Distributed Averaging and Balancing in Network Systems
with Applications to Coordination and Control

Christoforos N. Hadjicostis
University of Cyprus
chadjic@ucy.ac.cy

Alejandro D. Domínguez-García
University of Illinois
aledan@illinois.edu

Themistokis Charalambous
Aalto University
themistoklis.charalambous@aalto.fi
# Contents

1 Introduction .......................... 2
   1.1 Background and Motivation .................. 2
   1.2 Scope of the Monograph ................. 3
   1.3 Organization ............................ 6

2 Theory .................................. 7

2 Preliminaries .......................... 8
   2.1 Mathematical Background ................. 8
   2.2 Models for Distributed Control Systems .... 14
   2.3 Distributed Algorithms .................. 15
   2.4 Further Reading ........................... 19

3 Distributed Asymptotic Average Consensus .... 20
   3.1 Introduction and Motivation ............... 20
   3.2 Preliminaries and Problem Formulation ...... 22
   3.3 Distributed Averaging in Undirected Graphs . 26
   3.4 Distributed Averaging in Digraphs ........... 31
   3.5 Overcoming Transmission Delays and Packet Drops . 46
   3.6 Privacy and Security Considerations .......... 54
   3.7 Discussion .............................. 60
4 Distributed Finite-Time Average Consensus 62
4.1 Introduction and Motivation 62
4.2 Exact Finite-Time Average Consensus 63
4.3 Approximate Finite-Time Average Consensus 80
4.4 Discussion 86

5 Distributed Weight Balancing 89
5.1 Introduction and Motivation 89
5.2 Preliminaries and Problem Formulation 90
5.3 Distributed Balancing with Unconstrained Weights 96
5.4 Distributed Balancing with Constrained Weights 103
5.5 Balancing in Undirected Graphs 120
5.6 Discussion 122

II Applications 125

6 Coordination of Distributed Energy Resources 126
6.1 Introduction and Motivation 126
6.2 Power Distribution Network Model 127
6.3 Fair-Splitting Allocation 130
6.4 Optimal Allocation 133
6.5 Flow-Feasible Allocation 139
6.6 Discussion 145

7 The PageRank Problem 147
7.1 Introduction and Motivation 147
7.2 The PageRank Problem 149
7.3 Centralized Computation of PageRank Values 150
7.4 Asynchronous Computation of PageRank Values 151
7.5 The Asynchronous PageRank Algorithm 157
7.6 Discussion 159

References 160
Distributed Averaging and Balancing in Network Systems

Christoforos N. Hadjicostis\(^1\), Alejandro D. Domínguez-García\(^2\) and Themistoklis Charalambous\(^3\)

\(^1\)University of Cyprus, Cyprus
\(^2\)University of Illinois, USA
\(^3\)Aalto University, Finland

ABSTRACT

Rapid developments in digital system and networking technologies have led to the emergence of complex systems that are \textit{de facto} managed and controlled over cyber infrastructures, such as wireless and wired broadband networks. The emergence of this type of network systems, which range from smart grids and traffic networks of various sorts, to embedded electronic devices and robotic networks, has sparked huge interest in distributed control problems. This is due to the need to properly coordinate the information exchange between sensors, actuators, and controllers in order to enforce a desirable behavior without relying on a centralized decision maker. In this monograph, we present some recent progress in this area by focusing on the key operations of distributed average consensus and weight/flow balancing under a variety of communication topologies and adversarial network conditions, e.g., delays, and packet drops. These operations are key in control, coordination, and optimization tasks in many emerging applications; two of these, which we discuss in detail, are the coordination of distributed energy resources, and the computation of PageRank values.
1

Introduction

1.1 Background and Motivation

Over the past few decades, the design of protocols and algorithms for distributed control and coordination in network systems, and more generally distributed function calculation, has attracted significant attention by the control, communication, and computer science communities (see, e.g., Bertsekas and Tsitsiklis, 1989; Lynch, 1996; Olfati-Saber et al., 2007; Bullo et al., 2009; Mesbahi and Egerstedt, 2010, and the references therein). For example, given a set of interconnected nodes (which could be sensors in a sensor network, routers in a communication network, or unmanned vehicles in a multi-agent system), the nodes may be interested in (i) averaging a set of measurements, (ii) coordinating their speed or direction, (iii) jointly regulating/协调 traffic in an urban network, (iv) transmitting data from one/multiple sources to one/multiple sinks, or (v) electing a leader with each node casting a vote.

Within the control community, one coordination problem that has been the subject of extensive work by many researchers is the so-called consensus (or agreement) problem (see, e.g., Olfati-Saber et al., 2007). In this problem, each node in the network initially possesses some value, which is in general different for every node. The objective is then for the nodes to obtain the value of some function, the argument of which is a vector comprised by the values
possessed by the nodes. In general, the nodes may not know what the function is, or they may not have access to all the entries of the argument to be passed to it.

The average consensus problem is a special case of the consensus problem described above, where the nodes’ task is to compute the average of the values they possess initially (see, e.g., Xiao and Boyd, 2004). In this case, nodes know a priori the function to be calculated (i.e., the average), but as in the consensus problem, they do not necessarily have access to all the entries of the vector-valued argument to be passed to the function. More generally, the nodes might be interested in computing a weighted average of the values they possess, where, perhaps, the weight associated to each value is private information only known to the particular node. In this case, while the nodes may know what the structure of the function is (i.e., a weighted average), they do not know the function parameters.

While there are many ways to solve averaging problems in a distributed fashion, e.g., flooding (see, e.g., Ho et al., 1999), a popular approach is to use one (or more) linear iteration, where each node repeatedly updates some variables as a weighted linear combination of the previous values these variables take and those maintained by its neighbors. Each of the iterations in these algorithms can be thought of as an autonomous discrete-time linear system with a transition matrix (possibly time-varying), also referred to as weight matrix, that is defined by the coefficients (weights) used in the linear updates. While it is true some of the iterative-type algorithms can be translated to continuous-time formulations, our focus here is on the former descriptions because they allow flexibility to overcome abnormalities that appear in practice (such as communication delays and faulty/malicious components), and are more suitable for a variety of emerging systems with hybrid dynamics and event-driven control.

1.2 Scope of the Monograph

The focus of this monograph is to address several issues that are key when attempting to utilize iterative-type algorithms for distributed averaging and balancing, i.e., ensuring that the values of certain quantities (weights/flows) associated to certain pairs of nodes satisfy desirable properties, in practical settings.
Communication Constraints

The communication constraints imposed by the distributed nature of the system need to be taken into account when designing distributed algorithms, either for averaging or for weight/flow balancing. A number of protocols have been developed to address these two problems in distributed systems in which no component may have the capability to communicate directly with all other components. Chapters 3 and 4 address distributed averaging, and Chapter 5 addresses distributed balancing, for both the cases of bidirectional and non-bidirectional exchange of information among pairs of nodes.

Imperfect Communications

The unreliability of the communication channels connecting pairs of nodes in the network is an important challenge. For example, if a communication link between two nodes fails permanently, the nodes need to be able to detect this issue and compensate for it by adapting their weights or via some other means. Also, unreliable communication links can cause certain transmissions at certain time steps to be delayed or completely dropped. For example, in wireless networks, each node should generally be able to communicate with its neighbors; however, such transmissions may become unreliable and temporarily lost, due to, for example, channel fading and interference from other sources. Acknowledgements allow senders to know whether their transmissions have been received, but this imposes additional overhead and delay, and might not be as straightforward to implement in the case of non-bidirectional exchange of information among pairs of nodes. Chapter 3 discusses these challenges in more detail, and provides algorithms for distributed averaging that overcome them.

Weight Choice

Depending on the task to be performed, the weights to be used in iteration-type algorithms need to be chosen in a particular way. In this regard, and assuming there are no constraints on the values the weights can take, if the exchange of information among the nodes in the system is bidirectional, the choice of weights is usually relatively straightforward and the nodes can do it in a distributed fashion with fairly minimal information. On the other hand, if the
1.2. Scope of the Monograph

exchange of information among the nodes is non-bidirectional, the problem of choosing the weights is much harder even if there are no constraints on the weight choice, and the computations needed for such choice are performed by a single processor with access to all the information defining the problem. The problem complicates even further if there are restrictions on the values that the weights can take. Chapters 3 and 5 address weight choice under different constraints for different types of network systems.

Precision vs Execution and Time Complexity

There are many tradeoffs in distributed algorithms for averaging, including the computational/communication complexity, the execution time of the algorithm, and the precision of the outcome. For example, flooding techniques (see, e.g., Ho et al., 1999) have the components exchange messages until each component in the system becomes aware of all values that need to be averaged; thus, they enable them to compute the average after a finite number of iterations (that depends on the size and structure of the network), at the cost of high memory requirements and communication complexity (especially if the size of the network is large). On the other hand, iterative strategies that rely on each node updating its value using information that is available from its immediate neighbors, do not impose high memory requirements, but they require an indefinite number of iterations; they typically converge only asymptotically, implying a compromise in precision if they are aborted after a finite number of iterations. Chapter 3 focuses on asymptotic strategies for average consensus, whereas Chapter 4 focuses on finite-time strategies, including strategies that guarantee an approximation of the average within an \textit{a priori} chosen precision and strategies that guarantee the exact average (at the cost of higher computational complexity).

Interplay between Cyber and Physical Layers

Most literature on distributed averaging is focused on algorithm development and the cyber layer for communication and computation on which the algorithms developed are implemented. Unless the end goal is the computation itself—this is the case in a distributed computing system—this may not be enough. In general, the nodes in the cyber layer are controlling some actua-
tors or measuring some quantities in a physical system with the objective of making the physical system behave in some particular way. While taking into account the physical layer is key in understanding the overall system behavior, the physical layer model depends heavily on the particular application, and thus it is difficult to abstract out and generalize. For example, in an electric power distribution network, the nodes in the cyber layer may be controlling the amount of power injected by power generating resources. This in turn will result in power flows across the electrical lines connecting the different nodes in the electrical distribution network; these flows are governed by the physics of the system, i.e., Kirchhoff’s laws. Under certain assumptions on the electrical network topology and the operating conditions, the flow of power in an electrical network can be described by a network-flow-theoretic model (see, e.g., Ford and Fulkerson, 2010), which is also used in transportation networks. Thus, in Chapters 5 and 6, we consider such network-flow-theoretic models and their interaction with the cyber layer controlling them.

1.3 Organization

The monograph is divided into two parts; these are Part I: Theory (Chapters 2–5), and Part II: Applications (Chapters 6–7). A reader interested mainly in the averaging problem can focus only Chapters 2, 3, and 4. A reader interested in weight choice and weight/flow balancing can focus on Chapters 2 and 5. A reader interested only in electric power applications can read Chapters 2, 3, 5, and 6. A reader interested only in the PageRank problem can read Chapters 2, 3, and 7.

We include proofs that are easy to present, do not break the flow of the document, and provide some intuition for the results; however, we omit the more complex proofs, and refer the reader to particular references.

Instead of having a “centralized” literature review in this introductory chapter, and in following with the spirit of this monograph, we adopt a “distributed” literature review approach, where each chapter contains a review of the references that are relevant to the particular chapter content.
Part I

Theory
In this chapter, we introduce the notation we will be using throughout the monograph, as well as necessary mathematical preliminaries on graph theory and nonnegative matrix theory. We also discuss our models of distributed control systems and algorithms.

2.1 Mathematical Background

2.1.1 Symbols and Notation

The sets of real, integer, and natural numbers are denoted by $\mathbb{R}$, $\mathbb{Z}$, and $\mathbb{N}$, respectively; the nonnegative 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. We use lowercase letters to denote vectors (e.g., $x$ or $x$), and uppercase letters to denote matrices (e.g., $A$ or $A$). We use $1_n$ ($0_n$) to denote the $n$-dimensional all-ones (-zeros) vector. Additionally, we use $I_n$ to denote the $(n \times n)$-dimensional identity matrix, and $0_{n \times n}$ to denote the $(n \times n)$-dimensional all-zeros matrix. The $j^{th}$ entry of a column vector $x$ is denoted by $x_j$ or $x(j)$, whereas the entry at the $j^{th}$ row, $i^{th}$ column of matrix $A$ is denoted by $A_{ji}$ or $A(j, i)$. We use $\text{diag}(a_1, a_2, ..., a_n)$ to denote an $n \times n$ diagonal matrix with elements $a_1, a_2, ..., a_n$ on its main diagonal. More generally, we use $\text{diag}(A_1, A_2, ..., A_m)$ to
denote a block diagonal matrix (of appropriate dimensions) with square blocks \( A_1, A_2, \ldots, A_m \) on its main diagonal. We denote the transpose of a vector \( x \) (or a matrix \( A \)) by \( x^T \) (or \( A^T \)).

We use uppercase calligraphic symbols to denote sets of elements (e.g., \( S \)). In particular, \( S \setminus T \) denotes the set of elements in \( S \) that do not belong in \( T \). The cardinality of a set \( S \) is denoted by \( |S| \).

We use the notation \( x := y \) to define \( x \) to be equal to \( y \), and \( x =: y \) to define \( y \) to be equal to \( x \). We reserve the symbol \( k \) to index discrete time (or iterations) and explicitly show the time dependence of a scalar/vector variable \( x \) or matrix \( A \) as \( x[k] \) or \( A[k] \), respectively.

### 2.1.2 Graph Theory

In this section we review basic concepts from graph theory that are important for later developments in the monograph; more details can be found in graph theory textbooks (e.g., West, 2001).

A digraph of order \( n \) \((n \geq 2)\) is captured by \( G = (V, E) \), where \( V = \{v_1, v_2, \ldots, v_n\} \) is the set of nodes or vertices, and \( E \subseteq (V \times V) \setminus \{(v_j, v_j) \mid |v_j \in V\} \) is the set of edges (with self-edges excluded). A directed edge from node \( v_i \) to node \( v_j \) is denoted by \((v_j, v_i) \in E\), and indicates (depending on the application) that node \( v_i \) physically influences node \( v_j \), or there is a flow of information or some physical quantity from node \( v_i \) to node \( v_j \).

A digraph \( G = (V, E) \) is called **strongly connected** if for each pair of nodes \( v_j, v_i \in V, v_j \neq v_i \), there exists a directed path from \( v_i \) to \( v_j \), i.e., we can find a sequence of nodes \( v_i =: v_{l_0}, v_{l_1}, \ldots, v_{l_t} := v_j \) such that \((v_{l_{\tau+1}}, v_{l_\tau}) \in E\) for \( \tau = 0, 1, \ldots, t - 1 \). The digraph is called **weakly connected** or simply **connected**, if such a path can be found ignoring the direction of the edges (i.e., we can find a sequence of nodes \( v_i =: v_{l_0}, v_{l_1}, \ldots, v_{l_t} := v_j \) such that \((v_{l_{\tau+1}}, v_{l_\tau}) \in E\) or \((v_{l_\tau}, v_{l_{\tau+1}}) \in E\) for \( \tau = 0, 1, \ldots, t - 1 \)). Clearly, a strongly connected digraph is also (weakly) connected.

All nodes that are directly connected to node \( v_j \) comprise the set of **in-neighbors** of node \( v_j \), and are denoted by \( N^-_j := \{v_l \in V \mid (v_l, v_j) \in E\} \). The cardinality of \( N^-_j \) is called the **in-degree** of \( v_j \), and is denoted by \( d^-_j := |N^-_j| \). The nodes to which node \( v_j \) is connected comprise its **out-neighbors**, and are denoted by \( N^+_j := \{v_l \in V \mid (v_l, v_j) \in E\} \). The cardinality of \( N^+_j \) is called the **out-degree** of \( v_j \), and is denoted by \( d^+_j := |N^+_j| \).
By assigning a nonnegative weight \( w_{ji} \) to each edge \((v_j, v_i) \in E\) of a
given digraph \( G = (V, E) \), we obtain a weighted digraph \( G_w = (V, E, W) \) of
order \( n \). Here, \( W = [w_{ji}] \) is an \( n \times n \) weight matrix where \( W(j, i) = w_{ji} \) are
weights. We require that \( w_{ji} \neq 0 \) if and only if \((v_j, v_i) \in E\).

A special case of a weight matrix is the adjacency matrix of the graph,
denoted by \( A_d \). This is a binary \( n \times n \) matrix whose entries satisfy
\( A_d(j, i) = 1 \) if and only if \((v_j, v_i) \in E \) (otherwise, \( A_d(j, i) = 0 \)). Note that the adjacency
matrix \( A_d \) is an alternative (equivalent) representation for a digraph. Another
representation for a digraph that we use in Chapter 6 is the node-to-edge
incidence matrix, \( M \), defined as follows. If we assume that there are
\( m \) edges (i.e., \(|E| = m\)) and we give an arbitrary label to each edge so that
\( E = \{e_1, e_2, \ldots, e_m\} \), then \( M \) is an \( n \times m \) matrix, the entries of which are
defined as follows:

\[
M(j, l) = \begin{cases} 
1, & \text{if } e_l = (v_i, v_j) \text{ for some } v_i \in V, \\
-1, & \text{if } e_l = (v_j, v_i) \text{ for some } v_i \in V, \\
0, & \text{otherwise}, 
\end{cases}
\]

(2.1)

where \( j \in \{1, 2, \ldots, n\} \) and \( l \in \{1, 2, \ldots, m\} \). Since we are not dealing with
multigraphs (i.e., graphs that can admit multiple edges from some node, say
\( v_i \), to some other node, say \( v_j \)), no two columns of \( M \) can be identical.

An important special case of a digraph is the case of an undirected graph
\( G_u = (V, E_u) \), which can be seen as a digraph that satisfies \((v_j, v_i) \in E_u\) if
and only if \((v_i, v_j) \in E_u \). If the digraph is undirected, then we have \( N_j^+ = N_j^- := N_j \), and \( d_j^+ = d_j^- := d_j \) for all nodes \( v_j \in V \); we refer to \( N_j \) as
the set of neighbors of node \( v_j \), and to \( d_j = |N_j| \) as the degree of node \( v_j \).
The adjacency matrix of an undirected graph is easily seen to be symmetric
(i.e., \( A_d = A_d^T \)). A weighted undirected graph \( G_{u,w} = (V, E_u, W) \) is typically
associated with a weight matrix \( W \) that is symmetric (i.e., \( W = W^T \)).

An undirected graph \( G_u = (V, E_u) \) can always be viewed as a special
case of a digraph (in which \( E_u \) satisfies the key property mentioned above).
An alternative is to treat edges as unordered pairs of vertices (i.e., a subset
of \( \{(v_i, v_j) \mid v_i, v_j \in V \text{ such that } v_i \neq v_j\} \)). In such case, edge \( \{v_j, v_i\} \)
indicates that the two nodes are connected in both directions, i.e., \((v_j, v_i) \in
E_u \) and \((v_i, v_j) \in E_u \). Note that in undirected graphs the notions of strong
connectivity and connectivity are equivalent.
2.1. Mathematical Background

2.1.3 Nonnegative Matrices

In this section we review some results on nonnegative matrices (see, for example, Seneta, 2006; Horn and Johnson, 1985) that are important for later developments in the monograph.

A matrix $A$ with nonnegative entries, called nonnegative matrix, is denoted by $A \geq 0$, and a matrix $A$ with positive entries, called positive matrix, is denoted by $A > 0$. We will typically use the symbol $P_c$ to denote a column stochastic matrix (i.e., a nonnegative matrix whose entries in each column sum to unity), and the symbol $P_d$ to denote a doubly stochastic matrix (i.e., a nonnegative matrix whose entries in each row and each column sum to unity).

**Definition 2.1.** A nonnegative square matrix $A$ is said to be reducible if and only if there exists a permutation matrix, $P$, such that $P^T AP$ is a block upper triangular matrix (recall that $T$ denotes matrix transposition). If the square matrix $A$ is not reducible, it is said to be irreducible.

**Definition 2.2.** A nonnegative square matrix $A$ is said to be primitive if there exists $k > 0$ such that $A^k > 0$.

Any $n \times n$ matrix $A$ can be associated to its corresponding (unique) digraph $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ and $E = \{(v_j, v_i) \mid A(j, i) \neq 0\}$. For the case of nonnegative matrices, this digraph captures important properties. For example, it can be shown that the following are equivalent:

1. $A$ is an irreducible (nonnegative) matrix.
2. The digraph associated to $A$ is strongly connected.
3. Given any indices $i$ and $j$ such that $1 \leq i, j \leq n$, we can find $k > 0$ such that the $(j, i)^{th}$ entry of $A^k$ is greater than zero.
4. For any partition $P_1$ and $P_2$ of the index set $\{1, 2, ..., n\}$, and for any $k > 0$, we can find $j \in P_1$ and $i \in P_2$ such that the $(j, i)^{th}$ entry of $A^k$ is greater than zero.

---

1 An $n \times n$ permutation matrix $P$ is a matrix that has exactly one nonzero entry with value “1” at each row and each column.

2 Though the definition of a digraph typically excludes self-loops (i.e., edges of the form $(v_j, v_j)$ for some $v_j \in V$), we allow self-loops when making the association between a matrix and its corresponding digraph.
A sufficient condition for a nonnegative matrix to be primitive is for the matrix to be irreducible with at least one positive element on the main diagonal.

Notation $\sigma(A)$ denotes the spectrum of an $n \times n$ matrix $A$, i.e., $\sigma(A) = \{\lambda_1(A), \lambda_2(A), ..., \lambda_n(A)\}$, where $\lambda_i(A)$ denotes the $i^{th}$ eigenvalue of matrix $A$. Typically, the $\lambda_i(A)$’s are arranged according to their magnitude so that $\lambda_1(A)$ is the eigenvalue (possibly non-unique) of the largest magnitude; this magnitude is the spectral radius of matrix $A$, denoted by $\rho(A) = \max_i \{ |\lambda_i(A)| \mid \lambda_i(A) \in \sigma(A) \}$. The following theorem (which comes in many variations, see, e.g., Horn and Johnson, 1985) captures a key spectral property of nonnegative matrices.

**Theorem 2.1.** (Perron-Frobenius) Given an irreducible nonnegative matrix $A \in \mathbb{R}_{+}^{n \times n}$, the spectral radius $\rho(A)$ is a simple eigenvalue of $A$ (called the Perron-Frobenius eigenvalue) that corresponds to a unique nonnegative left eigenvector $w^T$ (such that $w^T A = \rho(A) w^T$), and a unique nonnegative right eigenvector $v$ such that $A v = \rho(A) v$.

When the irreducible (nonnegative) matrix $A$ in the above theorem is also primitive, then $\rho(A)$ is guaranteed to be the unique positive eigenvalue of largest magnitude. In the special case when one is dealing with an irreducible column stochastic matrix $P_c$, then $\rho(P_c) = 1$ and $w = 1_n$. If we are dealing with a doubly stochastic matrix, $P_d$, then $w = v = 1_n$.

Theorem 2.1 can be used to understand how the limit $\lim_{k \to \infty} A^k$ of a nonnegative matrix $A$ behaves. For example, the following theorem establishes that the product $A^k$ tends, for large $k$, to a rank one matrix.

**Theorem 2.2.** Suppose $A$ is a primitive nonnegative matrix, with spectral radius $\rho(A)$ and left and right eigenvectors $w^T$ and $v$, respectively, normalized so that $w^T v = 1$. Then,

$$\lim_{k \to \infty} \left( \frac{1}{\rho(A)} A \right)^k = vw^T.$$ 

It turns out that, under certain conditions, the product of matrices from a collection of square ($n \times n$)-dimensional nonnegative matrices also tends to a rank-one nonnegative matrix. The following theorem by Wolfowitz was originally stated for row stochastic matrices in Wolfowitz, 1963, but it is adapted here for column stochastic matrices.
Definition 2.3. A column stochastic matrix $P_c$ is called SIA (stochastic, indecomposable and aperiodic) if $Q = \lim_{k \to \infty} P_c^k$ exists and has identical columns.

Definition 2.4. Given a column stochastic matrix $P_c$, let

$$\delta(P_c) = \max_j \max_{i_1, i_2} |P_c(j, i_1) - P_c(j, i_2)|.$$ 

Thus, $\delta(P_c)$ measures how different the columns of $P_c$ are; in particular, $\delta(P_c) = 0$ if and only if $P_c$ has identical columns.

Theorem 2.3. Let $P_1, P_2, ..., P_m$ be $n \times n$ column stochastic matrices such that any finite product of them is SIA. For any $\epsilon > 0$, there exists an integer $\nu(\epsilon)$ such that for $t \geq \nu(\epsilon)$ any product $B = P_{i_1}P_{i_2}...P_{i_t}$, where $i_k \in \{1, 2, ..., m\}$ for $k = 1, 2, ..., t$, satisfies

$$\delta(B) < \epsilon.$$ 

In other words, for sufficiently large $t$, any product of $t$ matrices from the collection $\{P_1, P_2, ..., P_m\}$ (repetitions allowed) will result in a product matrix $B$ with columns that are approximately the same.

The following lemma describes the limiting behavior of a product of primitive column stochastic matrices when the sequence of matrices in the product converges to a limiting matrix $P_c$; the result follows from Theorem 4.14 in Seneta, 2006.

Lemma 2.4. Let $P[0], P[1], ...$ be a sequence of column stochastic matrices such that $\lim_{k \to \infty} P[k] = P_c$ elementwise, where the matrix $P_c$ is column stochastic and primitive. Then $\lim_{k \to \infty} P[k] \cdots P[1]P[0] = v1_n^\top$, where $v$ is the unique solution to $v = P_c v$, normalized so that $\sum_{j=1}^n v_j = 1$.

Proof. Let $M_k = (P[k] \cdots P[1]P[0])^\top = P[1]^\top P[0]^\top P[k]$, where $P[1]^\top P[0]^\top$, $k \geq 0$, is a row stochastic matrix, and $\lim_{k \to \infty} P[1]^\top P[k] = P_c^\top$, where $P_c^\top$ is primitive and row stochastic, and $v^\top = v^\top P_c^\top$. By Theorem 4.14 in Seneta, 2006, $M := \lim_{k \to \infty} M_k = 1_n v^\top$. Therefore

$$\lim_{k \to \infty} P[k] \cdots P[1]P[0] = \lim_{k \to \infty} (P[1]^\top P[0]^\top P[k])^\top = \lim_{k \to \infty} M_k^\top = M^\top = v1_n^\top;$$

this concludes the proof.
2.2 Models for Distributed Control Systems

A distributed control system (or a network system or a multi-component system) consists of a set of subsystems (components), which can have physical interactions (e.g., exchanges of certain quantities of interest, such as weights or flows of masses), and/or cyber interactions (e.g., exchanges of information messages between sensors, actuators, control sites that are in charge of making decisions, etc.). Depending on the underlying application, either of these types of interactions can be directed; thus, in general, the interactions in a distributed control system are captured by two directed graphs (digraphs), one representing the physical interactions and one representing the cyber interactions, as described below.

The physical interactions between components can be conveniently represented by a digraph \( G_p = (V_p, E_p) \), where \( V_p = \{v_{p,1}, v_{p,2}, \ldots, v_{p,|V_p|}\} \) is the set of nodes or vertices, which correspond to the physical components of the system, and \( E_p \subseteq (V_p \times V_p) \setminus \{(v_{p,j}, v_{p,j}) | v_{p,j} \in V_p\} \) is the set of edges, which correspond to the (physical) interactions among them. A directed edge from node \( v_{p,i} \) to node \( v_{p,j} \) is denoted by \( (v_{p,j}, v_{p,i}) \in E_p \), and indicates that node \( v_{p,i} \) physically influences node \( v_{p,j} \), or there is a flow of some physical quantity (e.g., electric power or traffic) from node \( v_{p,i} \) to node \( v_{p,j} \). We will refer to the digraph \( G_p \) as the **physical topology**.

Similarly, the cyber interactions among the system components can be captured by a digraph \( G_c = (V_c, E_c) \), where \( V_c = \{v_{c,1}, v_{c,2}, \ldots, v_{c,|V_c|}\} \) is the set of nodes or vertices, which correspond to the cyber components of the system (such as computational elements, transmitters, receivers, storage devices, etc.), and \( E_c \subseteq (V_c \times V_c) \setminus \{(v_{c,j}, v_{c,j}) | v_{c,j} \in V_c\} \) is the set of edges, which correspond to the (cyber) interactions among them. In this case, directed edge \( (v_{c,j}, v_{c,i}) \in E_c \) indicates that communication from node \( v_{c,i} \) to node \( v_{c,j} \) is possible; implicitly, this assumes that \( v_{c,i} \) (respectively, \( v_{c,j} \)) has transmitting (respectively, receiving) capability. We will refer to the digraph \( G_c \) as the **cyber topology** or the **communication topology**.

In general, the union of the physical and cyber topologies (i.e., the directed graph \( G_p \cup G_c = (V_p \cup V_c, E_p \cup E_c \cup E_{pc}) \), where \( E_{pc} \subseteq (V_c \times V_p) \cup (V_p \times V_c) \) is the set of edges between physical and cyber nodes, via sensor and actuators, captured respectively by edges in \( V_c \times V_p \) and \( V_p \times V_c \), describes the overall distributed control system. For notational simplicity, we will assume
(without much loss of generality) that $V_p = V_c =: V$ and talk about the physical topology $G_p = (V, E_p)$ and the cyber topology $G_c = (V, E_c)$ of the given distributed control system. Implicitly, this assumes that each (physical) system component (represented as a node in the physical graph) also acts as a node in the communication graph, i.e., it is equipped with individual communication/computation capabilities. More generally, there could be components with no such capability or multiple components that share a single communication/computational unit. The case of a component with no associated communication/computational capability can easily be captured in our model by having the corresponding node in the cyber topology be isolated (i.e., without any incoming or outgoing edges). Similarly, the case of multiple components sharing the same communication/control node can be captured by having the corresponding nodes in the cyber topology posses all possible pairwise edges (i.e., form a fully connected subgraph of the cyber topology); however, in such case, one might have to carefully track the time instants (iterations) at which different information might be available at different components. For simplicity, we assume that each physical component of a distributed control system is associated with a unique node in the cyber topology (so that we can associate the same set of nodes $V$ to both $G_p$ and $G_c$). In fact, in some of the examples we will consider, we will have the graphs of physical and cyber interactions coincide, in which case we will refer to the digraph of interactions as $G$.

2.3 Distributed Algorithms

Given a digraph $G_c = (V, E_c)$ that captures the cyber (communication) topology of a given distributed control system, we can perform a variety of computational tasks, collectively executed on the (cyber) components of the system (captured by the nodes in the set $V$) via distributed algorithms.

The distributed algorithms of interest in this monograph operate over several iterations, indexed by $k = 0, 1, 2, \ldots$; at each iteration $k$, each node $v_j \in V$ updates some variable of interest it maintains by performing some local computations, based on the values of its own variable(s), as well as the values of variables that are sent to it by its in-neighbors (i.e., the nodes in the set $N^{-}_j$ that can send information to node $v_j$). More specifically, if we use $x_j[k]$ to denote (collectively) the internal variables (state) of node $v_j$ at
iteration $k$, then node $v_j$ can update its internal variables using an update function that takes as arguments some (or all) of the entries of the vector

$$X_j[k] := [x_j^T[k], x_{i_1}^T[k], x_{i_2}^T[k], ..., x_{i_{d_j}}^T[k]]^T,$$

where $\{v_{i_1}, v_{i_2}, ..., v_{i_{d_j}}\} = \mathcal{N}_j^-$. The update function could be nonlinear and even time-varying, but it is restricted to only take as arguments (a subset of) the above values. [In principle, a node could also maintain a history of the values $X_j[0], X_j[1], ..., X_j[k]$, and use it to perform the update, but that case can be reduced to the one that relies exclusively on $X_j[k]$ by properly redefining the state at each node.]

Unless otherwise stated, the distributed algorithms considered in this monograph are synchronized, i.e., all nodes perform their updates at the same time in a synchronous manner. In the basic case, the distributed algorithms we will develop will operate as follows: after proper initialization of the internal variables at each node (i.e., proper initialization of $x_j[0]$ for each $v_j \in \mathcal{V}$), each node $v_j$ will take the following steps at iteration $k$:

1. Send $x_j[k]$ to each of its (cyber) out-neighbors (i.e., to each $v_l$ in $\mathcal{N}_j^+$);
2. Receive $x_i[k]$ from each of its (cyber) in-neighbors (i.e., from each $v_i$ in $\mathcal{N}_j^-$);
3. Form $X_j[k]$ (as defined above), and update $x_j[k+1]$ based on some or all of the entries of $X_j[k]$.

Under nominal conditions, we make the following assumptions about the execution of the distributed algorithms we develop:

**A1** The communication topology $\mathcal{G}_c = (\mathcal{V}, \mathcal{E}_c)$ is fixed (time-invariant) and forms a strongly connected digraph.

**A2** Nodes are synchronized at the granularity of the iterations.

**A3** Each node is capable of simultaneously receiving messages from each of its in-neighbors; the receiving node is able to identify the sending node.
2.3. Distributed Algorithms

A4 Communication links are perfectly reliable, i.e., message exchanges between nodes do not exhibit delays or packet drops, and are received uncorrupted. We also assume that messages are long enough to be able to represent the real value(s) that are being transmitted with sufficient accuracy (thus, we ignore quantization effects that arise due to the fact that messages are digitized).

A5 All transmissions from each node \(v_j\) are broadcasts, i.e., each node \(v_j\) sends identical information to all of its out-neighbors in \(N_j^+\). [This is actually rather convenient in a wireless communication setting.]

An additional assumption made for some of the algorithms we develop for distributed averaging in directed graphs is the following:

A6 Each node \(v_j\) is aware of its out-degree \(d_j^+\).

2.3.1 Max-Consensus

One example of a distributed algorithm is the max-consensus algorithm, in which each node \(v_j\) is given an initial value \(x_j[0] = V_j\), and the distributed algorithm allows the nodes to calculate the maximum of these initial values in at most \(|\mathcal{V}| - 1 = n - 1\) steps. The max-consensus algorithm has each node update its value at iteration \(k, k = 0, 1, 2, \ldots, n - 2\), as

\[
x_j[k + 1] = \max_{v_i \in N_j^- \cup \{v_j\}} \{x_i[k]\}.
\]

It can be shown that the execution of the above updates under Assumptions [A1]–[A5] allows each node \(v_j\) to obtain a value \(x_j[n - 1]\) that satisfies

\[
x_j[n - 1] = \max_l \{x_l[0]\} = \max_l \{V_l\}, \forall v_j \in \mathcal{V}.
\]

Max-consensus is a prototypical example of a finite-time algorithm (as it completes in a finite number of steps). The pseudocode for the max-consensus algorithm is shown in Algorithm 1.

2.3.2 Distributed Averaging via Linear Iterations

Another popular distributed algorithm is the distributed average consensus algorithm, in which each node \(v_j\) is given an initial value \(x_j[0] = V_j\) and the
Algorithm 1: Max-Consensus

**Input:** A strongly connected digraph \( G_c = (\mathcal{V}, \mathcal{E}_c) \) with \( n = |\mathcal{V}| \) nodes and \( m = |\mathcal{E}_c| \) edges (and no self-loops). Each node \( v_j \in \mathcal{V} \) has some initial value \( V_j \).

**Output:** \( \max_{v_j \in \mathcal{V}} V_j \).

**Initialization:** Each node \( v_j \in \mathcal{V} \) sets \( x_j[0] = V_j \).

**Iteration:** For \( k = 0, 1, 2, \ldots, n - 1 \), each node \( v_j \in \mathcal{V} \) does the following:

1. It transmits \( x_j[k] \) to each out-neighbor \( v_l \in \mathcal{N}_j^+ \).
2. It receives \( x_i[k] \) from each in-neighbor \( v_i \in \mathcal{N}_j^- \).
3. It updates its own variable as
   \[
   x_j[k + 1] = \max_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} \{x_i[k]\}.
   \]

distributed algorithm allows the nodes to asymptotically calculate the average of these initial values. The distributed average consensus algorithm has each node update its value at iteration \( k, k = 0, 1, 2, \ldots, \) via a linear function of the form

\[
x_j[k + 1] = p_{jj} x_j[k] + \sum_{v_i \in \mathcal{N}_j^-} p_{ji} x_i[k],
\]

where \( p_{ji} \) is the weight via which node \( v_j \) weighs information from its in-neighbor \( v_i \), and \( p_{jj} \) is its self-weight. Depending on the values of the weights, linear updates can have several interesting properties. For example, assuming that the matrix \( P = [p_{ji}] \) \((p_{ji} = 0 \text{ if } j \neq i \text{ and } (v_j, v_i) \notin \mathcal{E}_c)\) is primitive doubly stochastic, it can be shown (see Chapter 3) that

\[
\lim_{k \to \infty} x_j[k] = \frac{\sum_{\ell} x_\ell[0]}{n} = \frac{\sum_{v_\ell \in \mathcal{V}} V_\ell}{n}, \quad \forall v_j \in \mathcal{V},
\]

which allows the nodes to asymptotically compute the average of their initial values. Average consensus with linear updates is a prototypical example of an asymptotic distributed algorithm. The pseudocode for the distributed algorithm is shown in Algorithm 2.
Algorithm 2: Distributed Average

**Input:** A strongly connected digraph $G = (V, E_c)$ with $n = |V|$ nodes and $m = |E_c|$ edges (and no self-loops).

Each node $v_j \in V$ has some initial value $V_j$.

Each edge $(v_j, v_i) \in E_c$ is associated with a weight $p_{ji}$ such that the matrix $P = [p_{ji}]$ (with nonzero diagonal, and $p_{ji} = 0$ when $j \neq i$ and $(v_j, v_i) \notin E_c$) is doubly stochastic.

**Output:** $\sum_{v_j \in V} V_j$.

**Initialization:** Each node $v_j \in V$ sets $x_j[0] = V_j$.

**Iteration:** For $k = 0, 1, 2, \ldots$, each node $v_j \in V$ does the following:

Step 1: It transmits $x_j[k]$ to its out-neighbors $v_l \in N_j^+$. 

Step 2: It receives $x_i[k]$ from each in-neighbor $v_i \in N_j^-$. 

Step 3: It updates its own variable as 

$$x_j[k + 1] = p_{jj}x_j[k] + \sum_{v_i \in N_j^-} p_{ji}x_i[k] .$$

2.4 Further Reading

Graph theory can be explored further by referring to one of many good reference books, such as the introductory references West, 2001; Godsil and Roy, 2000 and the more advanced Bollobás, 2013.

Similarly, the reader who is interested in studying nonnegative matrix theory in more depth can start from one of several good introductions to the topic Strang, 1993; Meyer, 2000 as well as more advanced textbooks on theory Roman, 2005 and applications Horn and Johnson, 1985; Trefethen and Bau III, 1997. Nonnegative matrices and their properties are studied in Seneta, 2006; Berman and Plemmons, 1994.

Distributed algorithms are discussed in a variety of books (e.g., Lynch, 1996); distributed algorithms for control systems are also discussed in Shamma, 2008; Bullo *et al.*, 2009; Mesbah and Egerstedt, 2010.
In this chapter, we provide strategies for distributed average consensus in undirected graphs (where each link has bidirectional communication capability). We also provide strategies that can be used in the more general case of a directed graph. In addition, we discuss strategies for distributed average consensus that can overcome transmission delays and packet drops. We also discuss techniques for ensuring privacy and handling faulty/malicious nodes. We conclude the chapter by discussing further extensions and open problems.

3.1 Introduction and Motivation

As discussed in Chapter 1, in the distributed consensus (or agreement) problem, each node in a network system possesses an initial value, and the nodes need to agree (“reach consensus”) on the same value, typically by calculating, via some distributed algorithm (e.g., a linear iteration of the type described in Chapter 2, Section 2.3), the same function of their initial values. A prototypical application of consensus is a network of sensors with noisy measurements of the same quantity (e.g., the temperature in a region) that try to average (or, more generally, obtain a weighted linear combination of) their measurements so as to have a more accurate estimate of the measured quantity.
3.1. Introduction and Motivation

As demonstrated in the works by Tsitsiklis, 1984; Jadababaie et al., 2003; Xiao and Boyd, 2004; Moreau, 2005; Sundaram and Hadjicostis, 2008, and others, consensus can be reached under exceedingly weak conditions on the interactions among the nodes of the system (e.g., nonlinear updates, time-varying interconnections with relatively few connections at any given time, etc.). Algorithm 1 in Chapter 2 is an example of a distributed iterative scheme that allows the nodes in a network system to reach (in finite time) consensus to the maximum of their initial values, even under time-varying interconnections and asynchronous operation.

As discussed in in Chapter 1, when the value to which the nodes agree is the average of the initial values, we say that the nodes reach average consensus. Asymptotic average consensus is reached if, while running an iterative procedure, the nodes asymptotically converge to the average of their initial values; this approach has been the focus of study by several researchers who have considered a variety of pertinent problems, including convergence properties Chatterjee and Seneta, 1977; Tsitsiklis, 1984; Blondel et al., 2005; Olshevsky and Tsitsiklis, 2009, and weight choices that achieve faster convergence Xiao and Boyd, 2004. In contrast, finite-time average consensus is reached if the nodes, after running an iterative procedure for a finite number of steps, are able to compute a value that is the exact average (or an approximation of the average) of their initial values. This chapter focuses on iterative strategies for asymptotic average consensus in a distributed fashion, whereas Chapter 4 discusses strategies that achieve finite-time (exact or approximate) average consensus.

There are a number of different characteristics that are important in average consensus problems, including distributivity constraints (e.g., the need for each node to rely solely on locally available information), the possibility of a time-varying interconnection topology, the computational/communication/storage requirements (overall and at each node individually), convergence to the average (or to a value close to it), speed of convergence, and others. This chapter mainly focuses on the average consensus problem when the interconnection topology is described by a fixed graph that could be either undirected or directed; some extensions to time-varying interconnections (dealing with transmission delays, and packet drops) and pointers to more advanced studies are also included.
3.2 Preliminaries and Problem Formulation

Consider a distributed system described by a (possibly directed) cyber topology $G_c = (\mathcal{V}, \mathcal{E}_c)$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes (system components). Since the physical topology is not invoked in this chapter, we drop the subscript $c$ and simply refer to the (cyber) topology as $G = (\mathcal{V}, \mathcal{E})$. Unless explicitly stated otherwise, pertinent definitions (such as $N_j^+, d_j^-$, etc.) are defined with respect to the topology captured by $G$.

Assume that each node $v_j$ has some value $V_j$. The objective of average consensus is to have all the nodes calculate the average of these values, which we denote by $\mu := \frac{\sum_{i=1}^n V_i}{n}$. Depending on the assumptions, nodes may or may not know $n$ (the total number of nodes), and they will presumably require several rounds of message exchanges in order to obtain $\mu$ (perhaps obtaining some of the values in $\{V_1, V_2, \ldots, V_n\}$ in the process). In the algorithms we present in this chapter, each node $v_j$ maintains some state $x_j[k]$ at iteration $k$, and updates it iteratively based on the values received from its in-neighbors. The state $x_j[k]$ is assumed to be scalar-valued in the immediate discussion below, but, more generally, it could be vector-valued.

