University of Groningen

Distributed Computation of Graph Parameters in Finite Time

Friso de Jong

Supervisor: Prof. Dr. M.K. Camlibel

Second assessor: Dr. A.E. Sterk

July 28, 2017

Mathematics

Faculty of Science and Engineering

Abstract

This project is about computing graph parameters of a directed graph in a distributed way and in finite time. Graph parameters can be used for the analysis of the underlying structure of a network. We will consider the following parameters: the left-eigenvector, the out-degree and the spectrum. The left-eigenvector is often used for determining the final state of components of a network and can also be used to compute other parameters. The out-degree is important for the analysis on the out-neighbourhood of each node. The spectrum gives much information regarding the structure of a network and can be used for acceleration of computations. The key point of this project is that we want to compute the parameters in a distributed way.

Contents

Introduction

In this project, graph parameter computation is considered. Graph parameters can reveal the underlying structure of a graph. Given the structure, one can do computations in order to reach a certain goal. The goal depends on the type of graph and so on the type of network considered. Graphs represent networks. Networks, on their part, represent real-life processes. Networks consists of nodes together with linking edges. These edges can be directed. For example, consider a traffic network. The destinations are linked by roads. The goal could be, for instance, to get from one destination to another as quickly as possible. The time needed for this depends also on the traffic flow. That is, more traffic on a road can slow down reaching a certain destination. The computation will require some knowledge of the structure on how the destinations are related. There are many more examples of networks. One can think of computer networks, sensor networks, mobile robots, social networks or biological processes in a human body for instance. So graph parameters play an important role in fields such as computer science, physics, sociology and biology. The translation from networks to graphs is explained in Chapter 1. The translation is done by means of two applications. The applications also reveal the need/use of graph parameters.

As mentioned in the abstract, we want to compute three parameters: the left-eigenvector of the adjacency matrix, the out-degree of each node and the spectrum of the adjacency matrix. In this project we consider two applications which use these graph parameters: (average) consensus and forming bistochastic matrices. Reaching average consensus is a principle which tries to achieve a common goal among all the nodes in the network. A node, as part of a network, reaches average consensus if the final state of all nodes reach the same value. The computation of this can be done, among other things, by using the left-eigenvector (see for example [1]). In Chapter 2 the computation of the left-eigenvector is considered. The spectrum can be used in order to reach consensus in the fastest possible time. Also, the computation of the spectrum, will give rise to many applications in coding theory, quantum chemistry, social sciences, and so on and so forth. The eigenvalues can be used for counting structures (acyclic digraphs, spanning trees, Hamiltonian cycles, et cetera). This is discussed in Chapter 4. Forming bistochastic matrices is a principle which tries to balance for each node in the network, the incoming and the outgoing information flow. Forming bistochastic matrices is actually a special case of weight-balancing principles. Proposed algorithms needed the out-degree [2] as well as the left-eigenvector. The out-degree computation will be discussed in Chapter 3. Bistochastic matrices find their usefulness in matrix scaling problems [3].

So, there are many applications of the three graph parameters. These parameters depend on the type of graph. The type of graph depends on the network. There exists undirected graphs, (strongly) connected graphs, regular graphs, et cetera. In this project we will consider static strongly connected digraphs. A static digraph is a directed graph which does not change topology. Strongly connected means that each component is reachable from each component by a path. On a graph we can make certain assumptions. For example, one can assume that global parameters such as the left-eigenvector or the out-degree are known to each component. We will discuss these assumptions throughout the paper.

Above we mentioned a central knowledge property (global parameter) of a graph. But what does this mean? Central knowledge of parameters for instance, means that every component of the network has knowledge of these parameters. In this project, we assume that components are not aware of these parameters and so are not aware of the graph structure. For example, the out-neighbourhood of a node may be unknown. Out-neighbourhood is the set of nodes which receive information of the node considered. It is still possible to do computations, even though components are not aware of global parameters. These computations are called distributed (or decentralized) computations. Distributed computation is desirable, since it is not always possible to do computations centralized. This may be due to the fact that a network is too large, so it would require too much computational power. Also the memory at each node is limited. So for example, a mobile phone cannot do heavy computations and has limited storage capacity. So the key point in this project is that it is desirable to do the computations in a distributed way.

At last we want to do computations in finite time. This speaks for itself. Estimations of graph parameters through termination conditions for instance, are not desirable. Namely, a termination condition has to be set a priori, requiring some global knowledge. In this paper, we will show that the graph parameters can be determined in finite time.

1 From networks to graphs

As mentioned in the introduction, we are interested in finding three graph parameters. In this chapter we translate networks into graphs by means of two applications. Also, the need of graph parameters will become clear from these applications.

The applications we consider in this project are (average) consensus and forming bistochastic matrices. It is important to notice that we focus on finding the graph parameters. So the computation of the applications considered are not in our interest regarding this thesis.

In order to state our problem properly, we need to say what kind of graphs we consider and what terminology we will use. In this project, we consider a graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$ to be a set of n nodes $\mathcal{N} = \{v_1, v_2, \ldots, v_n\}$, together with a set of connecting edges $\mathcal{E}$. The connecting edges are denoted by ε_{ji}, which represent the link between nodes v_j and v_i. This link enables node v_i to send information to node v_j. Each node has some connection in the network. This means that it can transmit data to other nodes as well as receive incoming data from other nodes. The transmitting and receiving of node v_j can be captured by the state of a node $x_j[k]$, which is a real number and where $k \geq 0$.

1.1 Reaching consensus

One of the main features of distributed computation in a graph is reaching consensus. Consensus is a principle which tries to achieve a common goal. In a graph this means that the nodes try to converge asymptotically to a one-dimensional agreement space: $x_1[k] = x_2[k] = \cdots = x_n[k]$ for some k. Let us illustrate this by means of an example.

Example 1.1.1. We can look at a group of people, who try to make an agreement by taking their opinions into account. The state of a person then correspond to the opinion. So in this case, we would like to have the opinions of each person to be the same for some time instant k. In this thesis, we are interested in the distributed computation of this agreement. Meaning that not all the members of the group know the opinions of their fellows, but only a few. The connection of a person's opinion can be compared to social networks. The opinion of a friend for example, probably counts heavier than the opinion of some stranger.

An attempt for solving the problem could be by averaging the opinions of all the people involved. That is, $\frac{\sum_{v_i \in \mathcal{N}} x_i[k]}{n}$, where n is the number of people involved. This is called average consensus and is explained below. The example is part of a well-known field of interest: opinion

dynamics. See for instance [4]. This field is concerned with social networks. A commonly asked question is whether consensus can be reached or whether fragmentation or polarization arises.

How consensus works can be nicely illustrated using an applied version: average consensus. This principle tries to average the state of all the nodes. More precisely, we say that the nodes reach asymptotically average consensus if

$$\lim_{k\to\infty} x_j[k] = \frac{\sum_{v_i\in\mathcal{N}} x_i[0]}{n}, \qquad \forall v_j \in \mathcal{N}, \tag{1}$$

where $x_i[0]$ denotes the initial state of node v_i. We denote $\mathbf{x}[0]$ to be the matrix $\begin{pmatrix} x_1[0] & x_2[0] & \cdots & x_n[0] \end{pmatrix}^T$. The average consensus can be computed by choosing an initial condition $\mathbf{x}[0]$ and modify it such that $\tilde{\mathbf{x}}[0] = \mathbf{x}[0] + \gamma$, where $\gamma = \begin{pmatrix} \gamma_1 & \cdots & \gamma_n \end{pmatrix}$ is the adapting term. In [5] one can find the following relation:

$$\frac{\sum_{v_i\in\mathcal{N}} x_i[0]}{n} = \sum_{v_i\in\mathcal{N}} w_i(x_i[0] + \gamma_i), \tag{2}$$

where w_i will be defined after we have looked at the general dynamics of the network. In order to determine the adapting term, we have to know whether the limit (1) exists, and so, when reaching average convergence is possible. Many algorithms have been proposed which try to estimate the limit. But it is desirable to know whether this limit can be computed exactly or when there is some stopping criterion.

Remark. The initial states are not known to each node. Also, the total number of nodes n in a network is not known to every node. However, n can be determined by each node, which is explained later on.

Each node is somehow part of a network. So in order to analyse whether or not nodes can reach average consensus, we need to look at the whole system and its dynamics.

