of the proposition. ■

Remark 5. *The bound we obtain is actually $m^2\frac{\varepsilon[0]}{2}$ and can easily be improved (if one looks at the last equation in Section IV and realizes that it is in our best interest to have cycle 1 have size $|C_1| = m-(N-1)$ and all other cycles have size equal to 1) to $m^2\frac{\varepsilon[0]}{2}$. In terms of the example we provide, this bound suggests $m^2/2$ iterations which is 113. Nevertheless, the main motivation in obtaining our bound was to establish that the number of iterations is bounded by a number that is polynomial (and not exponential) in the size of the graph.*

V. SIMULATIONS AND COMPARISONS

We compare the proposed algorithms with the current state-of-the-art. Specifically, we run Algorithm 1 and Algorithm 2 in large digraphs (of size $n = 50$ and 200) and compare their performance against four other algorithms: (a) the weight balancing algorithm in [2] in which each node v_j with positive imbalance $x_j > 0$, increases the weights of its outgoing edges by equal integer amounts (if possible) so that it becomes weight-balanced, (b) the imbalance-correcting algorithm in [21] in which every node v_j with positive imbalance $x_j > 0$ adds all of its weight imbalance x_j to the outgoing node with the lowest weight w, (c) the weight-balancing algorithm presented in [16] which relies on the decentralized estimation of the left eigenvector associated to the zero structural eigenvalue of the Laplacian matrix and (d) the weight balancing algorithm in [15] in which each node v_j with positive imbalance $x_j > 0$, increases the weights of its outgoing edges by an equal amount so that it becomes weight-balanced.

Figure 9 shows the cases of: (i) 1000 averaged digraphs of 50 nodes each, where every edge is initialized to 1, (ii) 1000 averaged digraphs of 50 nodes each, where every edge is initialized to 1, and (iii) 1000 averaged digraphs of 50 nodes each, where every edge is initialized to 50, respectively. The first figure shows that Algorithm 1 outperforms the algorithms presented in [15] and [16]. The second figure shows that Algorithm 2 converges as fast as the one in [2] (as expected due to the particular initialization used). The last figure shows that when the edge's initialization is greater than 1 then Algorithm 2 is the fastest among algorithms [2] and [21].

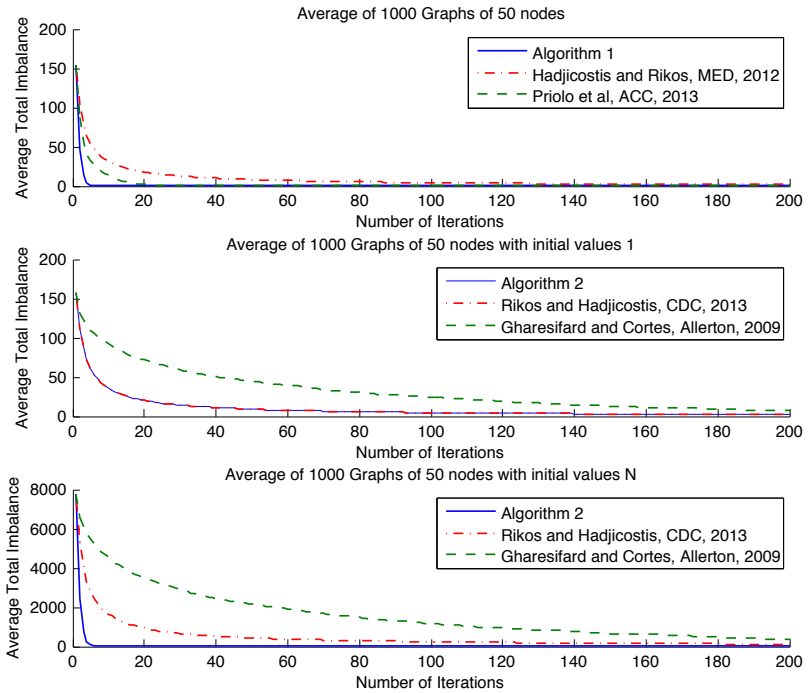

Fig. 9. Comparison between Algorithm 1, Algorithm 2, the weight-balancing algorithm proposed in [2], the imbalance-correcting algorithm [21], the weight-balancing algorithm presented in [16], and the weight-balancing algorithm proposed in [15]. *Top figure:* Average total imbalance plotted against the number of iterations for 1000 random digraphs of 50 nodes. *Middle figure:* Average total imbalance plotted against the number of iterations for 1000 random digraphs of 50 nodes. *Bottom figure:* Average total imbalance plotted against the number of iterations for 1000 random digraphs of 50 nodes.