In a typical consensus formulation, each node $v_j$ updates its state at iteration $k$ using a linear, possibly time-varying, update of the form

$$x_j[k+1] = p_{jj}[k]x_j[k] + \sum_{v_i \in N_j^-[k]} p_{ji}[k]x_i[k], \tag{3.1}$$

where $x_j[0] = V_j$, the set $N_j^-[k]$ denotes the subset of nodes that send values to node $v_j$ at iteration $k$ (sometimes referred to as the in-neighbors of node $v_j$ at iteration $k$), and the weights $p_{ji}[k]$ form a set of time-varying weights. More generally, the update in (3.1) could be replaced by a nonlinear function, as long as the arguments to these functions are the same (i.e., the values of $x_j[k]$ and $\{x_i[k] \mid v_i \in N_j^-[k]\}$ at iteration $k$).

If we let $x[k] = [x_1[k], x_2[k], \ldots, x_n[k]]^T$, then for analysis purposes the iteration in (3.1) can be written in matrix form as

$$x[k+1] = P[k]x[k],$$
$$x[0] = x_0, \tag{3.2}$$

where $x_0 = [V_1, V_2, \ldots, V_n]^T$ and $P[k] = [p_{ji}[k]]$ is the weight matrix (with the constraint that $p_{ji}[k] = 0$ if $v_i \notin N_j^-[k] \cup \{v_j\}$).
3.2. Preliminaries and Problem Formulation

In this chapter, we are interested in iterations like the one in (3.1) that allow the nodes to reach asymptotic consensus (i.e., to asymptotically compute the same value, which is a function of their initial values) and, in particular, to reach average consensus (i.e., to asymptotically compute the average of their initial values); these are formally defined below.

**Definition 3.1.** The nodes are said to reach asymptotic consensus if
\[
\lim_{k \to \infty} x_j[k] = f(x_1[0], x_2[0], \ldots, x_n[0]) , \quad \forall v_j \in \mathcal{V} ,
\]
where \( f : \mathbb{R}^n \to \mathbb{R} \) is some function of the initial values.

**Definition 3.2.** The nodes are said to reach asymptotic average consensus if
\[
\lim_{k \to \infty} x_j[k] = \frac{1}{n} \sum_{\ell=1}^{n} x_\ell[0] = \mu , \quad \forall v_j \in \mathcal{V} ,
\]
where \( \mu \) is the average of the initial values.

**Remark 3.1.** Whether the nodes reach (average) consensus, or not, depends critically on the choice of the time-varying weights \( p_{ji}[k] \)'s in (3.1). This choice has to be made by the nodes involved in a way that respects the distributivity constraints imposed. For example, if node \( v_i \) transmits to node \( v_j \) the variable \( x_i[k] \) and node \( v_j \) is able to identify the transmitting node (Assumption [A3] in Chapter 2), then \( p_{ji}[k] \) can be chosen by node \( v_j \); alternatively, if node \( v_i \) transmits to node \( v_j \) the variable \( p_{ji}[k]x_i[k] \), then \( p_{ji}[k] \) is effectively chosen by node \( v_i \) (and, assuming all in-neighbors of node \( v_j \) do the same, node \( v_j \) does not need to be able to identify the transmitting node, since it simply has to sum up the received values). However, for node \( v_i \) to be able to determine an appropriate weight \( p_{ji} \) for its out-neighbor \( v_j \), it might need to know the number of its out-neighbors (Assumption [A6] in Chapter 2) and in some cases their ID’s (so that they can direct different messages to them). Regardless of how the choice of weights is made, it should be clear that \( p_{ji}[k] = 0 \) for all \( i \neq j \), such that \( v_i \notin \mathcal{N}_j^-[k] \).

### 3.2.1 Time-Invariant Linear Iterations

If the weights (and interconnections) in iteration (3.1) are not changing over time, we are led to the linear time-invariant iteration
\[
x_j[k+1] = p_{jj}x_j[k] + \sum_{v_i \in \mathcal{N}_j^-} p_{ji}x_i[k] , \quad (3.3)
\]
which was implemented in a distributed fashion as Algorithm 2 in Chapter 2. Again, letting $x[k] = [x_1[k], x_2[k], \ldots, x_n[k]]^T$, we have

$$x[k + 1] = Px[k], \ x[0] = x_0,$$

(3.4)

where $x_0 = [V_1, V_2, \ldots, V_n]^T$ and $P = [p_{ji}]$ is the weight matrix (with the constraint that $p_{ji} = 0$ for all $i \neq j$, such that $v_i \notin N_j^\ominus$).

As stated in Xiao and Boyd, 2004; Olfati-Saber and Murray, 2004; Sundaram and Hadjicostis, 2008 in various forms, the necessary and sufficient conditions for the iteration in (3.4) to asymptotically reach average consensus are: (i) $P$ has a simple eigenvalue at 1, with left eigenvector $1_n^T$, and right eigenvector $1_n$, and (ii) all other eigenvalues of $P$ have magnitude strictly less than 1. If one focuses on nonnegative weights, these conditions are equivalent to $P$ being a primitive doubly stochastic matrix. In such case, from the Perron-Frobenius Theorem and Theorem 2.2 in Chapter 2, since $v = w = \frac{1}{\sqrt{n}} 1_n$ (normalized so that $w^T v = 1$) and $\rho(P) = 1$, it readily follows that

$$\lim_{k \to \infty} x[k] = \lim_{k \to \infty} P^k x_0 = v w^T x_0 = \frac{1}{n} 1_n (1_n^T x_0) = \frac{1}{n} \sum_{l} V_l 1_n = \mu 1_n,$$

which effectively establishes that the nodes asymptotically reach average consensus.

In subsequent sections of this chapter, we discuss how such weights can be chosen, in either undirected or directed graphs, so that the corresponding weight matrix $P$ is a nonnegative matrix with certain desirable properties. In particular, in undirected graphs, we discuss choices that lead to a weight matrix $P$ that is primitive doubly stochastic (which, as argued above, will lead to asymptotic average consensus). In directed graphs, we will discuss choices that lead to a weight matrix $P$ that is primitive column stochastic, and we will also describe how to use these weights to obtain iterative strategies that lead to asymptotic average consensus. At this point, it is also instructive to observe that, following the above approach, we can establish the following facts, which will be useful in our analysis later on.
3.2. Preliminaries and Problem Formulation

Row Stochastic Matrix

When $P$ is a primitive row stochastic matrix, then $\rho(P) = 1$ and $v = 1_n$, so that

$$\lim_{k \to \infty} x[k] = \lim_{k \to \infty} P^k x_0 = \frac{1}{w^T v} w^T x_0 = \frac{1}{w^T 1_n} 1_n (w^T x_0) = \frac{w^T x_0}{w^T 1_n} 1_n,$$

i.e., the nodes reach asymptotic consensus to the value $\frac{w^T x_0}{w^T 1_n}$. Note that nodes do not need to know $w$, as long as they are only interested in reaching consensus. Also, under Assumption [A3], each node can easily choose the weights on its incoming links to form a row stochastic matrix.

Column Stochastic Matrix

When $P$ is a primitive column stochastic matrix, then $\rho(P) = 1$ and $w = 1_n$, so that

$$\lim_{k \to \infty} x[k] = \lim_{k \to \infty} P^k x_0 = \frac{1}{w^T v} w^T x_0 = \frac{1}{1_n^T v} v (1_n^T x_0) = \frac{1_n^T x_0}{1_n^T v} v,$$

i.e., the nodes do not reach asymptotic consensus but they reach steady-state. Node $v_j$ reaches a value that is proportional, depending on $x_0$, to the $j^{th}$ entry of vector $v$ (the normalization of $v$ does not matter). Note that under Assumption [A6], nodes can easily choose the weight on their outgoing edges (e.g., set $P(l, j) = \frac{1}{1+d_j^+}$ for all $v_l \in \mathcal{N}_j^+ \cup \{v_j\}$), so that the resulting weight matrix $P$ is column stochastic.
3.2.2 Time-Varying Linear Iterations

Under relatively mild conditions on the weight matrices $P[k]$, the time-varying iteration in (3.2) also reaches asymptotic (average) consensus: more specifically, one can invoke Wolfowitz’s theorem (Theorem 2.3 in Chapter 2) to argue that the following are sufficient conditions for the iteration to reach asymptotic consensus:

- **C1** At each iteration $k$, the weight matrix $P[k]$ is a row stochastic matrix whose nonzero entries are bounded away from zero (i.e., they are lower bounded by some positive constant).

- **C2** There exists a finite window $K$, such that the matrix products

  $$P[mK + K - 1] \ldots P[mK + 1]P[mK], \ m = 0, 1, 2, \ldots,$$

  form primitive row stochastic matrices.

[Note that the product of row stochastic matrices is necessarily a row stochastic matrix; thus, the requirement is that the matrix product is a primitive matrix.]

**Remark 3.2.** If we let $G[k] = (\mathcal{V}, \mathcal{E}[k])$ (where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$) be the digraph that corresponds to matrix $P[k]$, then Condition [C2] is satisfied if the matrices $P[k]$ have positive elements on their diagonals, and the union graphs $G[m] := (\mathcal{V}, \cup_{k=mK}^{mK+K-1} \mathcal{E}[k]), \ m = 0, 1, 2, \ldots$, are strongly connected. This was explored for row stochastic matrices in Jadbabaie *et al.*, 2003 to establish conditions for asymptotic consensus, in the presence of time-varying topologies. In fact, the authors of Kingston and Beard, 2006 used the above approach to establish conditions for asymptotic average consensus, in the presence of time-varying topologies, by simply requiring the matrices $P[k]$ at each iteration in Condition [C1] to be doubly stochastic matrices, and the matrix products in Condition [C2] to be primitive doubly stochastic matrices.

### 3.3 Distributed Averaging in Undirected Graphs

In this section, we assume that the graph $\mathcal{G}$ that describes the cyber topology is undirected, and we discuss ways in which the nodes can reach asymptotic average consensus. The easiest strategy is to use the linear iteration in (3.3) (i.e., run Algorithm 2 in Chapter 2) with a proper choice of the weights.
3.3. Distributed Averaging in Undirected Graphs

$p_{ji}$, so that they form a primitive doubly stochastic matrix $P = [p_{ji}]$ (i.e., $\sum_l p_{lj} = \sum_i p_{ji} = 1, \forall j$). It turns out that there are several ways in which this can be achieved in an undirected graph. In all of the ways described below, the resulting doubly stochastic matrix $P$ is symmetric, with positive elements on its main diagonal; in fact, the matrix $P$ will be primitive as long as the given graph $G$ is connected. We assume for simplicity that the elements in the set $\{p_{lj} \mid v_l \in \mathcal{N}_j^+ \cup \{v_j\}\}$ are determined by node $v_j$ (i.e., each node is in charge of setting its self-weight and the weights on its outgoing links); this, however, can easily be changed in the undirected topology since communication between any two nodes is bidirectional. In what follows, we discuss two methods by which nodes can choose the weights in a distributed fashion.

- **Equal weights:** Assuming the nodes know the total number of nodes $n$ or an upper bound $n' \geq n$, each node $v_j$ sets the weights on all of its out-going links (recall that in an undirected graph $d_j = d_j^+ = d_j^-$) to be

$$p_{lj} = \begin{cases} \frac{1}{n'}, & \forall v_l \in \mathcal{N}_j^+, \\ 1 - \frac{d_j}{n'}, & l = j, \\ 0, & \text{otherwise.} \end{cases}$$

Note that $n'$ has to be greater or equal to $n$ to ensure that the diagonal elements will not be negative or zero. [At least one positive diagonal element ensures that $P$ is primitive (assuming the graph $G$ is connected).]

- **Metropolis weights:** Another simple choice that results in a doubly stochastic (symmetric) weight matrix $P$ are the Metropolis weights utilized in Xiao et al., 2005, where each node $v_j$ sets the weights on all of its out-going links to be

$$p_{lj} = \begin{cases} \frac{1}{1+\max(d_l,d_j)}, & \forall v_l \in \mathcal{N}_j^+, \\ 1 - \sum_{v_l \in \mathcal{N}_j^+} p_{lj}, & l = j, \\ 0, & \text{otherwise}. \end{cases}$$

Note that to determine the weight for $(v_l, v_j)$ (and also $(v_j, v_l)$), nodes $v_l$ and $v_j$ need to exchange their degrees (once, at the initialization of the algorithm). Also, note that $p_{jj} \geq \frac{1}{1+d_j} > 0.$
The rate of convergence to average consensus depends on the eigenvalue of the doubly stochastic matrix $P$ that has the second largest magnitude. This eigenvalue is denoted by $\lambda_2$ (the eigenvalue with largest magnitude is $\lambda_1 = 1$, which is unique if $P$ is primitive). Larger diagonal entries typically imply an increase in the magnitude of $\lambda_2$, thus leading to slower convergence. For example, a larger value of $n'$ when choosing equal weights leads to slower convergence. For this reason, Metropolis weights generally lead to faster convergence over equal weights.

Even if the connections between nodes are time-varying, effectively leading to a sequence of undirected graphs $G[k] = (V, E[k])$, either of the above choices (at each iteration $k$, for the given graph $G[k]$), would lead to a sequence of doubly stochastic matrices $P[k]$, which (under mild conditions, as described at the end of the previous section) allow the nodes to reach asymptotic average consensus.

**Remark 3.3.** Another popular approach for the calculation of the average value of the network is to use a randomized gossip protocol Boyd et al., 2005; Dimakis et al., 2010. In a randomized gossip protocol, a pair of nodes is randomly selected at each iteration (e.g., node $v_j$ wakes up at some random time instant, and selects randomly one of its neighboring nodes $v_i, v_i \in N_j$). The pair of nodes simply performs local averaging on their values, i.e., at iteration $k$, each node updates its value as

$$x_j[k+1] = x_i[k+1] = \frac{x_i[k] + x_j[k]}{2}. \quad (3.6)$$

[The remaining nodes do not update their values, i.e., $x_l[k+1] = x_l[k]$ for all nodes $v_l, v_l \neq v_j$ and $v_l \neq v_i$.] In terms of the linear time-varying iteration in (3.2), the update is governed by a doubly stochastic matrix $P[k]$ defined as follows:

$$
\begin{align*}
P[k](i, i) &= 1/2, \\
P[k](j, j) &= 1/2, \\
P[k](i, j) &= 1/2, \\
P[k](j, i) &= 1/2, \\
P[k](l, l) &= 1, \ l \neq i, l \neq j,
\end{align*}
$$

with all other entries being zero. It can be shown that, if all possible node pairs (as allowed by the topology) are active frequently enough, then, as long
as the graph is connected, they are guaranteed to reach, with probability one, asymptotic average consensus, i.e., \( \lim_{k \to \infty} x_j[k] = \mu, \forall v_j \in \mathcal{V} \).

**Example 3.1.** We now illustrate the strategies described above using as an example the undirected graph \( \mathcal{G}_u = (\mathcal{V}, \mathcal{E}) \) in Figure 3.1. In this particular case, the weight matrices \( P_{eq} \) for the strategy that uses equal weights \( (1/n, \text{ with } n = 5) \), and the weight matrix \( P_{metro} \) for the strategy that uses Metropolis weights are as follows:

\[
P_{eq} = \begin{bmatrix}
0.4 & 0.2 & 0.2 & 0.2 & 0 \\
0.2 & 0.4 & 0.2 & 0 & 0.2 \\
0.2 & 0.2 & 0.4 & 0 & 0.2 \\
0.2 & 0 & 0 & 0.6 & 0.2 \\
0 & 0.2 & 0.2 & 0.2 & 0.4
\end{bmatrix},
\]

\[
P_{metro} = \begin{bmatrix}
0.25 & 0.25 & 0.25 & 0.25 & 0 \\
0.25 & 0.25 & 0.25 & 0 & 0.25 \\
0.25 & 0.25 & 0.25 & 0 & 0.25 \\
0.25 & 0 & 0 & 0.5 & 0.25 \\
0 & 0.25 & 0.25 & 0.25 & 0.25
\end{bmatrix}.
\]

Note that Metropolis weights in this example happened to be equal in value, but that will not always be the case.

![Figure 3.1](image-url)

**Figure 3.1:** Connected undirected graph \( \mathcal{G}_u \) used to illustrate the execution of distributed averaging schemes with weights that form a doubly stochastic weight matrix.

Suppose that the initial values for the five nodes are

\[
x[0] = [V_1, V_2, V_3, V_4, V_5]^T = [1, 2, 3, 4, 5]^T,
\]
with average $\mu = 3$. Figure 3.2 shows 25 iterations of runs of two different distributed averaging strategies, one using equal weights (left), and one using Metropolis weights (right). As we can see, the strategy that uses Metropolis weights converges at a slightly faster pace; this is consistent with the fact that, for the two matrices in our example, we have have $\lambda_2(P_{eq}) = 0.6$ and $\lambda_2(P_{metro}) = 0.5$ (recall that $\lambda_2(P)$ is eigenvalue of matrix $P$ with the second largest magnitude, and governs the rate of convergence).

![Distributed Averaging with Equal Weights](image1)

![Distributed Averaging with Metropolis Weights](image2)

**Figure 3.2**: Execution of the distributed averaging algorithm with equal weights (left), and with Metropolis weights (right) on the undirected graph of Figure 3.1, shown in terms of the values $x_j[k], j \in \{1, 2, 3, 4, 5\}$, at each iteration.

A sample run of the gossip-based strategy in (3.6) is shown in Figure 3.3. At each iteration $k$, a pair of neighboring nodes $v_i$ and $v_j$ ($v_i \neq v_j$ and $\{v_i, v_j\} \in \mathcal{E}$) is chosen at random and the corresponding nodes locally average their values (i.e., $x_j[k+1] = x_i[k+1] = \frac{x_i[k]+x_j[k]}{2}$); the remaining nodes do not change their values. A larger number of iterations is needed for convergence, but one should keep in mind that in the gossip-based strategy only two nodes update their values (i.e., each iteration involves a single communication exchange between a pair of nodes), whereas in the strategies with equal and Metropolis weights illustrated in Figure 3.2, all node values are updated at each iteration.
3.4 Distributed Averaging in Digraphs

In this section, we relax the assumption that the underlying topology corresponds to an undirected graph by considering directed topologies. Asymmetric topologies can arise in a variety of realistic scenarios (e.g., if nodes transmit at different power strengths or if interference levels are not uniform at each node). The presence of asymmetry in the information structure makes the average consensus task challenging due to the fact that nodes cannot (easily) provide acknowledgements; more generally, nodes cannot directly inform the nodes that guide their updates about the actions they are taking.

In a digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a given node $v_j \in \mathcal{V}$ will likely not satisfy $d_j^+ = d_j^-$, which implies that the weight assignments for undirected graphs described in the previous section will not necessarily result in a doubly stochastic matrix. In fact, a weight assignment that results in a doubly stochastic matrix is not as straightforward in the case of a digraph. Strategies to obtain doubly stochastic matrices (in a centralized or distributed manner) for a given topology have been studied under the umbrellas of weight balancing and matrix scaling, which are explicitly discussed in Chapter 5.

In the class of algorithms we present next, we adopt Assumption [A6], and assume that each node $v_j$ knows its out-degree $d_j^+$ (i.e., the number of nodes it transmits information to). This requirement is rather mild, as, in most protocols for ad-hoc network discovery, each node not only knows which nodes it receives information from but also which nodes it transmits information to.
3.4.1 Ratio-Consensus

This section summarizes the *ratio-consensus* algorithm, a distributed algorithm that enables the nodes of a multi-component system to reach average consensus (see, e.g., Domínguez-García and Hadjicostis, 2010; Domínguez-García and Hadjicostis, 2011). For gossiping-type algorithms, a corresponding approach was also proposed in Bénézit *et al.*, 2010, which is a generalization of the algorithm proposed in Kempe *et al.*, 2003; however, the idea of ratio-consensus can be traced back much earlier (see the discussion on weak convergence in Seneta, 2006, pp. 88-89).

The ratio-consensus algorithm performs two iterative computations of the type in (3.3) in parallel, and allows each node to asymptotically obtain the exact average as the ratio of the two state variables each node maintains. The following theorem (see, e.g., Domínguez-García and Hadjicostis, 2010; Domínguez-García and Hadjicostis, 2011) summarizes the basic version of ratio-consensus.

**Theorem 3.1.** Consider a strongly connected digraph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$ where each node $v_j \in V$ has some initial value $V_j$. Each node $v_j$ maintains, at iteration $k$, state variables $y_j[k]$ and $z_j[k]$, and updates them as follows:

\[
y_j[k + 1] = \frac{1}{1 + d_j^+} y_j[k] + \frac{\sum_{v_i \in N^{-}_j} y_i[k]}{1 + d_i^+}, \quad (3.7)
\]

\[
z_j[k + 1] = \frac{1}{1 + d_j^+} z_j[k] + \frac{\sum_{v_i \in N^{-}_j} z_i[k]}{1 + d_i^+}, \quad (3.8)
\]

where $y_j[0] = V_j$, and $z_j[0] = 1$, for all $v_j \in V$. Let $\pi_j[k] := \frac{y_j[k]}{z_j[k]}$; then,

\[
\lim_{k \to \infty} \pi_j[k] = \frac{\sum_{\ell} \frac{y_j[0]}{z_j[0]}}{\sum_{\ell} \frac{V_j}{n}} = \frac{V_j}{n} = \mu, \quad \forall v_j \in V.
\]

**Proof.** Letting $y[k] = [y_1[k], \ldots, y_n[k]]^T$ and $z[k] = [z_1[k], \ldots, z_n[k]]^T$, the iterations in the above theorem can be written in matrix form as

\[
y[k + 1] = P_c y[k],
\]

\[
z[k + 1] = P_c z[k],
\]
where the weight matrix $P_c$ has entries $P_c(l, j) = \frac{1}{1 + d_j}$ for all $v_l \in N_j^+ \cup \{v_j\}$ (zero otherwise).

Notice that $P_c$ is a column stochastic matrix that is also primitive as long as the underlying digraph is strongly connected. By utilizing the result in (3.5) twice (once for each iteration), we realize that the ratio $\pi_j[k]$ asymptotically converges to

$$
\lim_{k \to \infty} \pi_j[k] = \frac{\lim_{k \to \infty} y_j[k]}{\lim_{k \to \infty} z_j[k]} = \frac{1^T y[0]}{1^T z[0]} = \frac{\sum_{\ell} y_{\ell}[0]}{\sum_{\ell} z_{\ell}[0]}
$$

Thus, with the chosen initial conditions, the ratio asymptotically converges to the average. 

In practice, at iteration $k$, node $v_j$ will send to its out-neighbors the values $\frac{y_j[k]}{1 + d_j}$ and $\frac{z_j[k]}{1 + d_j}$. This means that nodes do not necessarily need to know the identities of the nodes they receive values from. However, in addition to Assumptions [A1]–[A5] in Chapter 2, each node $v_j$ needs to know its out-degree, i.e., Assumption [A6] needs to hold. The pseudocode for the ratio-consensus algorithm is included as Algorithm 3.

**Remark 3.4.** By appropriately choosing the initial conditions of the ratio-consensus algorithm, we can compute arbitrary weighted linear combinations of the initial values of the nodes. For example, if $y_j[0] = c_j V_j$ and $z_j[0] = c'_j$, then $\lim_{k \to \infty} \pi_j[k] = \frac{\sum_{\ell} c_{\ell} V_{\ell}}{\sum_{\ell} c'_{\ell}}$ for all $v_j \in V$ (in other words, the ratios converge to a weighted linear combination $\sum_{\ell} w_{\ell} V_{\ell}$ of the initial values, where $w_{\ell} = \frac{c_{\ell}}{\sum_{\ell} c'_{\ell}}$). Another interesting observation is that, since the matrix $P_c$ is column stochastic, we have $\sum_{\ell} y_{\ell}[k] = \sum_{\ell} y_{\ell}[0]$ and $\sum_{\ell} z_{\ell}[k] = \sum_{\ell} z_{\ell}[0]$ for all $k$.

The term ratio-consensus is typically used for the case when the update weights are fixed, so that the two iterations are captured by $y[k + 1] = P_c y[k]$. 

Algorithm 3: Distributed Average via Ratio-Consensus

**Input:** A strongly connected digraph $G = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges (and no self-loops). Each node $v_j \in \mathcal{V}$ has an initial value $V_j$ and knows its out-degree $d_j^+$. 

**Output:** $\mu = \frac{\sum_{v \in \mathcal{V}} V_v}{n}$. 

**Initialization:** Each node $v_j \in \mathcal{V}$ sets $y_j[0] = V_j$ and $z_j[0] = 1$.

**Iteration:** For $k = 0, 1, 2, \ldots$, each node $v_j \in \mathcal{V}$ does the following:

1. It transmits $\frac{y_j[k]}{1+d_j^+}$ and $\frac{z_j[k]}{1+d_j^+}$ to its out-neighbors $v_l \in \mathcal{N}_j^+$. 
2. It receives $\frac{y_i[k]}{1+d_i^+}$ and $\frac{z_i[k]}{1+d_i^+}$ from each in-neighbor $v_i \in \mathcal{N}_j^-$. 
3. It updates its own variables as
   \[ y_j[k+1] = \frac{1}{1+d_j^+} y_j[k] + \sum_{v_i \in \mathcal{N}_j^-} \frac{y_i[k]}{1+d_i^+}, \]
   \[ z_j[k+1] = \frac{1}{1+d_j^+} z_j[k] + \sum_{v_i \in \mathcal{N}_j^-} \frac{z_i[k]}{1+d_i^+}. \]
4. It sets $\pi_j[k+1] = \frac{y_j[k+1]}{z_j[k+1]}$.

and $z[k+1] = P_c z[k]$ for a (time-invariant) weight matrix $P_c$ that is column stochastic. More generally, one can have iterations of the form

\[ y[k+1] = P_c[k] y[k], \]
\[ z[k+1] = P_c[k] z[k], \]

where, at each iteration $k$, the weight matrix $P_c[k]$ is a column stochastic matrix. Under mild conditions on the weight matrices $P_c[k]$, $k = 0, 1, 2, \ldots$, the coupled time-varying iterations above also lead to ratios $\pi_j[k] = \frac{y_j[k]}{z_j[k]}$ for all $v_j \in \mathcal{V}$ that asymptotically converge to the average $\mu$ of the initial values. For example, Conditions [C1] and [C2] in Section 3.2.2 are sufficient conditions to invoke Wolfowitz’s theorem (Theorem 2.3) in Chapter 2 to argue that

\[ \lim_{k \to \infty} \pi_j[k] = \frac{\sum_{\ell} y_{\ell}[0]}{\sum_{\ell} z_{\ell}[0]}, \forall v_j \in \mathcal{V}. \]

This time-varying approach, which does not necessarily lead to steady-state values for $y_j[k]$ and/or $z_j[k]$ (though the ratio $\pi_j[k]$ converges), is referred to as push-sum averaging (Kempe et al., 2003; Bénédit et al., 2010). These
3.4. Distributed Averaging in Digraphs

Techniques are also exploited in Hadjicostis and Charalambous, 2014 and Domínguez-García et al., 2012b; Hadjicostis et al., 2016 in order to achieve asymptotic average consensus despite the presence of, respectively, bounded delays and packet drops in the transmissions between pairs of nodes; this is discussed further in Section 3.5.

The original push-sum proposal in Kempe et al., 2003 operates like gossip: at each iteration \( k \), a pair of nodes \( v_j \) and \( v_l \) is selected (e.g., node \( v_j \) wakes up at some random time instant, and selects randomly one of its out-neighboring nodes \( v_l, v_l \in N_j^+ \)); then node \( v_j \) keeps half of its value and sends half of its value to node \( v_l \). In terms of the linear time-varying coupled iterations above, the update is governed by a column stochastic matrix \( P_c[k] \) defined as follows:

\[
P_c[k](j, j) = \frac{1}{2},
\]

\[
P_c[k](l, j) = \frac{1}{2},
\]

\[
P_c[k](i, i) = 1, \ i \neq j,
\]

with all other entries being zero. As long as the graph is strongly connected and each pair of nodes that is connected is selected with a nonzero probability, it can be shown that the ratios at the nodes will asymptotically become, with probability one, equal to the average of the initial values.

Another approach related to push-sum is the so-called broadcast gossip algorithm in Franceschelli et al., 2011, which runs two coupled time-varying linear iterations (as described above) where, at a particular iteration \( k \), a node \( v_j \) is (randomly or otherwise) selected and sends a fraction of \( \frac{1}{1+d_j^+} \) of its values \( y_j[k] \) and \( z_j[k] \) to all of its out-neighbors (including itself). In other words, matrix \( P_c[k] \) in this case is of the form:

\[
P_c[k](l, j) = \frac{1}{1+d_j^+}, v_l \in N_j^+ \cup \{v_j\},
\]

\[
P_c[k](i, i) = 1, \ i \neq j,
\]

with all other entries being zero. As long as each node \( v_j \in \mathcal{V} \) is selected frequently enough and the underlying graph is strongly connected, broadcast gossip can also be shown to lead to ratios \( \pi_j[k] = \frac{y_j[k]}{z_j[k]} \) that asymptotically converge, with probability one, to the average of the initial values.

**Example 3.2.** We illustrate ratio-consensus strategies for distributed averaging on the strongly connected digraph \( \mathcal{G} = (\mathcal{V}, \mathcal{E}) \) in Figure 3.4. We assume
(again) that the initial values for the five nodes are given by

\[ x[0] = [V_1, V_2, V_3, V_4, V_5]^T \]

\[ = [1, 2, 3, 4, 5]^T, \]

with average \( \mu = 3 \). Figure 3.5 shows 25 iterations of the ratio-consensus algorithm, which uses weights that form the column stochastic weight matrix

\[
P_c = \begin{bmatrix}
1/3 & 0 & 0 & 1/2 & 0 \\
1/3 & 1/3 & 0 & 0 & 0 \\
1/3 & 1/3 & 1/2 & 0 & 1/3 \\
0 & 0 & 0 & 1/2 & 1/3 \\
0 & 1/3 & 1/2 & 0 & 1/3
\end{bmatrix}.
\]

The plots show the progression of \( y_j[k] \) (left), \( z_j[k] \) (middle), and \( \pi_j[k] \) (right), for \( j \in \{1, 2, 3, 4, 5\} \), as a function of the iteration \( k \).

**Figure 3.4:** Strongly connected digraph used to illustrate the execution of averaging schemes.

**Figure 3.5:** Execution of the ratio-consensus algorithm with weights captured by \( P_c \) on the digraph of Figure 3.4, shown in terms of the values \( y_j[k] \) (left), \( z_j[k] \) (middle), and \( \pi_j[k] \) (right), for \( j \in \{1, 2, 3, 4, 5\} \), at each iteration.
Note that there is nothing that prohibits us from running ratio-consensus on the undirected graph in Figure 3.1. In that case the matrix \( P_{c,un} \) is given by

\[
P_{c,un} = \begin{bmatrix}
0.25 & 0.25 & 0.25 & 1/3 & 0 \\
0.25 & 0.25 & 0.25 & 0 & 0.25 \\
0.25 & 0.25 & 0.25 & 0 & 0.25 \\
0.25 & 0 & 0 & 1/3 & 0.25 \\
0 & 0.25 & 0.25 & 1/3 & 0.25 \\
\end{bmatrix},
\]

and a sample run is shown in Figure 3.6. One should notice that ratio-consensus converges faster than using the strategies described in Example 3.1. In fact, \( \lambda_2(P_c) = 0.3977 \), which is smaller in magnitude than both \( \lambda_2(P_{eq}) \) and \( \lambda_2(P_{metro}) \) of the averaging strategies that were studied in Example 3.1.

![Figure 3.6](image)

**Figure 3.6:** Execution of the distributed averaging algorithm using ratio-consensus on the undirected graph in Figure 3.1.

Figure 3.7 illustrates sample runs of distributed averaging based on the push-sum strategy. At each iteration \( k \), a node \( v_j \) wakes up and selects randomly one of its out-neighboring nodes \( v_l \) \( (v_l \in \mathcal{N}^+_j) \); then node \( v_j \) keeps half of its value and sends half of its value to node \( v_l \). For example, if node \( v_4 \) selects its out-neighbor \( v_1 \) (it does not have any other choice), then the update
matrix at iteration $k$ is

$$P[k] = \begin{bmatrix}
1 & 0 & 0 & 1/2 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1/2 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}. $$

The plots show the progression of $y_j[k]$ (left), $z_j[k]$ (middle), and $\pi_j[k]$ (right), for $j \in \{1, 2, 3, 4, 5\}$, as a function of iteration $k$. At each iteration, matrix $P[k]$ takes one of $|E| = 8$ different values, depending on which pair of nodes is selected. It is worth noting that the values of $y_j$ and $z_j$ do not converge but their ratios do converge (with probability one, assuming that each edge in the graph is selected with nonzero probability, as long as the graph is strongly connected).

**Figure 3.7:** Execution of the push-sum algorithm on the directed graph of Figure 3.4, shown in terms of the values $y_j[k]$ (left), $z_j[k]$ (middle), and $\pi_j[k]$ (right) for $j \in \{1, 2, 3, 4, 5\}$, at each iteration.

Finally, a sample run of broadcast gossip is shown in Figure 3.8. At each iteration $k$, a node (e.g., $v_j$) wakes up and sends to all of its out-neighbors (including itself) a fraction $\frac{1}{1+d_j^F}$ of its value. For example, if node $v_2$ is broadcasting at iteration $k$, then the update matrix is

$$P[k] = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1/3 & 0 & 0 & 0 \\
0 & 1/3 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 1/3 & 0 & 0 & 1
\end{bmatrix}. $$
The plots show the progression of $y_j[k]$ (left), $z_j[k]$ (middle), and $\pi_j[k]$ (right), for $j \in \{1, 2, 3, 4, 5\}$, as a function of iteration $k$. At each iteration, matrix $P[k]$ takes one of $|\mathcal{V}| = 5$ possible different values, depending on which node is broadcasting. Again, the values of $y_j$ and $z_j$ do not converge but their ratios do converge (with probability one, assuming that each node in the graph is selected with nonzero probability, as long as the graph is strongly connected). As expected, broadcast gossip is faster than push-sum (since the latter is constrained to pairwise interactions).

![Figure 3.8](image-url) Figure 3.8: Execution of the broadcast gossip algorithm on the directed graph of Figure 3.4, shown in terms of the values $y_j[k]$ (left), $z_j[k]$ (middle), and $\pi_j[k]$ (right), for $j \in \{1, 2, 3, 4, 5\}$, at each iteration.

### 3.4.2 Distributed Averaging by Distributed Weight Adjustment

Clearly, one strategy for achieving asymptotic average consensus is to first obtain a set of weights that form a doubly stochastic matrix, $P_d$, and then use these weights to perform a linear iteration of the type discussed in Section 3.2.1. Distributed strategies for obtaining a set of weights that form a doubly stochastic matrix $P_d$ are discussed in Chapter 5 in the context of weight balancing. However, such algorithms are typically asymptotic, which makes their use for reaching asymptotic average consensus challenging. An approach that overcomes this problem was proposed in Domínguez-García and Hadjicostis, 2013. The algorithm proposed in that reference operates by having each node adjust the weights on its outgoing links (so as to asymptotically reach weights that form a doubly stochastic matrix $P_d$), while at the same time also performing an iteration that allows the nodes to reach asymptotic average consensus. More specifically, the weights are updated, in a distributed fashion,
so that the weight matrix $P[k]$ at each iteration is a column stochastic matrix and reaches asymptotically a primitive doubly stochastic matrix, i.e.,

$$\lim_{k \to \infty} P[k] = P_d.$$ 

Under these conditions, we can invoke Lemma 2.4 in Chapter 2 to establish that the iteration

$$x[k + 1] = P[k]x[k],$$

with $x[0] = [V_1, V_2, \ldots, V_n]^T$, will allow the nodes to reach asymptotic average consensus.

The keys in the algorithm proposed in Domínguez-García and Hadjicostis, 2013 is to ensure that the weights are chosen, in a distributed manner, so that all weight matrices $P[k]$ are column stochastic, and eventually lead to a doubly stochastic matrix $P_d$. As in the case of ratio-consensus, the algorithm assumes that each node $v_j$ knows its out-degree $d_j^+$, and maintains, at each iteration $k$, equal weights on its outgoing edges, $p_{lj}[k]$, $\forall v_l \in \mathcal{N}_j^+$, and a self-weight, $p_{jj}[k]$, so that the sum of all these weights is unity (if all nodes do this, the resulting weight matrix $P[k]$ at each iteration will be column stochastic). In order to eventually reach weights that form a doubly stochastic matrix, each node $v_j$ takes into account the sum of the weights on its incoming edges, which is given by

$$s_j^-[k] := \sum_{v_i \in \mathcal{N}_j^-} p_{ji}[k],$$

and the sum of the weights on its outgoing edges, which is given by

$$s_j^+[k] := \sum_{v_l \in \mathcal{N}_j^+} p_{lj}[k] = d_j^+ p_{lj}$$

(since all weights on its outgoing edges are equal). Then, node $v_j$ updates the weights on its outgoing edges (increasing them or decreasing them by an equal amount), so as to drive the sum $s_j^+$ closer to $s_j^-$ (and below unity). Node $v_j$ also adjusts its self-weight so that the sum of the weights on its outgoing edges and its self-weight is unity. The procedure described above is outlined in detail in Algorithm 4.
Algorithm 4: Distributed Average via Adjustment of Weights to Form a Doubly Stochastic Weight Matrix

**Input:** A strongly connected digraph $G(V, E)$ with $n = |V|$ nodes and $m = |E|$ edges (and no self-loops). Each node $v_j$ has an initial value $V_j$ and knows its out-degree $d_j^+$.  

**Output:** $\mu = \frac{1}{n} \sum_{v \in V} V_v$.  

**Initialization:** Each node $v_j$ initializes the following variables:  
1) It sets $p_{lj}[0] = 1/(1 + d_j^+)$, $\forall v_l \in N_j^+ \cup \{v_j\}$.  
2) It chooses $\alpha_j \in (0, 1)$.  
3) It sets $x_j[0] = V_j$.  

**Iteration:** For $k = 0, 1, 2, \ldots$, each node $v_j \in V$ does the following:  

Step 1: It transmits $x_j[k], p_{lj}[k]$ to each of its out-neighbors $v_l \in N_j^+$.  

Step 2: It receives $x_i[k], p_{ji}[k]$ from each of its in-neighbors $v_i \in N_j^-$.  

Step 3: It updates $x_j[k+1] = \sum_{v_i \in N_j^- \cup \{v_j\}} p_{ji}[k] x_j[k]$.  

Step 4: It chooses $\beta_j[k]$ as follows:  
\[
\beta_j[k] = \begin{cases} 
\frac{1-s_j^-[k]}{s_j^-[k] - s_j^+[k]}, & s_j^-[k] > s_j^+[k], \\
\alpha_j, & \text{otherwise}. 
\end{cases} \tag{3.10}
\]

5) It updates  
\[
p_{lj}[k+1] = p_{lj}[k] + \beta_j[k] \left( \frac{s_j^-[k]}{d_j^+} - p_{lj}[k] \right), v_l \in N_j^+. \tag{3.11}
\]

6) It assigns $p_{jj}[k+1] \geq 0$ as follows:  
\[
p_{jj}[k+1] = 1 - \sum_{v_l \in N_j^+} p_{lj}[k+1]. \tag{3.12}
\]
**Remark 3.5.** This is the same update as in (5.4) of Algorithm 12 (which is discussed in the context of weight balancing in Chapter 5), with the difference that the proportionality constant $\beta_j$ can be adapted at each time step $k$, and is chosen so that $s_j^+[k+1] \leq 1$ (so as to ensure that $p_{jj}[k+1]$ can be chosen in Step 3 to be nonnegative and satisfy $p_{jj}[k+1] + s_j^+[k+1] = 1$).

**Proposition 3.1.** If a digraph is strongly connected or is a collection of strongly connected digraphs, then Algorithm 4 reaches some steady-state weights $p_{ji}$, i.e.,

$$\lim_{k \to \infty} p_{ji}[k] = p_{ji},$$

such that the limiting weight matrix $P_d = [p_{ji}]$ is doubly stochastic. Furthermore, the weights of all edges (including self-edges) in the graph (assumed to be strongly connected in Assumption [A1] in Chapter 2) are nonzero, which implies that $P_d$ is primitive.

**Proof.** See Charalambous and Hadjicostis, 2013; Domínguez-García and Hadjicostis, 2013.

---

**3.4.3 Distributed Averaging using a Surplus Variable**

When trying to perform distributed averaging on digraphs using a linear iteration of the type discussed in Section 3.2.1, but without weights that form a doubly stochastic matrix, one of the main difficulties is that the state sum $\sum_{\ell=1}^{n} x_{\ell}[k]$ does not necessarily remain invariant, which, in turn, can result in losing track of the average of the initial values $\mu := \sum_{\ell} x_{\ell}[0] / n$. To alleviate this problem, the authors in Cai and Ishii, 2012 proposed an approach in which, apart from variable $x_{j}[k]$ (initially set to $x_{j}[0] = V_{j}$) an additional variable $s_{j}[k]$, called *surplus*, is introduced for each node $v_{j}$ (initially set to $s_{j}[0] = 0$). Nodes use and update their surpluses to track their value changes and to ensure that $\sum_{\ell}(x_{\ell}[k] + s_{\ell}[k]) = \sum_{\ell} x_{\ell}[0] = \sum_{\ell} V_{\ell}$ for all time steps $k$. The key novelty of this approach is to augment an additional variable for each agent whose purpose is to locally record individual state updates.

There are three operations that every node $v_{j}$ performs at each time step $k$:

1. First, node $v_{j}$ sends its state information $x_{j}[k]$ and weighted surplus $b_{lj}s_{j}[k]$ to each out-neighbor $v_{l} \in \mathcal{N}_{j}^{+}$. The *sending weight* $b_{lj}$ is assumed to satisfy $b_{lj} \in (0, 1)$ if $v_{l} \in \mathcal{N}_{j}^{+}$, $b_{lj} = 0$ if $v_{l} \notin \mathcal{N}_{j}^{+}$, and
3.4. Distributed Averaging in Digraphs

\[ \sum_{v_l \in \mathcal{N}_j^+} b_{lj} < 1. \text{ [For instance, if node } v_j \text{ knows its out-degree, then it can choose } b_{lj} = \frac{1}{1+d_j}.] \]

2. Second, node \( v_j \) receives state information \( x_i[k] \) and weighted surplus \( b_{ji} s_i[k] \) from each in-neighbor \( v_i \in \mathcal{N}_j^- \).

3. Third, node \( v_j \) updates its own state \( x_j[k] \) and surplus \( s_j[k] \) as follows:

\[
x_j[k+1] = x_j[k] + \sum_{v_i \in \mathcal{N}_j^-} a_{ji} (x_i[k] - x_j[k]) + \epsilon s_j[k], \tag{3.13}
\]

\[
s_j[k+1] = \left( (1 - \sum_{v_l \in \mathcal{N}_j^+} b_{lj}) s_j[k] + \sum_{v_i \in \mathcal{N}_j^-} b_{ji} s_i[k] \right) - \left( x_j[k+1] - x_j[k] \right), \tag{3.14}
\]

where the updating weights \( a_{ji} \)'s are assumed to satisfy that \( a_{ji} \in (0, 1) \) if \( v_i \in \mathcal{N}_j^- \), \( a_{ji} = 0 \) if \( v_j \not\in \mathcal{N}_j^- \), and \( \sum_{v_i \in \mathcal{N}_j^-} a_{ji} < 1 \); in addition, the parameter \( \epsilon \) is a positive number which specifies the amount of surplus used to update the state.