1.1.a Dynamics of the network

The dynamics of the network results from the information flow between the nodes. Each node updates its state by gathering information from incoming arrows of nodes. The set of incoming links at v_j is denoted by $\mathcal{N}_j^-$. The update of the state of a node depends on other nodes. In this project we consider information flows between nodes as positive weights. This may seem a bit odd at first sight, but in the example of consensus, one can think of weight as the significance of one's opinion. The more priority is given to someone's opinion, the more 'weight' it is assigned to. So the information flow from node v_i to v_j can be captured by a positive weight p_{ji}. In this project we assume that each node can choose its self-weight and the weight of its incoming arrows only and that the total weight adds up to exactly 1. We also assume that the self-weight is between 0 and 1. That is, $p_{jj} \in (0, 1)$. Considering the example again, this means that a person can choose the significance of his or her own opinion and the significance of opinions from people to which he or she is connected to. So the significance of other's opinions depends

on what the person himself considers to be important.
From the analysis above, we get the following equation:

$$x_j[k+1] = p_{jj}x_j[k] + \sum_{v_i \in \mathcal{N}_j^-} p_{ji}x_i[k]. \tag{3}$$

for $k \geq 0$ and where $x_j[0] \in \mathbb{R}$ is the initial state of node v_j. Recall that $\mathcal{N}_j^-$ denotes the set of incoming links at node v_j. So the update of the state of node v_j, depends in its self-weight p_{jj} and on the sum of all incoming weights p_{ji}, where each incoming weight should have a connection from node v_i to v_j. Let $\mathbf{x}[k]=\begin{pmatrix} x_1[k] & x_2[k] & \cdots & x_n[k] \end{pmatrix}^T$. Then we get the following system:

$$\mathbf{x}[k+1] = P\mathbf{x}[k] \tag{4}$$

where $\mathbf{x}[0]=\begin{pmatrix} x_1[0] & x_2[0] & \cdots & x_n[0] \end{pmatrix}^T$ is the initial state of the system and P is a real, positive square matrix. This matrix P is called the weighted adjacency matrix.

Remark. The nodes are not aware of the global parameters such as the matrix P. In this sense, the nodes are able to do computations locally (distributedly). That is, a node is able to receive information and updates its state, without need for additional communication.

We can now apply the analysis above to Figure 1 below. It has 4 nodes and connecting arrows (edges). The nodes can give the incoming information a certain weight. Each node can also choose its self-weight, which is represented by the arrows out- and in-going at the same node.

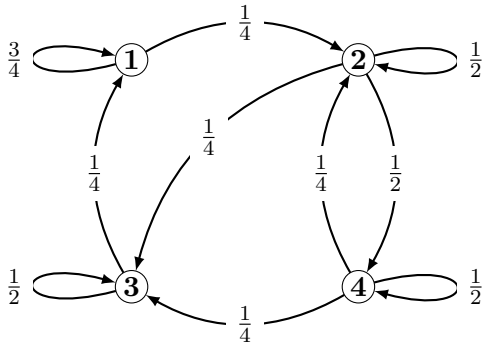


Figure 1: An example.

Recall that we want to reach average consensus. Since we have translated the network into a graph, we can analyse the matrix P and impose certain conditions, which guarantee average consensus.

1.1.b Conditions on the matrix P

Whether each node can reach average consensus depends on the matrix P. The matrix P can be given conditions such that average consenus is reached. The necessary and sufficient condition for reaching average consensus is that the matrix P should be primitive. That is, there should exists

a k such that $P^k > 0$. A sufficient condition for P being primitive is: it should be non-negative, irreducible and it should have at least a positive element on the maindiagonal. In this project we assume that the weights are strictly positive. Also the self-weights (the main diagonal elements) are strictly positive. One may wonder what it means for P to be irreducible. There is a theorem which states that P is irreducible if and only if the graph is strongly connected. Strongly connectedness of the graph means that there exits a path from every node to every node. That is, one should be able to find a sequence of connected nodes, such that one can reach every node from every node. In this project we also assume to deal with strongly connected graphs. The assumption however that the P should be irreducible is not always necessary. One can approach the situation as close as possible to the irreducible case. This is, however, a whole study on itself, and therefore we will not discuss it in this project.

The matrix $P_{\geq 0}$ (notation for having positive entries) can be given such that it adheres to the graph structure. It can be constructed in several ways. Recall that the in-degree at node v_j is denoted by $\mathcal{N}_j$. The cardinality of $\mathcal{N}_j$ (the number of incoming arrows) is denoted by D_j^-. We consider two examples of weight selection in distributed way.

Example 1.1.2. Distributed weight selection 1

$$P = [p_{ji}] := \begin{cases} 1 - c_j, & \text{if } i = j, \\ \frac{c_j}{D_j^-}, & \text{if } v_i \in \mathcal{N}_j^-, \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

where $c_j \in (0, 1)$.

Example 1.1.3. Distributed weight selection 2

$$P := I - \theta\mathcal{L}, \qquad 0 < \theta < \frac{1}{\max_i\{D_i^-\}}$$

In the first example c_j is called the proportionality gain of a node v_j. A node can freely choose this gain subject to the interval $(0, 1)$. In this way, the weights are determined by the nodes.

Assumption 1. In this project we consider static networks. That is, the matrix P does not change as k varies. This means that the proportionality gain is fixed. So c_j does not change as the nodes update their states.

Again, notice that it is called distributed weight selection, since the states of each node is updated based on its self-weight and the weight of its incoming arrows only. That is, each entry of P updates the state of a node based solely on the incoming information, as well as its current state.

In this thesis we assume that the P matrix is as in Example 1.1.2. For clarity let us consider Figure 1 from before. The matrix P is built up from nodes with self-weight, together with incoming weights. On the diagonal we get the updates regarding the self-weight. The off-diagonal

elements are depending on the structure of the graph. For example, the state of node v_1 depends on the incoming weight of node v_3, since there is an arrow going from v_3 to v_1. Excluding the count of its own incoming arrow, we have that $p_{13} = \frac{c_1}{D_1^-}$, since $v_3 \in \mathcal{N}_1^-$. Since node v_1 has only 1 incoming arrow, its in-degree $\mathcal{D}_1^-$ is equal to 1.

We get the following system:

$$\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \\ x_3[k+1] \\ x_4[k+1] \end{bmatrix} = \begin{bmatrix} 1-c_1 & 0 & c_1 & 0 \\ \frac{c_2}{2} & 1-c_2 & 0 & \frac{c_2}{2} \\ 0 & \frac{c_3}{2} & 1-c_3 & \frac{c_3}{2} \\ 0 & c_4 & 0 & 1-c_4 \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \\ x_3[k] \\ x_4[k] \end{bmatrix}$$

Remark. The in-degree does not count the incoming arrow of its own. Therefore, the in-degree of node v_1 for example, is equal to 1.

If each node chooses its proportionality gain c_j properly (that is, $c_j \in (0,1)$), one can easily see that each row adds up to exactly 1. So P is row-stochastic. Given the construction of P, we can now move on to the computation of the limit $\lim_{k\to\infty} x_j[k]$.

1.1.c Computation of the limit

In this subsection it will become clear that the left-eigenvector is needed for the computation of average consensus. Recall that we are interested in finding the limit $\lim_{k\to\infty} x_j[k]$ such that it is equal to $\frac{\sum_{v_i \in \mathcal{N}} x_i[0]}{n}$. We know that the state of node v_j depends on the state of other nodes. So it is convenient to include the matrix P in this limit. This can be done by (4). From this relation it follows that $\mathbf{x}[k] = P\mathbf{x}[k-1]$. Doing this k-times we get: $\mathbf{x}[k] = P^k\mathbf{x}[0]$. For the average consensus computation of node v_j we set $\mathbf{x}[0] = e_j$, where e_j is the vector with 1 at the jth component and zeros in the remaining entries. The limit becomes as follows:

$$\lim_{k\to\infty} \mathbf{x}[k] = \lim_{k\to\infty} P^k\mathbf{x}[0] = \lim_{k\to\infty} P^k e_j.$$

Now we want to show that this limit converges to the jth componenent of the left-eigenvector. For this, we use the Perron-Frobenius theorem. The Perron-Frobenius theorem states that if P is primitive, then three things hold:

1. one of the eigenvalues is positive and greater than or equal to (in absolute value) all other eigenvalues,
2. there is a positive eigenvector corresponding to that eigenvalue and
3. that eigenvalue is a simple root of the characteristic polynomial of P.

Now we can use the properties of this theorem in the computation of the limit. We can decompose P by the Jordan canonical form. Let T be a nonsingular matrix such that $P = TJT^{-1}$, which implies that $P^k = TJ^kT^{-1}$. Let $v_1, \ldots, v_n$ be the columns of T and $w_1^T, \ldots, w_n^T$ be the rows of T^{-1}. These vectors u_l's and w_m's are the generalized eigenvectors of P and are unique up

to positive scaling. By the Perron Frobenius theorem, we have that P has a simple, positive eigenvalue, say ρ and all other eigenvalues λ are strictly smaller than ρ. That is, $|\lambda| < \rho$. We know that $\rho = 1$ since P is row-stochastic. Matrix J is of the following form:

$$J = \left[\begin{array}{c|c} \rho & \mathbf{0} \\ \hline \mathbf{0} & \widetilde{J}_{ij} \end{array}\right]$$

Where $\widetilde{J}_{ij}$ is also in Jordan form. Since J is in Jordan form, we get:

$$J^k = \left[\begin{array}{c|c} \rho^k & \mathbf{0} \\ \hline \mathbf{0} & \widetilde{J}_{ij}^k \end{array}\right]$$