Figure 10 shows the same cases as Figure 9, with the difference that the network consists of 200 nodes. The performances do not change due to the network size and the conclusions are the same as in Figure 9.

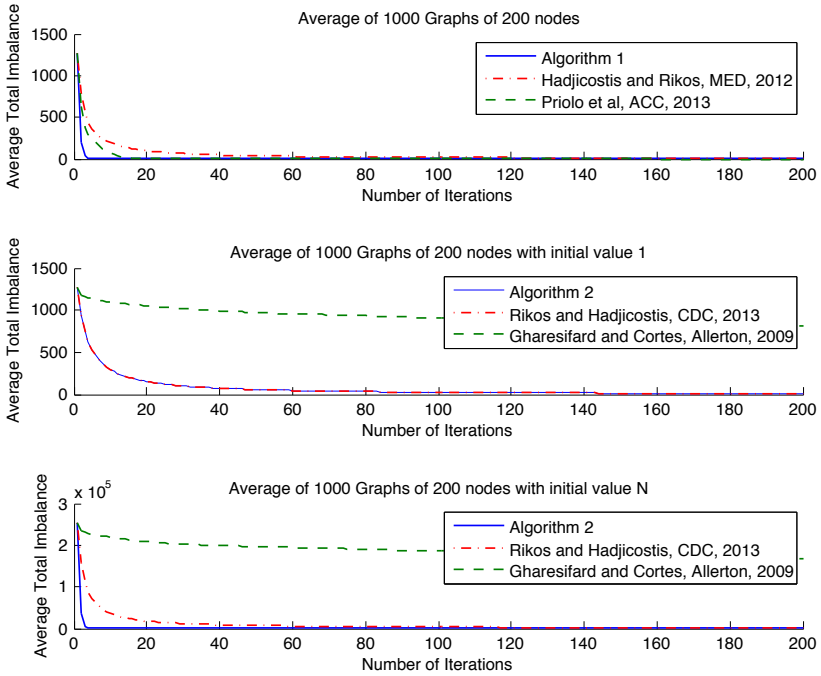

Fig. 10. Comparison between Algorithm 1, Algorithm 2, the weight-balancing algorithm proposed in [2], the imbalance-correcting algorithm [21], the weight-balancing algorithm presented in [16], and the weight-balancing algorithm proposed in [15]. *Top figure:* Average total imbalance plotted against the number of iterations for 1000 random digraphs of 200 nodes. *Middle figure:* Average total imbalance plotted against the number of iterations for 1000 random digraphs of 200 nodes. *Bottom figure:* Average total imbalance plotted against the number of iterations for 1000 random digraphs of 200 nodes.

VI. CONCLUSIONS

In this work, we have considered the weight balancing problem for a distributed system that forms a digraph where its nodes can exchange information iteratively in a distributed fashion. Weight-balanced digraphs play a key role in a number of applications, such as cooperative control, distributed optimization and distributed averaging. We proposed two distributed algorithms for solving the weight balancing problem when the weights are either (i) nonnegative real numbers or (ii) nonnegative integers. For the case of real weights, the proposed algorithm is shown to admit geometric convergence rate. For the case of integer weights, the proposed algorithm is shown to converge after a finite number of iterations that we explicitly bound. Illustrative examples show the operation, performance, and advantages of the proposed algorithms with respect to the current state-of-the-art.

In the future, we plan to apply these techniques to quantized average consensus and balancing problems, including possible applications to distributed power control in wireless transmitters with quantized power levels (e.g., [27]). We also plan to consider applications to flow problems in internet-like networks with constraints in capacity or usage of different flows (see, for example, [28] and references therein).

REFERENCES

[1] T. Charalambous and C. N. Hadjicostis, "Distributed formation of balanced and bistochastic weighted adjacency matrices in digraphs," *Proceedings of the European Control Conference (ECC)*, pp. 1752–1757, July 2013.