The pseudocode for the procedure described above is provided in Algorithm 5.

**Remark 3.6.** Note that parameter \( \epsilon \) affects the convergence of Algorithm 5. It must be chosen sufficiently small, and a valid bound for its value involves nonlocal information of the digraph. This problem is overcome by computing a bound offline, and making each node aware of the value of \( \epsilon \).

For more technical details and simulations, the interested reader can see Cai and Ishii, 2012.

3.4.4 Distributed Averaging via left Perron-Frobenious eigenvector estimation

In Priolo et al., 2014 a distributed algorithm is proposed in which the average consensus over any strongly connected weighted digraph is computed, by concurrently running an estimation procedure for the computation of the left
Algorithm 5: Distributed Average using a Surplus Variable

**Input:** A strongly connected digraph $G(V, E)$ with $n = |V|$ nodes and $m = |E|$ edges (and no self-loops). Each node $v_j$ has an initial value $V_j$ and knows an upper bound of its out-degree $d^+_j$ and the surplus parameter $\epsilon$.

**Output:** $\mu = \frac{\sum_{v_l \in V} V_l}{n}$.

**Initialization:** Each node $v_j$ initializes the following variables:

1) It sets $b_{lj} \in (0, 1), \forall v_l \in \mathcal{N}^+_j$ such that $\sum_{v_l \in \mathcal{N}^+_j} b_{lj} < 1$.

2) It chooses $a_{ji} \in (0, 1), \forall v_i \in \mathcal{N}^-_j$ such that $\sum_{v_i \in \mathcal{N}^-_j} a_{ji} < 1$.

3) It sets $x_j[0] = V_j$.

**Iteration:** For $k = 0, 1, 2, \ldots$, each node $v_j \in V$ does the following:

Step 1: It transmits $x_j[k], b_{lj}s_j[k]$ to each of its out-neighbors $v_l \in \mathcal{N}^+_j$.

Step 2: It receives $x_i[k], b_{ji}s_i[k]$ from each of its in-neighbors $v_i \in \mathcal{N}^-_j$.

Step 3: It updates

$$x_j[k+1] = x_j[k] + \sum_{v_i \in \mathcal{N}^-_j} a_{ji}(x_i[k] - x_j[k]) + \epsilon s_j[k]$$

$$s_j[k+1] = \left(1 - \sum_{v_l \in \mathcal{N}^+_j} b_{lj}\right)s_j[k] + \sum_{v_i \in \mathcal{N}^-_j} b_{ji}s_i[k]$$

$$- \left(x_j[k+1] - x_j[k]\right).$$
eigenvector associated with the zero eigenvalue of the Laplacian matrix and for which agents are not required to be aware of their out-neighbors.

The main idea of the approach is as follows. If the weights (and interconnections) in iteration (3.1) are not changing over time, the distributed consensus algorithm becomes $x[k + 1] = Px[k]$, with $x[0] = [V_1, V_2, \ldots, V_n]^T$, as described in (3.4). When nodes weigh the incoming messages $x_i[k]$ from their in-neighbors with $p_{ji}$ such that $\sum_{i \in N_j} p_{ji} = 1$, $\forall v_j \in \mathcal{V}$, and the digraph is strongly connected, then matrix $P$ is primitive and row stochastic. Hence, by the Perron-Frobenius Theorem (see Theorem 2.2 in Chapter 2),

$$
\lim_{k \to \infty} x[k] = \frac{1}{n} w^T 1_n x[0];
$$

(3.15)

thus, the nodes reach consensus to

$$
\bar{\mu} := \frac{1}{n} \lim_{k \to \infty} 1_n^T x[k]
$$

$$
= \frac{1}{n w^T 1_n} w^T x[0]
$$

$$
= \bar{w}^T x[0]
$$

$$
= \sum_{v_j \in \mathcal{V}} \bar{w}_j x_j[0],
$$

(3.16)

where $\bar{w} := [\bar{w}_1 \bar{w}_2 \ldots \bar{w}_n]^T$ is the normalized left eigenvector of the Laplacian matrix of the digraph. In order to reach the average, the actual initial condition $x[0]$ should be modified as $\tilde{x}[0] = x[0] + \Gamma$ so that

$$
\mu = \frac{1}{n} \sum_{v_j \in \mathcal{V}} x_j[0]
$$

$$
= \frac{1}{n} \sum_{v_j \in \mathcal{V}} \bar{w}_j (x_j[0] + \Gamma_j)
$$

(3.17)

where $\Gamma := [\Gamma_1 \Gamma_2 \ldots \Gamma_n]^T$ is the extra term, and according to (3.17),

$$
\Gamma_j = x_j[0] \left( \frac{1}{n \bar{w}_j} - 1 \right).
$$

Let each node have a variable $\delta_j$ with $\delta_j[0] = 1$. The iteration $\delta[k + 1] = P\delta[k]$, according to (3.15), $\delta_j[k]$ will converge to the normalized left
eigenvector \( \tilde{w}_i \). Let \( \tilde{\Gamma}_j[k] \) denote the estimate of \( \Gamma_j \) by using \( \delta_j[k] \) instead of \( \tilde{w}_j \), i.e.,

\[
\tilde{\Gamma}_j[k] := x_j[0] \left( \frac{1}{n\delta_j[k]} - 1 \right).
\]

Let also \( \varepsilon_j[k] := \tilde{\Gamma}_j[k] - \tilde{\Gamma}_j[k-1], \tilde{\Gamma}_j[-1] = 0 \forall j \in \mathcal{V} \). Then, iteration (3.1) is modified such that

\[
x[k+1] = P(x[k] + \varepsilon[k]),
\]

where \( \varepsilon := [\varepsilon_1 \varepsilon_2 \ldots \varepsilon_n]^T \).

The pseudocode for the procedure described above is provided in Algorithm 6.

For more technical details and simulations, the interested reader can see Priolo et al., 2014.

3.5 Overcoming Transmission Delays and Packet Drops

In this section, we relax Assumption [A4] in Chapter 2 and consider distributed average consensus strategies when communication links are not perfect, i.e., they might delay the delivery of a message by a (fixed or time-varying) bounded delay, or they may completely drop a packet. We first discuss the modeling of transmission delays and packet drops, and then develop distributed algorithms capable of reaching average consensus despite their presence.

3.5.1 Modeling Transmission Delays and Packet Drops

Transmission delays and packet drops might arise due to a variety of reasons (e.g., asynchronous operation or imperfectly synchronized clocks, unequal propagation/computational delays, network congestion, or other factors).

Transmission Delay Model

Transmission delays could be fixed and known in some special cases, but, more generally, they could be time-varying and a priori unknown (but bounded). 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
3.5. Overcoming Transmission Delays and Packet Drops

Algorithm 6: Distributed Average via Left Perron-Frobenious Eigen-vector Estimation

**Input:** A strongly connected digraph $G(V, E)$ with $n = |V|$ nodes and $m = |E|$ edges (and no self-loops).

**Output:** $\mu = \frac{\sum_{V \in V} V}{n}$.

**Initialization:** Each node $v_j$ initializes the following variables:

1) It chooses $p_{ji} \in (0, 1)$, $\forall v_i \in N_j^-$ such that $\sum_{v_i \in N_j^-} p_{ji} = 1$.
2) It sets $x_j[0] = V_j$, $\delta_j[0] = 1$, and $\bar{\Gamma}_j[-1] = 0$.

**Iteration:** For $k = 0, 1, 2, \ldots$, each node $v_j \in V$ does the following:

Step 1: It transmits $x_j[k]$, $\delta_j[k]$ to each of its out-neighbors $v_l \in N_j^+$.

Step 2: It receives $x_i[k]$, $\delta_i[k]$ from each of its in-neighbors $v_i \in N_j^-$. 

Step 3: It updates

$$\bar{\Gamma}_j[k] = x_j[0] \left( \frac{1}{n\delta_j[k]} - 1 \right),$$

$$\varepsilon_j[k] = \bar{\Gamma}_j[k] - \bar{\Gamma}_j[k - 1],$$

$$x_j[k + 1] = \sum_{v_i \in N_j^- \cup \{v_j\}} p_{ji}[k](x_i[k] + \varepsilon_i[k]),$$

$$\delta_j[k + 1] = \sum_{v_i \in N_j^- \cup \{v_j\}} p_{ji}[k]\delta_i[k].$$
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 at the ensuing iteration without delay).

In general, a delay profile might be available in some cases, i.e., we might have knowledge of the probability mass functions that describe the probability with which each \( \tau_{ji}[k] \) takes a particular value in the set \( \{0, 1, 2, \ldots, \bar{\tau}_{ji}\} \). A typical (straightforward) assumption would be for delays between different links and different iterations to be independent random variables. In the discussions below, the probabilistic profile of delays will not come into play because we are only interested in establishing asymptotic average consensus (for which the boundedness of delays is sufficient); however, the delay profile of each link certainly plays a role in the performance (e.g., in the convergence rate).

**Packet Drop Model**

To model packet drops, we assume that a transmission from node \( v_i \) to node \( v_j \) at a particular iteration \( k \) could be successful or unsuccessful. To make the discussion precise, at instant \( k \), we let \( \{\delta_{ji}[k] \mid (v_j, v_i) \in \mathcal{E}\} \), be indicator variables that take value \( \delta_{ji}[k] = 1 \) if the message from node \( v_i \) is received by node \( v_j \) at time instant \( k \) (i.e., there is no packet drop); otherwise, \( \delta_{ji}[k] = 0 \) (i.e., there is a packet drop).

For simplicity of our discussions later on, we assume that the successes of transmissions between different time steps are independent, but there could be dependencies between the random variables at each step. More specifically, if we let \( \Delta[k] \) be a random binary vector that contains the variables \( \{\delta_{ji}[k] \mid (v_j, v_i) \in \mathcal{E}\} \) in some arbitrary ordering, we assume that the random vectors \( \Delta[0], \Delta[1], \ldots, \Delta[k], \ldots \), are independent identically distributed (i.i.d.). Notice that we require that no random variable \( \delta_{ji}[k] \) is identically zero (i.e., there is a positive probability that \( \delta_{ji}[k] = 1 \)); this will be a sufficient condition for establishing asymptotic convergence.

**Remark 3.7.** One can combine the two models described above to analyze the case when transmissions could be dropped with some probability, or could be completed after some bounded delay (with a certain delay profile). This case can be treated by variations of the techniques we describe below; however, we chose not to explicitly address this case, as it would make our discussions (and notation) unnecessarily complicated.
3.5. Overcoming Transmission Delays and Packet Drops

3.5.2 Overcoming Transmission Delays

Here, we present a variation of the ratio-consensus algorithm presented in Section 3.4.1 that overcomes bounded (but \textit{a priori} unknown) transmission delays. The ratio-consensus algorithm is basically modified to process iteration values as soon as the packets containing them arrive at their destination. Note that, under our model for communication delays, a node could receive multiple packets from the same neighbor at the same iteration (these packets are sent at different time instances and are delayed by different amounts). The only additional requirement for asymptotic convergence of the ratio to the average is that delays are identical at each of the two iterations (e.g., both values are transmitted on the same packet).

More formally, we run ratio-consensus as in Section 3.4.1 with each node \( v_j \) broadcasting the values \( y_j[k] \) and \( z_j[k] = \frac{1}{1+d_j} \) to all of its out-neighbors \( v_l \in \mathcal{N}_j^+ \), at each iteration \( k \). These two values reach the out-neighbors of node \( v_j \) possibly at different iterations, depending on the delays incurred at the various communication links. At the same iteration \( k \), node \( v_j \) also receives (possibly delayed) values from all of its in-neighbors, which it processes as soon as they arrive. As in the basic version of the ratio-consensus algorithm, the initial values are set to \( y_j[0] = V_j \) and \( z_j[0] = 1 \). The following proposition, taken from Hadjicostis and Charalambous, 2011 formalizes this discussion.

**Proposition 3.2** (Hadjicostis and Charalambous, 2011). Consider a strongly connected digraph \( G(\mathcal{V}, \mathcal{E}) \), where each node \( v_j \in \mathcal{V} \) has some initial value \( V_j \). Let \( y_j[k] \) and \( z_j[k] \) (for all \( v_j \in \mathcal{V} \) and \( k = 0, 1, 2, \ldots \)) be the result of the iterations

\[
y_{j}[k+1] = p_{jj}y_{j}[k] + \sum_{v_i \in \mathcal{N}_j^-} \sum_{r=0}^{\hat{r}} p_{ji}y_{i}[k-r]I_{k-r,ji}[r], \tag{3.18a}
\]

\[
z_{j}[k+1] = p_{jj}z_{j}[k] + \sum_{v_i \in \mathcal{N}_j^-} \sum_{r=0}^{\hat{r}} p_{ji}z_{i}[k-r]I_{k-r,ji}[r], \tag{3.18b}
\]

where

1) the weight matrix \( P = [p_{ji}] \) is a primitive column stochastic matrix (e.g., for each \( v_j \in \mathcal{V} \), we could set \( p_{lj} = \frac{1}{1+d_j} \), for \( v_l \in \mathcal{N}_j^+ \cup \{v_j\} \));

2) \( y[0] = [V_1, V_2, \ldots, V_n]^T \) and \( z[0] = 1_n \); and
3) $I_{k,ji}$ is an indicator function that captures the bounded delay $\tau_{ji}[k] \leq \bar{\tau}$ ($\bar{\tau} < \infty$) on link $(v_j, v_i)$ at iteration $k$, defined as

$$I_{k,ji}[\tau] = \begin{cases} 1, & \text{if } \tau_{ji}[k] = \tau, \\ 0, & \text{otherwise.} \end{cases}$$

(3.19)

[Note that $y[k]$ and $z[k]$ are taken to be zero for negative $k$ in (3.18).]

Then, we have

$$\lim_{k \to \infty} \pi_j[k] = \frac{\sum_{\ell=0}^{\bar{\tau}} V_{\ell}}{n} = \mu, \quad \forall v_j \in \mathcal{V},$$

where

$$\pi_j[k] := \frac{y_j[k]}{z_j[k]}.$$

**Remark 3.8.** Note that the inner sums in the pair of iterations in (3.18) effectively sum up all the values contained in packets that arrive at node $v_j$ from node $v_i$ at iteration $k$ (and might have been delayed by $r = 0, 1, \ldots, \tau$ iteration steps).

Reference Hadjicostis and Charalambous, 2011 used a graph augmentation technique to prove the above proposition. The augmented graph representation adds extra “virtual” nodes and uses them to model the effect of delays. In particular, at most $\bar{\tau}$ “virtual” nodes $v_j^{(1)}, v_j^{(2)}, \ldots, v_j^{(\bar{\tau})}$ are introduced for each node $v_j \in \mathcal{V}$ (where $\tau$ is the maximum delay): node $v_j^{(d)}$ holds information that is destined to arrive at node $v_j$ after $d$ iterations. The augmented digraph has $(\bar{\tau} + 1)|\mathcal{V}|$ nodes and $(1 + \bar{\tau})|\mathcal{E}| + \bar{\tau}|\mathcal{V}|$ edges (an example can be seen in Figure 4.5) as follows:

- **Type 1 Edges:** For each $j \in \{1, 2, \ldots, n\}$, the set of edges $(v_j^{(\bar{\tau} - 1)}, v_j^{(\bar{\tau})}), (v_j^{(\bar{\tau} - 2)}, v_j^{(\bar{\tau} - 1)}), \ldots, (v_j^{(1)}, v_j^{(\bar{\tau})})$ are added in the augmented graph.

- **Type 2 Edges:** For each $(v_j, v_i) \in \mathcal{E}$ in the original graph, the edges $(v_j^{(1)}, v_i), (v_j^{(2)}, v_i), \ldots, (v_j^{(\bar{\tau})}, v_i)$ are added in the augmented graph.

In the above augmented graph, we have a total of $(\bar{\tau} + 1)n$ nodes. We can track the evolution of the state variables in (3.18a)–(3.18b) in the augmented graph via

$$\psi[k + 1] = \Xi[k] \psi[k],$$

(3.20)

$$\phi[k + 1] = \Xi[k] \phi[k],$$

(3.21)
where
1) $\psi[k] = \left[ y^T[k], \ y^{(1)}[k], \ldots, y^{(\tau)}[k] \right]^T$ and $y^{(r)}[k] = \left[ y_1^{(r)}[k], \ldots, y_n^{(r)}[k] \right]$,
   $r = 1, 2, \ldots \tau$ (at initialization, $y[0] = [V_1, \ldots, V_n]^T$ and $y^{(1)}[0] = \ldots = y^{(\tau)}[0] = 0$);
2) $\phi[k] = \left[ z^T[k], \ z^{(1)}[k], \ldots, z^{(\tau)}[k] \right]^T$ and $z^{(r)}[k] = \left[ z_1^{(r)}[k], \ldots, z_n^{(r)}[k] \right]$,
   $r = 1, 2, \ldots \tau$ (at initialization, $z[0] = 1_n$ and $z^{(1)}[0] = \ldots = z^{(\tau)}[0] = 0$);
3) The matrix $\Xi[k]$ is a column stochastic matrix that depends on the delays incurred to transmissions at iteration $k$ and is given by

$$\Xi[k] := \begin{bmatrix}
P_0[k] & I_{n \times n} & 0 & \ldots & 0 \\
0 & P_1[k] & I_{n \times n} & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & I_{n \times n} & 0 \\
P_{\tau-1}[k] & 0 & \ldots & 0 & 0
\end{bmatrix}.$$  \hspace{1cm} (3.22)

Note that $P_0[k]$, $P_1[k]$, $\ldots$, $P_\tau[k]$ are appropriately defined nonnegative matrices that depend on the link delays that are experienced by messages sent at time $k$. Specifically, $P_r[k]$ is a matrix associated only with the links of the graph for which the message is being delayed by $r$ steps at time step $k$, and satisfies

$$P_r[k](j, i) = \begin{cases} P(j, i), & \text{if } \tau_{ji}[k] = r, \ (v_j, v_i) \in \mathcal{E}, \\
0, & \text{otherwise.} \end{cases}$$

Note that, for each $(v_j, v_i) \in \mathcal{E}$, only one of $P_0[k](j, i)$, $P_1[k](j, i)$, $\ldots$, $P_\tau[k](j, i)$ is nonzero and is equal to $P(j, i)$. It is easily deduced that, at each iteration $k$, $\Xi[k]$ is a column stochastic, nonnegative matrix. Thus, we have a time-varying ratio-consensus iteration as in (3.9). It can be shown that Conditions [C1] and [C2] hold, thus the ratio $\tau_j[k]$ at each node converges to the average of the initial values.

Remark 3.9. The essential difference between the standard ratio-consensus in Section 3.4.1 and the variant that overcomes bounded time delays in this section is that the version in this section effectively operates with time-varying weights on the virtual topology. These time-varying weights depend on the set of edges that are “active” at each iteration. Specifically, at each iteration $k$, all of Type 1 Edges in the augmented graph are “active,” but only one (or none) of Type 2 Edges is “active,” depending on the delay $\tau_{ji}[k]$: edge
Distributed Asymptotic Average Consensus

$(v_j, v_i)$ is selected if there is no delay, whereas edge $(v_j^{(d)}, v_i)$ is selected if $\tau_{ji}[k] = d). The fact that weights are time-varying implies that, in the presence of packet drops, $y_j[k]$ and $z_j[k]$ do not converge in general (though their ratio $\pi_j[k] = \frac{y_j[k]}{z_j[k]}$ converges to the average).

3.5.3 Overcoming Packet Drops

In this section we first present the running-sum ratio-consensus algorithm in Domínguez-García et al., 2012b; Hadjicostis et al., 2016, which is a variation of the ratio-consensus algorithm in Section 3.4.1, and can be used to overcome packet drops.

The main idea in the running-sum algorithm is that, instead of sending $\frac{y_j[k]}{1+d_j^+}$ and $\frac{z_j[k]}{1+d_j^+}$, node $v_j$ sends to all of its out-neighbors the $y$-running sum and the $z$-running sum, given by

$$\sigma_j[k + 1] := \sum_{t=0}^{k} \frac{y_j[t]}{1 + d_j^+},$$
$$\eta_j[k + 1] := \sum_{t=0}^{k} \frac{z_j[t]}{1 + d_j^+}.$$

Clearly, each receiving node can recover $\frac{y_j[k]}{1+d_j^+}$ and $\frac{z_j[k]}{1+d_j^+}$, by taking the differences $\sigma_j[k + 1] - \sigma_j[k]$ and $\eta_j[k + 1] - \eta_j[k]$ (i.e., the difference between two consecutive receptions from node $v_j$). Thus, apart from sending/receiving $y$- and $z$-running sums, the ratio-consensus algorithm could be run in exactly the same way (by first recovering the needed values at each iteration), modulo some additional bookkeeping at each node.

In practice, when executing the running-sum variation of ratio consensus, each node $v_j$ needs to track its own running-sums (the two values it sends to its out-neighbors) and also maintain the values of the running-sums for each of its in-neighbors; the update procedure is summarized next.

- **Sending:** Each node $v_j$ maintains and broadcasts $\sigma_j[k+1] = \sum_{t=0}^{k} \frac{y_j[t]}{1+d_j^+}$, referred to as the $y$-running sum, and $\eta_j[k + 1] = \sum_{t=0}^{k} \frac{z_j[t]}{1+d_j^+}$, referred to as the $z$-running sum. Both running-sums are initialized at zero, i.e.,
\[
\sigma_j[0] = \eta_j[0] = 0, \forall v_j \in \mathcal{V}, \text{ and can be updated recursively as}
\]
\[
\begin{align*}
\sigma_j[k + 1] &= \sigma_j[k] + y_j[k] / (1 + d_j^+), \\
\eta_j[k + 1] &= \eta_j[k] + z_j[k] / (1 + d_j^+).
\end{align*}
\]

**Receiving:** In addition to state variables \(y_j[k], z_j[k], \sigma_j[k],\) and \(\eta_j[k],\) each node \(v_j\) also maintains state variables, \(\rho_{ji}[k],\) for each of its in-neighbors \(v_i \in \mathcal{N}_j^-\). As seen below, \(\rho_{ji}[k]\) keeps track of the \(y\)-running sum, last received at node \(v_j\) from node \(v_i,\) with \(\rho_{ji}[0] = 0, \forall v_i \in \mathcal{N}_j^-\). Similarly, \(\nu_{ji}[k]\) keeps track of the \(z\)-running sum, last received at node \(v_j\) from node \(v_i,\) with \(\nu_{ji}[0] = 0, \forall v_i \in \mathcal{N}_j^-\).

**Computing:** Given the availability of running-sums, the updates for the running-sum ratio-consensus algorithm can be written as
\[
\begin{align*}
y_j[k + 1] &= \sum_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} (\rho_{ji}[k + 1] - \rho_{ji}[k]), \\
z_j[k + 1] &= \sum_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} (\nu_{ji}[k + 1] - \nu_{ji}[k]),
\end{align*}
\]
where \(\rho_{ji}[k]\) and \(\nu_{ji}[k]\) were defined above (and \(\rho_{jj}[k] = \sigma_j[k],\) and \(\nu_{jj}[k] = \eta_j[k]\)). [Note that if there are no packet drops, we have \(\rho_{ji}[k] = \sigma_i[k]\) and \(\nu_{ji}[k] = \eta_i[k]\) for all \(v_i \in \mathcal{V}\) and all \(k,\) which implies that the above iterations reduce to the iterations in Theorem 3.1.]

**Remark 3.10.** The values of running-sums grow in an unbounded manner in an implementation of the running-sum ratio-consensus algorithm. This issue does not really cause problems because running-sums grow linearly with the number of iterations whereas convergence is exponential with the number of iterations.

In the presence of packet drops, some of the transmissions may be unsuccessful. This means that, at some particular iteration \(k,\) node \(v_j\) may not receive the values \((\sigma_i[k + 1], \eta_i[k + 1])\) that its in-neighbor \(v_i \in \mathcal{N}_j^-\) tries to
send (this will happen if there is a packet drop on the link from node $v_i$ to node $v_j$ at iteration $k$, which we denote by $\delta_{ji}[k] = 0$). In such case, node $v_j$ assumes that $\rho_{ji}[k+1] = \rho_{ji}[k]$ and $v_{ji}[k+1] = v_{ji}[k]$, i.e., it maintains the previous values for $\rho$ and $\nu$). In other words, we have

$$
\rho_{ji}[k+1] = \begin{cases} 
\sigma_i[k+1], & \text{if } \delta_{ji}[k] = 1, \\
\rho_{ji}[k], & \text{if } \delta_{ji}[k] = 0, 
\end{cases}
$$

$$
v_{ji}[k+1] = \begin{cases} 
\eta_i[k+1], & \text{if } \delta_{ji}[k] = 1, \\
v_{ji}[k], & \text{if } \delta_{ji}[k] = 0. 
\end{cases}
$$

The formal description of the running-sum algorithm for distributed averaging in the presence of packet drops is presented as Algorithm 7 below. In Hadjicostis et al., 2016 the following theorem is proved.

**Theorem 3.2** (Hadjicostis et al., 2016). Consider the execution of Algorithm 7 in a strongly connected digraph $G = (V, E)$, where each node $v_j \in V$ has initial value $V_j$. Assume that a transmission from node $v_i$ to node $v_j$ at iteration $k$ is successful (i.e., $\delta_{ji}[k] = 1$) with a nonzero probability, for all $(v_j, v_i) \in E$. Then, the execution of Algorithm 7 results, with probability one, in

$$
\lim_{k \to \infty} \pi_j[k] = \frac{\sum_{v_l \in V} V_l}{|V|} = \mu,
$$

for all $v_j \in V$.

### 3.6 Privacy and Security Considerations

There are several ways in which the techniques for distributed averaging of this chapter can be enhanced. In this section, we provide pointers for two such issues, privacy preservation in the presence of curious nodes, and the handling of faulty/malicious nodes.

#### 3.6.1 Privacy Preservation

Privacy-preserving average consensus considers the following setting: a set of interconnected components, each with some value, aim to compute the average of their values in a distributed manner, while at the same time preserving their privacy from curious (but properly performing) peers. [Curious nodes
Algorithm 7: Distributed Average via Running-Sum Ratio-Consensus for Handling Packet Drops

**Input:** A strongly connected digraph $G = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges (and no self-loops). Each node $v_j \in \mathcal{V}$ has some initial value $V_j$ and knows its out-degree $d_j^+$.

**Output:** $\mu = \frac{1}{n} \sum_{v \in \mathcal{V}} V_v$.

**Initialization:** Each node $v_j \in \mathcal{V}$ sets

\[
\begin{align*}
y_j[0] &= V_j, \\
\sigma_j[0] &= 0, \quad \text{and} \quad \rho_{ji}[0] = 0, \forall v_i \in \mathcal{N}_j^- , \\
z_j[0] &= 1, \\
\eta_j[0] &= 0, \quad \text{and} \quad \nu_{ji}[0] = 0, \forall v_i \in \mathcal{N}_j^- .
\end{align*}
\]

**Iteration:** For $k = 0, 1, 2, \ldots$, each node $v_j \in \mathcal{V}$ does the following:

**Step 1:** It computes

\[
\begin{align*}
\sigma_j[k+1] &= \sigma_j[k] + y_j[k]/(1 + d_j^+) , \\
\eta_j[k+1] &= \eta_j[k] + z_j[k]/(1 + d_j^+) .
\end{align*}
\]

**Step 2:** It broadcasts $\sigma_j[k+1]$ and $\eta_j[k+1]$ to all $v_l \in \mathcal{N}_j^+$.

**Step 3:** It receives $\sigma_i[k+1]$ and $\eta_i[k+1]$ if $\delta_{ji}[k] = 1$ from each $v_i \in \mathcal{N}_j^-$ and updates

\[
\begin{align*}
\rho_{ji}[k+1] &= \begin{cases} 
\sigma_i[k+1] , & \text{if } \delta_{ji}[k] = 1 \text{ or } i = j , \\
\rho_{ji}[k] , & \text{if } \delta_{ji}[k] = 0 ,
\end{cases} \\
\nu_{ji}[k+1] &= \begin{cases} 
\eta_i[k+1] , & \text{if } \delta_{ji}[k] = 1 \text{ or } i = j , \\
\nu_{ji}[k] , & \text{if } \delta_{ji}[k] = 0 .
\end{cases}
\end{align*}
\]

**Step 4:** It updates its own variables as

\[
\begin{align*}
y_j[k+1] &= \sum_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} (\rho_{ji}[k+1] - \rho_{ji}[k]) , \\
z_j[k+1] &= \sum_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} (\nu_{ji}[k+1] - \nu_{ji}[k]) .
\end{align*}
\]

**Step 5:** It sets $\pi_j[k+1] = \frac{y_j[k+1]}{z_j[k+1]}$.
are not malicious in the sense that they execute the distributed protocol as expected; in particular, they make no attempt to alter the outcome of the distributed computation.] Initial studies have been able to demonstrate some of the tradeoffs involved between cooperativeness and privacy, using notions of observability in Manitara and Hadjicostis, 2013 and differential privacy in Huang et al., 2012; Kia et al., 2015.

A simple transformation strategy for privacy preservation was proposed in Kefayati et al., 2007 using random offset values. Specifically, each node $v_j$ that wishes to protect its privacy adds a random offset value $O_j$ to its initial value $V_j$; thus, the nodes execute the distributed averaging protocol with values $\{V'_j := V_j + O_j \mid v_j \in \mathcal{V}\}$. The key observation is that, when an infinite number of nodes employ the protocol, their offsets will have a zero net effect on the average, allowing the nodes to converge to the true average of the initial values of the nodes in the network. To see this, notice that, following any of the asymptotic strategies for distributed averaging outlined in this chapter, the nodes will reach consensus to

$$\frac{\sum_\ell V'_\ell}{n} = \frac{\sum_\ell (V_\ell + O_\ell)}{n} = \frac{\sum_\ell 1 V_\ell}{n} + \frac{\sum_\ell O_\ell}{n},$$

where $\mu$ is the desirable (exact) average of the original values, and $O$ is a random variable that captures the net effect of the offsets. Assuming the $O_j$’s are zero mean random variables, then $O$ will also be zero mean; in fact, under a broad range of conditions, the variance of $O$ will go to zero, as $n$ goes to infinity. For example, if the $O_j$’s are i.i.d. random variables with zero mean and finite variance $\sigma^2$ (as proposed in Kefayati et al., 2007), then $O$ will be a zero mean random variable with variance $\frac{\sigma^2}{n}$; thus, as $n$ increases, the variance of $O$ goes to zero, allowing the nodes to converge to the true average. However, for a finite number of nodes, the accuracy of this method will be compromised.

The work in Manitara and Hadjicostis, 2013 proposes an alternative strategy where each node adds a random offset to its initial value, as well as random offsets on the result of its computation at each iteration during the execution of the protocol. The key difference from the approach in Kefayati et al., 2007, however, is that each node eventually subtracts the accumulated sum of the
offsets it has added, allowing the nodes to converge to the exact average of their initial values. The scheme that it is used in Manitara and Hadjicostis, 2013 assumes that the underlying network forms a strongly connected undirected graph, and makes use of linear iterations as in (3.3), where the weights \( p_{ji} \) form a doubly stochastic matrix \( P = [p_{ji}] \). The main difference is that node \( v_j \) preserves its privacy by (i) setting its initial value to \( x_j[0] = V_j + O_j \) (where \( O_j \) is some random offset), and (ii) subsequently updating its value as

\[
x_j[k + 1] = p_{jj}[k]x_j[k] + \sum_{v_i \in \mathcal{N}_j} p_{ji}x_i[k] + u_j[k], \quad k = 0, 1, \ldots,
\]

where \( u_j[k] \) is a pseudo-random value chosen by node \( v_j \) at time-step \( k \). The constraint is that \( u_j[k] = 0 \) for \( k > L_j \) (for some \( L_j \) known only to node \( v_j \)) and

\[
u_j[L_j] = - \sum_{k=0}^{L_j-1} u_j[k] - O_j.
\]

Thus, at time-step \( L_j \), node \( v_j \) effectively cancels out the pseudo-random value it has added initially, and the pseudo-random values added at subsequent steps during the information exchange in the network up to that point.

The work in Manitara and Hadjicostis, 2013 established topological conditions that ensure a certain degree of privacy for the nodes that follow the above protocol, despite the presence of curious nodes in the network. The work in Mo and Murray, 2017 proposes a strategy in which nodes asymptotically subtract their initial offsets, and characterizes the mean square convergence rate and the covariance matrix of the maximum likelihood estimate on the initial state.

More recently, approaches that try to preserve privacy via homomorphic encryption have also been proposed (e.g., Freris and Patrinos, 2016); however, a key obstacle is the need to have a node that is universally trusted.

### 3.6.2 Overcoming Faulty/Malicious Behavior

The presence of faulty (respectively, malicious) components, which inadvertently (respectively, purposely) alter the result of their computations while executing a distributed averaging computation, can significantly complicate the task. On one hand, the desire to obtain the average necessitates sharing of information and a certain level of trust to other components, while resiliency
against faulty or malicious peers implies the need to operate under the presumption that some of them may be misbehaving. Byzantine consensus Vaidya et al., 2012 and resilient in-network consensus LeBlanc et al., 2013 allow the components to compute a value (not necessarily the exact average) that is, respectively, approximately the same for normal components, or within the range of values of normal components.

When aiming to calculate the exact average of the initial values in a distributed manner, one can utilize the results in Sundaram and Hadjicostis, 2008, at least if the network is time-invariant and assuming that some preprocessing is done by the nodes at initialization. Consider a distributed control system with time-invariant communication topology, captured by a digraph $G = (\mathcal{V}, \mathcal{E})$. Each node $v_j$ has some initial value, $x_j[0]$, and, at each time-step $k$, follows a linear iteration as in (3.3). If we let

$$S_j = \{ v_i \mid \text{There exists a path from } v_i \text{ to } v_j \text{ in } G \} \cup \{ v_j \},$$

then Sundaram and Hadjicostis, 2008 establishes that, for almost any choice of weight matrix $P = [p_{ji}]$ (such that $p_{ji} = 0$ if $i \neq j$ and $(v_j, v_i) \notin \mathcal{E}$), node $v_j$ can obtain the values $\{ x_i[0] \mid x_i \in S_j \}$, after running the linear iteration for at most $|S_j|$ steps (and can therefore calculate any arbitrary function of these values). The constructions outlined in Sundaram and Hadjicostis, 2008 relate the row space of certain observability matrices to the calculation of linear functions of the data using observability theory for linear structured systems Dion et al., 2003.

The authors of Sundaram and Hadjicostis, 2011 model a faulty or malicious node $v_j$ in a very general way, allowing it to produce, at any given iteration $k$, the erroneous value

$$x_j[k + 1] = p_{jj}x_j[k] + \sum_{v_i \in \mathcal{V}_j^{-}} p_{ji}x_i[k] + e_j[k],$$

for some error term $e_j[k]$ that is either (pseudo-) random (in the faulty node case), or chosen in a specific manner, perhaps conspiring with other nodes (in the malicious node case). Thus, given a set of faulty (or malicious) nodes $F = \{ i_1, i_2, \ldots, i_f \}$, the iteration can be captured concisely via

$$x[k + 1] = P x[k] + B_F e[k],$$

(3.25)
where $x[k] = [x_1, x_2, \ldots, x_n]^T$, $e[k] = [e_{i_1}[k], e_{i_2}[k], \ldots, e_{i_f}[k]]^T$, and $B_F = [u_{i_1}, u_{i_2}, \ldots, u_{i_f}]$ with $u_{i_j}$ being a unit vector of dimension $n$ with a single nonzero entry with value “1” at the $i_j$ entry.

If we let $f$ denote the maximum number of malicious nodes that are to be tolerated in the network, and $C_{ji}$ denote the size of the smallest $(j, i)$-cut between any two nodes $v_j$ and $v_i$, then, regardless of the actions of the malicious nodes, node $v_j$ can uniquely determine all of the initial values in the network if and only if $\min_i \{C_{ji}\} \geq 2f + 1$ Sundaram and Hadjicostis, 2011. Furthermore, if this condition is satisfied, $v_j$ will be able to recover the initial values after the nodes run the linear iterative strategy, with almost any choice of weights, for at most $n$ time-steps.

Similar ideas have also been pursued in other contexts. For example, resiliency to faulty sensors/actuators in linear systems has been studied in Fawzi et al., 2014, resiliency in consensus computations has been studied in Pasqualetti et al., 2012, and fault detection/identification in cyber-physical systems has been studied in Pasqualetti et al., 2013.

One limitation of the techniques in Sundaram and Hadjicostis, 2011 is that they require the set of weights that the various nodes will use in their linear updates to be determined ahead of operation, which makes it difficult for such techniques to accommodate system or network changes (because they would require re-calculation or re-transmission of these weights). However, when the network remains invariant, Sundaram and Hadjicostis, 2011 presented a simple distributed scheme for the nodes to randomly choose their own weights and then recover the set of weights (or rather functions of the set of weights) that they need. A distributed approach that allows nodes to obtain the needed set of weights in the possible presence of malicious nodes is an open question, perhaps utilizing some of the techniques in Ho et al., 2008, which were developed in a related setting but for data that take values over finite fields.

In the above works, the structure of the cyber layer plays a decisive role for resiliency to faulty/malicious components. More generally, however, one has to investigate the role of the combined structures of the cyber and the physical layers, as well as the interactions among them. In fact, if one is interested in assessing the capabilities of malicious (possibly sophisticated) entities that might try to inflict damage to a system of interest, one has to
consider strategies that combine actions in the cyber and/or physical networks. Some efforts in this direction have been made in the context of power systems in Mohajerin Esfahani et al., 2010; Sou et al., 2011.

3.7 Discussion

A number of researchers have analyzed the behavior of linear iterations for (average) consensus in the presence of fixed or time-varying delays. Consensus protocols, for example (based on a single linear iteration as in (3.1) or (3.3)), have been shown to be relatively robust to disturbances due to delays (e.g., Fang and Antsaklis, 2005; Angeli and Bliman, 2006; Bliman and Ferrari-Trecate, 2008): nodes asymptotically reach consensus to the same value though this consensus value could be affected by the delays (and how they manifest themselves during the execution of the protocol).

Average consensus has been proven more challenging, particularly for directed graphs, because of the need to converge to the exact average of the initial values. The work in Patterson et al., 2007 assumes that the communication topology is captured by an undirected graph: when a communication link fails, it affects communication in both directions, and nodes can detect it and compensate for it; effectively, the nodes adjust the weights by ignoring the link that failed at that particular iteration and by choosing weights on the remaining links that form a doubly stochastic matrix (since all communication links are bidirectional, this can be done in a straightforward manner, similar to the one in Kingston and Beard, 2006 for time-varying topologies). Also focusing on undirected graph topologies, Kar and Moura, 2009; Kar and Moura, 2010 consider a much more general setting of random link failures, including noisy links and links that introduce quantization noise; the authors of Kar and Moura, 2009; Kar and Moura, 2010 do not aim for convergence to the exact average, but study instead the tradeoff between bias and variance after running a (possibly time-varying) consensus algorithm for a finite number of iterations.

The work in Fagnani and Zampieri, 2009 considers digraphs and proposes two compensation methods to account for communication link failures; however, these strategies do not necessarily lead to the exact average. The work in Chen et al., 2010 proposes a strategy that uses special corrective iterations to compensate for packet losses, and the work in Eyal et al., 2012 considers
distributed averaging with dynamic sensor measurements in asynchronous networks with undirected communication links. The results in Eyal et al., 2012 establish convergence to the correct average once the network settles (i.e., no updates in the values and no topology changes). The work in Cai and Ishii, 2014 has addressed the problem of exact average consensus over time-varying digraphs using “surplus” variables at each node; in that sense it extends the approach in Cai and Ishii, 2012, described in Section 3.4.3 of this chapter, to a setting that allows for packet dropping links.

In a directed graph setting, a key assumption made in most of the algorithms presented in this chapter is that each node \( v_j \) knows its out-degree \( d_j^+ \) (Assumption [A6] in Chapter 2. This assumption is removed in Hadjicostis et al., 2016, which proposes a scheme that combines running-sum ratio-consensus with flooding techniques; to achieve this, the approach requires each node in the network to be associated with a unique id (as well as additional memory, transmissions, and bookkeeping).

Having presented the four main approaches proposed in the literature for reaching average consensus asymptotically in directed topologies, we provide below a table comparing and summarizing the features of these methods: [1] refers to Domínguez-García and Hadjicostis, 2010, [2] refers to Domínguez-García and Hadjicostis, 2013, [3] refers to Franceschelli et al., 2011, and [4] refers to Priolo et al., 2014. Note that ratio consensus as described in Section 3.4.1 requires that each node \( v_j \) knows its out-degree \( d_j^+ \); however, the variation in Hadjicostis et al., 2016 mentioned earlier does not require knowledge of the out-degrees (at the cost of additional bookkeeping).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Asymptotic</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Global info</td>
<td>None</td>
<td>None</td>
<td>Surplus gain</td>
<td>None</td>
</tr>
<tr>
<td>Out-degree</td>
<td>No</td>
<td>Yes</td>
<td>Upper bound</td>
<td>No</td>
</tr>
<tr>
<td>Additional synchronization</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Handles delays</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Handles packet losses</td>
<td>Yes</td>
<td>—</td>
<td>—</td>
<td>No</td>
</tr>
<tr>
<td>Handles switchings</td>
<td>Yes</td>
<td>Yes</td>
<td>—</td>
<td>No</td>
</tr>
</tbody>
</table>
In this chapter, we describe strategies for distributed average consensus that, unlike the asymptotic strategies in Chapter 3, complete in finite time. In particular, we present strategies which provide the exact average in finite time, as well as (much simpler) strategies that provide an approximation of the average in finite time.

### 4.1 Introduction and Motivation

One of the main advantages of the popular approaches for distributed average consensus discussed in Chapter 3 is that using simple local rules nodes are able to calculate the average. However, the applicability of most of these algorithms for distributed averaging to real-world coordination and control applications is inhibited by the fact that they can only guarantee asymptotic convergence; thus, finite-time (average) consensus algorithms are, in general, more desirable. Besides finite-time convergence, it is reported that closed-loop systems under finite-time control usually demonstrate better disturbance rejection properties Bhat and Bernstein, 2000; this can be important in applications where the averaging operation is a first step towards more involved control or coordination tasks (see the applications in power systems in Chapter 6).
Finite-time consensus with continuous-time dynamics is investigated in Cortés, 2006; Hui et al., 2008; Wang and Xiao, 2010. Herein, and following the philosophy of this monograph, we focus on finite-time average consensus in systems with discrete-time dynamics. Typically, finite-time average consensus is handled by pre-determining the number of steps that all nodes in the network will perform, such that by the end of these iterations they have values sufficiently close to the average. This approach, however, requires some prior knowledge about the network and the convergence rate of the iteration process (in the case of the time-invariant iteration in (3.1), for example, that would require knowledge of the eigenvalue $\lambda_2$ that has the second largest magnitude). This chapter discusses approaches that do not require such prior network knowledge and still allow the nodes to reach either exact or approximate convergence to the average. More specifically, we divide these approaches into two main approaches: (i) those that aim at computing the exact average (see Section 4.2), and (ii) those that aim at approximating the average within an a priori specified precision (see Section 4.3).