The entries of the matrix $\widetilde{J}_{ij}^k$ will converge to 0 as k tends to infinity. This can be seen by considering the form of $\widetilde{J}_{ij}$. Let λ_i, $i = 1, ..., l$ denote the the distinct eigenvalues of P and let the dimension of $\widetilde{J}_{ij}$ be $n_{ij} \times n_{ij}$. In [6] one can find the kth power of $\widetilde{J}_{ij}$:

$$\widetilde{J}_{ij}^k = \begin{bmatrix} \lambda_i^k & \binom{k}{1}\lambda_i^{k-1} & \cdots & \binom{k}{n_{ij}-1}\lambda_i^{k-n_{ij}} \\ & \lambda_i^k & \binom{k}{1}\lambda_i^{k-1} & \cdots & \binom{k}{n_{ij}-2}\lambda_i^{k+1-n_{ij}} \\ & & \ddots & \ddots & \vdots \\ & & & \lambda_i^k & \binom{k}{1}\lambda_i^{k-1} \\ & & & & \lambda_i^k \end{bmatrix}$$

Each nonzero entry will converge to 0 as k tends to infinity since $|\lambda_i| < \rho$. Also, since $\rho = 1$ we have that $\rho^k \to 1$. Resulting in the following equation:

$$\lim_{k\to\infty} P^k = \lim_{k\to\infty} TJ^kT^{-1} = T\left[\begin{array}{c|c} 1 & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \end{array}\right]T^{-1} = v_1 w_1^T.$$

where v_1 and w_1^T are right- and left-eigenvectors of P, respectively. We will denote v_1 by $\mathbf{v}$ and w_1 by $\mathbf{w}$. We know from the analysis before that the right-eigenvector v_1 (corresponding to the eigenvalue 1) is equal to $\mathbf{1}$. Finally, we get the following result:

$$\lim_{k\to\infty} \mathbf{x}[k] = \lim_{k\to\infty} P^k\mathbf{x}[0] = \lim_{k\to\infty} P^k e_j = v_1 w_1^T e_j = \mathbf{v}\mathbf{w}^T e_j = \mathbf{1}w_j,$$

where w_j is the jth component of the left-eigenvector w_1. So we are able to compute limit. The consensus value is actually contained in this limit. It is $\mathbf{v}\mathbf{w}^T$ times the initial value $\mathbf{x}[0]$.

In order to reach average consensus, the initial values should be adjusted. Recal that we had the following relation:

$$\frac{\sum_{v_i\in\mathcal{N}} x_i[0]}{n} = \sum_{v_i\in\mathcal{N}} w_i(x_i[0] + \gamma_i),$$

The initial values can be expressed in terms of the left-eigenvector. So, the adjusted term can be computed at each node, given the left-eigenvector of P and also n. The number of nodes of a

network however, is not known to each node. But the calculation of n is actually inherited by message passing, which is described in Section 2.4. So given the left-eigenvector, we can reach asymptotically average consensus.

For the calculation of the left-eigenvector sometimes an estimation is made. For example, by setting a maximum on the number of iterations needed. This is not desirable since the maximum has to be determined a priori. Therefore, it remains to solve the problem of finding the left-eigenvector corresponding to the eigenvalue 1. This is discussed in Chapter 2.

The rate of convergence of the algorithms for finding the consensus value depend on the structure of the graph. The spectrum of weighted adjacency P of the graph reveals information on the speed of convergence. There are some algorithms proposed which use the spectrum in order to speed up the convergence (see for example [7]). These are called self-configuring protocols. The spectrum computation of the weighted adjacency matrix will be discussed in Chapter 4.

As mentioned earlier, the computation of average consensus can be done in several ways. Other approaches for computation of the average consensus rely on for example the knowledge of the out-degree. The computation of the out-degree is handled in Chapter 3.

In this thesis we focus mainly on the application of average consensus. Nevertheless, there are many other applications equally important. We give a short description of another application. Namely, forming bistochastic matrices.

1.2 Forming bistochastic matrices

Bistochastic matrices are matrices which are both row- and column-stochastic. It is a special case of weight-balancing problems. Weight-balancing is a principle which tries to balance the total amount of incoming weight and the total amount of outgoing weight. A simple example illustrates the usefullnes of this application. Consider a network of water pipelines. Since pipelines have a certain flow capacity, it is important to keep the flow steady and to balance it. Similarly, information networks can be optimized such that the information flow is maximal. In order to make it more concrete, we introduce some mathematical notation.

Let $\mathcal{S}_j^-$ denote the total amount of in-weight at node v_j. This is equal to $\mathcal{S}_j^- = \sum_{v_i \in \mathcal{N}_j^-} p_{ji}$. Defining the out-weight from node v_j to v_i to be $p_i j$, one can do the same for the total amount of out-weight: $\mathcal{S}_j^+ = \sum_{v_i \in \mathcal{N}_j^+} p_{ij}$. It follows that weight-balancing tries to achieve the following equation for each node v_j: $\mathcal{S}_j^- = \mathcal{S}_j^+$. We know that $\mathcal{S}_j^- = 1$, since P is row-stochastic. Now we would like to make P such that also the columns of P add up to 1. That is, $\mathcal{S}_j^+ = 1$. This can be done by choosing the initial state (dependent on the out-degree) at node v_j and then compare $\mathcal{S}_j^-$ and $\mathcal{S}_j^+$ at each iteration. The update of the state can be adjusted such that $\mathcal{S}_j^+$ gets closer to 1. The algorithm which enables this can be found in [8]. This algorithm will form a bistochastic matrix. However, the initializing of each node depends on the out-degree. So what follows is the need of the out-degree.

In a graph, it is not always possible to form a bistochastic (doubly stochastic) matrix. A necessary and sufficient condition for forming bistochastic matrices can be found in [9]. There

exist several algorithms which balance the weights at each node. However, it is often assumed that the out-degree is known. This again implies the need for the out-degree. The out-degree computation is considered in Chapter 3.

2 Left-eigenvector computation

The first graph parameter we consider is the left-eigenvector of the (weighted) adjacency matrix P corresponding to the eigenvalue 1. That is, $\mathbf{w}^T P = \mathbf{w}^T$. From Chapter 1 we know that the left-eigenvector is often needed for algorithms. Sometimes the left-eigenvector is estimated by assuming to know the termination condition of the algorithm a priori, although this may lead to an erroneous approximation.

We know that the limit will converge to the left-eigenvector under certain conditions. The actual computation of the left-eigenvector will be shown in this chapter. There are several ways to compute the left-eigenvector. We will use the exact computation of the left-eigenvector given in [10]. However, there are some assumptions which make the claim of being a distributed questionable. This is discussed throughout the chapter. The computation requires several steps. Starting with the final value theorem.

2.1 Final value theorem

Recall that the limit we consider is as follows: $\lim_{k\to\infty} x_j[k]$. The final value theorem can be used for the computation of the limit. The theorem is stated below.

Theorem 2.1.1. *(Final Value Theorem)* If $\lim_{k\to\infty} x_j[k]$ exists, then

$$\lim_{k\to\infty} x_j[k] = \lim_{z\to 1}(z-1)X_j(z), \tag{6}$$

where $X_j(z)$ is the unilateral z-transform of $x_j[k]$.

A z-transform basically converts a discrete-time sequence, into a complex frequency domain representation. For the unilateral case, it is defined as follows:

$$X_j(z) = \mathcal{Z}\{x_j[k]\} = \sum_{k=0}^{\infty} x_j[k]z^{-k}. \tag{7}$$

The transform has a time-shift property. For example, we can set $y_j[k] = x_j[k+1]$. The z-transform of $y_j[k]$ then becomes:

$$Y_j(z) = \sum_{k=0}^{\infty} y_[k]z^{-k} = \sum_{k=0}^{\infty} x_j[k+1]z^{-k} = z\sum_{k=0}^{\infty} x_j[k+1]z^{-k+1} = z(X_j(z) - x[0]),$$

where the last expression is obtained by noticing that $X_j(z) = x[0] + z\sum_{k=1}^{\infty} x[k]z^{-k}$. We are able to do this as many times as we like. Say, we want to do this i times. Then we set $y_j[k] = x_j[k+i]$. Resulting in the next equation.

$$Y_j(z) = \sum_{k=0}^{\infty} y_j[k]z^{-k} = \sum_{k=0}^{\infty} x_j[k+i]z^{-k} = z^i \sum_{k=0}^{\infty} x_j[k+1]z^{-k+i} = z^i(X_j(z) - \sum_{k=0}^{i-1} x[k]z^{-k}). \quad (8)$$

The computation of $X_j(z)$ can be done with use of the minimal polynomial of the matrix pair $[P, e_j^T]$.

2.1.a Minimal polynomial

The transform $X_j(z)$ can be computed with use of the minimal polynomial of P. In particular, the minimal polynomial of the matrix *pair* $[P, e_j^T]$. We state both the definitions since the minimal polynomial of a matrix pair becomes more clear when compared to the definition of a minimal polynomial. The reason for introducing the polynomial for the matrix *pair* will become clear later, when it is applied.

Definition 2.1.1. (Minimal polynomial of a matrix) The minimal polynomial of a matrix P denoted by

$$q(t) = t^{D+1} + \sum_{i=0}^{D} \alpha_i^{(j)} t^i$$

is the monic polynomial of minimum degree $D+1$ that satisfies $q(P) = 0$.

By definition, the degree of minimal polynomial divides the characteristic polynomial of a matrix.