[2] A. I. Rikos and C. N. Hadjicostis, "Distributed balancing of a digraph with integer weights," *Proceedings of the IEEE Conference on Decision and Control*, pp. 1983–1988, December 2013.

[3] A. Jadbabaie, J. Lin, and A. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 988–1001, June 2003.

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

[5] W. Ren and R. W. Beard, "Distributed consensus in multi-vehicle cooperative control: Theory and applications," *Communication and Control Engineering Series*, 2007.

[6] P. DeLellis, M. di Bernardo, F. Garofalo, and M. Porfiri, "Evolution of complex networks via edge snapping," *IEEE Transactions on Circuits and Systems*, vol. 57, no. 8, pp. 2132–2143, August 2010.

[7] W. Yu, P. DeLellis, G. Chen, M. di Bernardo, and J. Kurths, "Distributed adaptive control of synchronization in complex networks," *IEEE Transactions on Automatic Control*, vol. 57, no. 8, pp. 2153–2158, Aug. 2012.

[8] J. M. Hendrickx and J. N. Tsitsiklis, "Convergence of type-symmetric and cut-balanced consensus seeking systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 1, pp. 214–218, 2013.

[9] D. P. Bertsekas, *Network Optimization: Continuous and Discrete Models*. Belmont, Massachusetts: Athena Scientic, 1998.

[10] N. Lynch, *Distributed Algorithms*. San Mateo: CA: Morgan Kaufmann Publishers, 1996.

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

[12] R. Olfati-Saber, "Distributed Kalman filtering for sensor networks," *Proceedings of 46th IEEE Conference on Decision and Control*, pp. 5492–5498, December 2007.

[13] R. Carli, A. Chiuso, L. Schenato, and S. Zampieri, "Distributed Kalman filtering based on consensus strategies," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 4, pp. 622–633, May 2008.

[14] J. Tsitsiklis, "Problems in decentralized decision making and computation," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, Cambridge, 1984.

[15] C. N. Hadjicostis and A. I. Rikos, "Distributed strategies for balancing a weighted digraph," *Proceedings of* 20^{th} *Mediterranean Conference on Control Automation (MED)*, pp. 1141–1146, July 2012.

[16] A. Priolo, A. Gasparri, E. Montijano, and C. Sagues, "A decentralized algorithm for balancing a strongly connected weighted digraph," *Proceedings of the American Control Conference (ACC)*, pp. 6547–6552, June 2013.

[17] B. Gharesifard and J. Cortés, "Distributed strategies for generating weight-balanced and doubly stochastic digraphs," *European Journal of Control*, vol. 18, no. 6, pp. 539–557, 2012.

[18] A. Olshevsky and J. N. Tsitsiklis, "Convergence speed in distributed consensus and averaging," *SIAM Journal on Control and Optimization*, vol. 48, no. 1, pp. 33–55, February 2009.

[19] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science.*, pp. 482–491, October 2003.

[20] A. D. Domínguez-García and C. N. Hadjicostis, "Distributed matrix scaling and application to average consensus in directed graphs," *IEEE Trans. on Automatic Control*, vol. 58, no. 3, pp. 667–681, March 2013.

[21] B. Gharesifard and J. Cortés, "Distributed strategies for making a digraph weight-balanced," *Proceedings of the* 47^{th} *Annual Allerton Conference on Communication, Control, and Computing*, pp. 771–777, September 2009.

[22] R. A. Horn and C. R. Johnson, *Matrix* A*nalysis*. Cambridge, CB2 2RU, UK: Cambridge University Press, 1985.

[23] C. D. Meyer, *Matrix* A*nalysis and Applied Linear Algebra*. SIAM, 2000.

[24] B. Gharesifard and J. Cortés, "When does a digraph admit a doubly stochastic adjacency matrix?" *Proceedings of the American Control Conference (ACC)*, pp. 2440–2445, July 2010.

[25] R. Olfati-Saber and R. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1520–1533, September 2004.

[26] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation*. Prentice-Hall, 1989.

[27] C. Wu and D. P. Bertsekas, "Distributed power control algorithms for wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 50, no. 2, pp. 504–514, 2001.

[28] S. Shakkottai and R. Srikant, "Network optimization and control," *Journal Foundations and Trends in Networking*, vol. 2, no. 3, pp. 271–379, January 2007.