### 4.2 Exact Finite-Time Average Consensus

A strand of research is based on factorization of the averaging matrix, $\frac{1}{n}1_n1_n^T$, to orchestrate update matrices at design time, so that average consensus is achieved in finite-time (see, e.g., Ko, 2010; Georgopoulos, 2011; Kibangou, 2012; Hendrickx et al., 2014). In other words, one aims for finding a finite number of matrices, $P[0], P[1], ..., P[k_f]$, all adhering to the given structure of the communication topology, such that

$$\frac{1}{n}1_n1_n^T = P[k_f]P[k_f-1]...P[1]P[0].$$

These matrices define the weights that will be used at each iteration of the form $x[k + 1] = P[k]x[k]$ (with $x[0]$ being initial values of the nodes), such that $x[k_f + 1] = \mu 1_n$ where $\mu$ is the average (i.e., average consensus is reached at iteration $k_f + 1$. Note, however, that knowledge of the network topology is required to compute the matrices $P[0], P[1], ..., P[k_f]$.

Another strand of research aims to take advantage of the minimal polynomial of a matrix. In this regard, Sundaram and Hadjicostis, 2007 proposed a distributed method to compute the asymptotic final consensus value in finite-time; this method requires nodes to have enough memory to store previous
state values, knowledge of an upper bound on the total number of nodes, and computing power to check rank conditions on matrices. Specifically, by running a linear iteration of the form $x[k + 1] = Px[k]$, with $x[0]$ being the initial values and $P$ being a doubly stochastic matrix (so that asymptotically the nodes reach average consensus), these methods allow each node $v_j$ to track its own values (for at most $k_0 \leq n$ steps, where $n$ is the number of nodes) and use the values 

$$\{x_j[0], x_j[1], x_j[2], \ldots x_j[k_0]\}$$


to calculate the average $\mu$.

Following this line of research, Yuan et al., 2009; Yuan et al., 2013 proposed a distributed algorithm with which any component can compute the average consensus value in a minimum number of steps and without requiring any knowledge about the total number of nodes in the network. All the aforementioned works have considered average consensus in undirected graphs and assume synchronized operation, and timely exchange of information between neighboring components.

In this section, extending the work of Yuan et al., 2009; Yuan et al., 2013, we present a distributed protocol that allows nodes to find the exact average of the initial values in a finite and minimum number of steps on interconnection topologies described by strongly connected digraphs. In addition, the average can be obtained in a finite number of steps even when the information exchange is subject to delays. This algorithm, appearing in Charalambous et al., 2015, is the only one in the literature so far that allows for distributed computation of the exact average in digraphs in finite time, even in the presence of delays.

### 4.2.1 Delay-Free Protocol

In what follows, we propose an algorithm that is based on (a) the ratio consensus algorithm discussed in Chapter 3 (see Domínguez-García and Hadjicostis, 2010; Domínguez-García and Hadjicostis, 2011) and (b) the distributed algorithm proposed in Yuan et al., 2009; Yuan et al., 2013 in which any arbitrarily chosen node in an undirected graph can compute its asymptotic final consensus value in a finite number of steps.

The proposed algorithm is based on letting the nodes execute the ratio-consensus Algorithm 3 (see Theorem 3.1) and store a sequence of consecutive
iterates, based on which each node can compute \( \mu_j := \lim_{k \to \infty} \pi_j[k] \) in a minimum number of steps.

The result hinges on the use of the concept of the minimal polynomial associated with the linear dynamics of each of (3.7) and (3.8), in conjunction with the final value theorem. For this reason, we first provide definitions on minimal polynomials that are essential for the development of the analysis.

**Definition 4.1.** (Minimal polynomial of a matrix) The minimal polynomial associated with a matrix \( P \in \mathbb{R}^{n \times n} \), denoted by

\[
q(t) = t^{D+1} + \sum_{i=0}^{D} \alpha_i t^i,
\]

is the monic polynomial of minimum degree \((D + 1)\) that satisfies \( q(P) = 0_{n \times n} \).

Note that the minimal polynomial will have a degree at most \( n \) (i.e., \( D + 1 \leq n \)), which means that \( q(t) \), if not the same, divides the characteristic polynomial \( \chi(t) \) of a matrix \( P \), given by \( \chi(t) := \det(P - tI) \).

**Definition 4.2** (Minimal polynomial of a matrix pair). The minimal polynomial associated with the matrix pair \([P, e_j^T]\), \( P \in \mathbb{R}^{n \times n} \), \( e_j \in \mathbb{R}^n \), denoted by \( q_j(t) = t^{M_j+1} + \sum_{i=0}^{M_j} \alpha_i^{(j)} t^i \), \( \alpha_i^{(j)} \in \mathbb{R} \), is the monic polynomial of minimum degree \( M_j + 1 \) that satisfies \( e_j^T q_j(P) = 0_n^T \).

Considering the iteration in (3.7) (or (3.8)), which can be written in matrix form as \( y[k+1] = Py[k] \) (or \( z[k+1] = Pz[k] \)) for an appropriately defined column stochastic matrix \( P \), it is easy to show that

\[
\sum_{i=0}^{M_j+1} \alpha_i^{(j)} w_j[k + i] = 0, \quad \forall k \in \mathbb{Z}_+ ,
\]

where \( \alpha_{M_j+1}^{(j)} = 1 \) (and \( w_j \) is either \( y_j \) or \( z_j \)). Let us now denote the \( z \)-transform of \( w_j[k] \) as \( W_j(z) \triangleq \mathcal{Z}(w_j[k]) \). From (4.2) and the time-shift property of the \( z \)-transform, it is easy to show (see Yuan et al., 2009; Yuan et al., 2013) that

\[
W_j(z) = \frac{\sum_{i=1}^{M_j+1} \alpha_i^{(j)} \sum_{\ell=0}^{i-1} w_j[\ell] z^{i-\ell}}{q_j(z)}.
\]
If the network is strongly connected, the minimal polynomial of \([P, e_j^T]\), \(q_j(z)\), does not have any unstable poles apart from one at 1; thus, we can then define the following polynomial:

\[
p_j(z) := \frac{q_j(z)}{z - 1} = \sum_{i=0}^{M_j} \beta_i^{(j)} z^i.
\] (4.4)

The application of the final value theorem Yuan et al., 2009; Yuan et al., 2013 yields:

\[
\phi_y(j) = \lim_{k \to \infty} \ y_j[k] = \lim_{z \to 1} (z - 1) Y_j(z) = \frac{y_{M_j}^T \beta_j}{1^T \beta_j}, \quad (4.5a)
\]

\[
\phi_z(j) = \lim_{k \to \infty} \ z_j[k] = \lim_{z \to 1} (z - 1) Z_j(z) = \frac{z_{M_j}^T \beta_j}{1^T \beta_j}, \quad (4.5b)
\]

where

\[
y_{M_j}^T = [y_j[0], y_j[1], \ldots, y_j[M_j]],
\]

\[
z_{M_j}^T = [z_j[0], z_j[1], \ldots, z_j[M_j]],
\]

and \(\beta_j\) is the vector of coefficients of the polynomial \(p_j(z)\), defined in (4.4), i.e., \(\beta_j^T = [\beta_0^{(j)}, \ldots, \beta_{M_j}^{(j)}]\). The next question is how one can obtain the coefficient vector \(\beta_j\) that is needed for the computation of the final values in (4.5a) and (4.5b). Consider the vectors of \(2k + 1\) successive discrete-time values at node \(v_j\), given by

\[
y_{2k}^T = [y_j[0], y_j[1], \ldots, y_j[2k]],
\]

\[
z_{2k}^T = [z_j[0], z_j[1], \ldots, z_j[2k]],
\]

for the two iterations \(y_j[k]\) and \(z_j[k]\) at node \(v_j\) (as given in iterations (3.7) and (3.8)), respectively. Let us define their associated Hankel matrices as

\[
\Gamma\{y_{2k}^T\} \triangleq \begin{bmatrix}
y_j[0] & y_j[1] & \ldots & y_j[k] \\
\vdots & \vdots & \ddots & \vdots \\
y_j[k] & y_j[k+1] & \ldots & y_j[2k]
\end{bmatrix},
\]

\[
\Gamma\{x_{2k}^T\} \triangleq \begin{bmatrix}
z_j[0] & z_j[1] & \ldots & z_j[k] \\
\vdots & \vdots & \ddots & \vdots \\
z_j[k] & z_j[k+1] & \ldots & z_j[2k]
\end{bmatrix}.
\]
We also consider the vector of differences between successive values of $y_j[k]$ and $z_j[k]$:
\[
\begin{align*}
\bar{y}_{2k}^T &= \left[ y_j[1] - y_j[0], \ldots, y_j[2k+1] - y_j[2k] \right], \\
\bar{z}_{2k}^T &= \left[ z_j[1] - z_j[0], \ldots, z_j[2k+1] - z_j[2k] \right].
\end{align*}
\]

It was shown in Yuan et al., 2013 that $\beta_j$ can be computed as the kernel of the first defective Hankel matrices $\Gamma\{\bar{y}_{2k}^T\}$ and $\Gamma\{\bar{z}_{2k}^T\}$ for arbitrary initial conditions except a set of initial conditions with zero Lebesgue measure.\(^1\)

Next, we provide one of the main results in Charalambous et al., 2015, in which it is stated that the exact average $\mu$ can be distributively obtained in minimum number of steps in strongly connected digraphs.

**Theorem 4.1.** Consider a strongly connected graph $G(V, E)$. Let $y_j[k]$ and $z_j[k]$ (for all $v_j \in V$ and $k = 0, 1, 2, \ldots$) be the result of the iterations (3.7) and (3.8), where $P = [p_{ji}] \in \mathbb{R}_{+}^{n \times n}$ is a set of weights that adhere to the graph structure and form a primitive column stochastic weight matrix. Then, the limit of the linear iteration (which corresponds to the consensus value) can be distributively obtained in minimum number of steps at each node $v_j$, by computing
\[
\mu_j := \lim_{k \to \infty} \frac{y_j[k]}{z_j[k]} = \frac{\phi_y(j)}{\phi_z(j)} = \frac{y_{M_j}^T \beta_j}{z_{M_j}^T \beta_j},
\]
where $\phi_y(j)$ and $\phi_z(j)$ are given, respectively, by equations (4.5a) and (4.5b), and $\beta_j$ is the vector of coefficients, as defined in (4.4).

**Proof.** The consensus value of node $v_j$ for each of the iterations (3.7) (with initial condition $y[0] = [V_1, V_2, \ldots, V_n]^T$) and (3.8) (with initial condition $z[0] = 1_n$) is found by (4.5a) – (4.5b). Note that the vector $\beta_j$ does not depend on the initial conditions and hence it is the same for each node for every initial condition (except for initial conditions in a set of Lebesgue measure zero).

---

\(^{1}\)The Lebesgue measure is a way of assigning a measure to subsets of $n$-dimensional Euclidean space. A subset of $\mathbb{R}^n$ is said to have Lebesgue measure zero if, for every $\varepsilon > 0$, it can be covered with countably many products of $n$ intervals whose total volume is at most $\varepsilon$. All countable sets and all the subsets of $\mathbb{R}^n$ whose dimension is smaller than $n$ have Lebesgue measure zero in $\mathbb{R}^n$. 
Hence,

\[
\lim_{k \to \infty} y_j[k] = \frac{\phi_y(j)}{\phi_z(j)} = \frac{y_{M_j}^T \beta_j}{z_{M_j}^T \beta_j}.
\]

But, from ratio consensus and Theorem 3.1, we already know that, for all \( v_j \in \mathcal{V} \), we have

\[
\lim_{k \to \infty} \pi_j[k] = \lim_{k \to \infty} y_j[k]/z_j[k] = \sum_{v_l \in \mathcal{V}} y_l[0]/|\mathcal{V}| = \mu.
\]

Hence,

\[
\frac{y_{M_j}^T \beta_j}{z_{M_j}^T \beta_j} = \mu. \tag*{\square}
\]

**Remark 4.1.** It has been shown in Yuan et al., 2009 that the number of steps required for predicting \( y \) and \( z \) are less than \( 2n \), where \( n \) is the number of nodes in the network. Thus, an upper bound on the network size immediately yields an upper bound on the storage requirements and the convergence time of the algorithm. However, in many cases we do not need to store this many values: as soon as the square Hankel matrices lose rank, the nodes can stop storing information. Throughout this work we assume that nodes can store as many values as necessary to obtain the resulting defective matrix.

In Algorithm 8, we provide the pseudocode for an algorithm that, by leveraging the results presented earlier, allows the nodes to distributively compute the exact average of the initial values in a finite number of steps.

**Remark 4.2.** Note that the nodes cannot necessarily stop iterating as soon as they compute the exact average, because other nodes may require more steps to compute their Hankel matrices. If the nodes had knowledge of the number of nodes \( n \) in the network (or an upper bound \( n' \)), then each node could stop iterating after \( 2n \) \((2n')\) steps from the time it started the iterations.

**Example 4.1.** Consider the directed network shown in Figure 4.1 where each node \( v_j \) chooses its weight and the weights of its outgoing links to be \((1 + d_j^+)\)^{-1} (such that the sum of all weights assigned by each node \( v_j \) is equal to 1). When each node updates its information state using (3.7) – (3.8), the

\footnote{The special structure of Hankel matrices allows for efficient methods to compute their rank. For example, in Sendra and Llovet, 1992 a modular method is proposed with complexity \( O(r^2) \), where \( r \) is the order of the matrix.}
4.2. Exact Finite-Time Average Consensus

Algorithm 8: Distributed Minimum-Time Average Consensus in Di-graphs

**Input:** A strongly connected digraph $G(V, E)$ with $n = |V|$ nodes and $m = |E|$ edges.

**Data:** Successive observations for $y_j[k]$ and $z_j[k], \forall v_j \in V,$ $k = 0, 1, 2, \ldots,$ using iterations (3.7) and (3.8), with initial conditions $y[0] = [V_1, V_2, \ldots, V_n]^T$ and $z[0] = 1_n$, respectively.

For $k = 0, 1, 2, \ldots,$ each node $v_j \in V$ does the following:

**Step 1:** It runs the ratio consensus algorithm (Algorithm 3) and stores the vectors of differences $y_{M,j}^T$ and $z_{M,j}^T$ between successive values of $y_j[k]$ and $z_j[k]$, respectively.

**Step 2:** It increases the dimension $k$ of the square Hankel matrices $\Gamma\{y_{M,j}^T\}$ and $\Gamma\{z_{M,j}^T\}$ for each of the iterations; when they lose rank, it stores its first defective matrix.

**Step 3:** When the first defective matrix is available, it finds its kernel $\beta_j = [\beta_0, \ldots, \beta_{M,j-1}, 1]^T$, which gives the values $\phi_y$ and $\phi_z$, via (4.5a) and (4.5b), respectively.

**Step 4:** The average consensus value is computed as

$$\mu_j = \frac{\phi_y(j)}{\phi_z(j)} = \frac{y_{M,j}^T \beta_j}{z_{M,j}^T \beta_j}.$$
Figure 4.1: A digraph consisting of six nodes.

information update for the whole network is given by \( y[k + 1] = P y[k] \) and \( z[k + 1] = P z[k] \), where

\[
P = \begin{bmatrix}
\frac{1}{4} & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 \\
\frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 \\
\frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & 0 \\
\frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{2} & 0 & \frac{1}{3} \\
0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{3} \\
0 & 0 & \frac{1}{4} & \frac{1}{2} & 0 & \frac{1}{3}
\end{bmatrix},
\]

with initial conditions \( y[0] = [-1 \ 0 \ 1 \ 2 \ 3 \ 4]^T \) and \( z[0] = 1_6 \).

Since the update matrix is column (but not doubly) stochastic, the iteration (3.7) (or (3.8)) for this network does converge, but not necessarily to the average (as shown in Figure 4.2 for the case when the initial condition is \( y[0] \)). The final consensus vectors \( \phi_y \) and \( \phi_z \), for initial conditions \( y[0] \) and \( z[0] \), respectively, are given by

\[
\phi_y = \begin{bmatrix}
1.6119, & 1.0746, & 0.5373, & 2.4179, & 1.3433, & 2.0149
\end{bmatrix}^T, \\
\phi_z = \begin{bmatrix}
1.0746, & 0.7164, & 0.3582, & 1.6119, & 0.8955, & 1.3433
\end{bmatrix}^T.
\]

Then, each node can compute the exact average as \( \mu_j = \phi_y(j)/\phi_z(j) \). For example, for node \( v_1 \) he have

\[
\mu_1 = \frac{\phi_y(1)}{\phi_z(1)} = \frac{1.6119}{1.0746} = 1.5.
\]
The exact average for $v_1$ is computed in 12 steps (i.e., $2(M_1 + 1) = 2 \times 6 = 12$ steps). See Yuan et al., 2013 for more details on how to compute the minimum number of steps required.

Figure 4.2: Iteration (3.7) with initial condition $y[0] = [-1, 0, 1, 2, 3, 4]^T$ (left) and iteration (3.8) with initial condition $z[0] = 1_6$ (right), for the network in Figure 4.1 do not converge to the average (the average in this case is 1.5).

Figure 4.3: By running two iterations $y[k]$ and $z[k]$ as in (3.7) – (3.8) (using the weight matrix $P$ and initial conditions $y[0]$ and $z[0] = 1_6$, respectively), average consensus is asymptotically reached for the ratio $y_j[k]/z_j[k]$.

The exact average is also justified by running the ratio consensus algorithm showing asymptotic convergence to the exact average (see Figure 4.3). From the simulation we observe that the ratio consensus algorithm after 12 steps has almost converged; more specifically, the values of the ratio at time step 12 are given by $z[12] = [1.4998, 1.4946, 1.4995, 1.5001, 1.5043, 1.5002]^T$, illustrating that the minimum-time needed for distributed computation of the exact average is comparable with the time needed for approximate convergence of the ratio consensus algorithm (at least for this example).

Example 4.2. Next, we consider an example of a larger network, for which asymptotic convergence to the average takes a considerable amount of time.
More specifically, we consider a Leslie matrix (typically used for discrete, age-structured models of population growth) consisting of 100 nodes. The adjacency matrix $P \in \mathbb{R}^{100 \times 100}$ is as follows:

$$
P = \begin{bmatrix}
1/2 & 1/3 & 1/3 & 1/3 & \cdots & 1/3 & 1/2 \\
1/2 & 1/3 & 0 & 0 & \cdots & 0 & 0 \\
0 & 1/3 & 1/3 & 0 & \cdots & 0 & 0 \\
0 & 0 & 1/3 & 1/3 & \cdots & 0 & 0 \\
0 & 0 & 0 & 1/3 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & 1/3 & 0 \\
0 & 0 & 0 & 0 & \cdots & 1/3 & 1/2 \\
\end{bmatrix}.
$$

Due to the structure of the network, the asymptotic convergence time is considerable (see Figure 4.4). By running Algorithm 8, each node can compute the exact average value in a minimum number of steps; for example, node $v_{100}$ computes the average in 34 steps (i.e., $2(M_{100} + 1) = 2 \times 17 = 34$ steps), while the (asymptotic) ratio-consensus algorithm seems to need more than 160 steps for the error to be less than 0.1.

![Figure 4.4: Average consensus is asymptotically reached for the ratio $y_j[k]/z_j[k]$ of each node $v_j$ for a Leslie matrix of size 100.](image)

4.2.2 Delayed Case

In this section, we address the finite-time average consensus problem over directed graphs in the presence of bounded delays in the communication links, arising mainly due to propagation and computational delays. Towards this end, we postulate an asynchronous operation of Algorithm 8, where each agent updates its own value by using delayed information from neighboring nodes.
As in Section 3.5, 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 
\[
0 \leq \tau_{ji}[k] \leq \bar{\tau}_{ji} \leq \bar{\tau} \quad \text{for all} \quad k \geq 0,
\]
for some finite \( \bar{\tau} = \max\{\tau_{ji}\}, \bar{\tau} \in \mathbb{Z}_+ \).

Also, as in Section 3.5, we assume that \( \tau_{jj}[k] = 0, \forall v_j \in \mathcal{V} \), at all time instances \( k \) (i.e., the value of a node is always available to itself without delay). A protocol is employed where each node updates its information state \( w_j[k + 1] \) by combining the available (possibly delayed) information received by its neighbors using constant positive weights \( p_{ji} \). We make use of the algorithm proposed in Hadjicostis and Charalambous, 2011, presented in Section 3.5.2, with which the exact average is asymptotically reached in the presence of bounded delays, even if the digraph is not balanced. In what follows, we consider two cases:

(a) In the first case, the delay is assumed to be time-invariant. We show that the algorithm for computing the ratio consensus value is actually the same as Algorithm 8, however requiring a larger number of observations.

(b) In the second case, the delay is time-varying. By assuming knowledge of an upper bound of the time-varying delays, we can show that the algorithm is the same again as Algorithm 8.

### Time-Invariant Delay

When the delay is time-invariant, and we execute the Algorithm described in Section 3.5.2, matrix \( \Xi[k] \) in (3.20) – (3.21) satisfies \( \Xi[k] = \Xi \), where

\[
\psi[k + 1] = \Xi \psi[k] \quad \text{and} \quad \phi[k + 1] = \Xi \phi[k],
\]

with

\[
\Xi := \begin{bmatrix}
P_0 & I_{n \times n} & 0 & \cdots & 0 \\
P_1 & 0 & I_{n \times n} & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
P_{\bar{\tau} - 1} & 0 & 0 & \cdots & I_{n \times n} \\
P_{\bar{\tau}} & 0 & 0 & \cdots & 0
\end{bmatrix}.
\]

Let \( g(z) \) be the minimal polynomial of \( \Xi \). As proven in Horn and Johnson, 1985, the minimal polynomial of a matrix divides its characteristic polynomial.
Therefore, by direct calculation, we obtain that
\[
\det(zI - \Xi) = \det \left( zI - P_0z^{\bar{\tau}} - \ldots - P_{\bar{\tau}-1}z - P_{\bar{\tau}} \right).
\]

As a consequence, the degree of \( g(z) \) is at most \((\bar{\tau} + 1)n\) (this can be clearly seen by the size of the matrix describing the augmented graph), where \( n \) is the number of nodes in the network. Then, we have the following corollary, which follows from the fact that the maximum number of steps required for a node to compute its final value is 2n.

**Corollary 4.2.** Consider the system in (4.7). Any arbitrarily chosen component \( v_j \) can compute its corresponding final value (provided that the initial value does not belong to the Lebesgue measure zero set) in finite time using at most \( 2(\bar{\tau} + 1)n \) successive values of its own state.

**Remark 4.3.** From Corollary 4.2, it can be deduced that the algorithm for computing the average value in the presence of time-invariant delays is actually the same as Algorithm 8, however requiring a larger number of observations.

**Figure 4.5:** A simple example with two nodes. When the nodes experience no delays the graph is as shown at the top figure. When the nodes experience time-varying delays, then at each time instant \( k \), there will be a path from \( v_1 \) to \( v_2 \), either directly or through the “virtual” nodes of the augmented graph, as shown in the middle figure. For time-invariant graphs the augmented figure remains fixed (e.g., bottom figure for \( \tau_{21} = 2 \) and \( \tau_{12} = 1 \)) for all \( k \).
4.2. Exact Finite-Time Average Consensus

In the time-invariant case, the augmented graph is fixed, and hence, apart from the upper bound given in Corollary 4.2, we can find the exact number of steps required; this is illustrated via an example next.

**Example 4.3.** Consider the network of two nodes exchanging information shown in Figure 4.5. Note that the weights $p_{11}, p_{12}, p_{21},$ and $p_{22}$ are all strictly positive, and satisfy $p_{11} + p_{21} = 1$ and $p_{22} + p_{12} = 1$. Suppose the nodes experience delays that are bounded by $2 (\bar{\tau} = 2)$. Therefore, two extra “virtual” nodes will be added for each node (see Figure 4.5 middle figure), depicting the states at which the delayed messages reside before reaching their destination. However, if a node does not experience the maximum delay, then some of the “virtual” nodes (and therefore links) can be removed.

In the case of two nodes only, the graph representation of the network for which node $v_2$ sends information to node $v_1$ with delay $\tau_{12} = 1$, while node $v_1$ sends information to node $v_2$ with delay $\tau_{21} = 2$ is given in Figure 4.5 (bottom figure). Therefore, the degree of $g(z)$ is less than $(\bar{\tau} + 1)n$. 

**Example 4.4.** Consider again the network in Figure 4.1, for a delay profile that is given by

$$\tau = \begin{bmatrix} 0 & 4 & - & - & 1 & - \\ 5 & 0 & 0 & - & - & - \\ 1 & - & 0 & - & - & - \\ 5 & - & 1 & 0 & - & 4 \\ - & - & - & - & 0 & 0 \\ - & - & 1 & 1 & - & 0 \end{bmatrix},$$

where $\tau_{ji} := \tau(j, i)$, i.e., $\tau(j, i)$ captures the delay with which node $v_i$ sends information to $v_j$. [A “-” is inserted when there does not exist a directed link between two nodes.] Figure 4.6 shows the evolution of the ratio at each node.

The final consensus vectors $\phi_y$ and $\phi_z$, for initial conditions $y[0]$ and $z[0]$, respectively, are calculated using (4.5a) – (4.5b) and are given by

$$\phi_y = [0.7105, \ 0.4737, \ 0.2368, \ 1.0658, \ 0.5921, \ 0.8882]^T,$$

$$\phi_z = [0.4737, \ 0.3158, \ 0.1579, \ 0.7105, \ 0.3947, \ 0.5921]^T.$$
Distributed Finite-Time Average Consensus

Figure 4.6: The ratio $y_j[k]/z_j[k]$ of each node $v_j$ of the iterations converges to the exact average.

Figure 4.7: The iterations (3.18a) with initial condition $y[0] = [-1 \ 0 \ 1 \ 2 \ 3 \ 4]^T$ (left) and iteration (3.18b) with initial condition $z[0] = 1_6$ (right), for the network in Figure 4.1 do not converge.

Figure 4.8: The ratio $y_j[k]/z_j[k]$ of each node $v_j$ of the iterations (3.18a) – (3.18b) converges to the exact average.

Each node can compute the exact average via $\mu_j = \phi_y(j)/\phi_z(j)$, e.g., for node $v_1$, $\mu_1 = \phi_y(1)/\phi_z(1) = 0.7105/0.4737 = 1.5$. For this example, node $v_1$ computes the exact average value in 40 steps. □
4.2. Exact Finite-Time Average Consensus

Time-Varying Delay

It can be observed that the iterations in (3.18a) – (3.18b) themselves, do not necessarily converge in the presence of time-varying delays (see Figure 4.7, for the network in Figure 4.1).

However, the ratio of the iterations (3.18a) – (3.18b) does converge to the exact average (see Figure 4.8). Thus, if one attempts to apply Algorithm 8 in this case for iterations (3.18a) – (3.18b), the method will fail, since neither iteration converges to a final value. In addition, if someone attempts to apply the final value approach Yuan et al., 2013 directly to the ratio value \( \mu_j[k] \), even though the ratio itself converges, the approach initially fails to converge to a final value due to the nonlinearity of the ratio.

If it is possible for each node \( v_j \) to know an upper bound \( \tau_{ji} \) on the delay \( \tau_{ji}[k] \) (i.e., \( \tau_{ji}[k] \leq \tau_{ji} \)) for all the incoming links from the in-neighboring nodes \( v_i \in N_j^- \), then by having the nodes update their value \( \tau_{ji} + 1 \) steps after \( k \) using iterations (3.18a) – (3.18b), it can be easily deduced that the problem reduces to the case of time-invariant delays, studied in Section 4.2.2. In many cases, it is not possible to know an upper bound on the delay of each incoming link of the in-neighbors \( v_i \in N_j^- \), but instead an upper bound \( \tau_j \) of all the in-neighboring links or a global upper bound \( \bar{\tau} \) may be available. Both of these cases constitute special cases of the problem with time-invariant delays. When the nodes have knowledge of a global upper bound \( \bar{\tau} \), they can update their states at time instants \( k + \bar{\tau} \) for time instants \( k, k = 0, 1, 2, \ldots \), i.e., once all (delayed) packets for time instant \( k \) have arrived, and use the iteration (4.7) (for each initial value). This is stated in Theorem 4.3 below, and illustrated for the network in Figure 4.1, where nodes update their states at time instants \( k + \bar{\tau} \) steps (see Figure 4.9); the ratio of the iteration converges to the exact average (see Figure 4.10).

**Theorem 4.3.** Consider a strongly connected graph \( G(V, E) \) with weight matrix \( P \), where \( P = [p_{ji}] \in \mathbb{R}_+^{n \times n} \) that adheres to the graph structure and forms a primitive column stochastic weight matrix. Let \( y_j[k] \) and \( z_j[k] \) be the result of the iterations in (3.18a) – (3.18b) for all \( v_j \in V \). By choosing to update at time instants \( h = k + \bar{\tau} \) for time instants \( k, k = 0, 1, 2, \ldots \) (i.e., once all (delayed) packets for time instant \( k \) have arrived), and by letting \( \hat{y}_j[h] \)
Distributed Finite-Time Average Consensus

Figure 4.9: The iterations (3.18a) with initial condition \( y[0] = [-1 \ 0 \ 1 \ 2 \ 3 \ 4]^T \) (left) and iteration (3.18b) with initial condition \( z[0] = 1_6 \) (right) with maximum delay \( \bar{\tau} = 5 \), for the network in Figure 4.1 do converge if the updates of the states \( \hat{y}_j[h] \) and \( \hat{z}_j[h] \) occur every step after the first \( \bar{\tau} \) steps (using iterations (3.7) – (3.8) with \( h = k + \bar{\tau}, \ k = 0, 1, 2, \ldots \)).

Figure 4.10: The ratio \( y_j[k]/z_j[k] \) of each node \( v_j \) of the iterations (3.18a) – (3.18b) when updated every \( h = k + \bar{\tau} \) steps, \( k = 0, 1, 2, \ldots \), (a) for \( \bar{\tau} = 5 \), and (b) when the assumed maximum delay \( \bar{\tau} \) is considered as double (i.e., \( \bar{\tau} = 2\bar{\tau} = 10 \)) the steps, due to an overestimate of the maximum delay. The ratio converges for both scenarios to the exact average, but with different convergence rates.

and \( \hat{z}_j[h] \) be the result of the iterations (3.18a) – (3.18b), the solution to the average consensus is the same, i.e.,

\[
\mu_j = \lim_{k \to \infty} \frac{y_j[k]}{z_j[k]} = \lim_{h \to \infty} \frac{\hat{y}_j[h]}{\hat{z}_j[h]}.
\]  

(4.8)

Hence, the solution to the average consensus can be distributively obtained in finite-time at each node \( v_j \), by computing

\[
\mu_j := \lim_{h \to \infty} \frac{\hat{y}_j[h]}{\hat{z}_j[h]} = \frac{\phi_{\hat{y}}(j)}{\phi_{\hat{z}}(j)} = \frac{\hat{y}_j^T M_j \beta_j}{\hat{z}_j^T M_j \beta_j},
\]  

(4.9)

where \( \phi_{\hat{y}}(j) \) and \( \phi_{\hat{z}}(j) \) are given by equations (4.5a) and (4.5b), respectively and \( \beta_j \) is the vector of coefficients defined in (4.4).

**Proof.** Each node updates at time instant \( k + \bar{\tau} \) for time instants \( k, \ k = 0, 1, 2, \ldots \), by which time all the delayed packets for time instant \( k \) have
arrived. As a result, this is equivalent to the time-invariant delay case (4.7) in which all the delays are set to the maximum delay, i.e., $P_j = 0_{n \times n}$ for all $j = 1, 2, \ldots, \bar{\tau} - 1$. Then, the proof proceeds similarly to that of Theorem 4.1.

Algorithm 9 below provides a finite-time distributed algorithm for computing the exact average when delays in the network are time-varying but bounded, and all the nodes have knowledge of the upper bound of the delay.

**Algorithm 9: Distributed Finite-Time Average Consensus in Digraphs with Bounded Time-Varying Delays**

**Input:** A strongly connected digraph $G(\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges.

**Data:** Observations for $\hat{y}_j[h]$ and $\hat{z}_j[h]$, $\forall v_j \in \mathcal{V}$, $h = k + \bar{\tau}$, $k = 0, 1, 2, \ldots$, using iterations (3.7) – (3.8), with initial conditions $y[0] = [V_1, V_2, \ldots, V_n]^T$ and $z[0] = 1_n$, respectively.

For $h = k + \bar{\tau}$, $k = 0, 1, 2, \ldots$, each node $v_j \in \mathcal{V}$ does the following:

**Step 1:** It runs the ratio consensus algorithm (Algorithm 3) for fixed delays and stores the vectors of differences $\hat{y}_T^M_j$ and $\hat{z}_T^M_j$ between successive values of $\hat{y}_j[h]$ and $\hat{z}_j[h]$, respectively.

**Step 2:** It increases the dimension $h$ of the square Hankel matrices $\Gamma\{\hat{y}_T^M_j\}$ and $\Gamma\{\hat{z}_T^M_j\}$ for each of the iterations, until they both lose rank, at which point it stores the first defective matrix.

**Step 3:** It computes the kernel $\beta_j = [\beta_0, \ldots, \beta_{M_j-1}, 1]^T$ of the first defective matrix, which gives the values $\phi_y$ and $\phi_z$, via (4.5a) and (4.5b), respectively.

**Step 4:** The average consensus value can be computed as

$$
\mu_j = \lim_{h \to \infty} \frac{\hat{y}_j[h]}{\hat{z}_j[h]} = \frac{\phi_y(j)}{\phi_z(j)} = \frac{y_T^M_j \beta_j}{z_T^M_j \beta_j}.
$$
Example 4.5. Considering the network in Figure 4.1 and using Theorem 4.3, the final consensus vectors $\phi_y$ and $\phi_z$, for initial conditions $y[0]$ and $z[0]$, respectively, are given by

$$
\phi_y = \begin{bmatrix} 0.4045, & 0.2697, & 0.1348, & 0.6067, & 0.3371, & 0.5056 \end{bmatrix}^T, \\
\phi_z = \begin{bmatrix} 0.2697, & 0.1798, & 0.0899, & 0.4045, & 0.2247, & 0.3371 \end{bmatrix}^T.
$$

Then, each node can compute the exact average $\mu_j = \phi_y(j)/\phi_z(j)$, e.g., $\mu_1 = \phi_y(1)/\phi_z(1) = 1.5$. For this example, the maximum number of steps a node requires to compute the exact average value is 64 steps, while the values of the ratios for the algorithm described in Proposition 3.2 using the upper bound $\bar{\tau}$ is $z[64] = [1.4906, 1.4993, 1.4953, 1.5016, 1.5034, 1.4905]^T$ (see the top of Figure 4.10). If the delay is overestimated then Algorithm 9 needs more time to converge to the final value; for this specific example, nodes assume that the upper bound on the delay is 10 (i.e., $\bar{\tau} = 10$) and the maximum number of steps required to compute the exact average value is 114 steps. This observation suggests that while we have further assumptions on the upper bound of the delay, Algorithm 9 performs well even in the presence of time-varying delays and even if the delays are overestimated. The algorithm described in Proposition 3.2 using the overestimated upper bound $\bar{\tau} = 10$ is shown at the right of Figure 4.10. \hfill $\square$

4.3 Approximate Finite-Time Average Consensus

In this strand of research the idea is that some deviation from the exact average is tolerated in order to stop the iterations in a finite number of steps. In what follows, we will discuss two different approaches appearing in the literature:

- **Section 4.3.1** presents an approach where *all* nodes update at every time step until a certain step in which the value of the error (defined as the largest absolute difference of the values of a pair of nodes) is within a certain range that can be tolerated and, hence, all nodes simultaneously stop updating.

- **Section 4.3.2** presents an approach where each node updates and broadcasts its value if its own error is above what can be tolerated; nodes do not necessarily stop updating simultaneously.
4.3. Approximate Finite-Time Average Consensus

4.3.1 Distributed Finite-Time Termination for Ratio Consensus

One of the first works that proposed an average consensus with a criterion for stopping in finite time was Yadav and Salapaka, 2007, where each node, apart from the average consensus algorithm, also runs max- and min-consensus algorithms. The key property exploited in Yadav and Salapaka, 2007 is that if the max (min) consensus algorithm (see Algorithm 1 in Chapter 2) runs every $D$ iterations (where $D$ is the diameter of the network), each time initialized with the node values, its outcome, i.e., the maximum (minimum) of the values it was initialized with, will be is strictly decreasing (increasing). Hence, it is possible to quantify when there is convergence of the algorithm within a range from the exact average, in a finite number of steps and in a distributed fashion. However, for the method of Yadav and Salapaka, 2007 to work in directed graphs, the weighted adjacency matrix has to be doubly stochastic. To alleviate this shortcoming, Cady et al., 2015a proposed a similar idea which is based on the ratio-consensus protocol proposed in Chapter 3. The main idea there is that iterations can be used to check when the ratios of the nodes are sufficiently close.

In what follows we provide the distributed finite-time termination algorithm for ratio consensus, given in Cady et al., 2015a. As aforementioned, the proposed algorithm takes advantage of min- and max-consensus iterations to allow the nodes to determine the time step, $k_0$, when their ratios, $\{\pi_j[k_0] | v_j \in V\}$, are within $\epsilon$ of each other. More specifically,

- each node $j$ runs a ratio consensus iteration (as introduced in Chapter 3);

- in order to allow the nodes to determine when the ratios $\pi_j$ for all nodes are close to the asymptotic value, each node maintains two auxiliary states, $m_j[k]$ and $M_j[k]$, which are updated using min- and max-consensus, respectively.

- Every $D$ steps (where $D$ is the diameter of the graph) each node checks whether $|M_j[k] - m_j[k]| < \epsilon$. If this is the case, it stops iterating. Otherwise, $m_j[k]$ and $M_j[k]$ are reinitialized to $\pi_j[k]$.

The pseudocode for the algorithm described above is provided in Algorithm 10.
Algorithm 10: Distributed Finite-Time Termination for Ratio Consensus

**Input:** Initial values for the two iterations $y_j[0] = V_j$ and $z_j[0] = 1$, and tolerance $\epsilon$

set $M_j[0] = +\infty$, $m_j[0] = -\infty$, $u_j[0] = 0$, $\pi_j = \frac{y_j[0]}{z_j[0]}$

set $p_{lj} = \frac{1}{1+d_j^2} \forall v_l \in \mathcal{N}_j^+ \cup \{v_j\}$ (zero otherwise)

for $k \geq 0$ do

while $u_j[k] = 0$ do

if $k$ mod $D = 0$ and $k \neq 0$ then

if $|M_j[k] - m_j[k]| < \epsilon$ then

set $u_j[k] = 1$

end if

set $M_j[k] = m_j[k] = \pi_j[k] = \frac{y_j[k]}{z_j[k]}$

end if

broadcast to all $v_l \in \mathcal{N}_j^+ : p_{lj} y_j[k], p_{lj} z_j[k], M_j[k], m_j[k]$

receive from all $v_i \in \mathcal{N}_j^- : p_{ji} y_i[k], p_{ji} z_i[k], M_i[k], m_i[k]$

compute

$y_j[k] \leftarrow \sum_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} p_{ji} y_i[k]$

$z_j[k] \leftarrow \sum_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} p_{ji} z_i[k]$

$M_j[k] \leftarrow \max_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} M_i[k]$

$m_j[k] \leftarrow \min_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} m_i[k]$

end while

end for
Example 4.6. Consider the 5-node directed network shown in Figure 4.11. Each node $v_j$ chooses its weight and the weight of its outgoing links to be

\[
\frac{1}{1 + d_j^+} \quad \text{(such that the sum of all weights assigned by each node $v_j$ is equal to 1)} \quad \text{and the weight matrix is}
\]

\[
P = \begin{bmatrix}
\frac{1}{3} & 0 & 0 & \frac{1}{2} & 0 \\
\frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\
\frac{1}{3} & \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{3} \\
0 & 0 & 0 & \frac{1}{2} & \frac{1}{3} \\
0 & \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{3}
\end{bmatrix}.
\]

In this example, two iterations are executed simultaneously with initial conditions $y[0] = [1, 2, 3, 4, 5]^T$ and $z[0] = 1_5$. It is shown that the algorithm converges in a finite-number of steps to ratios approximately equal to the average. For this example, all the nodes terminate after 16 steps for a tolerance $\epsilon = 0.01$.

Figure 4.12: The ratio $y_j[k]/z_j[k]$ of each node $v_j$ of Algorithm 10 converges to values approximately equal to the average in 16 steps.
4.3.2 Distributed Event-Triggered Termination

Another approach for distributed stopping in directed graphs is via a randomized event-triggered strategy, proposed in Manitara and Hadjicostis, 2017. The key observation in the event-triggered algorithm is that each node $v_j \in \mathcal{V}$ decides whether to transmit its value, based on the difference between its calculated ratio and the ratios it receives from its in-neighbors, and stops communicating (with its out-neighbors) depending on whether these differences are smaller than the tolerance parameter $\epsilon$. Since some of the in-neighbors of node $v_j$ might stop transmitting at some time step, the above rule needs to be implemented by comparing the ratio last seen from each neighbor (i.e., the ratio each neighbor had the last time it transmitted values). Note that the last seen ratio of node $v_i$ will be the same from the perspective of all of its out-neighbors; thus, we use $\bar{\pi}_i[k]$ to denote the last transmitted ratio of node $v_i$ by time step $k$ (in reality, $\bar{\pi}_i$ is not a global variable, but a local variable that is identical for all out-neighbors of node $v_i$). Note also that node $v_j$ also checks the value $|\pi_j[k] - \bar{\pi}_j[k]|$ because its own ratio might change while it is not transmitting (due to the fact that it might be receiving values).

In what follows we provide the rules followed by each node while running the event-triggered strategy.

**Event-triggered rule 1**: At each time step $k$, each node $v_j$ compares its ratio $\pi_j[k]$ against the ratio $\bar{\pi}_j[k]$ of each in-neighbor $v_i \in \mathcal{N}_j^-$ and itself; if the absolute difference $|\pi_j[k] - \bar{\pi}_i[k]| > \epsilon$ for at least one of the in-neighbors $v_i \in \mathcal{N}_j^-$ or itself, then node $v_j$ transmits its values to all of its out-neighbors; if the absolute differences $|\pi_j[k] - \bar{\pi}_i[k]| \leq \epsilon$, and it does not receive any value from its in-neighbors, node $v_j$ remains silent.