Definition 2.1.2. (Minimal polynomial of a matrix pair) The minimal polynomial associated with the matrix pair $[P, e_j^T]$ denoted by

$$q_j(t) = t^{M_j+1} + \sum_{i=0}^{M_j} \alpha_i^{(j)} t^i \quad (9)$$

is the monic polynomial of minimum degree $M_j + 1$ that satisfies $e_j^T q_j(P) = 0$.

Note that the polynomial $q_j(t)$ is not necessarily the same as $q(t)$. It is proved in [11] that $q_j(t)$ divides $q(t)$. This will be used in Section 2.2.

Now we want to use the property of the minimal polynomial ($e_j^T q_j(P) = 0$), in order to compute $X_j(z)$. The derivation of the following equation will be explained later on.

$$\sum_{i=0}^{M_j+1} \alpha_i^{(j)} x_j[k+i] = 0, \qquad \forall k \in \mathbb{Z}_+ \quad (10)$$

where $\alpha_{M_j+1}^{(j)}$ is assumed to be 1, since the minimal polynomial is monic. This enables us to

take the z-transform of the whole thing, and equate it to zero. In other words,

$$\mathcal{Z}\{\sum_{i=0}^{M_j+1} \alpha_i^{(j)} x_j[k+i]\} = \sum_{i=0}^{M_j+1} \alpha_i^{(j)} \mathcal{Z}\{x_j[k+i]\} = 0.$$

From (8), we know what $\mathcal{Z}\{x_j[k+i]\}$ is. So we get the following:

$$\sum_{i=0}^{M_j+1} \alpha_i^{(j)} \mathcal{Z}\{x_j[k+i]\} = \sum_{i=0}^{M_j+1} \alpha_i^{(j)} (z^i [X_j(z) - \sum_{l=0}^{i-1} x[l] z^{-l}]) = 0.$$

Implying that

$$X_j(z) \sum_{i=0}^{M_j+1} \alpha_i^{(j)} z^i = \sum_{i=0}^{M_j+1} \alpha_i^{(j)} \sum_{l=0}^{i-1} x[l] z^{i-l}.$$

Note that $\sum_{i=0}^{M_j+1} \alpha_i^{(j)} z^i$ is equal to $q_j(z)$. So ultimately, this results in:

$$X_j(z) = \frac{\sum_{i=1}^{M_j+1} \alpha_i^{(j)} \sum_{l=0}^{i-1} x_j[l] z^{i-l}}{q_j(z)}. \tag{11}$$

where the summation in the numerator starts at $i = 1$. Otherwise the index in the second summation sign is not defined.

It remains to show, how (10) was obtained. For that, consider the following equation:

$$e_j^T q_j(P) x[k] = e_j^T (\sum_{i=0}^{M_j+1} \alpha_i^{(j)} P^i) x[k] = e_j^T \sum_{i=0}^{M_j+1} \alpha_i^{(j)} x[k+i] = \sum_{i=0}^{M_j+1} \alpha_i^{(j)} x_j[k+i] = 0,$$

where we used the facts that $e_j^T q_j(P) = 0$ and $P^i x[k] = x[k+i]$. The reason for introducing the definition of a minimal polynomial of a matrix *pair* can be seen in the equation above.

2.2 Computation of the limit

The matrix P is assumed to be irreducible. Recall that this means that P has a simple eigenvalue at 1. Therefore, the characteristic polynomial has one unstable eigenvalue at 1 and all the other eigenvalues are stable. Since the minimal polynomial of the matrix pair divides the characteristic polynomial, also the minimal polynomial of the matrix pair has one unstable eigenvalue at 1. We can capture this by defining a new polynomial $p_j(z)$ as follows:

$$p_j(z) := \frac{q_j(z)}{1-z} := \sum_{i=0}^{M_j} \beta_i^{(j)} z^i, \tag{12}$$

where $z \neq 1$.
The $\beta_i^{(j)}$'s are some linear combination of the $\alpha_i^{(j)}$'s. The exact combination can be found by just solving $q_j(z) = (1-z)\sum_{i=0}^{M_j} \beta_i^{(j)} z^i$, where α_{M_j+1} is still assumed to be 1. From (12) we get

the following relation:

$$\begin{aligned}
\alpha_0^{(j)} &= \beta_0^{(j)} & & & \beta_{M_j}^{(j)} &= \alpha_{M_j+1}^{(j)} = 1 \\
\alpha_1^{(j)} &= \beta_1^{(j)} - \beta_0^{(j)} & & & \beta_{M_j-1}^{(j)} &= 1 + \alpha_{M_j}^{(j)} \\
&\vdots & & \Rightarrow & \beta_{M_j-2}^{(j)} &= 1 + \alpha_{M_j}^{(j)} + \alpha_{M_j-1}^{(j)} \\
\alpha_{M_j}^{(j)} &= \beta_{M_j}^{(j)} - \beta_{M_j-1}^{(j)} & & & &\vdots \\
\alpha_{M_j+1}^{(j)} &= \beta_{M_j}^{(j)} & & & \beta_1^{(j)} &= 1 + \alpha_{M_j}^{(j)} + \cdots + \alpha_2^{(j)} \\
& & & & \beta_0^{(j)} &= 1 + \alpha_{M_j}^{(j)} + \cdots + \alpha_2^{(j)} + \alpha_1^{(j)}
\end{aligned}$$

This relation can be used for the computation of the limit. So we are now able to compute the limit with use of the final value theorem 2.1.1, the z-transform and the new polynomial $p_j(z)$:

$$\begin{aligned}
\lim_{z\to 1}(z-1)X_j(z) &= \lim_{z\to 1}(z-1)\frac{\sum_{i=1}^{M_j+1}\alpha_i^{(j)}\sum_{l=0}^{i-1}x_j[l]z^{i-l}}{q_j(z)} \\
&= \lim_{z\to 1}(z-1)\frac{\sum_{i=1}^{M_j+1}\alpha_i^{(j)}\sum_{l=0}^{i-1}x_j[l]z^{i-l}}{(z-1)\sum_{i=0}^{M_j}\beta_i^{(j)}z^i} \\
&= \lim_{z\to 1}\frac{\sum_{i=1}^{M_j+1}\alpha_i^{(j)}\sum_{l=0}^{i-1}x_j[l]z^{i-l}}{\sum_{i=0}^{M_j}\beta_i^{(j)}z^i}.
\end{aligned}$$

We can substitute $z = 1$ and write it out:

$$\begin{aligned}
\lim_{z\to 1}\frac{\sum_{i=1}^{M_j+1}\alpha_i^{(j)}\sum_{l=0}^{i-1}x_j[l]z^{i-l}}{\sum_{i=0}^{M_j}\beta_i^{(j)}z^i} &= \frac{\alpha_1^{(j)}(x_j[0]) + \alpha_2^{(j)}(x_j[0]+x_j[1]) + \cdots + \alpha_{M_j+1}(x_j[0]+\cdots+x_j[M_j])}{\beta_0^{(j)}+\beta_1^{(j)}+\cdots+\beta_{M_j}^{(j)}} \\
&= \frac{x_j[0](\alpha_1^{(j)}+\alpha_2^{(j)}+\cdots+\alpha_{M_j+1}) + \cdots + x_j[M_j]\alpha_{M_j+1}}{\beta_0^{(j)}+\beta_1^{(j)}+\cdots+\beta_{M_j}^{(j)}} \\
&= \frac{x_j[0]\beta_0^{(j)} + x_j[1]\beta_1^{(j)} + \cdots + x_j[M_j]\beta_{M_j}^{(j)}}{\beta_0^{(j)}+\beta_1^{(j)}+\cdots+\beta_{M_j}^{(j)}} \\
&= \frac{\mathbf{x}_{M_j}^T\boldsymbol{\beta}_j}{\mathbf{1}^T\boldsymbol{\beta}_j},
\end{aligned} \tag{13}$$

where $\mathbf{x}_{M_j}^T = \begin{pmatrix} x_j[0] & x_j[1] & \cdots & x_j[M_j] \end{pmatrix}$ and $\boldsymbol{\beta}_j = \begin{pmatrix} \beta_0^{(j)} & \beta_1^{(j)} & \cdots & \beta_{M_j}^{(j)} \end{pmatrix}^T$.

It remains to compute the degree M_j of the minimal polynomial of the matrix pair $[P, e_j^T]$ and the vector $\boldsymbol{\beta}_j$. This is done in the next section.

2.3 Special case of the realization problem

The computation of the limit requires knowledge of the vectors $\mathbf{x}_{M_j}^T$ and $\boldsymbol{\beta}_j$. The structure of the vector $\mathbf{x}_{M_j}^T$ depends on the degree of the minimal polynomial of the matrix pair. Finding the degree is actually a special case of the realization problem. First a general problem description on the realization will be given. Then we apply it to our case.

A linear system can be represented in two ways: internally and externally. The relation from an internal system to an external system is already known to most of us: with use of a transfer matrix. The realization problem focuses on the other way around. That is, given an external description, construct an internal or state variable description. The definition below can be found in [12].