What’s not decided yet in **Event-triggered rule 1** is what happens when $|\pi_j[k] - \bar{\pi}_i[k]| \leq \epsilon$ and node $v_j$ receives a value from at least one of its in-neighbors. If node $v_j$ decides not to take any action, an example was constructed in Manitara and Hadjicostis, 2017, in which the network does not converge to the desired range of values. This led to the need of having the nodes transmit in some or many of the cases, even when $|\pi_j[k] - \bar{\pi}_i[k]| \leq \epsilon$. In Manitara and Hadjicostis, 2017, two solutions were provided: a randomized one and a deterministic one. Both methods were proven to converge. Herein, we present the randomized strategy, but the interested reader can find both methods in Manitara and Hadjicostis, 2017. In the randomized solution,
additionally to **Event-triggered rule 1**, the following rule is used:

**Event-triggered rule 2**: At each time step $k$, even if all values seen at node $v_j$ satisfy $|\pi_j[k] - \bar{\pi}_i[k]| \leq \epsilon$ for all $v_i \in \mathcal{N}_j^- \cup \{v_j\}$, if node $v_j$ receives a value from at least one in-neighbor, then node $v_j$ broadcasts its value with some probability $p_j$ (where $0 < p_j < 1$).

The algorithm for distributed event-triggered termination is given in Algorithm 11.

**Algorithm 11**: Randomized Strategy for Distributed Event-Triggered Termination

**Input**: Initial values for the two iterations $y_j[0] = V_j, z_j[0] = 1$

**set** $\bar{\pi}_i[0] = \infty \forall v_i \in \mathcal{N}_j^- \cup \{v_j\}, \mathcal{N}_j^-[0] = \mathcal{N}_j^-$, and 

$p_{ij} = 1/(1 + d_j^+) \forall v_i \in \mathcal{N}_j^+ \cup \{v_j\}$ (zero otherwise)

**set** $\bar{y}_j[k] = y_j[k]$ and $\bar{z}_j[k] = z_j[k]$.

Each node $v_j$ executes the following procedure:

**for** $k \geq 0$ **do**

- **receive** from all $v_i \in \mathcal{N}_j^-[k]$ (if $\mathcal{N}_j^-[k] \neq \emptyset$): $\bar{y}_i[k] := p_{ji}y_i[k], \bar{z}_i[k] := p_{ji}z_i[k]$

- **update** the last seen ratios for the in-neighbors that transmitted at time $k$, $v_i \in \mathcal{N}_j^-[k]$, and $v_j$, i.e., for $v_i \in \mathcal{N}_j^-[k] \cup \{v_j\}$:

  $\bar{\pi}_i[k] = \bar{y}_i[k]/\bar{z}_i[k]$

- **if** $|\pi_j[k] - \bar{\pi}_i[k]| > \epsilon$ for any $v_i \in \mathcal{N}_j^-[k] \cup \{v_j\}$ **then**

  - **broadcast** to all $v_l \in \mathcal{N}_j^+$: $\bar{y}_j[k] := p_{lj}y_j[k], \bar{z}_j[k] := p_{lj}z_j[k]$

  - **set** $\pi_j[k] := \bar{y}_j[k]/\bar{z}_j[k]$

- **else** **if** $|\pi_j[k] - \bar{\pi}_i[k]| \leq \epsilon$ and a message is received **then**

  - **broadcast** with probability $p_j$ to all $v_l \in \mathcal{N}_j^+$: $\bar{y}_j[k] := p_{lj}y_j[k], \bar{z}_j[k] := p_{lj}z_j[k]$

  - **set** $\pi_j[k] := \bar{y}_j[k]/\bar{z}_j[k]$

**end if**

- **compute** (using any values received from in-neighbors)

  $y_j[k] \leftarrow \sum_{v_i \in \mathcal{N}_j^-[k] \cup \{v_j\}} \bar{y}_i[k]$

  $z_j[k] \leftarrow \sum_{v_i \in \mathcal{N}_j^-[k] \cup \{v_j\}} \bar{z}_i[k]$

  **set** $\pi_j[k] := y_j[k]/z_j[k]$

**end for**
Theorem 4.4. Consider a network described by a digraph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Each node $v_j$ has some initial value $y_j[0] = V_j$ and runs Algorithm 11. Then, with probability one, all nodes become non-transmitting after a finite number of steps $f$, at which point the ratios satisfy

$$|\pi_j[f] - \mu| \leq 2\epsilon D, \forall v_j \in \mathcal{V},$$

where $\mu = \frac{1}{n} \sum_{i=1}^{n} V_i$ and $D$ is the diameter of graph $\mathcal{G}$.

Proof. See Manitara and Hadjicostis, 2017.

Example 4.7. Consider again the 5-node digraph shown in Figure 4.11. The randomized strategy for distributed event-triggered termination is used under the same initial conditions and the outcome is shown in Figure 4.13.

![Figure 4.13](image-url)

Figure 4.13: The ratio $y_j[k]/z_j[k]$ of each node $v_j$ of Algorithm 11 converges to values approximately equal to the average; the red straight lines are $2\epsilon D$ from the average.

4.4 Discussion

Having presented the three main approaches proposed in the literature for reaching average consensus in a finite number of steps, in what follows we include Table 4.1 comparing these methods.

While the approach in Charalambous et al., 2015 converges to the exact average value and it does not need to know any global parameters, it cannot handle the case for which the state has uncertainties (either due to communication with limited capacity or noisy measurements). Among the approximate algorithms, Cady et al., 2015a can easily handle delays if an upper bound is

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Magnitude of error</td>
<td>0</td>
<td>$\epsilon$</td>
<td>$2\epsilon D$</td>
</tr>
<tr>
<td>Global parameters</td>
<td>None</td>
<td>Graph diameter</td>
<td>None</td>
</tr>
<tr>
<td>Handles state uncertainties</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Handles delays</td>
<td>If upper bound is known</td>
<td>If upper bound is known</td>
<td>–</td>
</tr>
</tbody>
</table>

known (in the simplest case, a node can update its value with a delay equal to the maximum and the approach follows the nominal one), but it requires knowledge of the graph diameter (or an upper bound). On the other hand, the event-triggered approach Manitara and Hadjicostis, 2017 has an error that depends on the diameter of the network and it is unknown whether it works under delays.

For exact finite-time average consensus, as described in Section 4.2, the computation of the rank of the Hankel matrix, which is instrumental for the finite-time algorithm, assumes that its block elements are exact. However, in realistic scenarios the communication between the agents does not facilitate the exchange of the exact values due to bandwidth and time limitations. The problem of how to handle uncertainties in the measurements when using approaches based on Hankel matrices is still not well understood. Furthermore, it is shown in Charalambous and Hadjicostis, 2014 that ratio consensus reaches average consensus even in the presence of switching topologies and time-varying delays. However, switching topologies imply that the system is no longer linear time-invariant and, hence, the approach proposed in Section 4.2 is not applicable. Thus, the problem of computing the average in a finite number of steps in networks with switching topologies is open. It would be interesting to also investigate whether we can reach approximate average consensus in finite-time when agents experience time-varying delays and switching.

In Algorithm 10, each node needs to know a priori the diameter of the network or an upper bound of the diameter, something which hinders the
distributed implementation of the algorithm. On the other hand, there are algorithms in the literature for computing the diameter in a distributed fashion (see, for example, Roditty and Vassilevska Williams, 2013) or the number of nodes (upper bound for the diameter) in the network (see, for example, Shames et al., 2012), but the main issue is that there is no clear way of orchestrating the completion of the diameter/network size estimation and the initiation of the consensus protocol. Therefore, an interesting extension would be to investigate an approach in which the diameter (or an upper bound) of the network is computed concurrently and the whole implementation is fully distributed. More generally, a problem that remains open in distributed stopping is the a priori characterization (e.g., in the form of an upper bound) of the number of iterations that are needed to guarantee that approximate average consensus is reached. For example, given a digraph and a distributed stopping strategy for allowing the nodes to reach approximate average consensus, one would like to have a bound on the maximal number of iterations that are needed in order to guarantee that the distributed averaging algorithm will complete. Clearly, this bound also depends on the distributed stopping algorithm that will be used.

As it can be seen in Table 4.1, it is unknown whether the event-triggered average consensus strategy in Manitara and Hadjicostis, 2017 can operate under time-varying delays and switchings. Thus, an interesting problem would be to investigate whether it is possible, with modifications if necessary, for Algorithm 11 to function under such conditions.
In this chapter we discuss ways to assign weights (sometimes also viewed as flows) on the edges of a given digraph so that the resulting weighted digraph is weight-balanced (or simply balanced), i.e., for each of its nodes, the sum of the weights on the edges incoming to the node is equal to the sum of the weights on the edges outgoing from the node. Most of the chapter will deal with real weights/flows, but we will also describe works that discuss the case of integer weights. In our development of distributed algorithms, we will make a distinction between two different cases as follows. The first case considers unconstrained weights/flows (i.e., with unbounded positive real values), whereas the second case considers constrained weights/flows (i.e., with some or all of the weights/flows restricted to lie within some non-empty interval of the set of positive real numbers).

5.1 Introduction and Motivation

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 DeLellis et al., 2010, and
distributed adaptive strategies to tune the coupling weights of a network based on local information of node dynamics Yu et al., 2012. Weight balancing can also be associated with the matrix balancing problem in network optimization, which is, in turn, associated with numerous applications, such as predicting the distribution matrix of telephone traffic Bertsekas, 1998.

In all of the above applications, weights are associated with the physical interactions in a distributed control system, and are assigned to edges of the physical digraph. There are also many applications where weight balancing plays a significant role in the cyber digraph of a given distributed control system. In particular, weight balance is closely related to weights that form a doubly stochastic matrix, which find applications in distributed averaging. As discussed in Chapter 3, if the weights form a primitive doubly stochastic matrix $P_d$, then the nodes can asymptotically reach average consensus via simple iterations in which each node updates a variable it maintains to be a weighted linear combination of its own variable and the variables of its neighbors. Given a balanced digraph, one can use a finite-time algorithm based on max-consensus to obtain a doubly stochastic matrix Gharesifard and Cortés, 2012; we make this connection explicit later in Section 5.2.2.

Apart from allowing the nodes to asymptotically reach average consensus, a weight choice that forms a primitive doubly stochastic matrix is also important for a number of other tasks that involve the cyber layer. For example, the work in Nedic et al., 2010 uses weights that form a doubly stochastic matrix to devise strategies that allow the nodes to distributively reach consensus to a value vector that satisfies local convex constraints at each node, and to also optimize a global objective function (that is a combination of local objective functions at each node).

The weight balancing problem is also related to matrix scaling problems, which are discussed in Section 5.6 once we have the opportunity to introduce notation for the problem and discuss some of the technical details.

5.2 Preliminaries and Problem Formulation

In this section, we provide a precise definition of the weight-balancing problem, as well as its connection to the problem of obtaining a set of weights that form a doubly stochastic matrix. We also, provide a summary of the two main settings considered in the remainder of the chapter.
5.2. Preliminaries and Problem Formulation

5.2.1 Problem Formulation

Given a digraph \( G = (V, \mathcal{E}) \) of order \( n \), we can associate nonnegative weights (sometimes, also viewed as flows) \( w_{ji} \in \mathbb{R}_+ \) on each edge \((v_j, v_i) \in \mathcal{E}\) to obtain a weighted digraph \( G_w = (V, \mathcal{E}, W) \) (where \( W = [w_{ji}] \), with \( w_{ji} = 0 \) when \((v_j, v_i) \notin \mathcal{E}\), is the \( n \times n \) weight matrix). In our discussions, these weights could be restricted to lie in an interval \([l_{ji}, u_{ji}]\) where \( 0 \leq l_{ji} \leq w_{ji} \leq u_{ji} \). We will also use matrix notation to denote (respectively) the lower limit and upper limit matrices by the \( n \times n \) matrices \( L = [l_{ji}] \) and \( U = [u_{ji}] \), where \( L(j,i) = l_{ji} \) and \( U(j,i) = u_{ji} \) (and \( w_{ji} = l_{ji} = u_{ji} = 0 \) when \((v_j, v_i) \notin \mathcal{E}\)).

**Definition 1.** Given a weighted digraph \( G_w = (V, \mathcal{E}, W) \) of order \( n \) with weight assignment \( W = [w_{ji}] \), the total in-weight of node \( v_j \) is

\[
b_j^- := \sum_{v_i \in \mathcal{N}_j^-} w_{ji} = \sum_{i=1}^{n} w_{ji}
\]

whereas the total out-weight of node \( v_j \) is

\[
b_j^+ := \sum_{v_i \in \mathcal{N}_j^+} w_{lj} = \sum_{l=1}^{n} w_{lj}
\]

**Definition 2.** Given a weighted digraph \( G_w = (V, \mathcal{E}, W) \) of order \( n \) with weight assignment \( W = [w_{ji}] \), the weight balance of node \( v_j \) is

\[
b_j := b_j^- - b_j^+ = \sum_{i=1}^{n} w_{ji} - \sum_{l=1}^{n} w_{lj}
\]

**Definition 3.** Given a weighted digraph \( G_w = (V, \mathcal{E}, W) \) of order \( n \) with weight assignment \( W = [w_{ji}] \), the absolute imbalance (or total imbalance) of digraph \( G_w \), denoted by \( \varepsilon \), is

\[
\varepsilon := \sum_{j=1}^{n} |b_j|
\]

**Definition 4.** A weighted digraph \( G_w = (V, \mathcal{E}, W) \) of order \( n \) with weight assignment \( W = [w_{ji}] \), is called weight-balanced if its absolute imbalance (or total imbalance) is zero, i.e., \( \varepsilon = \sum_{j=1}^{n} |b_j| = 0 \); this implies that \( b_j = 0 \) for all \( v_j \in V \).
We are interested in solving the following problem in a distributed manner.

**Definition 5.** (Distributed Weight/Flow Balancing Problem.) We are given a strongly connected digraph $G_p = (V, E_p)$ of order $n$ (physical topology), as well as lower and upper bounds $l_{ji}$ and $u_{ji}$ ($0 \leq l_{ji} \leq u_{ji}$) on each edge $(v_j, v_i) \in E_p$. We want to develop a distributed algorithm that allows the nodes to iteratively adjust the weights on their edges so that they eventually obtain a set of weights $\{w_{ji} \mid (v_j, v_i) \in E_p\}$ that satisfy the following:

1. **Interval constraints:** $l_{ji} \leq w_{ji} \leq u_{ji}$ for each edge $(v_j, v_i) \in E_p$;
2. **Balance constraints:** $b_j^+ = b_j^-$ for every $v_j \in V$, where $b_j^+$ and $b_j^-$ are given in Definition 1.

The distributed algorithm needs to respect the communication constraints imposed by the communication graph $G_c = (V, E_c)$ (cyber topology) that describes the communication capabilities between pairs of nodes. [In general, the communication and physical topologies do not have to be related, though we will focus on two important special cases: (i) the case when the two topologies are identical (i.e., $E_c = E_p$), and (ii) the communication topology corresponds to the undirected version of the physical topology (i.e., nodes that are connected physically can communicate in a bi-directional manner).]

**Remark 5.1.** Note that during the execution of the algorithm, the nodes do not necessarily have to adjust the actual weights; rather, one could simply view the iterative algorithm as a process that allows the nodes to adjust estimates of the weights; these estimates could be used eventually (or periodically) to adjust the actual weights.

A necessary and sufficient condition for the existence of a solution to the above problem is the condition described below (see, for example, Ford and Fulkerson, 2010; Hoffman, 1960).

**Definition 6** (Ford and Fulkerson, 2010; Hoffman, 1960). (Circulation Conditions: Necessary and Sufficient Condition for Existence of Balanced Weights/Flows.) Consider a strongly connected digraph $G_p = (V, E_p)$, with lower and upper bounds $l_{ji}$ and $u_{ji}$ ($0 \leq l_{ji} \leq u_{ji}$) on each edge $(v_j, v_i) \in E_p$. The necessary and sufficient condition for the existence of a set of weights $\{w_{ji} \mid (v_j, v_i) \in E\}$ that satisfy
5.2. Preliminaries and Problem Formulation

1. **Interval constraints:** \( l_{ji} \leq w_{ji} \leq u_{ji} \) for each edge \((v_j, v_i) \in E_p\),

2. **Balance constraints:** \( b_j^+ = b_j^- \) for every \( v_j \in V \),

is the following: for each \( S, S \subset V \), we have

\[
\sum_{(v_j, v_i) \in E^-_S} l_{ji} \leq \sum_{(v_l, v_j) \in E^+_S} u_{lj} \tag{5.1}
\]

where

\[
E^-_S = \{ (v_j, v_i) \in E \mid v_j \in S, \ v_i \in V \setminus S \} , \tag{5.2}
\]

\[
E^+_S = \{ (v_l, v_j) \in E \mid v_j \in S, \ v_l \in V \setminus S \} . \tag{5.3}
\]

The fact that the circulation conditions above are necessary might be obvious if one views weights as flows into nodes. Balancing requires the total flow that goes into a node to equal the total flow that comes out. This requirement also needs to be satisfied for sets of nodes: the total flow that goes into the set of nodes captured by \( S \) has to equal the total flow that comes out of these nodes. The fact that the circulation conditions are also sufficient is perhaps not as obvious; however, the discussion in this chapter is one way of establishing the sufficiency of these conditions.

### 5.2.2 Obtaining Weights that Form a Doubly Stochastic Matrix

Consider the setting in Definition 5 and assume (for now) that, for the given strongly connected digraph \( G_p = (V, E_p) \) of order \( n \), we have obtained a set of weights \( w_{ji} \) on each edge \((v_j, v_i) \in E_p\), such that the interval constraints and balance constraints are satisfied. In particular, the weight matrix \( W = [w_{ji}] \) with \( w_{ji} \) on its \((j, i)^{th}\) position \((w_{ji} = 0\) for \((v_j, v_i) \notin E_p\)) satisfies the following:

- The diagonal elements are zero, i.e., \( w_{jj} = 0 \), for all \( j \in \{1, 2, ..., n\} \).

- Letting \( r_j \) denote the sum of all elements on the \( j^{th} \) row (i.e., \( r_j = \sum_{i=1}^n w_{ji} \)) and \( c_j \) denote the sum of all elements on the \( j^{th} \) column (i.e., \( c_j = \sum_{l=1}^n w_{lj} \)), we have

\[
 r_j = c_j , \forall j \in \{1, 2, ..., n\} .
\]
Given the set of weights in $W$, we can easily obtain a set of weights that form a doubly stochastic matrix $P_d$ (e.g., to use it for average consensus as described in Chapter 3) via the following simple steps:

1. Calculate $s_{\text{max}} = \max_j \{c_j\} = \max_j \{r_j\}$.

2. Define $P_d$ by setting its elements to

$$P_d(j, i) = \begin{cases} \frac{w_{ji}}{s_{\text{max}}}, & j \neq i, \\ \frac{s_{\text{max}} - c_j}{s_{\text{max}}}, & j = i. \end{cases}$$

It is not hard to verify that the resulting matrix is doubly stochastic (all elements are positive and the sum of each row and each column is exactly one). The matrix $P_d$ will have at least one diagonal element that is zero (namely, the column(s) that corresponds (correspond) to the row(s)/column(s) with the maximum sum); matrix $P_d$ will be primitive as long as at least one diagonal entry is not zero (because the graph $G_p$ is assumed to be strongly connected). Note that matrix $P_d$ is a scaled version of $W$ with diagonal elements added so that the row/column sums are unity.

The scaling of the weights can easily be applied in a distributed manner if necessary. All that is needed by each node is the scaling $s_{\text{max}}$, which can be obtained easily using a max-consensus algorithm (see Chapter 2). For example, if each node $v_j$ is in charge of assigning the weights $P_d(l, j)$ on its outgoing edges $(v_l, v_j)$, then all it has to do is to scale the weights on its outgoing edges to $P_d(l, j) = \frac{w_{ji}}{s_{\text{max}}}$ for $(v_l, v_j) \in \mathcal{E}_p$, and set its self-weight to $P_d(j, j) = \frac{s_{\text{max}} - c_j}{s_{\text{max}}}$. It is worth noting that the value $s_{\text{max}}$ can be replaced by any other larger value (e.g., if the nodes already have knowledge of an upper bound on the maximum possible value of a column/row sum, they could use that without needing to resort to max-consensus).

5.2.3 Scope

In the remainder of this chapter we assume that we are given a strongly connected graph $G_p = (\mathcal{V}, \mathcal{E}_p)$ with lower and upper bounds $l_{ji}$ and $u_{ji}$ ($l_{ji} \leq u_{ji}$) on each edge $(v_j, v_i) \in \mathcal{E}_p$, such that the circulation conditions in Definition 6 are satisfied. We discuss two important instances of the problem
5.2. Preliminaries and Problem Formulation

...formulated in the previous section, as well as distributed iterative algorithms to solve them. As before, we use $k$ ($k = 0, 1, 2, \ldots$) to denote the iteration. For example, $b_j^+[k]$ will denote the value of the total out-weight of node $v_j$ at time instant $k$.

**Unrestricted Weights**

First, in Section 5.3, we consider the case when the weights (flows) are unrestricted, i.e., $l_{ji} = 0$ and $u_{ji} = \infty$ for each edge $(v_j, v_i) \in \mathcal{E}_p$. Note that for this case, the circulation conditions in Definition 6 reveal that a solution always exists (as long as the graph is strongly connected or is a collection of strongly connected graphs). In fact, a trivial (but not interesting) solution is to set all the weights to zero; it should also be clear that if we have a solution any scaling of the corresponding weights by the same positive constant is also a solution. Thus, in finding the solution in a distributed manner, we will require that weights on edges are strictly positive and the main challenge will be to overcome the constraints imposed by the communication topology, which is assumed to resemble the weight/flow (physical) topology, i.e., $\mathcal{G}_c = \mathcal{G}_p$. The development in this section assumes that each node $v_j \in \mathcal{V}$ is in charge of assigning the weights on its outgoing edges, i.e., the weights in $\{w_{lj} \mid v_l \in \mathcal{N}_j^+\}$.

**Restricted Weights**

We then consider in Section 5.4 the case when (some or all of) the weights (flows) are restricted, i.e., $0 \leq l_{ji} \leq u_{ji}$ for each edge $(v_j, v_i) \in \mathcal{E}_p$. In this case, we assume (for simplicity) that the circulation conditions in Definition 6 are satisfied, though the algorithm we develop can also be adapted to reveal to the nodes when the conditions are violated. For this weight/flow topology, we consider the communication topology that allows nodes that are physical neighbors to communicate in a bidirectional manner, i.e., $\mathcal{G}_c = (\mathcal{V}, \mathcal{E}_c)$ with

$$\mathcal{E}_c = \bigcup_{(v_j, v_i) \in \mathcal{E}_p} \{(v_j, v_i), (v_i, v_j)\}.$$

---

1As we illustrate via an example later in this chapter, the case when the communication topology matches the physical topology, i.e., $\mathcal{G}_c = \mathcal{G}_p$, is more challenging because a node that is unbalanced can only influence the weights on its outgoing edges (i.e., it can only signal that it is unbalanced to its out-neighbors).
This essentially assumes that a pair of nodes $v_j$ and $v_i$ that are connected by an edge in the digraph $G_p$ (i.e., $(v_j, v_i) \in E_p$ and/or $(v_i, v_j) \in E_p$) can exchange information among themselves (in both directions).

5.3 Distributed Balancing with Unconstrained Weights

In this section we allow weights to take positive\(^2\) 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 rate of convergence of the proposed 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. Before we discuss the distributed algorithm, which was presented in Rikos et al., 2014, we briefly discuss a centralized algorithm to achieve weight balance.

5.3.1 Centralized Algorithm for Weight Balancing

Suppose that we are given a digraph $G_p = (\mathcal{V}, \mathcal{E}_p)$ that is strongly connected. If we ignore the distributivity constraints, one simple centralized approach to obtain a set of positive weights that form a balanced graph is the iterative algorithm that we briefly describe below. Initially, we let $k = 0$, and set all of the weights to a small positive value, say $w_{ji}[0] = 1$ for each edge $(v_j, v_i) \in E_p$. At each subsequent iteration $k$, starting from $k = 0$, we perform the following steps:

1. We calculate the balance $b_j[k] = \sum_{v_i \in \mathcal{N}_j^-} w_{ji}[k] - \sum_{v_l \in \mathcal{N}_j^+} w_{lj}[k]$ for each node $v_j \in \mathcal{V}$.

2. We pick a node $v_+$ that has maximum positive balance $b_{v_+}[k] > 0$ and a node $v_-$ that has minimum negative balance $b_{v_-}[k] < 0$. The choices of the two nodes (one with positive balance and one with negative balance) is not crucial for the algorithm; it could have been random (i.e., choose $v_+$ randomly among the nodes with positive balance and choose $v_-$ randomly among the nodes with negative balance). If no

\(^2\)We exclude zero weights in order to avoid trivial solutions that essentially exclude some (or all) of the edges of the graph by assigning a zero weight to them.
such choice is possible, the graph is balanced and we can terminate the algorithm (it will become evident from our analysis later on that no weight assignment in a strongly connected graph can result in all nodes having nonnegative balance or all nodes having nonpositive balance, unless the assignment results in a balanced graph).

3. We find a directed acyclic path from node \( v_+ \) to node \( v_- \), say \( v_+ =: v_{i_0}, v_{i_1}, ..., v_{i_{t-1}}, v_{i_t} := v_- \) (where \( (v_{i_{\tau+1}}, v_{i_{\tau}}) \in E_p \) for \( \tau = 0, 1, ..., t - 1 \)) (this is always possible since the graph is strongly connected). We increase the weight of each edge in this path by \( b_{v_+}[k] \), i.e., we set

\[
  w_{i_{\tau+1}, i_{\tau}}[k + 1] = w_{i_{\tau+1}, i_{\tau}}[k] + b_{v_+}[k], \quad \tau = 0, 1, 2, ..., t - 1 .
\]

4. We set \( k := k + 1 \) and go to Step 1 (next iteration).

We will not analyze the algorithm in detail since the focus is mostly on distributed algorithms for weight balancing; however, its proper operation should become evident once we complete the analysis for the distributed algorithms in later sections. To gain some intuition about this strategy, observe the following about each iteration of the algorithm:

- \( b_{v_+}[k + 1] = 0 \) because we have increased (by \( b_{v_+}[k] \)) the weight \( w_{i_{1},i_{0}} \), which is associated with an outgoing edge of node \( v_+ \) (note that no other changes were made to weights on incoming or outgoing edges of node \( v_+ \));

- \( b_{v_-}[k + 1] = b_{v_-}[k] \) for \( \tau = 1, 2, ..., t - 1 \), because, for each of these nodes, we have increased by \( b_{v_+}[k] \) the weight of exactly one incoming and exactly one outgoing edge (which leaves their balance unchanged).

- \( b_{v_-}[k + 1] = b_{v_-}[k] + b_{v_+}[k] \), which satisfies \( b_{v_-}[k + 1] > b_{v_-}[k] \) (the balance of node \( v_- \) could remain negative or become positive).

The above conditions imply that at each iteration, one node with positive balance (namely \( v_+ \)) gets to be balanced, one node with negative balance (namely \( v_- \)) gets its balance increased (and could become positive or zero), whereas all other nodes have their balances unchanged. Thus, after at most \( n-1 \) iterations, all nodes will have zero balance (since at each step, a node with positive balance gets to be balanced and remains balanced for all subsequent iterations of the algorithm).
5.3.2 Distributed Algorithm for Weight Balancing

Suppose that we are given a digraph $G_p = (V, \mathcal{E}_p)$ that is strongly connected. We are also given a communication topology, $G_c$, that coincides with the physical topology (i.e., $G_c = G_p$). We are interested in developing a distributed algorithm that respects the given communication topology and allows the nodes to obtain edge weights $w_{ji} > 0$ on each edge $(v_j, v_i) \in \mathcal{E}_p$ such that the resulting weighted digraph $G_p = (V, \mathcal{E}_p, W)$ (where $W = [w_{ji}]$, with $w_{ji} = 0$ if $(v_j, v_i) \notin \mathcal{E}_p$) is balanced. We assume that each node observes (but cannot set) the weights on its incoming edges and is responsible for determining the weights on its outgoing edges. For notational simplicity, since the physical and communication topologies coincide, we refer to a single digraph $G = (V, \mathcal{E})$.

The iterative algorithm we present below is asymptotic and allows each node to maintain equal weights on all of its outgoing edges, which makes implementation particularly easy in settings where broadcasting is possible (because a single transmission by node $v_j$ ensures that each out-neighbor $v_l$, $v_l \in \mathcal{N}_j^+$, obtains the value of the weight $w_{lj}$). What is perhaps not clear at this point is whether it will be possible to have a weight-balanced graph such that the weights on the outgoing edges of each node are equal; it turns out that such assignment is always possible as long as the given digraph is strongly connected (or is a collection of strongly connected digraphs), which is the necessary and sufficient condition for balancing with positive weights to be feasible Gharesifard and Cortés, 2012. In other words, restricting the outgoing weights of each node to be equal does not limit the feasibility of a balanced solution under the setting considered in this section.

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 computes its balance $b_j[k]$ and, if $b_j[k]$ is positive (respectively negative), it increases (respectively decreases) all of the weights on its outgoing edges by an equal amount that is proportional to $b_j[k]$. More specifically, node $v_j$ updates the weights on its outgoing edges to be

$$w_{lj}[k + 1] = w_{lj}[k] + \beta_j \frac{b_j[k]}{d_j^+}, \forall v_l \in \mathcal{N}_j^+, \quad (5.4)$$

where $\beta_j$ is a constant satisfying $0 < \beta_j < 1$. [Recall that $d_j^+$ is the out-degree of node $v_j$.] A pseudocode description of the algorithm (referred to as
Algorithm 12) is provided below. We assume that all \( n \) nodes make the updates on the weights on their outgoing edges (and subsequently transmit them to their out-neighbors) in a synchronized manner. In particular, we assume that Assumptions [A1]–[A5] in Section 2.3 hold.

**Algorithm 12: Distributed Balancing with Unconstrained Weights**

**Input:** A strongly connected digraph \( G = (\mathcal{V}, \mathcal{E}) \) with \( n = |\mathcal{V}| \) nodes and \( m = |\mathcal{E}| \) edges (and no self-loops).

**Initialization:** Each node \( v_j \in \mathcal{V} \) initializes the following:

1) It sets \( w_{lj}[0] = 1, \forall v_l \in \mathcal{N}_j^+ \).

2) It chooses a proportionality gain \( \beta_j \in (0, 1) \).

**Iteration:** For \( k = 0, 1, 2, \ldots \), each node \( v_j \in \mathcal{V} \) does the following:

Step 1: It transmits \( w_{lj}[k] \) to its out-neighbors \( v_l \in \mathcal{N}_j^+ \).

Step 2: It receives \( w_{ji}[k] \) from each in-neighbor \( v_i \in \mathcal{N}_j^- \).

Step 3: It calculates its balance as

\[
b_j[k] = \sum_{v_i \in \mathcal{N}_j^-} w_{ji}[k] - \sum_{v_l \in \mathcal{N}_j^+} w_{lj}[k].
\]

Step 4: It updates the weights of each of its outgoing edges \( w_{lj} \), \( \forall v_l \in \mathcal{N}_j^+ \), as

\[
w_{lj}[k + 1] = w_{lj}[k] + \beta_j \frac{b_j[k]}{d_j^+}, \quad \forall v_l \in \mathcal{N}_j^+ .
\]

To gain intuition about the operation of the proposed algorithm, one can make the following observation: at iteration \( k \), node \( v_j \) adjusts the weights on its outgoing edges so that, if none of its in-neighbors changes the weights on its incoming edges, then node \( v_j \) will improve its balance at iteration \( k + 1 \) by a factor of \( (1 - \beta_j) \). This can be seen by realizing that

\[
b_j[k + 1] = \sum_{v_i \in \mathcal{N}_j^-} w_{ji}[k + 1] - \sum_{v_l \in \mathcal{N}_j^+} w_{lj}[k + 1]
\]

\[
= \sum_{v_i \in \mathcal{N}_j^-} w_{ji}[k] - \sum_{v_l \in \mathcal{N}_j^+} (w_{lj}[k] + \beta_j \frac{b_j[k]}{d_j^+})
\]

\[
= b_j[k] - \beta_j b_j[k] = (1 - \beta_j) b_j[k],
\]
where on the second line we used the fact that the weights on the incoming edges do not change and on the third line we used the fact that the number of out-neighbors of node \( v_j \) is \( d_j^+ \). Clearly, if \( \beta_j = 1 \), node \( v_j \) would become balanced at iteration \( k+1 \) (assuming no changes on the weights on its incoming edges); however, since all nodes are changing their weights simultaneously, we will see that \( \beta_j \) should not be unity (at least not for all nodes) because it might lead to oscillatory behavior and non-convergence. In particular, we will see that in such case the weighted adjacency matrix \( P \) (defined below) may not necessarily be primitive and hence the algorithm may not converge to weights that form a weight-balanced digraph. The following proposition is established in Rikos et al., 2014.

**Proposition 1** (Rikos et al., 2014). If digraph \( \mathcal{G}(\mathcal{V}, \mathcal{E}) \) is strongly connected, Algorithm 12 asymptotically reaches a steady-state weight matrix \( W^* \) (with identical columns) that forms a weight-balanced digraph. The rate of convergence is geometric and is captured by \( |\lambda_2(P)| \), where \( P = [p_{ji}] \) is defined as

\[
p_{ji} := \begin{cases} 
1 - \beta_j, & \text{if } i = j, \\
\beta_j / d_j^+, & \text{if } v_i \in \mathcal{N}_j^-, 
\end{cases}
\]

and \( |\lambda_2(P)| := \max\{|\lambda| : \lambda \in \sigma(P), \lambda \neq 1\} \equiv \rho(P - \frac{vu^T}{u^Tv}) \), where \( v \) and \( u \) are the unique (up to scaling) right and left eigenvectors of \( P \) at eigenvalue 1, respectively.

**Proof.** Since all outgoing edges of each node 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), we can denote the weight on any outgoing edge of node \( v_j \), at iteration \( k \), as \( w_j[k] \) and define \( w[k] = [w_1[k], w_2[k], \ldots, w_n[k]]^T \) with \( w_j[k] = w_{lj}[k] \) \( (v_l \in \mathcal{N}_j^+) \). Then, the balance for node \( v_j \) at iteration \( k \) is given by

\[
b_j[k] = \sum_{v_i \in \mathcal{N}_j^-} w_{ji}[k] - \sum_{v_l \in \mathcal{N}_j^+} w_{lj}[k] \\
= \sum_{v_i \in \mathcal{N}_j^-} w_i[k] - d_j^+ w_j[k] .
\]
Using the above and realizing that the update (5.4) for the weights can be written as
\[ w_j[k + 1] = w_j[k] + \frac{\beta_j}{d_j^+} \left( \sum_{v_i \in N_j^-} w_i[k] - d_j^+ w_j[k] \right), \]
we are led to the following update equation in matrix form
\[ w[k + 1] = P w[k], \]
\[ w[0] = w_0, \]
(5.6)
where \( w_0 = 1_n \) and \( P \) is as defined in the proposition. It should be clear from the above update equation that the weights remain nonnegative during the execution of the algorithm (because \( w_0 \) is nonnegative and \( P \) is a nonnegative matrix).

Matrix \( P \) can be written as \( P = I_n - B + BD^{-1} A_d \), where \( I_n \) is the \( n \times n \) identity matrix, \( B = \text{diag}(\beta_1, \beta_2, \ldots, \beta_n) \), \( D = \text{diag}(d_1^+, d_2^+, \ldots, d_n^+) \) and \( A_d \) is the adjacency matrix (with \( A_d(j, i) = 1 \) if \( (v_j, v_i) \in E \), and \( A_d(j, i) = 0 \) otherwise). Since \( \sigma(D^{-1} A_d) = \sigma(A_d D^{-1}) \), then \( \rho(D^{-1} A_d) = \rho(A_d D^{-1}) \).

In addition, \( \rho(A_d D^{-1}) = 1 \) because matrix \( A_d D^{-1} \) is column stochastic. As a result, \( \rho(D^{-1} A_d) = 1. \) Also, note that \( \bar{P} := I_n - B + A_d D^{-1} B \) is column stochastic and therefore \( \rho(\bar{P}) = 1 \). Furthermore,
\[ \rho(\bar{P}) = \rho(\bar{P} B^{-1} D D^{-1} B) \]
\[ = \rho(D^{-1} B \bar{P} B^{-1} D) \]
\[ = \rho(I_n - B + BD^{-1} A_d) \]
\[ = \rho(P) \]
\[ = 1. \]

The nonnegative matrix \( P \) can easily be shown to be primitive (the corresponding graph is strongly connected and at least one entry on the main diagonal entries is positive). Hence, there is no other eigenvalue with modulus equal to the spectral radius. We conclude that Algorithm 12 has a geometric convergence rate \( R_\infty(P) = -\ln |\lambda_2(P)| \), where \( |\lambda_2(P)| := \max\{|\lambda| : \lambda \in \sigma(P), \lambda \neq 1\} \); by the Perron-Frobenius theorem (see Theorem 2.1 in Chapter 2), this is equal to \( \rho(P - \frac{w u^T}{u^T v}) \), where \( v \) and \( u^T \) are the right and left eigenvectors of \( P \) at eigenvalue 1, respectively. \( \square \)
**Example 5.1.** In this example (borrowed from Gharesifard and Cortés, 2010 and Rikos *et al.*, 2014), we demonstrate the proposed algorithm in the network system depicted in Figure 5.1. In Figure 5.2, we plot the absolute (total) imbalance of the system as a function of the number of iterations when we run Algorithm 12 with $\beta_j = 0.1$ (red solid line) or 0.5 (green dotted line) or 0.9 (blue dash-dotted line) for all $v_j \in \mathcal{V}$. These plots agree with the claims in Proposition 1 and validate that the algorithm converges to a weight-balanced digraph with geometric convergence rate.

![Figure 5.1: Strongly connected digraph used to demonstrated the execution of Algorithm 12.](image)

![Figure 5.2: Execution of Algorithm 12 on the digraph of Figure 5.1, shown in terms of the absolute (total) imbalance at each iteration for different choices of the $\beta$'s (identical for all $v_j \in \mathcal{V}$): $\beta_j = 0.1$ (red solid line) or 0.5 (green dotted line) or 0.9 (blue dash-dotted line).](image)
5.4. Distributed Balancing with Constrained Weights

From the above, it is also clear that a more aggressive choice of $\beta_j$’s (closer to unity) will not always lead to faster convergence (though, generally, that might be the case). In fact, setting $\beta_j = 1$ could lead into problems. In particular, if each node $v_j$ in the graph in Figure 5.3 sets its $\beta_j$ value to unity, then we get oscillatory behavior (and non-convergence). The reason is that the matrix $P$ in Proposition 1 is not primitive anymore (because the graph is periodic). This is relatively easy to see in this case: at iteration 1, nodes $v_1$ and $v_2$ will set their outgoing weights to $1/2$, whereas nodes $v_3$ and $v_4$ will set their outgoing weight to 2; at iteration 2, all nodes will set their outgoing weights to 1; at iteration 3, nodes $v_1$ and $v_2$ will set their outgoing weights to $1/2$, whereas nodes $v_3$ and $v_4$ will set their outgoing weight to 2; and so forth.

![Figure 5.3: Strongly connected (periodic) digraph used to demonstrate potential problems in case $\beta_j = 1$ for all nodes $v_j \in \mathcal{V}$.](image)

**Figure 5.3:** Strongly connected (periodic) digraph used to demonstrate potential problems in case $\beta_j = 1$ for all nodes $v_j \in \mathcal{V}$.

5.4 Distributed Balancing with Constrained Weights

In this section, we consider weights that are restricted to lie in some interval subset of the set of positive real numbers. Unlike the situation in the previous section where we were able to assign equal weights on the outgoing edges of each node, in this section we need to assume that each edge can be assigned its own weight, independently from other edges. This is a necessary relaxation because the assignment of equal weights might not even be feasible (e.g., if the weights of the outgoing edges of the given node are constrained to lie in non-intersecting intervals). Indirectly, this implies that each node is aware of the
edges associated with its in- and out-neighbors; more importantly, each node needs to be aware of at least the identity of its out-neighbors and in-neighbors. We also assume (for simplicity) that the circulation conditions in Definition 6 are satisfied, though the algorithm we describe can also be adapted to reveal to the nodes when these conditions are violated.

As in the previous section, the described distributed algorithm is iterative, and we use \( k \) (\( k = 0, 1, 2, \ldots \)) to denote the iteration. We can still treat the weight values during the iterative process as estimates of the eventual weights and, without loss of generality, we can think of each node as being responsible for eventually assigning weights to all of its outgoing edges.

The distributed algorithm assumes a communication topology that allows nodes that are physical neighbors to communicate in a bidirectional manner, i.e., \( G_c = (V, E_c) \) with \( E_c = \cup_{(v_j,v_i) \in E_p} \{ (v_j,v_i), (v_i,v_j) \} \). This essentially means that a pair of nodes \( v_j \) and \( v_i \) that are connected by an edge in the digraph \( G_p \) (i.e., \( (v_j,v_i) \in E_p \) and/or \( (v_i,v_j) \in E_p \)) can exchange information among themselves (in both directions). Thus, each node can easily obtain the needed identities of its in- and out-neighbors.

The algorithm we develop operates by having, at each iteration, nodes with positive weight balance attempt to change the weights on both their incoming and outgoing edges, so as to get closer to being balanced. In the process, since the weight on each edge affects the balance of two nodes (both of which may be simultaneously attempting to adjust the edge weight), the nodes need to coordinate with the corresponding neighbor (which could be an in-neighbor or an out-neighbor) in order to reach an agreement on the weight for that particular edge. Naturally, the nodes need to assign weights that respect the lower and upper limits on each edge.

**Remark 5.2.** We do not allow nodes with negative weight balance to make a weight adjustment, in order to avoid problematic situations like the oscillations in the example in Figure 5.3; alternatively, one could allow all nodes (with both positive and negative weight balance) to make adjustments, as long as not all of them try to become fully balanced at the next iteration (that would be the equivalent of having \( \beta_j = 1 \) for all nodes in Example 5.3). In the particular version of the algorithm we present in this section, due to the averaging of the flow estimates, this is the case; thus, one could actually allow all nodes to make weight adjustments though the proof of convergence becomes more...
tedious. Simulations indicate that the version of the algorithm that allows all nodes to make adjustments generally converges faster to a set of weights that satisfy the interval constraints and balance requirements.

The distributed weight balancing algorithm assumes that at initialization each node is aware of the feasible weight interval on each of its incoming and outgoing edges, i.e., node $v_j$ is aware of $l_{ji}, u_{ji}$ for each $v_i \in \mathcal{N}_j^-$ and $l_{lj}, u_{lj}$ for each $v_l \in \mathcal{N}_j^+$ (note that $\mathcal{N}_j^-$ and $\mathcal{N}_j^+$ are defined with respect to the physical topology captured by $\mathcal{G}_p = (\mathcal{V}, \mathcal{E}_p)$). Furthermore, the weights are initialized at the middle of the feasible interval, i.e., $w_{ji}[0] = (l_{ji} + u_{ji})/2$ for each $(v_j, v_i) \in \mathcal{E}_p$. [This initialization is not critical and could be any value in the feasible weight interval $[l_{ji}, u_{ji}]$, as long as both node $v_j$ and node $v_i$ agree on the same value.] Subsequently, the nodes enter the iterative stage of the algorithm. At each iteration $k \geq 0$, node $v_j$ is aware of the weights on its incoming edges $\{w_{ji}[k] \mid v_i \in \mathcal{N}_j^-\}$ and outgoing edges $\{w_{lj}[k] \mid v_l \in \mathcal{N}_j^+\}$. The update of the edge weights involves a three-stage process.

**Stage 1.** In this stage, each node $v_j$ with positive balance attempts to change the weights on both its incoming edges and its outgoing edges. In particular, if $b_j[k] > 0$, node $v_j$ attempts to change the weights at both its incoming edges $\{w_{ji}[k + 1] \mid v_i \in \mathcal{N}_j^-\}$ and outgoing edges $\{w_{lj}[k + 1] \mid v_l \in \mathcal{N}_j^+\}$, in a way that drives its balance $b_j[k + 1]$ (at the next iteration) to zero, at least if no other changes are inflicted on the weights. More specifically, since node $v_j$ is associated with $D_j := d_j^- + d_j^+$ (physical) edges, it attempts to change each incoming weight by $-\frac{b_j[k]}{D_j}$ and each outgoing weight by $+\frac{b_j[k]}{D_j}$, i.e., from the perspective of node $v_j$, the desirable weights at the next iteration are

$$w_{ji}^{(j)}[k + 1] = w_{ji}[k] - \frac{b_j[k]}{D_j}, \quad v_i \in \mathcal{N}_j^-,$$

$$w_{lj}^{(j)}[k + 1] = w_{lj}[k] + \frac{b_j[k]}{D_j}, \quad v_l \in \mathcal{N}_j^+,$$

where $b_j[k] > 0$. Note that if the above changes on the weights were adopted, then the new balance of node $v_j$ would be

$$b_j[k + 1] = \sum_{v_i \in \mathcal{N}_j^-} w_{ji}^{(j)}[k + 1] - \sum_{v_l \in \mathcal{N}_j^+} w_{lj}^{(j)}[k + 1]$$

$$= \sum_{v_i \in \mathcal{N}_j^-} (w_{ji}[k] - \frac{b_j[k]}{D_j}) - \sum_{v_l \in \mathcal{N}_j^+} (w_{lj}[k] + \frac{b_j[k]}{D_j})$$

$$= b_j[k] - d_j^- \frac{b_j[k]}{D_j} - d_j^+ \frac{b_j[k]}{D_j} = 0.$$
If node \( v_j \) has balance \( b_j[k] \) that is negative or zero (\( b_j[k] \leq 0 \)), then node \( v_j \) does not attempt to make any weight changes. Note that no desirable change on the weights can also be captured by (5.7)–(5.8) with \( b_j[k] = 0 \). Thus, regardless of whether node \( v_j \) has positive imbalance or not, we can capture the desirable new weights on each incoming and outgoing edge as

\[
\begin{align*}
\tilde{w}_{ji}[k+1] &= \frac{1}{2} \left( \tilde{b}_j[k] - \frac{\tilde{b}_j[k]}{D_j} \right), & v_i &\in \mathcal{N}_j^- , \\
\tilde{w}_{lj}[k+1] &= \frac{1}{2} \left( \tilde{b}_j[k] + \frac{\tilde{b}_j[k]}{D_j} \right), & v_l &\in \mathcal{N}_j^+ ,
\end{align*}
\]

(5.9)\hspace{1cm} (5.10)

where \( \tilde{b}_j[k] \) is defined as

\[
\tilde{b}_j[k] = \begin{cases} 
    b_j[k] , & \text{if } b_j[k] > 0 , \\
    0 , & \text{otherwise.}
\end{cases}
\]

[Stage 2.] Since the weight \( w_{ji} \) on each edge \((v_j, v_i) \in \mathcal{E}_p \) affects positively the weight balance \( b_j[k] \) of node \( v_j \) and negatively the weight balance of node \( v_i \), we need to account for the possibility of both nodes attempting to inflict changes on the weights. Thus, the new weight on each edge \((v_j, v_i) \in \mathcal{E}_p \) is taken to be

\[
\tilde{w}_{ji}[k+1] = \frac{1}{2} \left( w_{ji}[k] + w_{ij}[k] \right) = w_{ji}[k] + \frac{1}{2} \left( \frac{\tilde{b}_i[k]}{D_i} - \frac{\tilde{b}_j[k]}{D_j} \right).
\]

(5.11)

[Stage 3.] If the above value is in the interval \([l_{ji}, u_{ji}]\), then \( w_{ji}[k+1] = \tilde{w}_{ji}[k+1] \); otherwise, if it is above \( u_{ji} \) (respectively, below \( l_{ji} \)), it is set to the upper bound \( u_{ji} \) (respectively, the lower bound \( l_{ji} \)):

\[
\tilde{w}_{ji}[k+1] = \begin{cases} 
    \tilde{w}_{ji}[k+1] , & \text{if } l_{ji} \leq \tilde{w}_{ji}[k+1] \leq u_{ji} , \\
    u_{ji} , & \text{if } \tilde{w}_{ji}[k+1] > u_{ji} , \\
    l_{ji} , & \text{if } \tilde{w}_{ji}[k+1] < l_{ji} .
\end{cases}
\]

(5.12)

Once the values \( \{w_{ji}[k+1] \mid (v_j, v_i) \in \mathcal{E}_p\} \) are obtained, the iteration is repeated.

The pseudocode for the algorithm described by iterations (5.7)–(5.12) is provided as Algorithm 13 below.
Algorithm 13: Distributed Balancing with Constrained Weights

**Input:** A strongly connected digraph $G_p = (\mathcal{V}, \mathcal{E}_p)$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}_p|$ edges (and no self-loops).

A communication graph $G_c = (\mathcal{V}, \mathcal{E}_c)$ with $\mathcal{E}_c = \bigcup_{(v_j, v_i) \in \mathcal{E}_p} \{(v_j, v_i), (v_i, v_j)\}$.

Each node $v_j$ is aware of lower and upper bounds on its incoming and outgoing edges ($l_{ji}, u_{ji}$, $\forall v_i \in \mathcal{N}_j^-$ and $l_{lj}, u_{lj}$, $\forall v_l \in \mathcal{N}_j^+$).

**Initialization:** Each node $v_j \in \mathcal{V}$ initializes the following:

1) It sets $w_{ji}[0] = \frac{l_{ji} + u_{ji}}{2}$, $\forall v_i \in \mathcal{N}_j^-$.

2) It sets $w_{lj}[0] = \frac{l_{lj} + u_{lj}}{2}$, $\forall v_l \in \mathcal{N}_j^+$.

3) It sets $D_j = d_j^− + d_j^+$.

**Iteration:** For $k = 0, 1, 2, \ldots$, each node $v_j \in \mathcal{V}$ does the following:

Step 1: It calculates $b_j[k] = \sum_{v_i \in \mathcal{N}_j^-} w_{ji}[k] - \sum_{v_i \in \mathcal{N}_j^+} w_{lj}[k]$ and sets $\tilde{b}_j[k] = \max(b_j[k], 0)$.

Step 2: It transmits $\frac{\tilde{b}_j[k]}{D_j}$ to $v_i \in \mathcal{N}_j^-$ and $v_l \in \mathcal{N}_j^+$.

Step 3: It receives $\frac{\tilde{b}_i[k]}{D_i}$ from all $v_i \in \mathcal{N}_j^-$ and $\frac{\tilde{b}_l[k]}{D_l}$ from all $v_l \in \mathcal{N}_j^+$.

Step 4: It calculates

$\tilde{w}_{ji}[k+1] = w_{ji}[k] + \frac{1}{2} \left( \frac{\tilde{b}_i[k]}{D_i} - \frac{\tilde{b}_j[k]}{D_j} \right)$, $\forall v_i \in \mathcal{N}_j^-$,

$\tilde{w}_{lj}[k+1] = w_{lj}[k] + \frac{1}{2} \left( \frac{\tilde{b}_j[k]}{D_j} - \frac{\tilde{b}_l[k]}{D_l} \right)$, $\forall v_l \in \mathcal{N}_j^+$,

and sets

$w_{ji}[k+1] = \begin{cases} \tilde{w}_{ji}[k+1], & \text{if } l_{ji} \leq \tilde{w}_{ji}[k+1] \leq u_{ji}, \\ u_{ji}, & \text{if } \tilde{w}_{ji}[k+1] > u_{ji}, \\ l_{ji}, & \text{if } \tilde{w}_{ji}[k+1] < l_{ji}, \end{cases}$

$w_{lj}[k+1] = \begin{cases} \tilde{w}_{lj}[k+1], & \text{if } l_{lj} \leq \tilde{w}_{lj}[k+1] \leq u_{lj}, \\ u_{lj}, & \text{if } \tilde{w}_{lj}[k+1] > u_{lj}, \\ l_{lj}, & \text{if } \tilde{w}_{lj}[k+1] < l_{lj}. \end{cases}$
Distributed Weight Balancing

\[
L = \begin{bmatrix}
0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
\end{bmatrix} \quad U = \begin{bmatrix}
0 & 0 & 2 & 3 \\
2 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 \\
0 & 0 & 2 & 0 \\
\end{bmatrix}
\]

Figure 5.4: Matrix of lower bounds \(L\) and matrix of upper bounds \(U\), used in Example 5.2 for the digraph in Figure 5.1.

Remark 5.3. For simplicity, the algorithm could be implemented by having each node \(v_j\) transmit its \(\tilde{b}_j[k]/D_j\) value at iteration \(k\) to all of its neighbors, both in-neighbors and out-neighbors. This approach could be advantageous in a wireless (broadcast) setting because a single transmission would serve all in- and out-neighbors. Of course, each node should maintain separate values for the weight of each incoming and outgoing edge.

Example 5.2. In this example, we illustrate the operation of Algorithm 13 for the digraph in Figure 5.1 with interval constraints on its edges, captured by matrices \(L\) and \(U\) in Figure 5.4. With these choices, it can be verified that the circulation conditions in Definition 6 are satisfied.
5.4. Distributed Balancing with Constrained Weights

Figure 5.5 plots the weight balance, $b_j[k], j = 1, 2, 3, 4$ of each of the four nodes against the iteration $k$ during the execution of Algorithm 13. Notice that the sum $\sum_{j=1}^{4} b_j[k]$ is identically zero for all $k$ as expected (this is discussed in the outline of the proof for convergence later in this section). Moreover, nodes with a positive weight balance retain a positive weight balance as $k$ increases; in the end, only one node retains negative balance, and all node weight balances asymptotically go to zero (this is also something established in the proof of the convergence of the algorithm). Figure 5.6 provides a plot of the evolution of the absolute balance $\Delta[k]$ against the iteration $k$. Notice that $\Delta[k]$ monotonically goes to zero (again, this is a key result in the proof of convergence).

It can be observed that the weights eventually stabilize to fixed values, as given in the following weight matrix:

$$W = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix};$$

the weights are easily seen to be feasible and result in a balanced digraph. $\square$
5.4.1 Proof of Convergence

Algorithm 13 converges to a set of feasible and balanced weights, as long as the necessary and sufficient condition in (5.1) holds. In particular, the absolute (total) imbalance $\varepsilon[k]$ in Definition 3 goes to zero as $k$ goes to infinity, with a geometric rate. This also implies that the weight balance $b_j[k]$ for each node $v_j \in \mathcal{V}$ goes to zero, and thus (from the weight updates in (5.11) and (5.12)) the weight $w_{ji}[k]$ on each edge $(v_j, v_i) \in \mathcal{E}_p$ stabilizes to a value $w_{ji}^*$ within the given lower and upper limits, i.e., $l_{ji} \leq w_{ji}^* \leq u_{ji}$ for all $(v_j, v_i) \in \mathcal{E}_p$.

Next, we describe the setting that is considered in this section and outline the convergence of Algorithm 13; the details of the proof can be found in Hadjicostis and Domínguez-García, 2016.

For easing subsequent developments, we will also make use of the following notation.

**Definition 7.** The weight change incurred at the weight of edge $(v_j, v_i) \in \mathcal{E}_p$ at iteration $k$ is denoted by $\Delta w_{ji}[k]$, i.e.,

$$
\Delta w_{ji}[k] := w_{ji}[k + 1] - w_{ji}[k],
$$

a variable that captures the combined effect of both (5.11) and (5.12). Similarly, we define the changes in the weight balances at each node $v_j \in \mathcal{V}$ and the absolute imbalance of the network:

$$
\Delta b_j[k] \equiv b_j[k + 1] - b_j[k], \forall v_j \in \mathcal{V},
$$

$$
\Delta \varepsilon[k] \equiv \varepsilon[k + 1] - \varepsilon[k].
$$

The bulk of this section will be devoted to proving the following theorem.

**Theorem 5.1.** Consider the setting described above where the circulation conditions in Definition 6 hold. During the execution of the algorithm described by (5.7)–(5.12), it holds that

$$
\varepsilon[k + n] \leq (1 - c)\varepsilon[k], \forall k \geq 0,
$$

where $n = |\mathcal{V}|$ is the number of nodes and $\varepsilon[k] \geq 0$ is the absolute imbalance of the network at iteration $k$ (refer to Definition 3), with

$$
c = \frac{1}{2n} \left( \frac{1}{2D_{\text{max}}} \right)^n,
$$

where $D_{\text{max}} = \max_{v_j \in \mathcal{V}} D_j$. [Note that $D_{\text{max}}$ necessarily satisfies $1 \leq D_{\text{max}} \leq 2(n - 1)$.]
Corollary 5.2. Consider the setting described at the beginning of this section. The execution of the algorithm described by (5.7)-(5.12) asymptotically leads to a set of weights \( \{ w_{ji}^* \mid (v_j, v_i) \in E_p \} \) that satisfy the interval constraints and balance constraints, i.e., we have \( \lim_{k \to \infty} w_{ji}[k] = w_{ji}^* \), \( \forall (v_j, v_i) \in E_p \), where the set of weights \( \{ w_{ji}^* \mid (v_j, v_i) \in E_p \} \) satisfy

1. \( l_{ji} \leq w_{ji}^* \leq u_{ji}, \forall (v_j, v_i) \in E_p \);
2. \( \sum_{v_i \in N_j^{-}} w_{ji}^* = \sum_{v_l \in N_j^{+}} w_{lj}^*, \forall v_j \in \mathcal{V} \).

Proof. From Theorem 5.1, we have that \( \lim_{k \to \infty} \varepsilon[k] = \lim_{k \to \infty} \sum_{j=1}^{n} |b_j[k]| = 0 \), which implies that \( \lim_{k \to \infty} b_j[k] = 0 \), \( \forall v_j \in \mathcal{V} \). From the weight updates in (5.11) and (5.12), the weight \( w_{ji}[k] \) on each edge \( (v_j, v_i) \in E_p \) stabilizes to a value \( w_{ji}^* \), i.e., \( w_{ji}^* = \lim_{k \to \infty} w_{ji}[k] \) exist for all edges \( (v_j, v_i) \in E_p \). Clearly, the algorithm described by (5.7)-(5.12) results in weights \( w_{ji}^* \) that are within the lower and upper bounds on each edge (i.e., \( l_{ji} \leq w_{ji}^* \leq u_{ji} \)). Furthermore, since \( \lim_{k \to \infty} b_j[k] = 0 \), we easily obtain that \( \sum_{v_i \in N_j^{-}} w_{ji}^* = \sum_{v_l \in N_j^{+}} w_{lj}^* \). \( \square \)

We next provide some propositions that are useful for proving the main result in Theorem 5.1, the proof of which can be found in Hadjicostis and Domínguez-García, 2016.

Proposition 2. At each iteration \( k \) during the execution of the algorithm described by (5.7)-(5.12), it holds that

1. For any subset of nodes \( S \subset \mathcal{V} \), if we let \( \mathcal{E}_S^- \) and \( \mathcal{E}_S^+ \) be defined by (5.2) and (5.3) respectively, we have \( \sum_{v_j \in S} b_j[k] = \sum_{(v_j,v_i) \in \mathcal{E}_S^-} w_{ji}[k] - \sum_{(v_l,v_j) \in \mathcal{E}_S^+} w_{lj}[k] \);
2. \( \sum_{j=1}^{n} b_j[k] = 0 \);
3. \( \varepsilon[k] = 2 \sum_{v_j \in \mathcal{V}^+[k]} b_j[k] \) where \( \mathcal{V}^+[k] = \{ v_j \in \mathcal{V} \mid b_j[k] > 0 \} \).

Proof. To prove the first statement, let \( \mathcal{E}_S = \{ (v_j, v_i) \in E_p \mid v_j \in S, v_i \in S \} \)
be the set of edges that are internal to the set \( S \). From the definition of the weight balance for node \( v_j \), we have (after re-arranging the summations)

\[
\sum_{v_j \in S} b_j[k] = \sum_{v_j \in S} \left( \sum_{v_i \in N_j^-} w_{ji}[k] - \sum_{v_i \in N_j^+} w_{lj}[k] \right)
\]

\[
= \sum_{(v_j,v_i) \in E^-} w_{ji}[k] - \sum_{(v_i,v_j) \in E^+} w_{lj}[k] + \sum_{(v_j,v_i) \in E^-} w_{ji}[k] - \sum_{(v_i,v_j) \in E^+} w_{lj}[k]
\]

\[
= \sum_{(v_j,v_i) \in E^-} w_{ji}[k] - \sum_{(v_i,v_j) \in E^+} w_{lj}[k] .
\]

For the second statement, we can take any \( S \subset V \) and argue (using the first statement twice, once for \( S \) and once for \( V \setminus S \) and the facts that \( E^+ = E^- \setminus S \) and \( E^- = E^+ \setminus S \))

\[
\sum_{v_j \in V} b_j[k] = \sum_{v_j \in S} b_j[k] + \sum_{v_j \in V \setminus S} b_j[k] = 0 .
\]

For the third statement, notice that, from the definition of \( \epsilon[k] \) in Definition 3, we have

\[
\epsilon[k] = \sum_{v_j \in V} |b_j[k]| = \sum_{v_j \in V^+[k]} |b_j[k]| + \sum_{v_j \in V \setminus V^+[k]} |b_j[k]|
\]

\[
= \sum_{v_j \in V^+[k]} b_j[k] - \sum_{v_j \in V \setminus V^+[k]} b_j[k] = 2 \sum_{v_j \in V^+[k]} b_j[k] ,
\]

utilizing the definition of \( V^+[k] \) (all nodes have positive balance) and the second statement of this proposition. \( \square \)

**Proposition 3.** Let \( V^+[k] \subset V \) be the set of nodes with positive weight balance at iteration \( k \), i.e., \( V^+[k] = \{ v_j \in V \mid b_j[k] > 0 \} \). During the execution of the algorithm described by (5.7)–(5.12), we have the following:

1. \( b_j[k + 1] \geq \frac{1}{2} b_j[k] > 0 \), for all \( v_j \in V^+[k] \);

2. \( V^+[k] \subseteq V^+[k + 1] \).
5.4. Distributed Balancing with Constrained Weights

Proof. Consider a node \( v_j \in V^+[k] \) with balance \( b_j[k] > 0 \). Suppose for now that (i) all neighbors of node \( v_j \) do not belong in \( V^+[k] \) (i.e., \( N_j^- \cup N_j^+ ) \cap V^+[k] = \emptyset \)) and that (ii) during the update of the weights (following (5.11)), the weights on each edge of node \( v_j \) are away from the lower or upper limits. Then, since nodes outside the set \( V^+[k] \) posses a \( b \) of zero, it follows that the weights are updated as

\[
\begin{align*}
w_{ji}[k + 1] &= w_{ji}[k] - \frac{b_j[k]}{2D_j}, \forall v_i \in N_j^- , \\
w_{lj}[k + 1] &= w_{lj}[k] + \frac{b_j[k]}{2D_j}, \forall v_l \in N_j^+ .
\end{align*}
\]

Thus, the weight balance of node \( v_j \) satisfies

\[
\begin{align*}
b_j[k + 1] &= \sum_{v_i \in N_j^-} w_{ji}[k + 1] - \sum_{v_l \in N_j^+} w_{lj}[k + 1] \\
&= \sum_{v_i \in N_j^-} w_{ji}[k] - \frac{b_j[k]}{2D_j} - \sum_{v_l \in N_j^+} (w_{lj}[k] + \frac{b_j[k]}{2D_j}) \\
&= -D_j \frac{b_j[k]}{2D_j} - d_j^+ \frac{b_j[k]}{2D_j} + b_j[k] \\
&= -D_j \frac{b_j[k]}{2D_j} + b_j[k] = \frac{1}{2} b_j[k].
\end{align*}
\]

For the general case (when node \( v_j \) may have neighbors with positive weight balance and/or the weights reach the lower or upper weight limits on the corresponding edges), we make the following observations:

1. If an in-neighbor \( v_i \in N_j^- \) has positive weight balance, the weight \( \tilde{w}_{ji}[k + 1] \) will satisfy \( \tilde{w}_{ji}[k + 1] \geq w_{ji}[k] - \frac{b_j[k]}{2D_j} \) (from (5.11)). Clearly, if the weight \( \tilde{w}_{ji}[k + 1] \) is within the lower and upper limits (in (5.12)), we have \( w_{ji}[k + 1] = \tilde{w}_{ji}[k + 1] \geq w_{ji}[k] - \frac{b_j[k]}{2D_j} \). Even if the weights are not within the lower and upper limits, we still have

\[
w_{ji}[k + 1] \geq w_{ji}[k] - \frac{b_j[k]}{2D_j}, \forall v_i \in N_j^- .
\]

To see this, notice that:

(i) If \( \tilde{w}_{ji}[k + 1] > w_{ji} \), then \( w_{ji}[k + 1] = w_{ji} \geq \tilde{w}_{ji}[k + 1] = w_{ji}[k] - \frac{b_j[k]}{2D_j} \) (because \( l_{ji} \leq w_{ji}[k] \leq u_{ji} \)).

(ii) If \( \tilde{w}_{ji}[k + 1] < l_{ji} \), then \( w_{ji}[k + 1] = l_{ji} \geq \tilde{w}_{ji}[k + 1] = w_{ji}[k] - \frac{b_j[k]}{2D_j} \).

2. Similar arguments (but reversed) can be used to establish that if an out-neighbor \( v_l \in N_j^+ \) has positive weight balance, the weight \( w_{lj}[k + 1] \) will
satisfy
\[ w_{lj}[k + 1] \leq w_{lj}[k] + \frac{b_j[k]}{2D_j}, \forall v_l \in \mathcal{N}_j^+ . \]

Thus, we conclude that in the general case we still have \( b_j[k + 1] \geq \sum_{v_i \in \mathcal{N}_j^-} (w_{ji}[k] - \frac{b_j[k]}{2D_j}) - \sum_{v_l \in \mathcal{N}_j^+} (w_{lj}[k] + \frac{b_j[k]}{2D_j}) = -D_j \frac{b_j[k]}{2D_j} + b_j[k] = \frac{1}{2} b_j[k]. \)

The second statement in the proposition follows trivially from the first: a positive node remains positive; thus, the set \( \mathcal{V}^+[k] \) can only be enlarged.

**Proposition 4.** During the execution of the algorithm described by (5.7)–(5.12), it holds that \( 0 \leq \varepsilon[k + 1] \leq \varepsilon[k] \).

**Proof.** Let \( \mathcal{V}^+[k] \subset \mathcal{V} \) be the set of nodes with positive weight balance at iteration \( k \), i.e., \( \mathcal{V}^+[k] = \{v_j \in \mathcal{V} \mid b_j[k] > 0\} \) (note that \( \mathcal{V}^+[k] \) has to be a strict subset of \( \mathcal{V} \), e.g., because of the second statement of Proposition 2). Taking \( \mathcal{S} = \mathcal{V}^+[k] \subset \mathcal{V} \), define \( \mathcal{E}_S^- \) and \( \mathcal{E}_S^+ \) as in (5.2) and (5.3) respectively, and let \( \mathcal{E}_S = \{(v_j, v_i) \in \mathcal{E} \mid v_j \in \mathcal{S}, v_i \in \mathcal{S}\} \) be the set of edges that are internal to the set \( \mathcal{S} \). Note that \( \Delta w_{ji}[k] \) necessarily satisfies (e.g., see the discussion in the beginning of the proof of Proposition 3):

\[
\begin{align*}
\Delta w_{ji}[k] &\leq 0, \forall (v_j, v_i) \in \mathcal{E}_S^- , \\
\Delta w_{ji}[k] &\geq 0, \forall (v_j, v_i) \in \mathcal{E}_S^+ , \\
\Delta w_{ji}[k] &= 0, \forall (v_j, v_i) \in \mathcal{E}_S \setminus (\mathcal{E}_S^+ \cup \mathcal{E}_S^- \cup \mathcal{E}_S) .
\end{align*}
\]

Since all nodes in \( \mathcal{S} = \mathcal{V}^+[k] \) have positive weight balance at iteration \( k \) and all remaining nodes have nonpositive weight balance at iteration \( k \), we have

\[
\varepsilon[k] = \sum_{v_j \in \mathcal{V}} |b_j[k]| = \sum_{v_j \in \mathcal{S}} b_j[k] + \sum_{v_j \notin \mathcal{S}} |b_j[k]| = \sum_{(v_j, v_i) \in \mathcal{E}_S^-} w_{ji}[k] - \sum_{(v_i, v_j) \in \mathcal{E}_S^+} w_{lj}[k] + \sum_{v_j \notin \mathcal{S}} |b_j[k]| ,
\]

where in the last line we utilized the result of the first statement of Proposition 2.
5.4. Distributed Balancing with Constrained Weights

Similarly, since from Proposition 3, it holds $S = \mathcal{V}^+[k] \subseteq \mathcal{V}^+[k + 1]$, we have

$$
\varepsilon[k + 1] = \sum_{v_j \in S} b_j[k + 1] + \sum_{v_j \in \mathcal{V} \setminus S} |b_j[k + 1]|
= \sum_{(v_j,v_i) \in \mathcal{E}_S^-} w_{ji}[k + 1] - \sum_{(v_i,v_j) \in \mathcal{E}_S^+} w_{lj}[k + 1]
+ \sum_{v_j \in \mathcal{V} \setminus S} |b_j[k + 1]|.
$$

Putting the two above equations together, we obtain:

$$
\Delta \varepsilon[k] = \sum_{(v_j,v_i) \in \mathcal{E}_S^-} \Delta w_{ji}[k] - \sum_{(v_i,v_j) \in \mathcal{E}_S^+} \Delta w_{lj}[k]
+ \sum_{v_j \in \mathcal{V} \setminus S} (|b_j[k + 1]| - |b_j[k]|).
$$

For $v_j \in \mathcal{V} \setminus S$ we have

$$
b_j[k + 1] = \sum_{v_i \in \mathcal{N}_j^-} w_{ji}[k + 1] - \sum_{v_i \in \mathcal{N}_j^+} w_{lj}[k + 1]
= \sum_{v_i \in \mathcal{N}_j^-} (w_{ji}[k + 1] + \Delta w_{ji}[k]) - \sum_{v_i \in \mathcal{N}_j^+} (w_{lj}[k + 1] + \Delta w_{lj}[k])
= b_j[k] + \sum_{v_i \in \mathcal{N}_j^-} \Delta w_{ji}[k] - \sum_{v_i \in \mathcal{N}_j^+} \Delta w_{lj}[k]
$$

and by the triangle inequality

$$
|b_j[k + 1]| \leq |b_j[k]| + \sum_{v_i \in \mathcal{N}_j^-} |\Delta w_{ji}[k]| + \sum_{v_i \in \mathcal{N}_j^+} |\Delta w_{lj}[k]|.
$$

Thus the last term in (5.14) satisfies

$$
\sum_{v_j \in \mathcal{V} \setminus S} (|b_j[k + 1]| - |b_j[k]|)
\leq \sum_{v_j \in \mathcal{V} \setminus S} \left( \sum_{v_i \in \mathcal{N}_j^-} |\Delta w_{ji}[k]| + \sum_{v_i \in \mathcal{N}_j^+} |\Delta w_{lj}[k]| \right)
\leq \sum_{(v_j, v_i) \in \mathcal{E}_S^-} |\Delta w_{ji}[k]| + \sum_{(v_i, v_j) \in \mathcal{E}_S^+} |\Delta w_{lj}[k]|,
$$

where the last line follows from the fact that the edges, the weights of which change and are incident to nodes in the set $\mathcal{V} \setminus S$, are the edges in $\mathcal{E}_S^-$ and $\mathcal{E}_S^+$ (the other weights in question do not change).
Going back to (5.14), we have that
\[
\Delta \varepsilon [k] = \sum_{(v_j,v_i) \in \mathcal{E}_-^S} \Delta w_{ji}[k] - \sum_{(v_i,v_j) \in \mathcal{E}_+^S} \Delta w_{lj}[k] + \sum_{v_j \in \mathcal{V} \setminus \mathcal{S}} (|b_j[k+1]| - |b_j[k]|)
\leq \sum_{(v_j,v_i) \in \mathcal{E}_-^S} \Delta w_{ji}[k] - \sum_{(v_i,v_j) \in \mathcal{E}_+^S} \Delta w_{lj}[k] + \sum_{(v_j,v_i) \in \mathcal{E}_-^S} |\Delta w_{ji}[k]| + \sum_{(v_i,v_j) \in \mathcal{E}_+^S} |\Delta w_{lj}[k]|
= 0 ,
\]
where the last line follows form the fact that \(\Delta w_{ji}[k] \leq 0 \forall (v_j, v_i) \in \mathcal{E}_-^S\), and \(\Delta w_{lj}[k] \geq 0, \forall (v_l, v_j) \in \mathcal{E}_+^S\).

The following proposition is a refinement of Proposition 4. It basically states that at each iteration \(k\) of the algorithm described by (5.7)–(5.12), the change in the absolute imbalance of the network depends exclusively on (i) the changes in weights on edges that connect positive nodes in \(\mathcal{V}^+ [k]\) and negative nodes in \(\mathcal{V} \setminus \mathcal{V}^+ [k]\), and (ii) the changes in the weight balances of nodes with nonpositive balance that are directly connected with nodes with positive weight balance.

**Proposition 5.** At time step \(k\) of the execution of the algorithm described by (5.7)–(5.12), let \(\mathcal{S} = \mathcal{V}^+ [k] \subset \mathcal{V}\) be the set of nodes with positive weight balance at iteration \(k\) (i.e., \(\mathcal{V}^+ [k] = \{v_j \in \mathcal{V} \mid b_j[k] > 0\}\)) and let \(\overline{\mathcal{S}} = \mathcal{V} \setminus \mathcal{S}\) be the remaining nodes (with zero or negative weight balance). Define \(\mathcal{E}_-^\mathcal{S}\) and \(\mathcal{E}_+^\mathcal{S}\) as in (5.2) and (5.3) respectively, and let \(\mathcal{T} \subseteq \overline{\mathcal{S}}\) be the subset of nodes in \(\overline{\mathcal{S}}\) directly connected to nodes in \(\mathcal{S}\) (i.e., \(\mathcal{T} = \{v_j \in \overline{\mathcal{S}} \mid \exists v_i \ s.t. \ (v_i, v_j) \in \mathcal{E}_-^\mathcal{S} \ or \ (v_j, v_i) \in \mathcal{E}_+^\mathcal{S}\}\)). We have
\[
\Delta \varepsilon [k] = 2 \left( \sum_{(v_j,v_i) \in \mathcal{E}_-^\mathcal{S}} \Delta w_{ji}[k] - \sum_{(v_i,v_j) \in \mathcal{E}_+^\mathcal{S}} \Delta w_{lj}[k] \right) + \sum_{v_j \in \mathcal{T}} (|b_j[k+1]| + b_j[k+1])
\]
\begin{equation}
= \sum_{v_j \in \mathcal{T}} \Delta \varepsilon_j [k] ,
\end{equation}
\begin{equation}
(5.15)
\end{equation}
\begin{equation}
(5.16)
\end{equation}
5.4. Distributed Balancing with Constrained Weights

where

\[ \Delta \varepsilon_j[k] = |b_j[k + 1]| + b_j[k + 1] + \sum_{v_i \in \mathcal{N}_j^- \cap S} 2\Delta w_{ji}[k] + \sum_{v_l \in \mathcal{N}_j^+ \cap S} 2\Delta w_{lj}[k] \leq 0. \]  

(5.17)

Proof. From (5.14) in the proof of Proposition 4, we have

\[ \Delta \varepsilon[k] = \sum_{(v_j, v_i) \in \mathcal{E}_S^-} \Delta w_{ji}[k] - \sum_{(v_i, v_j) \in \mathcal{E}_S^+} \Delta w_{lj}[k] + \sum_{v_j \in \overline{S}} (|b_j[k + 1]| + b_j[k]) \]

\[ = \sum_{(v_j, v_i) \in \mathcal{E}_S^-} \Delta w_{ji}[k] - \sum_{(v_i, v_j) \in \mathcal{E}_S^+} \Delta w_{lj}[k] + \sum_{v_j \in \overline{S}} (|b_j[k + 1]| + b_j[k + 1]) - \sum_{v_j \in \overline{S}} (b_j[k + 1] - b_j[k]), \]

where we used the fact that \( b_j[k] \leq 0 \) for \( v_j \in \overline{S} \). Furthermore, we have

\[- \sum_{v_j \in \overline{S}} (b_j[k + 1] - b_j[k]) = \sum_{v_j \in S} (b_j[k + 1] - b_j[k]) = \sum_{(v_j, v_i) \in \mathcal{E}_S^-} \Delta w_{ji}[k] - \sum_{(v_i, v_j) \in \mathcal{E}_S^+} \Delta w_{lj}[k], \]

where the second (third) line follows from Statement 2 (Statement 1) in Proposition 2 and the definition of \( \Delta w_{ji}[k] \).

To arrive at (5.15), we realize that the term \( |b_j[k + 1]| + b_j[k + 1] \) is necessarily zero for all nodes in the set \( \overline{S} \setminus T \): the reason is that the weights on edges incident to these nodes do not change at all; thus, \( b_j[k + 1] = b_j[k] \leq 0 \) and therefore \( |b_j[k + 1]| = -b_j[k + 1] \).

To complete the proof, we reorder the two summations of the weight changes in the set of edges in \( \mathcal{E}_S^- (\mathcal{E}_S^+) \) in terms of outgoing (incoming) edges of nodes in the set \( T \), which leads us to (5.16). The fact that \( \Delta \varepsilon_j[k] \leq 0 \) can be seen as follows:

(i) If \( b_j[k + 1] \leq 0 \), then \( |b_j[k + 1]| + b_j[k + 1] = 0 \); thus, since \( \Delta w_{ji}[k] \geq 0 \)
for all \( v_j \in \mathcal{T} \) and \( v_i \in \mathcal{N}_{j}^- \cap \mathcal{S} \), and \( \Delta w_{lj}[k] \leq 0 \) for all \( v_j \in \mathcal{T} \) and \( v_i \in \mathcal{N}_{j}^+ \cap \mathcal{S} \), we have \( \Delta \varepsilon_j[k] \leq 0 \).

(ii) If \( b_j[k+1] \geq 0 \), then since
\[
 b_j[k+1] - b_j[k] = \sum_{v_i \in \mathcal{N}_j^- \cap \mathcal{S}} \Delta w_{ji}[k] - \sum_{v_i \in \mathcal{N}_j^+ \cap \mathcal{S}} \Delta w_{lj}[k] \leq 0,
\]
we have
\[
 0 \leq b_j[k+1] \leq \sum_{v_i \in \mathcal{N}_j^- \cap \mathcal{S}} \Delta w_{ji}[k] - \sum_{v_i \in \mathcal{N}_j^+ \cap \mathcal{S}} \Delta w_{lj}[k].
\]
Thus, \( |b_j[k+1]| = b_j[k+1] \) and
\[
 2b_j[k+1] \leq \sum_{v_i \in \mathcal{N}_j^- \cap \mathcal{S}} 2\Delta w_{ji}[k] - \sum_{v_i \in \mathcal{N}_j^+ \cap \mathcal{S}} 2\Delta w_{lj}[k];
\]
thus, \( \Delta \varepsilon_j[k] \leq 0. \)

### 5.4.2 Edges with Unidirectional Communication Capability

If communication capability is not necessarily bidirectional between neighboring nodes, one might be tempted to consider a variation of Algorithm 13, in which, at each iteration, nodes with positive weight balance attempt to change the weights on their outgoing edges only (since nodes have no way of sending information to their in-neighbors, they do not attempt to change the weights on their incoming edges). It quickly becomes evident, however, that such a variation of Algorithm 13 fails. One obvious limitation of this variation of the algorithm is that weights on outgoing edges can only be increased (because only nodes with positive balance make changes, and these changes are reflected as positive changes on the weights of their outgoing edges). However, even if we initialize the weights on each edge at the lower bounds (i.e., set \( w_{lj}[0] = l_{lj}, \forall v_i \in \mathcal{N}_j^+ \)) and/or we allow all nodes (with either positive or negative balance) to make changes, this variation of the algorithm fails. The following example illustrates this.

**Example 5.3.** Consider the digraph \( \mathcal{G}_p = (\mathcal{V}, \mathcal{E}_p) \) shown in Figure 5.7 with four nodes \( \mathcal{V} = \{v_1, v_2, v_3, v_4\} \) and five edges, with lower and upper bounds on the edges as shown in the figure. It is easy to verify that the graph satisfies the circulation conditions in Definition 6. In fact, if we run Algorithm 13
5.4. Distributed Balancing with Constrained Weights

Figure 5.7: Digraph used in Example 5.3 to demonstrate the ineffectiveness of the variation of Algorithm 13.

Figure 5.8: Execution of the variation of the Algorithm 13 on the digraph of Figure 5.7, shown in terms of absolute (total) imbalance at each iteration (left) and node balances at each iteration (right).

(assuming, of course, a bidirectional communication topology), we arrive at the following solution after approximately 200 iterations: $w_{21} = 5$, $w_{32} = 1$, $w_{42} = 4$, $w_{13} = 1$, $w_{14} = 4$. However, if we restrict ourselves to a communication topology that matches the physical topology (i.e., $G_c = G_p$) and run the aforementioned variation of Algorithm 13, we see that the algorithm fails to reach a balanced solution because it gets stuck (after a few steps) at a set of weights that does not achieve balance: $w_{21} = 5$, $w_{32} = 2$, $w_{42} = 3$, $w_{13} = 1$, $w_{14} = 3$. Figure 5.8 shows plots of the absolute balance (left) and the node balances (right) at each iteration during the execution of the variation of Algorithm 13.
Distributed Weight Balancing

The problem in the above execution of the variation of Algorithm 13 is that information cannot be propagated backwards from node $v_3$, which ideally would like to have the edge $(v_3, v_2)$ have a weight of one. Meanwhile, node $v_2$ is under the impression that it has to increase/maintain the weights on both of its outgoing edges. More generally, a node with positive balance cannot inform/request that its in-neighbors decrease the weights on its incoming edges; it can only attempt to increase its outgoing edges, which may have already reached their maximum possible values. This problem arises even if the algorithm is modified to allow nodes with negative balances to make updates on the weights of their outgoing edges.

As demonstrated by the above example, there is a need for developing distributed algorithms for weight/flow balancing under more general/restrictive communication topologies that do not necessarily allow for bidirectional communication capability between a pair of nodes that admits a nonzero flow in the physical topology.

5.5 Balancing in Undirected Graphs

In many applications of interest, the underlying physical topology $G_p = (V, E_p)$ may include links which can admit positive weights in either direction. For example, a particular link $\{v_j, v_i\}$ may admit a positive weight in the interval $[0, u]$, in the direction from $v_i$ to $v_j$, or vice-versa. Such situations arise naturally in power distribution applications where power flow on a particular line can be in either direction, as long as it is bounded by some upper limit (presumably determined by the capacity of the line). Clearly, in such situations we can choose an arbitrary direction for the link, say the direction from $v_i$ to $v_j$, and require that its weight be constrained to lie in the interval $[-u_{ji}, u_{ji}]$ (where $u_{ji} = u$). More generally, we can think of the following problem formulation, whose main difference from the problem in Definition 5 is that the lower/upper bounds are not restricted to be nonnegative (which means that the circulation conditions in Definition 6 cannot be applied, at least not directly).

Definition 8. (Distributed Weight/Flow Balancing Problem with Undirected Flows.) We are given a digraph $G_p = (V, E_p)$ of order $n$ (physical topology), as well as lower and upper bounds $l_{ji}$ and $u_{ji}$ ($l_{ji} \leq u_{ji}$) on each edge
(v_j, v_i) \in \mathcal{E}_p$. We want to develop a distributed algorithm that allows the nodes to iteratively adjust the weights on their edges so that they eventually obtain a set of weights \{w_{ji} \mid (v_j, v_i) \in \mathcal{E}_p\} that satisfy the following:

1. **Interval constraints**: \(l_{ji} \leq w_{ji} \leq u_{ji}\) for each edge \((v_j, v_i) \in \mathcal{E}_p\);

2. **Balance constraints**: \(b^+_j = b^-_j\) for every \(v_j \in \mathcal{V}\), where \(b^+_j\) and \(b^-_j\) are given in Definition 1.

As in the algorithms developed earlier in this chapter, the distributed algorithm needs to respect the communication constraints imposed by the communication graph \(G_c = (\mathcal{V}, \mathcal{E}_c)\) (cyber topology) that describes the communication capabilities between pairs of nodes.

Though the circulation conditions in Definition 6 cannot be applied directly, we can construct a modified physical topology \(\tilde{\mathcal{G}}_p = (\mathcal{V}, \tilde{\mathcal{E}}_p)\), which admits feasible and balanced flows if and only if \(\tilde{\mathcal{G}}_p\) admits feasible and balanced flows, and which can be checked against the circulation conditions. More specifically, for each edge \((v_j, v_i) \in \mathcal{E}_p\), depending on the values of the lower and upper bounds \((l_{ji}\) and \(u_{ji}\)), we add one or two edges in \(\tilde{\mathcal{E}}_p\), as follows:

- **Case 1**: If \(0 \leq l_{ji} \leq u_{ji}\), then we add \((v_j, v_i) \in \tilde{\mathcal{E}}_p\) with \(\tilde{l}_{ji} = l_{ji}\) and \(\tilde{u}_{ji} = u_{ji}\).
- **Case 2**: \(l_{ji} \leq u_{ji} \leq 0\), then we add \((v_i, v_j) \in \tilde{\mathcal{E}}_p\) with \(\tilde{l}_{ij} = -u_{ji}\) and \(\tilde{u}_{ij} = -l_{ji}\).
- **Case 3**: \(l_{ji} \leq 0 \leq u_{ji}\), then we add \((v_j, v_i) \in \tilde{\mathcal{E}}_p\) with \(\tilde{l}_{ji} = 0\) and \(\tilde{u}_{ji} = u_{ji}\) and \((v_i, v_j) \in \tilde{\mathcal{E}}_p\) with \(\tilde{l}_{ij} = 0\) and \(\tilde{u}_{ij} = -l_{ji}\).

**Note:** In both Case 2 and Case 3, we assume that \((v_i, v_j) \notin \mathcal{E}_p\), otherwise we could run into a multigraph situation, with multiple edges between a pair of nodes.