Definition 2.3.1. Given a sequence of $p \times m$ matrices $h[k]$, $k > 0$, the realization problem consists of finding a positive integer n and constant matrices (C, A, B) such that

$$h[k] = CA^{k-1}B, \quad C \in \mathbb{R}^{p\times n}, \quad A \in \mathbb{R}^{n\times n}, \quad B \in \mathbb{R}^{n\times m}, \quad k = 1, 2, ...$$

The triple (A, B, C) is then called the realization of the sequence $h[k]$, and the latter is called a realizable sequence. (C, A, B) is a minimal realization if among all realizations of the sequence, its dimension is the smallest possible.

Before stating how our case is a special case of the problem, let us first define what a Hankel matrix is and state in addition a lemma. A Hankel matrix $\Gamma\{h[k]\}$ of the parameters $h[k]$, where $k \geq 0$, has infinite many rows, infinite many columns, and block Hankel structure. It is defined as follows:

$$\Gamma\{h[k]\} := \begin{bmatrix} h[0] & h[1] & \cdots & h[k-1] & h[k] & \cdots \\ h[1] & h[2] & \cdots & h[k] & h[k+1] & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \\ h[k-1] & h[k] & \cdots & h[2k-2] & h[2k-1] & \cdots \\ h[k] & h[k+1] & \cdots & h[2k-1] & h[2k] & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \end{bmatrix}$$

The proof of the following lemma can be found in [12].

Lemma 2.3.1. The following statements are equivalent:

1. The sequence $h[k]$ is realizable.
2. The formal power series $\sum_{k>0} h[k]s^{-k}$ is rational
3. The sequence $h[k]$, $k > 0$, satisfies a recursion with constant coefficients, i.e., there exists a positive integer r and constant α_i, $0 \leq i < r$, such that

$$h[r+k] + \alpha_{r-1}h[r+k-1] + \alpha_{r-2}h[r+k-2] + \cdots + \alpha_1 h[k+1] + \alpha_0 h[k] = 0, \qquad k > 0.$$

4. The rank of the Hankel matrix is finite.

The lemma reveals what we will actually use. The computation of the degree M_j can be done with use of the Hankel matrix. The Hankel matrix can also be defined for our case. That is, we can consider the sequence of states of node v_j: $x_j[k]$. Usually, the Hankel matrix is represented by writing the first k components in the first row (although the row is of infinite length) and consequently, also for the first k components of the first column. For our case, this is denoted as $\Gamma\{\mathbf{x}_{2k}^T\}$. So we get the following Hankel matrix:

$$\Gamma\{\mathbf{x}_{2k}^T\} := \begin{bmatrix} x_j[0] & x_j[1] & \cdots & x_j[k] & \cdots \\ x_j[1] & x_j[2] & \cdots & x_j[k+1] & \cdots \\ \vdots & \vdots & \ddots & \vdots & \\ x_j[k] & x_j[k+1] & \cdots & x_j[2k] & \cdots \\ \vdots & \vdots & \ddots & \vdots & \end{bmatrix}$$

Now we can use (10). Recall that the equation was defined for all k:

$$\sum_{i=0}^{M_j+1} \alpha_i^{(j)} x_j[k+i] = \alpha_0^{(j)} x_j[k] + \alpha_1^{(j)} x_j[k+1] + \cdots + \alpha_{M_j}^{(j)} x_j[k+M_j] + \alpha_{M_j+1}^{(j)} x_j[k+M_j+1] = 0.$$

Introducing $\boldsymbol{\alpha}_j = \begin{pmatrix} \alpha_0 & \alpha_1 & \cdots & \alpha_{M_j} & \alpha_{M_j+1} \end{pmatrix}^T$, we get:

$$\begin{aligned} \Gamma\{\mathbf{x}_{2k}^T\}\boldsymbol{\alpha}_j : &= \begin{bmatrix} x_j[0] & x_j[1] & \cdots & x_j[k] \\ x_j[1] & x_j[2] & \cdots & x_j[k+1] \\ \vdots & \vdots & \ddots & \vdots \\ x_j[k] & x_j[k+1] & \cdots & x_j[2k] \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{M_j+1} \end{bmatrix} \\ &= \begin{bmatrix} x_j[0]\alpha_0 + x_j[1]\alpha_1 + \cdots + x_j[k]\alpha_{M_j+1} \\ x_j[1]\alpha_0 + x_j[2]\alpha_1 + \cdots + x_j[k+1]\alpha_{M_j+1} \\ \vdots \\ x_j[k]\alpha_0 + x_j[k+1]\alpha_1 + \cdots + x_j[2k]\alpha_{M_j+1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \end{aligned}$$

where the last equality follows from (10). The size of $\Gamma\{\mathbf{x}_{2k}^T\}$, for which we get the zero vector, depends on the M_j. In order to determine whether we get the zero vector, we need to increase the Hankel matrix, by increasing k. When the zero vector is obtained, the degree M_j is then equal to the number of elements of $\boldsymbol{\alpha}_j$ minus 2. When the zero vector is obtained, the Hankel matrix is of maximal rank. The proof can be found in [12]. This means that increasing k will result in rows (or columns) being linearly dependent. So the rank of the Hankel matrix will increase as k increases until M_j has been reached. Increasing k with 1 will result in a matrix with the same rank. This is called the first defective Hankel matrix. So if the rank has not changed when k increased its value with 1, then one knows that M_j has been reached and the first defective Hankel matrix is obtained.

Remember that we are interested in finding the vector $\boldsymbol{\beta}_j$ instead of $\boldsymbol{\alpha}_j$. From (12) we had a relation between the α_j's and β_j's. So a similar computation can be done for the vector $\boldsymbol{\beta}_j$. Introducing the vector of differences between successive values $\overline{\mathbf{x}}_{2k}^T = \begin{pmatrix} x_j[0] - x_j[1] & \cdots & x_j[2k] - x_j[2k+1] \end{pmatrix}$, we get the following Hankel matrix:

$$\Gamma(\overline{\mathbf{x}}_{2k}^T) := \begin{bmatrix} x_j[0] - x_j[1] & x_j[1] - x_j[2] & \cdots & x_j[k] - x_j[k+1] \\ x_j[1] - x_j[2] & x_j[2] - x_j[3] & \cdots & x_j[k+1] - x_j[k+2] \\ \vdots & \vdots & \ddots & \vdots \\ x_j[k] - x_j[k+1] & x_j[k+1] - x_j[k+2] & \cdots & x_j[2k] - x_j[2k+1] \end{bmatrix}$$

Multiplying the above matrix with $\boldsymbol{\beta}_j$ gives again the zero vector. That is,

$$\begin{aligned} \Gamma(\overline{\mathbf{x}}_{2k}^T) &= \begin{bmatrix} (x_j[0] - x_j[1])\beta_0^{(j)} + (x_j[1] - x_j[2])\beta_1^{(j)} + \cdots + (x_j[k] - x_j[k+1])\beta_{M_j}^{(j)} \\ (x_j[1] - x_j[2])\beta_0^{(j)} + (x_j[2] - x_j[3])\beta_1^{(j)} + \cdots + (x_j[k+1] - x_j[k+2])\beta_{M_j}^{(j)} \\ \vdots \\ (x_j[k] - x_j[k+1])\beta_0^{(j)} + (x_j[k+1] - x_j[k+2])\beta_1^{(j)} + \cdots + (x_j[2k] - x_j[2k+1])\beta_{M_j}^{(j)} \end{bmatrix} \\ &= \begin{bmatrix} x_j[0]\beta_0^{(j)} + x_j[1](\beta_1^{(j)} - \beta_0^{(j)}) + \cdots + x_j[k]\beta_{M_j}^{(j)} \\ x_j[1]\beta_0^{(j)} + x_j[2](\beta_1^{(j)} - \beta_0^{(j)}) + \cdots + x_j[k+1]\beta_{M_j}^{(j)} \\ \vdots \\ x_j[k]\beta_0^{(j)} + x_j[k+1](\beta_1^{(j)} - \beta_0^{(j)}) + \cdots + x_j[2k]\beta_{M_j}^{(j)} \end{bmatrix} \\ &= \begin{bmatrix} x_j[0]\alpha_0 + x_j[1]\alpha_1 + \cdots + x_j[k]\alpha_{M_j+1} \\ x_j[1]\alpha_0 + x_j[2]\alpha_1 + \cdots + x_j[k+1]\alpha_{M_j+1} \\ \vdots \\ x_j[k]\alpha_0 + x_j[k+1]\alpha_1 + \cdots + x_j[2k]\alpha_{M_j+1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \end{aligned}$$

So we are able to compute the degree M_j and also the vector $\boldsymbol{\beta}_j$. The degree M_j can also be computed using the vector of differences together with $\boldsymbol{\beta}_j$ (as the kernel of the first defective Hankel matrix). That is, the zero vector will still be obtained when using the vector of differences together with $\boldsymbol{\beta}_j$. This is more convenient when one wants to compute it on a computer, since it is requires only the Hankel matrices of the vector of differences.

Given (3) we can compute the left-eigenvector from the analysis above. This equation, however, together with the resulting system (4), needs to be implemented. This is due to the fact that nodes need to know which component of the left-eigenvector has to be computed. The initial conditions are depended on which component of the left-eigenvector is computed. So in order to let nodes iterate the system (4) with the proper set of the initial values, we need to start an iteration at a node, such that the other nodes will adhere to this starting iteration. So an implementation is needed and it should be based on an important assumption. That is,

Assumption 2. Each node has a unique identification and is aware of this.