Clearly, \(\tilde{\mathcal{G}}_p\) has lower/upper limits on its edges that satisfy the requirements for applying the circulation conditions. Moreover, we argue below that any feasible and balanced weight assignment in \(\tilde{\mathcal{G}}_p\) corresponds to a feasible and balanced weight assignment in \(\mathcal{G}_p\), and vice-versa. In particular, if we have a feasible and balanced weight assignment \(\tilde{w}_{ji}, \forall (v_j, v_i) \in \tilde{\mathcal{E}}_p\) then we can find a feasible and balanced assignment in \(\mathcal{G}_p\) as follows:

- For edge \((v_j, v_i) \in \mathcal{E}_p\) that satisfies Case 1, we choose \(w_{ji} = \tilde{w}_{ji}\).
• For edge \((v_j, v_i) \in \mathcal{E}_p\) that satisfies Case 2, we choose \(w_{ji} = -\tilde{w}_{ij}\).

• For edge \((v_j, v_i) \in \mathcal{E}_p\) that satisfies Case 3, we choose \(w_{ji} = \tilde{w}_{ij} - \tilde{w}_{ij}\) (which can easily be shown to lie in the interval \([l_{ji}, u_{ji}]\) assuming \(\tilde{w}_{ji}\) and \(\tilde{w}_{ij}\) are within the appropriate bounds for the edges \((v_j, v_i), (v_i, v_j) \in \mathcal{E}_p\)).

Since direction and weight values are chosen so that they contribute in the same way to the balances of nodes \(v_j\) and \(v_i\), the assignment of weights in \(\mathcal{G}_p\) will result in balanced nodes.

In the reverse direction, if we have a feasible and balanced weight assignment \(w_{ji}, \forall (v_j, v_i) \in \mathcal{E}_p\), then we can find a feasible and balanced assignment in \(\tilde{\mathcal{G}}_p\) as follows:

• For edge \((v_j, v_i) \in \mathcal{E}_p\) that satisfies Case 1, we set \(\tilde{w}_{ji} = w_{ji}\).

• For edge \((v_j, v_i) \in \mathcal{E}_p\) that satisfies Case 2, we set \(\tilde{w}_{ij} = -w_{ji}\).

• For edge \((v_j, v_i) \in \mathcal{E}_p\) that satisfies Case 3, we set \(\tilde{w}_{ji} = w_{ji}\) and \(\tilde{w}_{ij} = 0\) if \(w_{ji} \geq 0\), whereas we set \(\tilde{w}_{ij} = -w_{ji}\) and \(\tilde{w}_{ji} = 0\) if \(w_{ji} < 0\).

The above choices can be seen to contribute in exactly the same way to the balances of nodes \(v_j\) and \(v_i\), thus the assignment of weights in \(\tilde{\mathcal{G}}_p\) will result in balanced nodes.

The above modified physical topology \(\tilde{\mathcal{G}}_p\) can be used to check whether a feasible and balanced weight assignment is possible for \(\mathcal{G}_p\). In fact, the above modified topology can be used by the distributed algorithms described earlier in this chapter to obtain a feasible and balanced weight assignment. In the algorithms that we develop in Chapter 6, we take this approach one step further, and have the algorithm, at any given iteration \(k\), combine the weights \(\tilde{w}_{ji}[k]\) and \(\tilde{w}_{ij}[k]\) to obtain a single nonzero weight value \(w_{ji}[k] = \tilde{w}_{ji}[k] - \tilde{w}_{ij}[k]\) (which is arbitrarily chosen here to have direction from \(v_i\) to \(v_j\)).

5.6 Discussion

The problem of obtaining a weight balanced graph is related to matrix scaling problems, which have been addressed in the context of nonnegative matrices (see, for example, early work by Marshall and Olkin, 1968; Djokovic,
The prototypical version of the matrix scaling problem considers the following setup: given a nonnegative $n \times n$ matrix $P = [p_{ji}]$ and two positive $n$-dimensional vectors $r$ and $c$, find positive diagonal matrices $\Lambda = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_n)$ and $M = \text{diag}(\mu_1, \mu_2, ..., \mu_n)$ such that the scaled matrix $\hat{P} = \Lambda PM$ (note that $\hat{P}(j, i) = \lambda_j p_{ji} \mu_i$) satisfies:

$$\hat{P} \mathbf{1}_n = r, \quad \mathbf{1}_n^T \hat{P} = c^T,$$

i.e., the $j^{th}$ row of matrix $\hat{P}$ sums to $r_j$ and the $i^{th}$ column of matrix $\hat{P}$ sums to $c_i$ (of course, one must have that $\sum_j r_j = \sum_i c_i$). A special case of the matrix scaling problem is the case when $r = c = \mathbf{1}_n$, which leads to a doubly stochastic scaled matrix $\hat{P}$. If the diagonal of matrix $P$ is zero and $r = c$, then the matrix $\hat{P}$ of a scaled solution will be a balanced matrix (restricted to have the particular form $\Lambda PM$).

One of the early motivations for the matrix scaling problem was the desire to start from the stochastic matrix of a Markov chain and obtain a scaled version of it that is doubly stochastic and adheres to the sparsity structure of the original one Seneta, 2006. Many applications of matrix scaling can also be found in economics, urban planning, statistics, and demography—see, for instance, the discussions in Schneider and Zenios, 1990. Note that the matrix scaling problem, which also makes sense for rectangular matrices, has been studied mainly without considering any distributivity constraints; instead, our focus in this chapter has been on distributed algorithms for obtaining a weight balanced matrix (i.e., we have $r = c$ in the matrix scaling problem), without restricting the weights to form a balanced matrix of the form $\Lambda PM$ (but in some cases restricting individual weights to lie within specified intervals). Note that the solution presented by (5.4) and Algorithm 12, there is a restriction (imposed by the algorithmic solution) that the outgoing weights of each node are equal.

A line of research that is related to the distributed weight balancing approaches described in this chapter concerns balancing with weights that are restricted to be positive integers. In this line of work, one is given a digraph and the goal is to assign positive integers on its edges so that the resulting digraph is balanced. Starting with the work in Gharesifard and Cortés, 2009, several researchers proposed algorithms for integer weight balancing Gharesifard and Cortés, 2012; Rikos and Hadjicostis, 2013; Rikos et al., 2014. A key
characteristic of all these works is that the weights are unrestricted (weights need to be positive integers but there is no upper limit on their value); furthermore, each node needs to be able to assign different weights on its outgoing edges (which was not the case for the balancing algorithms presented in this chapter for real weights that are unrestricted). In contrast to the asymptotic algorithms for real weight balancing described in this chapter, integer weight balancing algorithms complete in finite time; in fact, the work in Rikos et al., 2014 established an upper bound on the number of iterations needed to reach an integer weight balanced digraph. Extensions of integer weight balancing have also been pursued in directions that involve communication delays Rikos and Hadjicostis, 2017, following the delay models described in Section 3.5 of Chapter 3.

Most works on integer weight balancing deal with positive integer weights that are unconstrained in the sense that there is no upper bound in their value. Recent work in Rikos and Hadjicostis, 2016 has also looked at the case when the positive integer weights are constrained to lie in an interval, i.e., the weight $w_{j_i}$ on edge $(v_j, v_i)$ has to be an integer in an interval $[l_{j_i}, u_{j_i}]$ where $0 < l_{j_i} \leq u_{j_i}$. 
Part II

Applications
6

Coordination of Distributed Energy Resources

In this chapter, we illustrate the application of some of the distributed algorithms presented in earlier chapters to electric power networks. Specifically, we look at the problem of determining the power injections in certain nodes of the power network so as to satisfy desirable objectives, e.g., the total power injected in these nodes is equal to some pre-specified value, and each individual injection lies within some interval. Rather than attempting to utilize very detailed models, we utilize fairly simplified models for the system physical layer, which can be succinctly described by a network-flow-theoretic model. The reason for adopting such a simplified model is to allow the reader to focus on the application of the algorithms without having to spend a substantial amount of time understanding the physical layer model.

6.1 Introduction and Motivation

The structure and functionality of the electricity infrastructure is undergoing a radical transformation in a push to increase efficiency and reliability. This transformation is enabled by the adoption of advanced communication and control; and the integration of distributed energy resources (DERs), e.g., photovoltaic systems, micro-turbines, and plug-in hybrid vehicles. Such DERs
can provide functionality to the electric grid they are connected to beyond their intended purpose, e.g., provision of reactive power for voltage control, and active power for frequency regulation (see, e.g., Joos et al., 2000).

In order to enable this additional functionality, it is necessary to develop appropriate control and coordination schemes. One such potential scheme relies on a centralized control architecture in which each DER is directly coordinated by (and communicates with) a central decision maker. An alternative approach is to distribute the decision making, which obviates the need for a central decision maker to coordinate the DERs. In both cases, the decision making involves solving an optimal resource allocation problem for coordinating the DERs to collectively provide a certain amount of active (or reactive) power, subject to their capacity constraints, and constraints on the active (or reactive) power flows in the electrical network interconnecting the DERs, while minimizing some cost.

In this chapter, we address the optimal resource allocation problem for DER coordination when active power is the resource to be allocated, and provide distributed algorithms to solve it. Specifically, we first consider the case when there are constraints on DER capacity but there are no constraints on the amount of power that can flow across the electrical lines in the (physical) network interconnecting the DERs, and the objective function is identically zero, i.e., the objective is to simply find a feasible solution. Then, we consider the case when there are constraints on DER capacity but no constraints on the network flows, and the objective function is to minimize the sum of quadratic costs associated to the DERs. Finally, we consider the case when there are constraints on DER capacity and on the network flows, but the objective function is identically zero, i.e., the objective is to simply find a feasible solution.

### 6.2 Power Distribution Network Model

In this section, we first develop a network-flow-theoretic model to describe the flow of active power among buses in the power distribution network (physical layer) to which the DERs are connected. Then, we develop another graph-theoretic model to describe the exchange of information among buses (cyber layer).
6.2.1 Physical Layer Model

Consider a loop-less electric power distribution network with \( n \) buses, \( v_1, v_2, v_n \), where each pair of buses (nodes), \( v_i, v_j \), are connected by, at most, one power distribution line. Assume that all power distribution lines in the network are short and lossless. For every distribution line connecting a pair of nodes, \( v_i \) and \( v_j \), choose an arbitrary sign convention for the flow of power across the line. Then, we can define a connected directed graph, \( G_p = (\mathcal{V}_p, \mathcal{E}_p) \), where \( \mathcal{V}_p = \{v_1, v_2, \ldots, v_n\} \) is the vertex set (each element corresponds to a bus in the network), and \( \mathcal{E}_p \subseteq (\mathcal{V}_p \times \mathcal{V}_p) \setminus \{(v_j, v_j) \mid v_j \in \mathcal{V}_p\} \) is the edge set, where \((v_j, v_i) \in \mathcal{E}_p\) if buses \( v_j \) and \( v_i \) are electrically connected by a distribution line, and the flow of power from \( v_i \) to \( v_j \) is positive according to the sign convention chosen.\(^1\) Note that since there are no loops in the network and the digraph is connected, we have that \(|\mathcal{E}_p| = |\mathcal{V}_p| - 1 = n - 1\).

Associated with each \((v_j, v_i) \in \mathcal{E}_p\), we can associate a label \( e_m \), and define a unique variable \( f_{e_m} \), \( m = 1, 2, \ldots, |\mathcal{E}_p| \), taking values in the interval \([f_{e_m}, \overline{f}_{e_m}]\), that describes the flow of active power across the distribution line connecting buses \( v_i \) and \( v_j \), where \( \overline{f}_{e_m} > 0 \) and \( f_{e_m} := -\overline{f}_{e_m} \) are determined by the voltage magnitudes of nodes \( v_i \) and \( v_j \) and the susceptance of the power distribution line. Connected to each node \( v_j \), there could be several DERs the output power of which can be varied within some interval. Also, connected to each bus \( v_j \), there could be electrical loads the output power of which is constant and cannot be controlled. Let \( p_j \) denote the amount of active power that the DERs connected to node \( v_j \), \( j = 1, 2, \ldots, n \), collectively inject into the network after serving the power demand of the non-controllable electrical loads that may be also connected at bus \( v_j \). Also, let \( p_j \) and \( \overline{p}_j \) denote the lower and upper limits on \( p_j \), \( j = 1, 2, \ldots, n \), which are determined by the limits on the output power of the DERs connected to node \( v_j \). Define the following vectors: \( f = [f_{e_1}, f_{e_2}, \ldots, f_{e_{n-1}}]^T \), \( \overline{f} = [\overline{f}_{e_1}, \overline{f}_{e_2}, \ldots, \overline{f}_{e_{n-1}}]^T \), and \( \overline{f} = [\overline{f}_{e_1}, \overline{f}_{e_2}, \ldots, \overline{f}_{e_{n-1}}]^T \). Similarly, define \( p = [p_1, p_2, \ldots, p_n]^T \), \( \overline{p} = [\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_n]^T \), and \( \overline{p} = [\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_n]^T \). Then, the pair \((p, f)\) must

\(^1\)If one ignores the power flow sign convention, the electrical connectivity among nodes can be described by an undirected graph, \( G_u = (\mathcal{V}_u, \mathcal{E}_u) \), where \( \mathcal{V}_u = \{v_1, v_2, \ldots, v_n\} \) is the vertex set (each element corresponds to a bus in the network), and \( \mathcal{E}_u \subseteq \{(v_i, v_j) \mid v_i, v_j \in \mathcal{V}_u, v_i \neq v_j\} \) is the edge set, where \( \{v_j, v_i\} \in \mathcal{E}_p \) if buses \( v_j \) and \( v_i \) are electrically connected by a distribution line.
satisfy:

\[ p = Mf, \tag{6.1} \]
\[ p \leq p \leq \bar{p}, \tag{6.2} \]
\[ f \leq f \leq \bar{f}, \tag{6.3} \]

where \( M = [m_{im}] \in \mathbb{R}^{n \times (n-1)} \) is the node-to-edge incidence matrix of \( G_p \), i.e.,

\[ m_{im} = \begin{cases} 
1, & \text{if } e_m = (v_j, v_i), \\
-1, & \text{if } e_m = (v_i, v_j), \\
0, & \text{otherwise.} 
\end{cases} \tag{6.4} \]

Although not explicitly written in (6.1) – (6.3), by adding the equations in (6.1), one can see that the active power injection vector, \( p \), must be such that:

\[ 0 = 1^T_n p, \tag{6.5} \]

which is consistent with the fact that the distribution lines are lossless; thus, the total power injected into the network must be identically equal to zero as stated in (6.5).

### 6.2.2 Cyber Layer Model

We assume that each bus in the power network is endowed with a processor that is capable of performing low-complexity computations, e.g., addition and multiplication, based on information obtained (i) locally, e.g., from measurements, and (ii) from other processors via some communication channel not necessarily bidirectional. Thus, the exchange of information between the processors can be described by a directed graph \( G_c = (V_c, E_c) \), where each element in \( V_c \) corresponds to a processor located at a particular bus, and \( E_c \subseteq V_c \times V_c \), with the ordered pair \((v_j, v_i) \in E_c\) if node \( v_j \) can receive information from node \( v_i \). Note that since each bus in the electrical (physical) network is endowed with a processor, there is a one-to-one correspondence between elements in \( V_c \) and \( V_p \); therefore, without loss of generality, we will use the same index for each bus in the power network and the processor it is endowed with, i.e., \( V_c := \{ v_1, v_2, \ldots, v_n \} \), where each \( v_i \in V_c \) corresponds to the processor with which bus \( v_i \in V_p \) is endowed. In the remainder, we will assume that the directed graph \( G_c \) is strongly connected.
6.3 Fair-Splitting Allocation

Assume that for every $p \in \mathbb{R}^n$ satisfying the constraints in (6.2) and (6.5), i.e., for every $p \in \mathcal{P} = \{ p \in \mathbb{R}^n : p \leq p \leq \overline{p}, \ 1^T_n p = 0 \}$, the vector $f$ that results from (6.1) is such that the constraint in (6.3) is satisfied, i.e., $\underline{f} \leq f \leq \overline{f}$. Then, the model requirements in (6.1)–(6.5) can be simplified to

$0 = 1^T_n p,$ \hspace{1cm} (6.6)

$p \leq p \leq \overline{p}.$ \hspace{1cm} (6.7)

Consider now the problem of choosing the values of the first $n - 1$ injections, $p_1, p_2, \ldots, p_{n-1}$, so that the the value of the $n^{th}$ injection is equal to some prescribed value, $-P$, $p_n \leq -P \leq \overline{p}_n$; then, clearly, the first $n - 1$ injections must satisfy

$$\sum_{i=1}^{n-1} p_i = P,$$

$$\underline{p}_i \leq p_i \leq \overline{p}_i, \quad i = 1, 2, \ldots, n - 1.$$ \hspace{1cm} (6.8)

Assume that $\sum_{j=1}^{n-1} \underline{p}_j \leq P \leq \sum_{j=1}^{n-1} \overline{p}_j$; then, one possible choice is to allocate the total amount of power to be injected into node $n$, $p_n = -P$, among the other nodes by setting their individual contributions as follows:

$$p_j = \underline{p}_j + \frac{P - \sum_{\ell=1}^{n-1} \underline{p}_\ell}{\sum_{\ell=1}^{n-1} (\overline{p}_\ell - \underline{p}_\ell)} (\overline{p}_j - \underline{p}_j), \quad j = 1, 2, \ldots, n - 1.$$ \hspace{1cm} (6.9)

**Equal Upper and Lower Injection Limits**

In this case, $\underline{p}_i = \underline{p}$ and $\overline{p}_i = \overline{p}$, with $\underline{p} \leq \overline{p}$. Then, the expression in (6.9) reduces to

$$p_j = \frac{P}{n-1}, \quad j = 1, 2, \ldots, n - 1.$$ \hspace{1cm} (6.10)

Assume now that only node $v_n$ is aware of the value of $P$, and each node is aware of the total number of nodes in the network, $n$. Further, as described in Section 6.2.2, assume that on top of the physical network, there exists a cyber

\footnote{It should be clear that, given $p$, there is a unique solution to (6.1) due to the lack of loops in the network.}
network, characterized by a strongly connected graph $G_c = \{V_c, E_c\}$, $V_c := V_p$, that allows nodes in the physical network to exchange information and perform computations locally. Then, under Assumptions [A1]–[A5], the nodes can obtain their contribution in a distributed fashion as follows. Each node executes a linear average-consensus iteration of the form

$$x_j[k + 1] = \sum_{v_i \in N_j^c \cup \{v_j\}} w_{ji} x_i[k],$$

$$x_j[0] = \begin{cases} 0, & \text{if } j = 1, 2, \ldots, n - 1, \\ P, & \text{if } j = n, \end{cases} \quad (6.11)$$

with the $w_{ji}$’s forming a doubly stochastic matrix $W = [w_{ji}]$, which the nodes can obtain in distributed fashion as discussed in Chapter 3 for the undirected graph case, and in Chapter 5 for the directed graph case. From the developments in Chapter 3, one can easily see that

$$\lim_{k \to \infty} x_i[k] = \frac{P}{n}. \quad (6.12)$$

Then, since each node $i$ is aware of the total number of nodes in the network, it can compute its contribution as follows:

$$p_j = \frac{n}{n - 1} \lim_{k \to \infty} x_j[k] = \frac{P}{n - 1}. \quad (6.13)$$

The above scheme is not ideal because it requires each node to know the total number of nodes in the network. Additionally, one would like to ensure that nodes can correctly compute their contributions even in the presence of packet drops or asynchronous operation. To address the first problem, nodes could execute the ratio-consensus algorithm described in Chapter 3. Thus, if we use its simplest variant described in Section 3.4.1, each node $v_j$ will maintain, at iteration $k$, state variables $y_j[k]$ and $z_j[k]$ and update them as follows:

$$y_j[k + 1] = \sum_{v_i \in N_j^c \cup \{v_j\}} \frac{y_i[k]}{1 + d_i^+}, \quad (6.14)$$

$$z_j[k + 1] = \sum_{v_i \in N_j^c \cup \{v_j\}} \frac{z_i[k]}{1 + d_i^+}, \quad (6.15)$$
with

\[
y_j[0] = \begin{cases} 
0, & \text{if } j = 1, 2, \ldots, n - 1, \\
\bar{P}, & \text{if } j = n,
\end{cases}
\]  
(6.16)

\[
z_j[0] = \begin{cases} 
1, & \text{if } j = 1, 2, \ldots, n - 1, \\
0, & \text{if } j = n.
\end{cases}
\]  
(6.17)

Then, as discussed in Chapter 3, we have that, for all \( v_j \in \mathcal{V}_c \),

\[
\lim_{k \to \infty} \frac{y_i[k]}{z_j[k]} = \frac{\sum_{\ell=1}^{n} y_{\ell}[0]}{\sum_{\ell=1}^{n} z_{\ell}[0]} = \frac{\bar{P}}{n-1};
\]  
(6.18)

therefore, each node \( v_i \) can compute its contribution as

\[
p_j = \lim_{k \to \infty} \frac{y_j[k]}{z_j[k]}.
\]  
(6.19)

To address the second problem mentioned above, i.e., the possibility of having packet drops and/or asynchronous operation, one could implement the robust version of the ratio-consensus algorithm as described in Section 3.5, which would enable the nodes to compute the correct contributions, i.e., \( p_j = \bar{P}/(n - 1) \), \( j = 1, 2, \ldots, n - 1 \), with probability one.

**Unequal Injection Upper and Lower Limits**

Consider now the general case where, for some \( v_j, v_i \in \mathcal{V}_c \), \( v_j \neq v_i \), \( \bar{p}_j \neq \bar{p}_i \) and/or \( \underline{p}_j \neq \underline{p}_i \). Assume that only node \( n \) is aware of the value of \( p_n = -\bar{P} \), and each other node \( v_j, j \in \{1, 2, \ldots, n - 1\} \) is only aware of its own upper and lower power injection limits, \( \bar{p}_j \), and \( \underline{p}_j \). Then, in order for each \( v_j j \in \{1, 2, \ldots, n - 1\} \) to compute its contribution, as determined by (6.9), it needs to obtain \( \gamma := \frac{p - \sum_{\ell=1}^{n-1} p_{\ell}}{\sum_{\ell=1}^{n-1} (\bar{p}_\ell - \underline{p}_\ell)} \), which clearly depends on information that node \( v_j \) does not possess. However, note that \( \gamma \) is the ratio of two quantities, each of which is a sum of quantities known to individual nodes. Thus, a natural way for the nodes to obtain \( \gamma \) in a distributed fashion is via the ratio-consensus
6.4. Optimal Allocation

iterations in (6.14) – (6.15) with

\[ y_j[0] = \begin{cases} \frac{-P_j}{P}, & \text{if } j = n, \\ \frac{y_j[0]}{y_j[0]} + \frac{P_j}{P}, & \text{if } j = n \end{cases} \]  

(6.20)

\[ z_j[0] = \begin{cases} \frac{P_j - P}{P}, & \text{if } j = 1, 2, \ldots, n - 1, \\ 0, & \text{if } j = n. \end{cases} \]  

(6.21)

Let \( \gamma_j[k] := \frac{y_j[k]}{z_j[k]}, j = 1, 2, \ldots, n; \) then, clearly,

\[
\lim_{k \to \infty} \gamma_j[k] = \frac{\sum_{\ell=1}^{n} y_{\ell}[0]}{\sum_{\ell=1}^{n} z_{\ell}[0]} = \frac{P - \sum_{\ell=1}^{n-1} p_{\ell}}{\sum_{\ell=1}^{n-1} (p_{\ell} - p_{\ell})} = \gamma; \quad (6.22)
\]

therefore, at each iteration \( k, \) each node \( v_j \) estimates its contribution as follows

\[
p_j[k] = p_j + \gamma[k](p_j - p_j); \quad (6.23)
\]

in the limit we have

\[
p_j = p_j + \frac{P - \sum_{\ell=1}^{n-1} p_{\ell}}{\sum_{\ell=1}^{n-1} (p_{\ell} - p_{\ell})} (p_j - p_j). \quad (6.24)
\]

6.4 Optimal Allocation

In this section, we consider the same setting as in the previous section, i.e., we assume that for every \( p \) satisfying (6.2) and (6.5), the corresponding flow vector \( f \) that results from (6.1) is such that the constraint in (6.3) is satisfied. Here, we assume that, associated to each \( p_j, j = 1, 2, \ldots, n - 1, \) there is a quadratic cost of the form \( \frac{(p_j - \alpha_j)^2}{2\beta_j}, \alpha_j \leq 0, \beta_j > 0 \) (see, e.g., Bergen and Vittal, 2000). Then, for a given \( P, \) \( p_n \leq -P \leq p_n, \) we are interested in determining the active power injections in the first \( n - 1 \) buses that result in minimum cost, so that \( p_n = -P. \) The problem can be cast as the following
optimization program:

\[
\begin{align*}
\text{minimize} & \quad \sum_{j=1}^{n-1} \frac{(p_j - \alpha_j)^2}{2\beta_j} \\
\text{subject to} & \quad \sum_{j=1}^{n-1} p_j = P \\
& \quad p_j \leq \bar{p}_j \leq \underline{p}_j, \quad j = 1, 2, \ldots, n - 1,
\end{align*}
\]

which we refer to as the optimal allocation problem.

**A Centralized Solution to the Optimal Allocation Problem**

The optimization program in (6.25) is convex and has a separable structure

[Bertsekas, 2004](pp. 502-506); its solution can be easily found by solving its

Lagrange dual (see, e.g., [Bertsekas, 2004](#); [Boyd and Vandenberghe, 2004](#)),

which is given by

\[
\begin{align*}
\text{maximize} & \quad \lambda P + \sum_{j=1}^{n-1} c_j(\lambda) \\
\text{subject to} & \quad \lambda \geq 0,
\end{align*}
\]

where \( c_j(\lambda), \ j = 1, 2, \ldots, n - 1, \) are (see [Domínguez-García et al., 2012a](#); [Cady et al., 2015b](#) for its derivation)

\[
c_j(\lambda) = \begin{cases} 
\frac{(p_j - \alpha_j)^2}{2\beta_j} - \lambda p_j, & 0 \leq \lambda < \lambda_{2j-1}, \\
-\lambda(\alpha_j + \frac{\beta_j}{2}), & \lambda_{2j-1} \leq \lambda \leq \lambda_{2j}, \\
\frac{2\beta_j}{\alpha_j - \alpha_j} - \lambda x_j, & \lambda_{2j} < \lambda,
\end{cases}
\]

with \( \lambda_{2j-1} = \frac{p_j - \alpha_j}{\beta_j} > 0 \) and \( \lambda_{2j} = \frac{\bar{p}_j - \alpha_j}{\beta_j} > 0 \). The Lagrange dual problem in (6.26) is convex and, in this case, its solution also provides the optimal solution to the primal problem in (6.25). It is easy to see that all the \( c_j(\cdot) \)'s are continuously differentiable; therefore, the cost function in (6.26) is also continuously differentiable. Thus, if (6.26) is feasible, the optimal solution \( \lambda^* \) must satisfy

\[
P - \sum_{j=1}^{n-1} g_j(\lambda^*) = 0, \quad (6.28)
\]
6.4. Optimal Allocation

where

\[ g_j(\lambda) = -\frac{dc_j(\lambda)}{d\lambda} = \begin{cases} p_j, & 0 \leq \lambda < \lambda_{2j-1}, \\ \alpha_j + \lambda \beta_j, & \lambda_{2j-1} \leq \lambda \leq \lambda_{2j}, \\ \bar{p}_j, & \lambda_{2j} < \lambda. \end{cases} \] (6.29)

Now, obtaining the value \( \lambda^* \) that satisfies (6.28) is equivalent to finding the point at which the functions \( g(\lambda) := \sum_{j=1}^{n-1} g_j(\lambda) \) and \( h(\lambda) := P \) intersect. In this regard, it is easy to see that \( g_j(\lambda) \) is a monotonically increasing piecewise linear function, thus \( g(\lambda) \) is also a monotonically increasing piecewise linear function with, at most, \( 2(n-1) + 1 \) line segments defined by the following \( 2(n-1) \) points:

\[ U = \{ \lambda_1, \lambda_2, \ldots, \lambda_{2(n-1)-1}, \lambda_{2(n-1)} \}. \]

Let \( U = U^+ \cup U^- \), where \( U^+ \) and \( U^- \) contain the \( \lambda_j \)'s in \( U \) that satisfy \( g(\lambda_i) \geq P \) and \( g(\lambda_j) < P \), respectively. Now, following the approach proposed in Madrigal and Quintana, 2000, in order to find \( \lambda^* \), since \( g(\lambda) \) is piecewise linear, we need to find \( \lambda^+ \) and \( \lambda^- \) satisfying

\[ \lambda^+ = \arg \min_{\lambda_j \in U^+} |g(\lambda_j) - P|, \] (6.30)

\[ \lambda^- = \arg \min_{\lambda_j \in U^-} |g(\lambda_j) - P|, \] (6.31)

and interpolate between these two values to obtain \( \lambda^* \) satisfying \( g(\lambda^*) = P \). Thus,

\[ \lambda^* = \lambda^+ - \left( g(\lambda^+) - P \right) \frac{\lambda^+ - \lambda^-}{g(\lambda^+) - g(\lambda^-)}, \] (6.32)

from where the solution to (6.25), denoted by \( p_j^* \), \( j = 1, 2, \ldots, n-1 \), can be obtained as

\[ p_j^* = g_j(\lambda^*). \] (6.33)

A Distributed Implementation to the Interpolation-Based Method

While (6.33) provides the unique global solution to the problem in (6.25) by solving its Lagrange dual, as formulated in (6.26), in order to determine \( g(\lambda^+) \)
and $g(\lambda^-)$, there is a need for a centralized decision maker with knowledge of all the individual $g_j(\lambda)$’s, as well as $P$. However, by rearranging (6.32), we will see that ratio consensus, together with a simple message-passing protocol, can be utilized to find $\lambda^*$ without requiring a centralized decision maker, or the need for each node $v_j, j \in \{1, 2, \ldots, n-1\}$, to obtain all the individual functions $g_j(\cdot), j \in \{1, 2, \ldots, n-1\}$; the values they take for $\lambda^+$ or $\lambda^-$; or the value of $P$.

First, rewrite (6.32) as follows:

$$\lambda^* = \lambda^+ - \left( \frac{g(\lambda^+)}{P} - 1 \right) \frac{\lambda^+ - \lambda^-}{\frac{g(\lambda^+)}{P} - \frac{g(\lambda^-)}{P}}; \quad (6.34)$$

where

$$g(\lambda^+) = \sum_{i=1}^{n-1} g_i(\lambda^+), \quad (6.35)$$

$$g(\lambda^-) = \sum_{i=1}^{n-1} g_i(\lambda^-). \quad (6.36)$$

As (6.35) and (6.36) imply, in order to find the optimal solution, each bus must know $\lambda^+$ and $\lambda^-$ a priori; that is, if each node knew which two $\lambda \in \mathcal{U}$ satisfied (6.30) and (6.31), ratio consensus could be used by the nodes to asymptotically obtain $g(\lambda^+)/P$ and $g(\lambda^-)/P$, and consequently $\lambda^*$. From (6.34), we see that the criteria for determining $\lambda^+$ and $\lambda^-$ are equivalent to finding $\lambda \in \mathcal{U}$ such that $g(\lambda)/P$ is closest to 1 from above and below, respectively; thus, if each bus obtains $g(\lambda)/P, \forall \lambda \in \mathcal{U}$, it can determine which ratios correspond to $g(\lambda^+)/P$ and $g(\lambda^-)/P$. Furthermore, if each bus also knows which $\lambda \in \mathcal{U}$ correspond to $\lambda^+$ and $\lambda^-$, it can compute $\lambda^*$, and therefore, by using (6.33), it can also compute the optimal solution to the primal problem. The procedure described above is amenable for a distributed implementation, and can be summarized by the following two steps.

**[S1.]** Assume that initially, each node $v_j, j \in \{1, 2, \ldots, n-1\}$, only knows its respective $\Delta_j$ and $\bar{\lambda}_j$. Then, by broadcasting these values, all nodes in the out-neighborhood of $v_j$ can obtain them, and in turn, broadcast them to their out-neighbors. Proceeding in this fashion, after a finite number of steps, bounded by the diameter of the graph representing the communication network, each node $v_j$ will obtain every $\lambda \in \mathcal{U}$. Note that there are no $\Delta_n$
and \( \bar{\lambda}_n \) associated with bus \( v_n \); however, we assume that the cyber node corresponding to bus \( v_n \) participates in the protocol.

[S2.] Once each node \( v_j, j \in \{1, 2, \ldots, n\} \), has acquired every \( \lambda \in \mathcal{U} \), it executes \( 2(n - 1) \) copies of the ratio-consensus numerator iteration in (6.14). For \( v_i, i = 1, \ldots, n - 1 \), let \( y_j^{(2i-1)}[k] \) and \( y_j^{(2i)}[k] \), denote the state variables at iteration \( k \) of the \((2i - 1)^{th}\) and \(2i^{th}\) such copies. Concurrently, each node \( v_j, j \in \{1, 2, \ldots, n\} \), executes a single copy of the ratio-consensus denominator iteration in (6.15); let \( z_j[k] \) denote the state variable maintained by node \( v_j \) at iteration \( k \) of this single copy. Then, if these state variables are initialized as

\[
y_j^{(2i-1)}[0] = \begin{cases} g_j(\lambda_{2i-1}), & \text{if } j = 1, 2, \ldots, n - 1, \\ 0, & \text{if } j = n, \end{cases}
\]

\[
y_j^{(2i)}[0] = \begin{cases} g_j(\lambda_{2i}), & \text{if } j = 1, 2, \ldots, n - 1, \\ 0, & \text{if } j = n, \end{cases}
\]

\[
z_j[0] = \begin{cases} 0, & \text{if } j = 1, 2, \ldots, n - 1, \\ P, & \text{if } j = n, \end{cases}
\]

(6.37)

it follows that

\[
\gamma_j^{(2i-1)} = \lim_{k \to \infty} \frac{y_j^{(2i-1)}[k]}{z_j[k]} = \frac{\sum_{l=1}^{n-1} g_l(\lambda_{2i-1})}{P} = \frac{g(\lambda_{2i-1})}{P} \tag{6.38}
\]

\[
\gamma_j^{(2i)} = \lim_{k \to \infty} \frac{y_j^{(2i)}[k]}{z_j[k]} = \frac{\sum_{l=1}^{n-1} g_l(\lambda_{2i})}{P} = \frac{g(\lambda_{2i})}{P}, \tag{6.39}
\]

for \( i = 1, \ldots, n - 1 \), and \( j = 1, \ldots, n \). Therefore, by determining which ratios are closest to 1 from above and below, and the corresponding \( \lambda^+ \) and \( \lambda^- \), respectively, each bus can compute \( \lambda^* \) using (6.34), and determine its optimal active power injection via (6.33).

**Remark 6.1.** Although the two-step procedure described above implies that each bus must obtain every \( \lambda \in \mathcal{U} \) (Step S1) before proceeding to execute the \( 2(n - 1) \) copies of the ratio-consensus algorithm (Step S2), it is possible, with a slight modification, to execute the message-passing protocol and ratio consensus in parallel and still guarantee convergence of (6.38) and (6.39); see Domínguez-García et al., 2012a; Cady et al., 2015b for a detailed description.
Algorithm 14: Distributed Optimal Allocation

Input: \( \alpha_j, \beta_j, p_j, \bar{p}_j, j = 1, 2, \ldots, n-1; P \)

Output: \( p_j^*, j \in \{1, 2, \ldots, n-1\} \)

Each node \( v_j, j \in \{1, \ldots, n\}, \) separately does the following:

\[
\begin{align*}
\text{begin} & \quad \text{initialize states} \\
& \quad \text{for } j = 1, 2, \ldots, n-1: \\
& \quad \quad \lambda_{2j-1} = \frac{p_j - \alpha_j}{\beta_j} \\
& \quad \quad \lambda_{2j} = \frac{p_j - \alpha_i}{\beta_j} \\
& \quad \quad y_j^{(2j-1)}[0] = g_j(\lambda_{2j-1}) \\
& \quad \quad y_j^{(2j)}[0] = g_j(\lambda_{2j}) \\
& \quad \quad y_{2j-1}[0] = 0 \\
& \quad \quad y_{2j}[0] = 0 \\
& \quad \quad z_j[0] = 0 \\
& \quad \quad \text{for } j = n: \\
& \quad \quad y_n^{(2i-1)}[0] = 0 \\
& \quad \quad y_n^{(2i)}[0] = 0, \forall i = 1, 2, \ldots, n-1 \\
& \quad \quad z_n[0] = P \\
& \quad \quad \text{foreach iteration, } k = 0, 1, \ldots, k_0, \text{ do} \\
& \quad \quad \quad \text{if } \lambda_{2j-1}, \lambda_{2j} \text{ not broadcasted then} \\
& \quad \quad \quad \quad \text{broadcast } \lambda_{2j-1}, \lambda_{2j} \\
& \quad \quad \quad \quad \text{if received } \lambda_{2i-1}, \lambda_{2i} \text{ then} \\
& \quad \quad \quad \quad \quad y_j^{(2i-1)}[k] = y_j^{(2i-1)}[k] + g_j(\lambda_{2i-1}), \\
& \quad \quad \quad \quad \quad y_j^{(2i)}[k] = y_j^{(2i)}[k] + g_j(\lambda_{2i}) \\
& \quad \quad \quad \quad \text{update and broadcast states:} \\
& \quad \quad \quad \quad \quad y_j^{(2i-1)}[k+1] = \sum_{v_l \in \mathcal{N}_j^- \cup \{v_j\}} \frac{1}{1 + d^{+}_j} y_{2j-1}^{(2j-1)}[k] \\
& \quad \quad \quad \quad \quad y_j^{(2i)}[k+1] = \sum_{v_l \in \mathcal{N}_j^- \cup \{v_j\}} \frac{1}{1 + d^{+}_j} y_{2j}^{(2j)}[k] \\
& \quad \quad \quad \quad \quad z_j[k+1] = \sum_{v_l \in \mathcal{N}_j^- \cup \{v_j\}} \frac{1}{1 + d^{+}_j} z_{2j}[k] \\
& \quad \quad \quad \quad \text{compute:} \\
& \quad \quad \quad \quad \quad \gamma_j^{(2i-1)}[k_0] = \frac{y_j^{(2i-1)}[k_0]}{z_j[k_0]}, \quad \gamma_j^{(2i)}[k_0] = \frac{y_j^{(2i)}[k_0]}{z_j[k_0]} \\
& \quad \quad \quad \quad \text{determine } g(\lambda^+)/\chi, g(\lambda^-)/\chi, \lambda^+, \lambda^- \\
& \quad \quad \quad \quad \text{compute } \lambda^* \text{ according to (6.34)} \\
& \quad \quad \quad \quad \text{return } p_j^* = g_j(\lambda^*) \\
\end{align*}
\]
Remark 6.2. In the pseudocode in Algorithm 14, we assume the ratio-consensus iterations are executed a finite number of times, \( k_0 \), a priori known to all the nodes, that is large enough to ensure the computed ratios at \( k = k_0 \) are sufficiently close to the asymptotic value. [The algorithm could also be modified to include the distributed stopping scheme described in Section 4.3 so as to determine when to stop in a distributed fashion.]

6.5 Flow-Feasible Allocation

In this section, we drop the assumption we made in earlier sections that for every \( p \in \mathcal{P} = \{ p \in \mathbb{R}^n : \underline{p} \leq p \leq \bar{p}, \mathbf{1}_n^T p = 0 \} \), the vector \( f \) that results from (6.1) is such that the constraint in (6.3) is satisfied; thus, we need to consider the full model in (6.1)–(6.3). As in Section 6.3, consider the problem of choosing the values of the first \( n - 1 \) injections, \( p_1, p_2, \ldots, p_{n-1} \), so that the value of the \( n^{th} \) injection is equal to some prescribed value, \(-P\), satisfying \( p_n \leq -P \leq \bar{p}_n \). This problem can be cast as a feasibility problem of the form:

\[
\text{find } \quad p, \ f \\
\text{subject to } \quad p = M f \\
\quad \underline{f} \leq f \leq \bar{f} \\
\quad p_j \leq p_j \leq \bar{p}_j, \quad j = 1, 2, \ldots, n - 1, \\
\quad p_n = -P. 
\] (6.40)

Existence of a Feasible Solution

In order to check whether or not the problem in (6.40) has a solution, we can utilize a similar approach to that outlined in Section 5.5 after defining an auxiliary graph \( \tilde{\mathcal{G}}_p = (\tilde{\mathcal{V}}_p, \tilde{\mathcal{E}}_p) \). In this graph, since \( \mathbf{1}_n^T p = 0 \), we can treat the entries of the active power injection vector, \( p \), as flows in and out of a “ground” node \( v_0 \), with the other nodes being identical to those in \( \mathcal{V}_p \); thus, \( \tilde{\mathcal{V}}_p = \{ v_0 \} \cup \mathcal{V}_p = \{ v_0, v_1, v_2, \ldots, v_n \} \). Then, for each \( v_j, \ j = 1, 2, \ldots, n - 1 \), we add one or two edges in \( \tilde{\mathcal{E}}_p \) as follows:

[GEN] if \( 0 \leq p_j \leq \bar{p}_j \), then we add \((v_j, v_0) \in \tilde{\mathcal{E}}_p \), with \( \tilde{f}_{j0} = p_j \), and \( \bar{f}_{j0} = \bar{p}_j \).
if $p_j \leq \bar{p}_j \leq 0$, then we add $(v_0, v_j) \in \tilde{E}_p$, with $\tilde{f}_{0j} = -\bar{p}_j$, and $\tilde{f}_{0j} = -p_j$.

if $p_j \leq 0 \leq \bar{p}_j$, then we add $(v_j, v_0) \in \tilde{E}_p$, with $\tilde{f}_{j0} = 0$, and $\tilde{f}_{j0} = \bar{p}_j$; and $(v_0, v_j) \in \tilde{E}_p$, with $\tilde{f}_{0j} = 0$, and $\tilde{f}_{0j} = -p_j$.

In addition, for $v_n$, we add one edge in $\tilde{E}_p$ as follows:

if $P > 0$, then we add $(v_n, v_0) \in \tilde{E}_p$, with $\tilde{f}_{n0} = P$, and $\tilde{f}_{n0} = P$.

if $P < 0$, then we add $(v_0, v_n) \in \tilde{E}_p$, with $\tilde{f}_{0n} = -P$, and $\tilde{f}_{0n} = -P$.

if $P = 0$, we do not add any edge to $\tilde{E}_p$.

Finally, for each $(v_j, v_i) \in E_p$, we add two edges in $\tilde{E}_p$ as follows:

$(v_j, v_i) \in \tilde{E}_p$ with the flow on this edge constrained to lie in the interval $[0, \tilde{f}_{ji}]$, with $\tilde{f}_{ji} = \tilde{f}_{ji}$.

$(v_i, v_j) \in \tilde{E}_p$ with the flow on this edge constrained to lie in the interval $[0, \tilde{f}_{ji}]$, with $\tilde{f}_{ji} = -\tilde{f}_{ji} = \tilde{f}_{ji}$.