This assumption is satisfied if each node has a unique identifier (for example, a MAC address). The assumption is not desirable to have since it requires knowledge on a global level. It is not

desirable since the purpose of the computations is to do it locally. Assuming that a node is aware of its identification, requires some knowledge, which is more than just receiving information. Therefore, the claim to be distributed is undermined. When considering the implementation, we comment on this a bit more. Given the assumption, an implementation of system (4) together with the initial values is discussed in the next section.

2.4 Message passing

In order to do the iterations described in the previous section with (4), we need the nodes of the graph to communicate with each other in such a way, that each node is able to set its initial values correctly. This can be done by message passing.

The idea of message passing is as follows: every node initiates an iteration regarding the computation of that specific component of the left-eigenvector. If we consider for example the node v_j and we would like to compute the jth component of the left-eigenvector, then v_j starts an iteration. The iteration is based on the incoming information and on the current state. It sets the initial values as follows: if v_j receives information from nodes v_i's then it sets its initial value equal to 0. And it sets the initial value equal to 1 regarding its own state. The update of the state of node v_j is depending on its incoming links v_i's. The state of the nodes v_i on their part are dependent on incoming links from other nodes. So having the initial values set, the update (iteration) of the node v_j can be computed. Since the computation depends on the states of all the other nodes, every nodes contribute to the computation of the jth component of the left-eigenvector. When all the nodes have contributed to the computation of the jth element of the left-eigenvector, a new iteration can be started by another node, regarding the computation of that specific component of the left-eigenvector. The number of iterations needed for each component of the left-eigenvector at each node is not required to be known a priori. Each node runs the iteration, as soon as it receives weights from other nodes together with the knowledge of knowing who initiated the iteration. In [6] it is shown that the maximum number of iterations needed for each node to compute the jth component of the left-eigenvector is equal to $2|\mathcal{N}|$. The number of nodes in the network, $|\mathcal{N}| = n$, is inheritly determined. Since each node initiates an iteration regarding the computation of a specific component of the left-eigenvector, there are $|\mathcal{N}|$ iterations initiated.

The approach given above can be implemented by Assumption 2 made in the previous section. Otherwise, a node cannot set the initial values accordingly. That is, it would not be able to distinguish its own identification with another node for instance.

In order to implement this method, we have to introduce new notation regarding the state of a node together with the specified component of the left-eigenvector. In this way, we can set the initial values. Let $x_{jl}[k]$ denote the state of node v_j (at time instant k) regarding the computation of w_l, where w_l denotes the lth component of the left-eigenvector $\mathbf{w}$. The following dynamics are obtained when considering the method of message passing: Each node x_j initiates an iteration regarding the computation w_l where the initial state $x_{jl}[0]$ is either 1 if $j = l$ or 0

otherwise. We get the following equation:

$$x_{jl}[k+1] = (1-c_j)x_{jl}[k] + \frac{c_j}{\mathcal{D}_j^-} \sum_{v_i \in \mathcal{N}_j^-} x_{il}[k] \tag{14}$$

where

$$x_{jl}[0] = \begin{cases} 1, & \text{if } l = j \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

Let $\mathbf{x}_l[k]=\begin{pmatrix} x_{1l}[k] & x_{2l}[k] & \cdots & x_{nl}[k] \end{pmatrix}^T$. Similar to the case of before, we can capture this in the following system:

$$\mathbf{x}_l[k+1] = P\mathbf{x}_l[k], \qquad \mathbf{x}_l[0] = e_l, \tag{16}$$

where P is defined as in Chapter 1. Let us revisit Figure 1 again from Chapter 1. Consider the first node: $j = 1$. The state update is then as follows: $x_{1l}[k+1] = (1-c_j)x_{1l}[k] + \frac{c_j}{1}x_{3l}[k]$. As can be seen, the P matrix will still be obtained. So the system will still adhere to the graph structure.

The convergence of the new system can be obtained using the same analysis as for the system (4). Recall that we had the following result: $\lim_{k\to\infty} P^k = \mathbf{1}\mathbf{w}^T$. We can use this in order to prove the following result.

Proposition 2.4.1. Given the system as in (14), with initial condition given by (15), the limit of the state each node v_j will converge to the l-component of the left-eigenvector of P. That is, $\lim_{k\to\infty} x_{jl}[k] = w_l$.

Proof. Using the Perron-Frobenius theorem, together with equation above the proposition, we get the following result: $\lim_{k\to\infty} P^k e_l = \mathbf{1}\mathbf{w}^T e_l = \mathbf{1}w_l$. □

As a result of the above proposition, we can state the following result. This basically means that the above result is the unique fixed point which can be obtained in finite time and in distribtuted way.

Proposition 2.4.2. Given a strongly connected digraph $\mathcal{G}(\mathcal{N}, \mathcal{E})$. Let $x_{ji}[k]$ be de result of the iteration (14)(for all $v_j \in \mathcal{N}$ and $k = 0, 1, 2, ...$), where P is any set of weights that adhere to the graph structure and form a primitive column-stochastic weight matrix. The unique fixed point of (14) can be obtained in finite time and in a distributed manner.

Given the fact that the implementation still guarantees a convergence to the left-eigenvector, we can introduce again the vector of differences between successive values $\overline{\mathbf{x}}_{l,2k}^T=$
$\begin{pmatrix} x_{lj}[0] - x_{lj}[1] & \cdots & x_{lj}[2k] - x_{lj}[2k+1] \end{pmatrix}^T$. Resulting into the following Hankel matrix:

$$\Gamma(\overline{\mathbf{x}}_{l,2k}^T) := \begin{bmatrix} x_{lj}[0] - x_{lj}[1] & x_{lj}[1] - x_{lj}[2] & \cdots & x_{lj}[k] - x_{lj}[k+1] \\ x_{lj}[1] - x_{lj}[2] & x_{lj}[2] - x_{lj}[3] & \cdots & x_{lj}[k+1] - x_{lj}[k+2] \\ \vdots & \vdots & \ddots & \vdots \\ x_{lj}[k] - x_{lj}[k+1] & x_{lj}[k+1] - x_{lj}[k+2] & \cdots & x_{lj}[2k] - x_{lj}[2k+1] \end{bmatrix}$$

From the analysis we had before, we can obtain the lth component of the left-eigenvector by just by letting each node iterate equation (14) with initial condition (15), regarding the computation of w_l. Recall that increasing k will give in the end M_j. Therefore we denote $\overline{\mathbf{x}}_{l,2k}^T$ by $\overline{\mathbf{x}}_{l,M_j}^T$. We get the following algorithm:

Algorithm 1 Distributed finite-time left-eigenvector computation

Input: A strongly connected digraph $\mathcal{G}(\mathcal{N}, \mathcal{E})$.
Data: Succesive observations for $x_{jl}[k]$, $\forall v_j \in \mathcal{N}$, $k = 0, 1, 2, ...$ via iteration (14), with initial values given by (15).
Step 1: For $k = 0, 1, 2, ...$, each node $v_j \in \mathcal{N}$ runs iteration (14) and stores vectors of differences $\overline{\mathbf{x}}_{lM_j}^T$ between succesive values of $x_{jl}[k]$.
Step 2: Increase the dimension k of the square Hankel matrices $\Gamma\{\overline{\mathbf{x}}_{lM_j}^T\}$ until they lose rank and store their first defective matrix. Extract the degree M_j from this.
Step 3: The kernel of $\Gamma\{\overline{\mathbf{x}}_{lM_j}^T\}$ gives the vector β_j.
Step 4: The final value is computed by

$$\frac{\mathbf{x}_{l_{M_j}}^T \beta_j}{\mathbf{1}^T \beta_j} = w_l$$

Output: w_l

So we are able to compute the left-eigenvector. As mentioned before, there are more ways for computing the left-eigenvector. The way we considered made use of several techniques, such as the final value theorem for example. In the next chapter, we want to use the left-eigenvector for the computation of the out-degree.

3 Out-degree computation

The second graph parameter we are interested in is the out-degree. This is the number of outgoing arrows of each node v_j, denoted by $\mathcal{N}_j^+$. From Chapter 1 we know that the out-degree is used for several algorithms for example. There are multiple ways to determine the out-degree of a node. One way of determining this has already been addressed in the literature. This is the so-called flooding approach, which is based on a neighbour discovery perspective. This method, however, requires big data chunks to be broadcast across the network and it imposes large memory requirements on nodes. Therefore, we would like to have a better method, which requires less storing of data for instance. The flooding approach will be discussed in Section 3.3 from this chapter. In the remaining sections we consider two methods. Namely, the out-degree computation via Integer Linear Programming (ILP) and computation via a consensus approach. The first one is faster than the second, but sometimes it gives multiple solutions although there is one correct answer. In that case we want to consider the second method. This method, however, requires additional communication between the nodes.

3.1 Computation via ILP

This method uses the left-eigenvector, which can be computed as described in the previous chapter. In this chapter we consider again the jth component of the left-eigenvector. Recall how we constructed the matrix P in Chapter 1. Each entry contains information on the incoming weights as well as its own weight. These were choosen by proportionality gains c_j's. The knowledge of the left-eigenvector enables the use of equation $\mathbf{w}^T = \mathbf{w}^T P$. For a node v_j, this is equivalent to $(\mathbf{w}^T P)_j = (\mathbf{w}^T)_j$. Writing it out gives the following relation:

$$(\mathbf{w}^T P)_j = \sum_i w_i P_{ji} = w_j(1 - c_j) + \sum_{v_i \in \mathcal{N}_j^+} w_i \frac{c_i}{D_i^-} = w_j = (\mathbf{w}^T)_j$$

From this we have $w_j c_j = \sum_{v_i \in \mathcal{N}_j^+} w_i \frac{c_i}{D_i^-}$. The computation of $\mathcal{N}_j^+$ can be done by rewriting the equation, such that we get a binary operator in the equation, which changes the sum. That is,

$$w_j c_j = \sum_{v_i \in \mathcal{N}} w_i \frac{c_i}{D_i^-} \mathbb{1}_i^j, \tag{17}$$

where

$$\mathbb{1}_i^j = \begin{cases} 1, & \text{if } v_i \in \mathcal{N}_j^+ \\ 0, & \text{otherwise.} \end{cases} \tag{18}$$

The out-degree of node v_j can be found by integer linear programming. That is, we are dealing with a *subset-sum* problem, which is a special case of the *knapsack* problem.

3.1.a Subset-sum problem

The knapsack problem is as follows: given n items with sizes $x_1, x_2, ..., x_n$ with $x_i > 0$ and capacity $W > 0$, maximize the sum, such that it is less then or equal to W. The sum being over the size of items times 0 or 1. That is,

$$\text{maximize} \quad \sum_{i=0}^{n} x_i y_i, \qquad \text{subject to} \quad \sum_{i=0}^{n} x_i y_i \leq W,$$

where $y_i \in \{0, 1\}$ for $i = 1, .., n$.

Translating it to our case gives: $W = w_j c_j$, $x_i = w_i \frac{c_i}{D_i^-}$ and $y_i = \mathbb{1}_i^j$. So the problem is maximizing the sum $\sum_{v_i \in \mathcal{N}} w_i \frac{c_i}{D_i^-} \mathbb{1}_i^j$, such that it will be less then or equal to $w_j c_j$. From (17), we know that the strict equality will be attained.

The problem can be solved using Integer Linear Programming. The ILP can give the solutions D_j^+ for all nodes v_j in a graph. It can happen that the ILP has multiple solutions. That is, each node can have a different out-degree such that (17) still holds for every node v_j . It can be shown (see for example [10]) in random networks (networks with nodes having a connection which depend on a certain probability) that the higher the connectivity, the lower the proportion of nodes for which the ILP does not have a unique solution. Having a higher connectivity is a relative term which we do not discuss in this project. In the case that the ILP gives more than one solution, it is desirable to come up with another approach, which is always able to give the correct out-degree for each node.

3.2 Consensus approach

The consensus approach is always able to find the correct out-degree, but it requires additional communication over the network. It concerns the knowledge of the out-neighbourhood. That is, node v_j requires the knowledge of $\mathcal{N}_j^+$. This is a drawback of the approach, since a node is not always immediately aware of its out-neighbours. Therefore, the computation via ILP is preferred. Nevertheless, the out-degree computation of this approach is still interesting, due to the fact that sometimes, nodes are able to determine their out-neigbourhood. Computations for this will not be discussed in this project.

The consensus approach also makes use of the left-eigenvector. This is the reason for calling it a consensus approach. It also uses almost the same algorithm as in the previous section. The only difference is the initial condition. In the new case, the initial conditions of the nodes are

depending on their out-neighbours. One runs the same iteration for each node v_j, but with a different initial condition. Regarding the iteration initiated by node v_l, the initial value of v_j is set equal to $1/w_j$ whenever it receives information from v_j, where w_j is equal to the jth component of the left-eigenvector. All other nodes set their initial value equal to zero. In other words,

$$x_{jl}[0] = \begin{cases} 1/w_j, \text{if } v_j \in \mathcal{N}_l^+ \\ 0, \text{otherwise.} \end{cases} \tag{19}$$

To see that this will give the out-degree of the node v_l, consider the analysis below, which is similar as convergence analysis before:

$$\lim_{k\to\infty} P^k \mathbf{x}[0] = \mathbf{1}\mathbf{w}^T\mathbf{x}[0] = \sum_{v_j \in \mathcal{N}_l^+} w_j(\frac{1}{w_j}) = \mathcal{D}_l^+$$

So one needs to proceed as follows: first the left-eigenvector needs to be computed using the linear iterations as in (14) with initial values given by (15). Secondly, the out-degree has to be computed using the same linear iterations as before (14), but with different initial values given in (19). It is convenient to summarise this in the following algorithm: Important to notice is the

Algorithm 2 Distributed finite-time out-degree computation

Input: A strongly connected digraph $\mathcal{G}(\mathcal{N}, \mathcal{E})$.
Data: Succesive observations for $x_{jl}[k]$, $\forall v_j \in \mathcal{N}$, $k = 0, 1, 2, ...$ via iteration (14), with initial values given by (19).
Step 1: For $k = 0, 1, 2, ...$, each node $v_j \in \mathcal{N}$ runs iteration (14) and stores vectors of differences $\overline{\mathbf{x}_l}_{M_j}^T$ between succesive values of $x_{jl}[k]$.
Step 2: Increase the dimension k of the square Hankel matrices $\Gamma\{\overline{\mathbf{x}_l}_{M_j}^T\}$ until they lose rank and store their first defective matrix. Extract the degree M_j from this.
Step 3: The kernel of $\Gamma\{\overline{\mathbf{x}_l}_{M_j}^T\}$ gives the vector β_j.
Step 4: The final value is computed by

$$\frac{\mathbf{x}_{l_{M_j}}^T \beta_j}{\mathbf{1}^T \beta_j} = \mathcal{D}_j^+$$

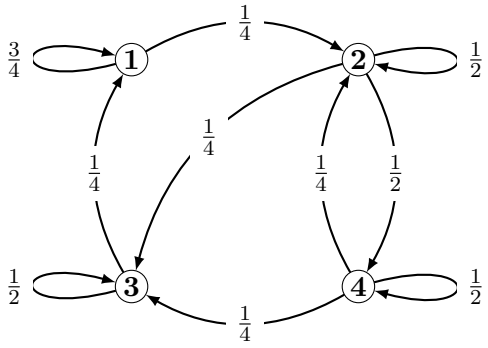
Output: $\mathcal{D}_j^+$

fact that the left-eigenvector should be computed beforehand. It is therefore not possible to combine the iterations in algorihtms 1 and 2.

As mentioned before, the fact that the out-neighbourhood is needed for this approach is a drawback. Additional communication is not desired, since one wants to do computations locally. But if the ILP produces more than one solution, and if it is the case that additional communication is possible, the solution can be obtained by algorithm 2. There is still another approach. This can be used if both the ILP and the consensus approach fails. This will be discussed in the next section.

3.3 Flooding approach

As mentioned earlier, the problem of finding the out-degree has already been addressed. The out-degree can be computed by neighbour discovery perspective. This is called the flooding approach. With this approach, each node v_j needs to broadcast D_j^- packets describing links of the form (v_j, v_i) where $v_i \in \mathcal{N}_j^-$ and all packets received from other nodes that have not been forwarded to its out-neighbors. Let us revisit Figure 1 from Chapter 1.



Node v_3 sends to node v_1 the links from which it has received information. That is, node v_3 sends its received information from nodes v_2 and v_4 to node v_1. So the links (v_2, v_3) and (v_4, v_3) are sent to v_1 by node v_3. Node v_1 has to store these two links. At the same time, nodes v_2 and v_4 can send their received information also to node v_3. That is, node v_2 received information from nodes v_1 and v_4. So v_2 broadcasts to node v_3 the links (v_1, v_2) and (v_4, v_2). Hence, node v_3 has to store these two links, and can also send it to node v_1 again. Node v_1 should now store four links in total. Similarly, node v_4 received information from node v_2, and so, sends to node v_3 the link (v_2, v_4). Sending this again to node v_1 means that node v_1 should store five links. At last the links (v_3, v_1) is send to v_2 by v_1 and (v_4, v_2) is send to v_4 by node v_2.

For clarity, let us deal with this step by step.

1. v_1 receives links from v_3: (v_2, v_3) and (v_4, v_3)
2. v_3 receives links from v_2 and v_4: (v_1, v_2), (v_4, v_2) and (v_2, v_4)
3. v_2 receives links from v_1: (v_3, v_1)
4. v_4 receives links from v_2: (v_4, v_2)

In this way, the graph structure is determined. And so, the out-degree can be made known to each node. As one can imagine, for very large graphs, the flooding approach takes a lot of time to determine the graph structure. In the example, it took four steps for each node, in order to determine their out-degree. In total, six links should be stored by each node.