Now, if the circulation conditions stated in Definition 6, Chapter 5, for the graph $\tilde{G}_p$, with the limits on the edge flows as assigned above, are satisfied, there exists a set of flows, $\{ \tilde{f}_{ji} \mid (v_j, v_i) \in \tilde{E}_p \}$, that balance $\tilde{G}_p$, i.e., the $\tilde{f}_{ji}$’s satisfy:

1. $\tilde{f}_{ji} \leq \tilde{f}_{ji} \leq \tilde{f}_{ji}$ for each edge $(v_j, v_i) \in \tilde{E}_p$,

2. $\sum_{v_i \in \tilde{N}_j^-} \tilde{f}_{ji} - \sum_{v_j \in \tilde{N}_j^+} \tilde{f}_{ij} = 0$ for every $v_j \in \tilde{V}_p$.

where $\tilde{N}_j^-$ and $\tilde{N}_j^+$ are respectively the in- and out-neighbors of node $v_j$ in the graph $\tilde{G}$.

Then, given $\{ \tilde{f}_{ji} \mid (v_j, v_i) \in \tilde{E}_p \}$, by using similar techniques as in Section 5.5, we can find a set of injections, $\{ p_j \mid j = 1, 2, \ldots, n \}$, and a set of flows, $\{ f_{ji} \mid (v_j, v_i) \in E_p \}$, that solve the feasibility problem in (6.40) by applying the following rules:
6.5. Flow-Feasible Allocation

**R1** For GEN-type nodes, we choose \( p_j = \tilde{f}_{j0} \).

**R2** For DEM-type nodes, we choose \( p_j = \tilde{f}_0j \).

**R3** For STO-type nodes, we choose \( f_{ji} = \tilde{f}_{j0} - \tilde{f}_0j \).

**R4** For \( v_n \), we choose \( p_n = -P \).

**R5** For \( (v_j, v_i) \in \mathcal{E}_p \), we choose \( f_{ji} = \tilde{f}_{ji} - \tilde{f}_{ij} \).

**Feasible Flow Distributed Algorithm**

The method discussed above to check the existence of a feasible solution can be leveraged to design an iterative algorithm for obtaining the set of injections, \( \{p_j \mid j = 1, 2, \ldots, n\} \), and a set of flows, \( \{f_{ji} \mid (v_j, v_i) \in \mathcal{E}_p\} \), that solve the feasibility problem in (6.40). Specifically, once the augmented graph \( \tilde{G}_p = (\tilde{V}_p, \tilde{E}_p) \) is defined, one can utilize the algorithm described in Section 5.4 to obtain a set of flows, \( \{\tilde{f}_{ji} \mid (v_j, v_i) \in \tilde{E}_p\} \), that balance \( \tilde{G}_p \), and then recover a solution to (6.40) by applying Rules R1-R5. However, this approach is not completely distributed as the ground node, which is not present in the original graph \( G_p \), would have to be emulated by one of the processors located at a particular bus of the electrical network. This would require such processor to communicate with all other processors as in the graph \( \tilde{G}_p \) there is an edge between the ground node and every other node. Next, we provide an alternative iterative algorithm that circumvents this issue.

The algorithm assumes there exists a cyber network that allows nodes that are physical neighbors to communicate in a bidirectional manner, i.e., \( G_c = (V_c, E_c) \) with \( V_c \equiv V_p \), and \( E_c = \cup_{(v_j, v_i) \in E_p} \{(v_j, v_i), (v_i, v_j)\} \). This means that a pair of nodes \( v_j \) and \( v_i \) that are electrically connected by a distribution line can exchange information among themselves (in both directions). Each node \( v_j \in V_c \) maintains and iteratively updates the following variables: \( \{f_{lj}^{(j)} \mid (v_l, v_j) \in E_c\} \), and \( p_{lj}^{(j)} \). The values that these variables take at iteration \( k \geq 0 \), denoted by \( f_{lj}^{(j)}[k] \), \( (v_l, v_j) \in E_c \), and \( p_{lj}^{(j)}[k] \), respectively, attempt to capture what the actual values of the corresponding line flows and power injection in (6.40) should be, as estimated by node \( v_j \). Thus, at each iteration \( k \), each node \( v_j \) can determine how close the variables it maintains are to
satisfying the corresponding equality constraint in (6.40) by computing

\[ b_j^{(j)}[k] = - \sum_{(v_l,v_j) \in E_c} f_{ij}^{(j)}[k] + p_j^{(j)}[k]. \]  
(6.41)

Initially, i.e., for \( k = 0 \), each node \( v_j \) sets \( p_j^{(j)}[0] \) and the \( f_{ij}^{(j)}[0] \)'s as follows:

\[ f_{ij}^{(j)}[0] = 0, \quad (v_l,v_j) \in E_c, \]
\[ p_j[0] = \frac{1}{2}(\bar{p}_j + \bar{p}_j). \]  
(6.42)

Then, at each iteration \( k = 1, 2, \ldots \), each node \( v_j \) follows a three-step process to update the values of \( p_j^{(j)}[k] \) and the \( f_{ij}^{(j)}[k] \)'s; this update process will drive \( b_j^{(j)}[k] \) to zero as \( k \to \infty \) if the problem in (6.40) is feasible. Next, we summarize each of the three steps involved in the update process; the pseudocode for the complete procedure is provided in Algorithm 15.

[S1.] Node \( v_j \) updates each \( f_{ij}^{(j)}[k] \), \( (v_l,v_j) \in E_c \), to an interim value \( \tilde{f}_{ij}^{(j)}[k+1] \), \( (v_l,v_j) \in E_c \), as follows:

\[ \tilde{f}_{ij}^{(j)}[k+1] = f_{ij}^{(j)}[k] + \frac{b_j^{(j)}[k]}{D_j}, \quad (v_l,v_j) \in E_c, \]  
(6.43)

with \( D_j := d_j^+ + d_j^- \), where \( d_j^+ \) is the number of out-neighbors of \( v_j \) in \( G_p \), and \( d_j^- \) is the number of in-neighbors of \( v_j \) in \( G_p \). Similarly, node \( v_j \) updates \( p_j^{(j)}[k] \), to an interim value, \( \hat{p}_j^{(j)}[k+1] \), as follows:

\[ \hat{p}_j^{(j)}[k+1] = p_j^{(j)}[k] - \frac{1}{2} \frac{b_j^{(j)}[k]}{D_j}. \]  
(6.44)

[S2.] After all nodes complete Step S1, it is possible that the estimates maintained by pairs of nodes that are electrically connected via a distribution line do not agree; thus, every pair of nodes \( v_j \) and \( v_i \) that are electrically connected will exchange the interim flow estimate values for the line that connects them, respectively \( \tilde{f}_{ij}^{(j)}[k+1] \) and \( \tilde{f}_{ji}^{(i)}[k+1] \), and both will compute another interim variable as follows:

\[ \tilde{f}_{ij}^{(j)}[k+1] = \frac{1}{2} \left( \tilde{f}_{ij}^{(j)}[k+1] - \tilde{f}_{ji}^{(i)}[k+1] \right), \]
\[ \tilde{f}_{ji}^{(i)}[k+1] = \frac{1}{2} \left( \tilde{f}_{ji}^{(i)}[k+1] - \tilde{f}_{ij}^{(j)}[k+1] \right). \]  
(6.45)
After all nodes complete Step S2, it is possible that some of the $\hat{f}_{ij}^j[k+1]$’s are outside the admissible range of flows for the line connecting

Algorithm 15: Distributed Feasible Flow Algorithm

Input: $f_{em}, \overline{f}_{em}, e_m = (v_j, v_i) \in E_p; p_j, \overline{p}_j, j = 1, 2, \ldots, n - 1; \overline{p}_n = p_n = -P$

Each node $j \in V_c$ separately does the following:

begin
initialize

\[
\begin{align*}
    f_{ij}^{(j)}[0] &= 0 \\
    p_{ij}^{(j)}[0] &= \frac{1}{2}(p_j + \overline{p}_j)
\end{align*}
\]

foreach iteration, $k = 0, 1, 2 \ldots$, do

compute

\[
\begin{align*}
    b_j[k] &= - \sum_{(v_i, v_j) \in E_c} f_{ij}^{(j)}[k] + p_{ij}^{(j)}[k]
\end{align*}
\]

transmit

\[
\begin{align*}
    b_{ij}^{(j)}[k] / D_j \text{ to all } v_i \in N_j^+ \text{ and to } v_i \in N_j^-
\end{align*}
\]

receive

\[
\begin{align*}
    b_{ij}^{(i)}[k] / D_i \text{ from all } i \in N_j^- \text{ and } b_{ij}^{(l)}[k] / D_l \text{ from all } l \in N_j^+
\end{align*}
\]

compute

\[
\begin{align*}
    \hat{f}_{ij}^{(j)}[k+1] &= f_{ij}^{(j)}[k] + \frac{1}{2} \left( \frac{b_{ij}^{(j)}[k]}{D_j} - \frac{b_{ij}^{(i)}[k]}{D_i} \right) \text{ for all } (v_i, v_j) \in E_c \\
    \hat{p}_{ij}^{(j)}[k+1] &= p_{ij}^{(j)}[k] - \frac{1}{2} \frac{b_{ij}^{(j)}[k]}{D_j}
\end{align*}
\]

set

\[
\begin{align*}
    \text{for all } (v_i, v_j) \in E_c, \\
    f_{ij}^{(j)}[k+1] &= \begin{cases} 
    \overline{f}_{ij}, & \text{if } \hat{f}_{ij}^{(j)}[k+1] > \overline{f}_{ij} \\
    \hat{f}_{ij}, & \text{if } \hat{f}_{ij}^{(j)}[k+1] < \hat{f}_{ij} \\
    \hat{f}_{ij}^{(j)}[k+1], & \text{otherwise}
\end{cases} \\
    p_{ij}^{(j)}[k+1] &= \begin{cases} 
    \overline{p}_j, & \text{if } \hat{p}_{ij}^{(j)}[k+1] > \overline{p}_j \\
    p_{ij}, & \text{if } \hat{p}_{ij}^{(j)}[k+1] < p_{ij} \\
    \hat{p}_{ij}^{(j)}[k+1], & \text{otherwise}
\end{cases}
\end{align*}
\]
nodes \(v_i\) and \(v_j\); thus, in order to obtain the new flow estimates, \(f_{ij}^{(j)}[k + 1]\), \((v_i, v_j) \in \mathcal{E}_c\), each node \(v_j\) performs the following computation:

\[
f_{ij}^{(j)}[k + 1] = \begin{cases} 
\bar{f}_{ij}, & \text{if } \hat{f}_{ij}^{(i)}[k + 1] > \bar{f}_{ij}, \\
\underline{f}_{ij}, & \text{if } \hat{f}_{ij}^{(i)}[k + 1] < \underline{f}_{ij}, \\
\hat{f}_{ij}^{(i)}[k + 1], & \text{otherwise}.
\end{cases}
\]  

(6.46)

Similarly, each node \(v_j\) needs to take into account the limits on its admissible power injection; thus, in order to obtain the new injection estimate, \(p_j[k + 1]\), it will perform the following computation:

\[
p_j^{(j)}[k + 1] = \begin{cases} 
\bar{p}_j, & \text{if } \hat{p}_j^{(j)}[k + 1] > \bar{p}_j, \\
p_j, & \text{if } \hat{p}_j^{(j)}[k + 1] < \bar{p}_j, \\
\hat{p}_j^{(j)}[k + 1], & \text{otherwise}.
\end{cases}
\]  

(6.47)

For each \((v_i, v_j) \in \mathcal{E}_p\), and each \(k = 0, 1, 2, \ldots\), we can define a variable \(f_{ij}[k]\) as follows:

\[
f_{ij}[k] := f_{ij}^{(j)}[k].
\]  

(6.48)

On the other hand, for every pair \((v_i, v_j), (v_j, v_i) \in \mathcal{E}_c\), it should be clear from (6.45) and (6.46) that \(f_{ij}^{(j)}[k] = -f_{ji}^{(i)}[k], \forall k\). Therefore, by letting \(p_j[k] := p_j^{(j)}[k], \forall k, j = 1, 2, \ldots, n\), the progress of the algorithm as determined by Steps S1–S3, can be described in terms of the \(f_{ij}[k]\)’s as follows:

\[
f_{ij}[k + 1] = \left[ f_{ij}[k] + \frac{1}{2} \left( \frac{b_j[k]}{D_j} - \frac{b_i[k]}{D_i} \right) \right] \bar{f}_{ij}, \quad \forall (v_i, v_j) \in \mathcal{E}_p,
\]  

(6.49)

\[
p_j[k + 1] = \left[ p_j[k] - \frac{1}{2} \frac{b_j[k]}{D_j} \right] \bar{p}_j, \quad \forall j = 1, 2, \ldots, n,
\]  

(6.50)

where

\[
b_j[k] := \sum_{(v_j, v_i) \in \mathcal{E}_p} f_{ji}[k] - \sum_{(v_i, v_j) \in \mathcal{E}_p} f_{ij}[k] + p_j[k]
\]  

\[
= - \sum_{(v_i, v_j) \in \mathcal{E}_c} f_{ij}^{(j)}[k] + p_j^{(j)}[k] = b_j^{(j)}[k].
\]  

(6.51)
As shown in Cady et al., 2017, if (6.40) is feasible, then iterations (6.49) – (6.50) will converge to a pair \((p, f)\) satisfying the constraints in (6.40). This, in turn, means that Algorithm 15 allows the nodes to obtain, in a distributed manner, a set of injections that will result in flows that are feasible. In fact, as also shown in Cady et al., 2017, the iterations (6.49) – (6.50) correspond to those of a gradient descent algorithm for a quadratic optimization program. In this program, the flows are constrained to lie within the corresponding edge upper- and lower-flow limits, and the cost function is the two-norm of the vector of node balances. Thus, finding a feasible flow assignment is equivalent to finding a zero-cost solution to this quadratic program. Therefore, if the algorithm converges to a solution in which the balance is non zero, it means that the quadratic program has a non-zero cost. This, in turn, means that there is no solution to the flow assignment problem.

The fact that iterations (6.49) – (6.50) converge to a feasible solution can also be established utilizing the techniques in Section 5.4, realizing that with the exception of the ground node, all nodes in the graph \(\tilde{G}\) execute Algorithm 13, thus driving their balances to zero. The key is that the sum of the balances is always zero (Proposition 2 in Chapter 5); thus, if all nodes but the ground nodes have zero balance, then the ground node necessarily has zero balance.

6.6Discussion

Using a centralized decision maker to solve the problem of optimally allocating a resource subject to some constraints is not new to electric power systems. Specifically, when the resource to be provided is active power, and the constraints reflect limits on nodal injection limits, voltage magnitudes, and line flows, and possible angle differences across transmission lines, the problem is commonly referred to as the optimal power flow problem (see, e.g., Wood and Wollenberg, 1996). The optimal power flow problem is in general non-convex, and in general does not have a unique globally optimal solution.

The different problem settings in this chapter are particular instances of the optimal power flow problem that result from specific simplifications of the power flow model. For example, the problem formulation in Section 6.4 is identical to that of the so-called classical economic dispatch problem (see, e.g., Bergen and Vittal, 2000). In this problem, there is a set of generator, each
of which can provide a certain amount of power (constrained to lie in some interval) at some cost. Then, the objective is for the generators to collectively provide a certain total amount of power, while minimizing the total cost of meeting this demand.

The problem in Section 6.4 is very related to problems addressed in distributed optimization. Next, we discuss some recent works in this area that are specifically related to the method discussed in Section 6.4 (the reader is also referred to Bertsekas and Tsitsiklis, 1989 for an account of earlier works). The paper by Xiao et al., 2006 propose a linear-iterative algorithm for optimal resource allocation over a network, where the the nodes need to collectively provide a fixed amount of a resource, while minimizing their individual costs; unlikely the assumptions we make in Section 6.4, Xiao et al., 2006 assume that: (i) the individual component contributions are not upper- and lower-bounded; and (ii) communication links are symmetric. Nedic et al., 2010 proposed an approach similar to the one by Xiao et al., 2006, except for: (i) the equality constraint is substituted by a set of convex inequalities corresponding to individual node constraints, and (ii) the communication graph is time-varying. Compared to the setting in Section 6.4, Nedic et al., 2010 impose no equality constraint on the sum of the individual contributions. Finally, the cost function considered by Zanella et al., 2011 is similar to the one considered by Xiao et al., 2006; Nedic et al., 2010, but without equality or inequality constraints, while the communication graph is undirected and time-invariant.

The problem in Section 6.5 can be also thought of as a particular case of the standard network flow problem (see, e.g., Ford and Fulkerson, 2010), where there is a cost associated to the flow on each link, and the objective is to minimize the total cost subject to the same constraints in the problem in Section 6.5. In this context, it is common to assume that the individual costs are described by convex functions on the flow, which makes the optimization problem convex. Then, its solution can be obtained via the Lagrange dual, the formulation of which is well suited for algorithms that can be executed, in a distributed fashion, over a cyber network that conforms to the same topology as that of the physical layer (see, e.g., Bertsekas et al., 1987). By contrast, the distributed algorithm proposed in Section 6.5 does not exploit duality notions, and instead acts directly on the primal variables, i.e., the flows.
The PageRank Problem

In this chapter, we illustrate the application of distributed averaging algorithms to the problem of calculating the so-called PageRank values in a given digraph. The problem has many applications, including the ranking of webpages in the World Wide Web.

7.1 Introduction and Motivation

Google’s positioning among search engines is partly due to the quality of its search results, which relies, to a great extent, on the PageRank algorithm Brin and Page, 1998, which Google uses for ranking the web pages in the World Wide Web. The importance of a web page is mainly characterized by its position in the Web graph, as captured by the web pages (nodes) and the inbound links, i.e., the hyperlinks that point to them. PageRank outputs the PageRank vector, which is equivalent to Bonacich centrality, a concept of centrality for social networks that was proposed in Bonacich, 1987. Its effectiveness in determining node importance in large graphs was understood when PageRank started as an algorithm for ranking web pages, and has subsequently extended its use to a wide variety of fields, including chemistry, bioinformatics, sports, literature, bibliometrics, and citations; the diversity of
applications of PageRank is comprehensively surveyed in Gleich, 2015. In practice, Google estimates the PageRank vector centrally once a month using the MapReduce paradigm; see Dean and Ghemawat, 2004. However, the ever increasing size of the web requires enormous computational power beyond that offered by the MapReduce paradigm; see Langville and Meyer, 2006.

While numerous efforts have been done to improve the computation via numerical methods (see, e.g., Nazin and Polyak, 2011; Lofgren, 2014; Csáji et al., 2014 and the references therein) and polyhedral approaches (see, e.g., Fercoq et al., 2013), recent efforts are turned towards the development of distributed approaches for solving the PageRank problem (see, e.g., Sankaralingam et al., 2003; Zhu et al., 2005; Ishii and Tempo, 2010; Ishii et al., 2012; Ishii and Tempo, 2014; Ravazzi et al., 2015; Clifton and Pasik-Duncan, 2015; You et al., 2015 and references therein), where each node (web page, or the server that hosts it or a server representing the web page on the Google cluster) is able to compute its own PageRank value by communicating, over several iterations, with neighboring pages connected to it via (directed) hyperlinks. In case the distributed algorithm is implemented on the Google cluster, the web graph should be duplicated by the servers or from virtual servers via linux containers or virtualization; see Qian et al., 2009.

It was shown that the PageRank problem is equivalent to estimating the right-eigenvector (corresponding to eigenvalue 1) of the weighted adjacency column-stochastic matrix of the Web graph. There have been works in the literature for obtaining the right-eigenvector (left-eigenvector) of a weighted adjacency column-stochastic (row-stochastic) matrix in a distributed fashion under the assumption that all nodes in the graph execute their updates in a synchronized manner (see, e.g., Qu et al., 2014; Priolo et al., 2014). However, global synchronization is not practical in the Web graph due to its enormous size. To overcome this problem, distributed randomized algorithms based on gossip-like algorithms have been recently proposed (see, e.g., Ishii and Tempo, 2010; Ishii et al., 2012; Ishii and Tempo, 2014; Ravazzi et al., 2015; Clifton and Pasik-Duncan, 2015).

All aforementioned approaches assume that the web pages reside on servers that are homogeneous in terms of the rates at which they execute their

\[1\] MapReduce is a programming paradigm for processing and generating large data sets with a parallel, distributed algorithm on a cluster.
updates. In practice, this assumption is not satisfied, because Internet servers are heterogeneous in terms of their clock rates and computational capabilities, due to the large number of different types employed over the years. Useful algorithms for distributed computation of PageRank must therefore allow for asynchronous operation, and must support server updating at different rates.

7.2 The PageRank Problem

There are many variations on the PageRank problem, yet there is a core definition that applies to almost all of them. Suppose that we are given a web graph $G = (\mathcal{V}, \mathcal{E})$. In terms of the system modeling of Chapter 2, this graph will serve as both the physical topology and the cyber topology. Graph parameters of interest, such as $d_j^+, N_j^+$, etc., are defined with respect to the given graph.

Let $A \in \mathbb{R}^{n \times n}_+$ denote a column-stochastic matrix where

$$A(l, j) = \begin{cases} 1/d_j^+, & v_l \in N_j^+, \\ 0, & \text{otherwise}. \end{cases}$$

The PageRank value of web page $v_j \in \mathcal{V}$ is given by $x_j^* \in [0, 1]$, where $x^*$ is the vector satisfying $1_n^T x^* = 1$ and $Ax^* = x^*$. Since $A$ is a nonnegative column-stochastic matrix, $x^*$ is the right-eigenvector corresponding to the Perron-Frobenius eigenvalue of matrix $A$, denoted by $\rho(A) = 1$. For existence and uniqueness of the right-eigenvector, it is required that the Web graph is strongly connected and aperiodic; see e.g., Horn and Johnson, 1985 (see also Theorem 2.1 in Chapter 2). One can view the vector $x^*$ as the steady-state distribution of a homogeneous Markov chain with transition matrix $A$ Brémaud, 2013; repeated iterations of the form $x[k + 1] = Ax[k]$ where $x[0]$ is some initial probability distribution (where $x_j[0]$ is nonnegative such that $\sum_\ell x_\ell[0] = 1$) lead to the steady-state vector, i.e., $\lim_{k \to \infty} x[k] = x^*$, when $A$ is a primitive matrix.

Strong connectivity can be guaranteed by the “back button” and/or teleportation models. In the “back button” model there is a small probability $b$ of the user hitting the back button in their browser, so we instead seek the stationary distribution of the Markov chain with transition matrix

$$P_{\text{back}} = (1 - b)A + bA^T.$$
The PageRank Problem

In the teleportation model there is a small probability \( t \) that the user will randomly jump to another website, according to a uniform distribution for simplicity, rather than following a hyperlink on their current website, and so we seek the stationary distribution of the Markov chain with transition matrix

\[
P_{\text{tele}} = (1 - t) A + (t/n) J,
\]

where \( J = 1_n 1_n^T \) is the matrix with all entries equal to one. In the “back button” model, if \( A \) was weakly connected then the matrix of interest will be strongly connected due to the symmetrization. In the teleportation model, we clearly have a strongly connected network since \( J \) is the adjacency matrix of the complete graph. For a detailed discussion, see Ishii and Tempo, 2014.

7.3 Centralized Computation of PageRank Values

Let \( P \in \mathbb{R}^{n \times n}_+ \) be a column-stochastic and primitive matrix representing the Web graph, including any additional links due to some model (“back button” and/or teleportation). Consider the following iteration

\[
x[k + 1] = Px[k],
\]

\[
x[0] = x_0,
\]

(7.1)
in which the initial condition \( x_0 \in [0, 1]^n \) satisfies \( 1_n^T x_0 = 1 \). The iteration in (7.1) converges to the steady-state distribution, i.e., since \( P \) is primitive column-stochastic

\[
\lim_{k \to \infty} x[k] = \lim_{k \to \infty} P^k x[0] = x^*,
\]

(7.2)

where \( x^* \) is the Perron-Frobenius eigenvector (normalized so that \( 1_n^T x^* = 1 \)). For very large graphs (e.g., the World Wide Web has about \( 10^{10} \) web pages), eigenvector decomposition of \( P \) is not feasible in practice, and a power method through (7.1) is used to compute \( x^* \). Even though the power method only requires a few iterations (about 50-100) to converge, it would take about a week to finish on the complete web graph; see Langville and Meyer, 2006; Ishii and Tempo, 2014.

In this chapter, we will describe a distributed algorithm of the type introduced in Chapter 2. Node \( v_j \) is in charge of calculating \( x_j[k] \) in (7.1) and can update its information state to \( x_j[k + 1] \) as a weighted linear combination of
its own value $x_j[k]$ and the available information received by its in-neighbors $\{x_i[k] \mid v_i \in \mathcal{N}_j^-[k]\}$ (here $\mathcal{N}_j^-[k]$ denotes the set of nodes that send information to node $v_j$ at iteration $k$). Since we deal with directed graphs, we assume that each node $v_j$ chooses its self-weight $p_{jj}[k]$ and the weights $p_{lj}[k]$ on its out-going links at time $k$, i.e., $\{p_{lj}[k] \mid v_l \in \mathcal{N}_j^+[k]\}$. Hence, each node updates its information state $x_j[k+1]$, $k = 0, 1, 2, \ldots$, according to (3.1) in Chapter 3 for an appropriate set of time-varying weights. As discussed in Chapter 3, it make sense for node $v_j$ to transmit to its out-neighbor $v_l \in \mathcal{N}_j^+$ the value $p_{lj}[k]x_j[k]$.

### 7.4 Asynchronous Computation of PageRank Values

Given a web digraph $G = (\mathcal{V}, \mathcal{E})$ of order $n$ (with $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$), we describe in this section a distributed algorithm that allows nodes to calculate their eigencentrality (which is the case in the PageRank value) while updating their state asynchronously, possibly with different update rates. Each node $v_j$ clearly knows its out-degree $d_j^+$, i.e., Assumption [A6] in Chapter 2 holds (because each node knows the nodes it points to via hyperlinks).

Consider a random process $\{\theta[k], k \in \mathbb{N}\}$, where $\theta[k] \in \mathcal{V}$ encodes the $k^{th}$ update event during the execution of the algorithm. In other words, $\theta[k] = v_j$ means that node $v_j$ broadcasts its value across its outgoing links at time $k$ (much like broadcast gossip in Chapter 3). At times when node $v_j$ updates, the update matrix is given by

$$P_j = \overline{P}_j D_j + (I_n - D_j), \quad (7.3)$$

where $\overline{P}_j = [\overline{p}_{ij}]$ is a matrix which differs from the identity matrix $I_n$ only at column $j$, the entries of which are $\overline{p}_{lj} = \frac{1}{d_j^+}$, $\forall (l, j) \in \mathcal{E}$, and zero otherwise (note that $\overline{p}_{jj} = 0$ because throughout this monograph we have adopted the convention that self-loops are excluded the definition of $G$), while $D_j$ is a diagonal matrix with

$$D_j(i, i) = \begin{cases} \alpha_j, & \text{if } i = j, \\ 1, & \text{otherwise,} \end{cases}$$

where $\alpha_j \in (0, 1)$; herein, we use $\alpha_j = d_j^+/(d_j^+ + 1)$.

If $\theta[k]$, $k = 0, 1, 2, 3, \ldots$, are assumed to be independent and identically distributed (i.i.d.) random variables with uniform probability mass (i.e., the
number of updates by each node in the long run is roughly the same), then
\[ P\{\theta[k] = v_j \} = \frac{1}{n}, \quad k = 0, 1, 2, \ldots. \]
The corresponding updates are given by (7.1) where \( P[k] = P_j \) if \( \theta[k] = v_j \). The average matrix \( \hat{P} \) is then given by
\[
\hat{P} = \frac{1}{n} \sum_{v_j \in \mathcal{V}} P_j = \frac{1}{n} \sum_{v_j \in \mathcal{V}} (P_j D_j + (I_n - D_j))
\]
\[
= \frac{1}{n} \sum_{v_j \in \mathcal{V}} P_j D_j + \frac{1}{n} (I_n - D), \quad (7.4)
\]
where \( D = \text{diag}(\alpha_1, \alpha_2, \ldots, \alpha_n) \).

Let \( x[0] \) be a stochastic vector and consider the iterates \( x[k] \) produced by (7.1) when \( P[k] \) are generated as just described. Then, due to ergodicity, the running average \( y[k+1] = \frac{1}{k+1} (ky[k] + x[k]) \), with \( y[0] = 0_n \), of the iterates converges to the stationary distribution \( x^* \) of the average matrix \( \hat{P} \) in the mean-square sense, i.e.,
\[
\lim_{k \to \infty} \mathbb{E}[\|y[k] - x^*\|^2] = 0.
\]
Here, the expectation \( \mathbb{E}[\cdot] \) is taken with respect to random process \( \theta[k] \). As a result, node \( v_j \) can estimate \( x_j^* \) by keeping track of the number of times \( k_j \) it updated its value and the running sum of these values, i.e.,
\[
x_j^* = \lim_{k_j \to \infty} \frac{1}{k_j + 1} \sum_{\ell=0}^{k_j} x_j[\ell] = \lim_{k_j \to \infty} y_j[k_j].
\]
When nodes update their values asynchronously and with different update rates, \( \theta[k] \) can be seen as an i.i.d. random variable with non-uniform probability mass function. Let
\[
P\{\theta[k] = v_j \} = q_j, \quad k = 0, 1, 2, \ldots, \quad (7.5)
\]
where \( q_j \) can be different for each \( j \) while \( \sum_{v_j \in \mathcal{V}} q_j = 1 \). We can view \( q_j \) as the normalized clock rate of node \( v_j \), i.e., \( q_j = r_j / \sum_{v_j \in \mathcal{V}} r_j \), where \( r_j \) is the clock rate of node \( v_j \).
Suppose that each node \( v_j \) updates its value based on the following update rule:

\[
\tilde{P}_j = \tilde{P}_j D_j \frac{c}{q_j n} + I_n - D_j \frac{c}{q_j n},
\]

where \( c \) is a positive constant. We will initially assume that \( c \) and \( n \) are known and show (both analytically and via simulations) convergence to the correct right-eigenvector. Then, we propose a way to compute everything in a distributed fashion and, finally, we present the fully distributed algorithm that is shown to converge to the correct right-eigenvector.

To ensure that \( \tilde{P}_j \) in (7.6) is nonnegative, it is required that

\[
I_n - D_j \frac{c}{q_j n} \geq 0.
\]

Notice that if \( c \leq q_j n \) for all \( j \), the above inequality is satisfied. When \( q_j = 1/n \) letting \( c = 1 \) renders the update rule (7.6) equivalent to (7.3). Later, we will discuss how \( \frac{c}{q_j n} \) can be computed in a distributed fashion. With the update rule defined by (7.6), as \( k \to \infty \), the average matrix \( \tilde{P} \) is given by

\[
\tilde{P} = \sum_{v_j \in V} q_j \tilde{P}_j = \sum_{v_j \in V} q_j \left( \tilde{P}_j D_j \frac{c}{q_j n} + I_n - D_j \frac{c}{q_j n} \right)
= \sum_{v_j \in V} \tilde{P}_j D_j \frac{c}{n} + I_n - \sum_{v_j \in V} D_j \frac{c}{n}
= \frac{c}{n} \sum_{v_j \in V} \tilde{P}_j D_j + \left( 1 - \frac{c n - 1}{n} \right) I_n - \frac{c}{n} D.
\]

Note that matrices \( \tilde{P} \) (as defined in (7.4)) and \( \tilde{P} \) (as defined in (7.7)) are nonnegative and column stochastic. By the Perron-Frobenius theorem Horn and Johnson, 1985 (see also Chapter 2), a primitive nonnegative matrix has an eigenvalue (called the Perron-Frobenius eigenvalue) which is simple and its magnitude is the largest among all the eigenvalues; its corresponding right eigenvector is nonnegative. In the following proposition, we show that \( \tilde{P} \) and \( \tilde{P} \) share the same right-eigenvector corresponding to the Perron-Frobenius eigenvalue.

**Proposition 7.1.** If \( c \leq q_j n \), then the right-eigenvector corresponding to the Perron-Frobenius eigenvalue for both matrices \( \tilde{P} \) and \( \tilde{P} \) is the same.
Proof. Let \( P_{\text{aux}} := \frac{1}{n} \sum_{i \in V} \tilde{P}_i D_i \). Suppose \( v \) is a right-eigenvector of matrix \( \tilde{P} \) in equation (7.4). Then,

\[
\left( P_{\text{aux}} + \frac{1}{n} (I_n - D) \right) v = v, \tag{7.8}
\]

which after algebraic manipulation, becomes

\[
P_{\text{aux}} v = \left( \frac{n - 1}{n} I_n + \frac{1}{n} D \right) v. \tag{7.9}
\]

For a vector \( w \), we have that

\[
\tilde{P} w = \left[ c P_{\text{aux}} + \left( 1 - c \frac{n - 1}{n} \right) I_n - \frac{c}{n} D \right] w
\]

\[
= c \left[ P_{\text{aux}} w - \left( \frac{n - 1}{n} I_n + \frac{1}{n} D \right) w \right] + w. \tag{7.10}
\]

Suppose \( w \) is a right-eigenvector of matrix \( \tilde{P} \), i.e., \( \tilde{P} w = w \). Then, from equation (7.10),

\[
P_{\text{aux}} w = \left( \frac{n - 1}{n} I_n + \frac{1}{n} D \right) w. \tag{7.11}
\]

After some algebraic manipulation, we have \( P_s w = w \) (where \( P_s \) is the matrix in (7.8)), and since \( P_s \) is primitive column stochastic \( w \) is unique when normalized to be a stochastic vector (and thus \( w = v \)). \hfill \Box

Proposition 7.1 states that even if the updates are totally asynchronous, the correct right-eigenvector can still be found in a distributed fashion, provided that each node has knowledge of or, at least, eventually computes \( \frac{c}{q_j n} \). The problem of computing \( \frac{c}{q_j n} \) can also be addressed in a distributed fashion. Let \( q_{\text{min}} := \min_j \{ q_j \} \). It would be sufficient for the global parameter \( c \) to satisfy \( c \leq q_{\text{min}} n \), in which case

\[
\frac{c}{q_j n} \leq q_{\text{min}} = \frac{r_{\text{min}}}{\sum_{v_j \in V} r_j} = \frac{r_{\text{min}}}{r_j}, \tag{7.12}
\]

where \( r_j \) is a local parameter and it is assumed to be known to each node executing the algorithm (since it has to do with the hardware characteristics of the node). The global value \( r_{\text{min}} \) can be computed by executing a
min-consensus algorithm (or a max-consensus algorithm of the reciprocal). The max-consensus protocol (Algorithm 1 in Chapter 2) has been shown to converge in a finite number of steps (information exchanges), even in the presence of arbitrary asynchronous updates and bounded delays, provided that the underlying communication network is strongly connected Giannini et al., 2016.

**Example 7.1.** Consider a strongly connected directed graph consisting of 7 nodes, with matrix $P$ given by

$$P = \begin{bmatrix}
0 & 0 & 0 & 0.33 & 0 & 0 & 0 \\
1 & 0 & 0.33 & 0 & 0.33 & 0.25 & 0.5 \\
0 & 0.33 & 0 & 0.33 & 0 & 0.25 & 0 \\
0 & 0 & 0.33 & 0 & 0 & 0 & 0 \\
0 & 0.33 & 0 & 0.33 & 0 & 0.25 & 0.5 \\
0 & 0 & 0 & 0 & 0.33 & 0 & 0 \\
0 & 0.33 & 0.33 & 0 & 0.33 & 0.25 & 0
\end{bmatrix}.$$ 

Assuming that each node knows $\frac{c}{q_j P}$, we demonstrate in Figure 7.1 that the evolution of the time average of the state of each of the seven nodes converges to the correct value corresponding to its entry in the right-eigenvector. In this example, we have simply used $c = \min_j \{q_j\}$.

![Figure 7.1](image-url) **Figure 7.1:** In the left plot, the value of each state is computed in a distributed fashion and it is shown to approximate the actual entry in the right-eigenvector, despite the different clock speeds. The length of each curve on the x-axis indicates the clock speed of the corresponding node. In the right plot, the evolution of the average of the state of each node for the first 7000 updates is shown, illustrating that the actual entry in the right-eigenvector is approximated.
The PageRank Problem

Figure 7.2: Each node $v_j$ updates its value based on a random process $\theta[k]$ with probability $q_j$; see equation 7.5.

Note that the length of each curve on the $x$-axis in Figure 7.1 indicates the different clock speed of each corresponding server. However, in the simulation each node updates its value according to an i.i.d. random variable with different probability masses, as shown in Figure 7.2. In this example, it is shown that despite the different clock speeds and asynchronous updates the actual entry in the right-eigenvector is approximated in a distributed fashion.

The following proposition combines the results of Proposition 7.1 and properties of the max-consensus protocol to show convergence of a distributed algorithm in which each node $v_j$ adjusts the weight of its out-going links to compensate for its clock rate.

**Proposition 7.2.** When each node $v_j$ updates its value at time $k_j$, $k_j \in T_j$, based on the following update rule:

$$\tilde{P}_j = P_j D_j f_j[k_j] + I_n - D_j f_j[k_j],$$  \hspace{1cm} (7.13)

where $f_j[k_j] = \mu_j[k_j]/r_j$, $\mu_j[0] = r_j$ and $\mu_j[k_j] = \min_{i \in N_j} \mu_i[k_j - 1]$, it satisfies $x^*_j = \lim_{k_j \to \infty} y_j[k_j]$, where $y_j[k_j]$ is the time average of the state trajectory and it is computed by

$$y_j[k_j + 1] = \frac{1}{k_j + 1} (k_j y_j[k_j] + x_j[k]),$$  \hspace{1cm} (7.14)

with $y_j[0] = 0$.

**Proof.** Proposition 7.1 shows that convergence to the correct ranking can be achieved for a fixed parameter $f_j = \frac{c}{q_j n}$, $\forall v_j \in \mathcal{V}$. When $f_j$ changes in the update (see Eq. (7.13)) column stochasticity (and hence $\sum_{v_j \in \mathcal{V}} x_j[0]$) is preserved. The simplest way to verify convergence is that $f_j$ converges
in a finite number of steps (due to the min-consensus algorithm), and then (7.13) becomes similar to (7.6). Also, since the number of times each node $v_j$ updates tends to infinity as $k$ tends to infinity, there exists a finite-time window length $K$ and an infinite sequence of time instants such that for any $m \in \mathbb{Z}_+, 0 < t_{m+1} - t_m \leq K < \infty$, and the union of graphs $G[t_m], G[t_m + 1], \ldots, G[t_{m+1} - 1]$ is strongly connected (see Charalambous and Hadjicostis, 2014, Lemma 2). In other words, Conditions [C1] and [C2] in Chapter 3 hold.

7.5 The Asynchronous PageRank Algorithm

As aforementioned, the PageRank problem has a wide variety of applications and in different fields (see, e.g., Gleich, 2015). The graph representation of the Internet and the fact that each node in the graph knows the number of its out-going links results in a nonnegative column-stochastic matrix. The randomized updating, however, due to asynchrony and different clock rates, renders current distributed approaches unsuitable to estimate the PageRank vector correctly.

Using the theoretical results of the previous section, we describe a distributed algorithm that aims to deal with the totally asynchronous nature of the internet, i.e., to allow for asynchronous updates between the servers and at different clock rates. More specifically, based on Proposition 7.2 we describe Algorithm 16, that each server should follow for updating the PageRank value of the web page it hosts. The following example shows how the algorithm works for the network in Example 7.1.

Figure 7.3: The parameter $f_j[k_j]$ using a min-consensus approach.
**Algorithm 16:** Distributed Totally Asynchronous Algorithm for the Calculation of the PageRank Values

**Input:** A strongly connected digraph $G(V, E)$ with $n = |V|$ nodes and $m = |E|$ edges.

**Data:** $x[0] = x_0$, $k_j = 0$, $y[k_j] = 0$, and $\mu_j[0] = r_j$.

For $k = 0, 1, 2, \ldots$, if $\theta[k] = v_j$, then node $v_j \in V$ is ready to update its information as follows:

**Step 1:** It updates its own value $x_j[k + 1]$ based on the following update rule:

$$x[k + 1] = P[k]x[k],$$

where $P[k] = \tilde{P}_j$, with

$$\tilde{P}_j = P_j D_j f_j[k_j] + I_n - D_j f_j[k_j],$$

(7.15)

where $f_j[k_j] = \mu_j[k_j]/r_j$ and $\mu_j[k_j] = \min_{i \in N_j^-} \mu_i[k_j - 1]$;

**Step 2:** It computes the time average of the state trajectory:

$$y_j[k_j + 1] = \frac{1}{k_j + 1}(k_j y_j[k_j] + x_j[k]).$$

**Step 3:** It updates $k_j \leftarrow k_j + 1$ and $x_j[k_j] \leftarrow x_j[k + 1]$.

**Output:** Node $v_j$ estimates its PageRank value by

$$x^*_j = \lim_{k_j \to \infty} y_j[k_j].$$
Example 7.2. First, in Figure 7.3, for the network considered in Example 7.1 the min-consensus converges in a finite number of steps to the smallest $r_j$. In Figure 7.4, one can see that the evolution of the time average of the state trajectory of each of the seven nodes in a randomly-generated strongly-connected directed network converges to the PageRank value.

Figure 7.4: In the left plot, the value of each state is computed in a distributed fashion and it is shown to approximate the actual PageRank value, despite the different clock speeds. As before, the length of the curves on the $x$-axis indicate their clock speed. In the right plot, the evolution of the average of the state of each node for the first 5000 updates is shown, illustrating that the actual entry in the right-eigenvector is approximated.

7.6 Discussion

A similar concept of adjusting the weights according to the update rate has been also considered for asynchronous decentralized optimization in Rabbat and Tsianos, 2014.

When nodes update their values asynchronously and with different update rates, we assume that the probability of a node to be chosen for update follows a non-uniform probability mass function. However, one can consider the probability of update are based on the transition probabilities of a Markov chain. Such an approach for a different algorithm than Algorithm 16 has been considered in You et al., 2017. Furthermore, It would be interesting to study the convergence rate of this algorithm and its dependence on these probabilities of update.
References


Hadjicostis, C. N. and T. Charalambous. 2011. “Asynchronous coordination of
distributed energy resources for the provisioning of ancillary services”. In:
1500–1507.
in digraphs under interval constraints”. In: Proc. of IEEE Conference on
Decision and Control. 1769–1774.
distributed average consensus via exchange of running sums”. IEEE Trans.
on Automatic Control. 61(6): 1492–1507.
“Graph diameter, eigenvalues, and minimum-time consensus”. Automatica.
reliable multicast in multi-hop ad hoc networks”. In: Proc. of 3rd int.
Workshop on Discrete Algorithms and Methods for Mobile Computing and
Communications. ACM. 64–71.
“Byzantine modification detection in multicast networks with random
network coding”. IEEE Transactions on Information Theory. 54(6): 2798–
2803.
Hoffman, A. J. 1960. “Some recent applications of the theory of linear in-
equalities to extremal combinatorial analysis”. In: Proc. of Symposium of
Cambridge University Press.
synchronous consensus”. In: Proc. of the 2012 ACM Workshop on Privacy
in the Electronic Society. 81–90.
sus for nonlinear dynamical networks”. IEEE Trans. on Automatic Control.
Ishii, H. and R. Tempo. 2010. “Distributed randomized algorithms for the
2002.