3.3.a Convergence analaysis and storage requirements

In general the number of steps and the number of links needed to store at each node is of a greater amount. This depends on the size of a graph together with the number of links. In [10],

it can be found that the maximal amount of links a node needs to store is equal to $O(n\mathcal{D}^-_{max})$, where the so-called 'big-O' notation is used and where $\mathcal{D}^-_{max}$ is the maximum in-degree among all nodes. The maximum in-degree is equal to 2 in the example. The number of steps needed for the nodes to determine their out-degree is equal to $O(n\mathcal{D}^-_{max}d_{max})$, where d_{max} is the diameter of the graph. The diameter is equal to 3 in the example above. It is the longest of all the calculated shortest paths in a graph. From Figure 1 it follows that at least 3 links are needed for the path from v_3 to v_4.

The flooding approach can be used in for the computation of the out-degree. It takes, however, much communication between the links, as well as much storage requirements. Therefore, the aforementioned methods (ILP and consensus approach) are preferred.

4 Spectrum computation

In this chapter we consider the spectrum of the weighted adjacency matrix, $\sigma(P)$. The study of the eigenvalues can reveal much information on a graph. For example, one can determine whether two graphs are isomorphic or not. Isomorphic in the sense that two graphs are the same up to relabeling. As mentioned in Chapter 1, the spectrum can also be used for protocols which try to reach average consensus in the fastest time possible. For the computation of the spectrum in general, one can look at the characteristic polynomial of the matrix. The eigenvalues are then given by the roots of this polynomial. It is important to notice that we want to compute the spectrum in a distributed way.

Although eigenvalues can be computed, it should be mentioned that some eigenvalues cannot be obtained by looking at the characteristic polynomial of the matrix. That is, some eigenvalues are not observable to the modes of the system. The observability of the eigenvalues will be discussed after the computation of the observable eigenvalues.

Considering the observable eigenvalues, we have, by definition, that the roots of a characteristic polynomial are the same as the roots of the minimal polynomial. The roots of the minimal polynomial are equal to the roots of the minimal polynomial of a matrix pair. Recall that we had (12) as the relation between the α's and β's. That is,

$$\alpha_0^{(j)} + \alpha_1^{(j)} z + \cdots + \alpha_{M_j}^{(j)} z^{M_j} + z^{M_j+1} = \beta_0^{(j)} + (\beta_1^{(j)} - \beta_0^{(j)}) z + \cdots + (\beta_{M_j}^{(j)} - \beta_{M_j-1}^{(j)}) z^{M_j} + \beta_{M_j}^{(j)} z^{M_j+1}. \tag{20}$$

We are able to compute the vectors β_j's in a distributed way as described in Chapter 2. Hence, we can compute the $\alpha_i^{(j)}$'s by the equation above. In this way, the roots of the minimal polynomial of the matrix pair $[P, e_j^T]$ can be computed and so the observable eigenvalues of the weighted adjacency matrix P.

Since the computation works for every initial value, there is no need for an implementation of (4) as was done in Section 2.4. Hence we can use (4) given in Chapter 1. That is,

$$\mathbf{x}[k+1] = P\mathbf{x}[k]$$

with an arbitrary initial value $\mathbf{x}[0] = x_0$. Each node chooses an arbitrary initial value and updates its state accordingly. Now we can introduce a similar algorithm as we had before. Algorithm 3 enables one to compute all the observable eigenvalues of P in a distributed way.

Algorithm 3 Distributed finite-time spectrum computation

Input: A strongly connected digraph $\mathcal{G}(\mathcal{N}, \mathcal{E})$.
Data: Succesive observations for $x_j[k]$, $\forall v_j \in \mathcal{N}$, $k = 0, 1, 2, ...$ via iteration (4), with initial values given by $x[0] = x_0$.
Step 1: For $k = 0, 1, 2, ...$, each node $v_j \in \mathcal{N}$ runs iteration (4) and stores vectors of differences $\overline{\mathbf{x}}_{M_j}^T$ between succesive values of $x_j[k]$.
Step 2: Increase the dimension k of the square Hankel matrices $\Gamma\{\overline{\mathbf{x}}_{M_j}^T\}$ until they lose rank and store their first defective matrix. Extract the degree M_j from this.
Step 3: The kernel of $\Gamma\{\overline{\mathbf{x}}_{M_j}^T\}$ gives the vector β_j.
Step 4: The final value is given by the roots of (20).
Output: $\sigma(P)$

4.1 Observability of the eigenvalues

We can study the observability of the eigenvalues of P by looking at the observability Gramian. Again we can use the minimal polynomial of the matrix pair. Recall that the definition of a minimal polynomial of a matrix pair ensures that $e_j^T q_j(P) = 0$. Writing out the minimal polynomial $q_j(P)$ gives:

$$\begin{pmatrix} \alpha_0 & \alpha_1 & \cdots & \alpha_{M_j} & \alpha_{M_j+1} \end{pmatrix} \mathcal{O}_{j,M_j+1} = 0,$$

where the observability Gramian for a matrix pair is given by

$$\mathcal{O}_{j,M_j+1} = \begin{pmatrix} e_j^T & e_j^T P & e_j^T P^2 & \cdots & e_j^T P^{M_j+1} \end{pmatrix}^T.$$

Therefore, the coefficients of the minimal polynomial can be found by left nullspace of $\mathcal{O}_{j,M_j+1}$. Given the coefficients, the eigenvalues can be found by the roots of the minimal polynomial of the matrix pair $[P, e_j^T]$. The rank of its observability Gramian is given by the observable modes of operation from a node v_j. In this way, one can determine for example the number of unobservable eigenvalues.

It is important to notice that the observability of the eigenvalues can only be studied if the global parameter P is known to each component of the network. Algorithm 3 does not require this global parameter. Hence, the observable eigenvalues can be computed in a distributed way.

5 Conclusion

In this project, distributed finite-time computation of three graph parameters were considered. Namely, the left-eigenvector of the weighted adjacency matrix, the out-degree of each node, and the spectrum of the weighted adjacency matrix. These parameters determine the underlying structure of the network considered. The knowledge of the structure enables reaching a certain desired goal. Two goals (applications) were considered in Chapter 1: (average) consensus and forming bistochastic matrices. The first goal could be achieved by the computation of the left-eigenvector. The spectrum could be used, among other things, to accelerate the achievement of consensus. The second goal could be achieved with use of the out-degree. The applications were an illustration for the need of the graph parameters.

The computation of the graph parameters required the translation from networks to graphs. Having done so in Chapter 1, the left-eigenvector was computed in several steps in Chapter 2. The limit of each node could be found by using the final value theorem. The z-tranform needed, was defined with the use of the time-shift-property together with a property of the minimal polynomial of the matrix pair. The equation could be solved as a special case of the realization problem. The out-degree was found in different ways in Chapter 3. The problem could be solved by computation via ILP, which used the left-eigenvector. However, the ILP could give multiple solutions. In that case, the out-degree could be determined using the consensus approach. The last manner considered in this project is called flooding approach. This computation required a lot of data storing, so the ILP is still preferred. The spectrum could be computed by determining the roots of the minimal polynomial of the matrix pair. This was done in Chapter 4.

For each graph parameter, we were able to give an algorithm. The algorithms enabled us to compute the parameters in finite time. However, some objections can be made on the fact of being distributed. This applies to some assumptions made. For example, the assumption that each node is aware of its identification. Or the consensus approach for computation of the out-degree, which requires additional communication regarding the out-neighborhood. Also, the computation of the unobservable eigenvalues requires knowledge of the weighted adjacency matrix at each node.

It would be interesting to see other (or adjusted) methods regarding distributed finite-time computation of the graph parameters, which do not require these assumptions or additional communication.

Bibliography

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

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

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

[4] R. Hegselmann, U. Krause, *et al.*, "Opinion dynamics and bounded confidence models, analysis, and simulation," *Journal of Artificial Societies and Social Simulation*, vol. 5, no. 3, 2002.

[5] A. Priolo, A. Gasparri, E. Montijano, and C. Sagues, "A distributed algorithm for average consensus on strongly connected weighted digraphs," *Automatica*, vol. 50, no. 3, pp. 946–951, 2014.

[6] Y. Yuan, G.-B. Stan, L. Shi, and J. Gonçalves, "Decentralised final value theorem for discrete-time lti systems with application to minimal-time distributed consensus," in *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pp. 2664–2669, IEEE, 2009.

[7] A. Y. Kibangou, "Graph laplacian based matrix design for finite-time distributed average consensus," in *American Control Conference (ACC), 2012*, pp. 1901–1906, IEEE, 2012.

[8] T. Charalambous and C. N. Hadjicostis, "Distributed formation of balanced and bistochastic weighted digraphs in multi-agent systems," in *2013 European, Control Conference (ECC)*, pp. 1752–1757, 2013.

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

[10] T. Charalambous, M. G. Rabbat, M. Johansson, and C. N. Hadjicostis, "Distributed finite-time computation of digraph parameters: Left-eigenvector, out-degree and spectrum," *IEEE Transactions on Control of Network Systems*, vol. 3, no. 2, pp. 137–148, 2016.

[11] S. Sundaram and C. N. Hadjicostis, "Finite-time distributed consensus in graphs with time-invariant topologies," in *American Control Conference, 2007. ACC'07*, pp. 711–716, IEEE, 2007.

[12] A. C. Antoulas, *Approximation of Large-Scale Dynamical Systems*. SIAM, 2005.