Optimized distributed average consensus and its applications in cloud detection.

Yang, Haitao

Optimized Distributed Average Consensus and Its Applications in Cloud Detection

Prifysgol Abertawe
Swansea University

Haitao Yang

College of Engineering

Swansea University

Submitted to Swansea University in fulfillment of the requirements
for the degree of

Master of Philosophy

2013

I would like to dedicate this thesis to my loving parents...

Abstract

Distributed information processing has attracted more interests recently because of its advantages in distributed network than centralized method. Some of the tools of distributed signal processing are the distributed consensus algorithms. If the consensus algorithms are to find the average value over the network, the algorithms are called distributed average consensus (DAC) algorithms, which are matrix iterative algorithms to find the dominant eigenvector of the matrix. Generally, the DAC algorithms with optimized convergence rate can return the result more quickly so that the distributed system can have higher signal processing speed. Therefore, many efforts have been devoted into its optimization. However, most of the optimizations are centralized methods so that the system can not be optimized in a distributed network. In addition, the existing distributed optimization algorithm converges very slowly.

Consequently, we proposed a distributed real-time optimization for the DAC algorithms. The optimization has advantages not only in short computation time but also can work simultaneously with the consensus algorithms. Simulation results show that the DAC algorithms using centralized optimization and proposed optimization have similar performance in convergence rate, when floating point number in double format is used and the network size is less than 32. Later, an application of cloud detection is presented where optimized DAC algorithms are applied to perform the hypothesis testing. In addition, the performance of the detection system with different number of sensors is evaluated using relative operating characteristic curves. It is shown that detection system with more sensors can have better performance.

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed [redacted]candidate)
Date......29/04/2013..........

STATEMENT 1

This thesis is the result of my own investigations, except where otherwise stated. Where correction services have been used, the extent and nature of the correction is clearly marked in a footnote(s). Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed... [redacted]ndidate)
Date..........29/04/2013..........

STATEMENT 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organizations.

Signed..... [redacted]didate)
Date..........29/04/2013..........

Contents

Acknowledgements

I would like to express my gratitude to my thesis advisor Dr. Xinheng Wang for his careful guidance and his valuable comments. His patience and the constructive criticism helped me to organize many concepts in a frame. His encouragement has been invaluable. Thank you so much Henry!

I would also like to thank Prof. Jinho Choi for his insightful comments.
Their suggestions contributed to broaden my view of the distributed information system and the distributed consensus problem. Thank you for your help and encouragement.

Thanks to my colleagues Frank Liu, Tommy To, Laurie Hughes, John Li.

At last, I would like to sincerely thank my parents for their encouragement and support during all phases of this work.

List of Figures

List of Notations

$\mathcal{G}$	Graph associated to the network
v_i	The i^{th} node
$\mathcal{V}$	Set of nodes in the graph $\mathcal{G}$, $\mathcal{V} = \{v_1, v_2, ..., v_n\}$
$\mathcal{E}$	Set of edges in the graph $\mathcal{G}$, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$
$\mathcal{A}$	Weighted adjacency matrix associated with graph $\mathcal{G}$
W	Weight matrix of first-order DAC algorithm
w_{ij}	Entry of weight matrix W
$\mathbf{H}$	Weight matrix of higher-order DAC algorithms
$\lambda_i(A)$	The i^{th}eigenvalue of matrix A
$S(A)$	Spectrum of matrix A
Q	Incidence matrix associated with graph $\mathcal{G}$
L	Laplacian matrix induced by $\mathcal{G}$
l_{ij}	Entry of Laplacian matrix L
$\lambda_i(L)$	The i^{th} Eigenvalue of Laplacian matrix
$S_i(L)$	Local spectrum of the Laplacian matrix estimated by node v_i
$\mathbf{x}(k)$	Local value vector at time index k
$x_i(k)$	Local value of node v_i at time index k
$\bar{\mathbf{x}}$	The consensus value vector. All entries are the consensus value.
ϵ	Step length of higher-order DAC algorithms
γ	Forgetting factor of higher-order DAC algorithms
$\epsilon_{opt,FO}$	Optimal step length of first-order DAC algorithm
$\epsilon_{opt,SO}$	Optimal step length of second-order DAC algorithm
$\gamma_{opt,SO}$	Optimal forgetting factor of second-order DAC algorithm
$p(\lambda)$	The minimal polynomial of W
$p_i(\lambda)$	Local minimal polynomial of W estimated by node v_i
$T_i(k, D_i)$	Toeplitz matrix in $\mathbb{R}^{D_i \times D_i}$ at v_i with $x_i(k)$ as diagonal entries.
$\mathbf{y}_i(k, D_i)$	The local value history vector at v_i constructed by $[x_i(k+1), x_i(k+2), \ldots, x_i(k+D_i)]^T$
$\mathbf{h}$	An FIR filter that can estimate the consensus value

List of Abbreviations

2D	Two Dimensional
3D	Three Dimensional
WSNs	Wireless Sensor Networks
DAC	Distributed Average Consensus
FIR filter	Finite Impulse Response Filter
FO-DAC	First-order Distributed Average Consensus
SO-DAC	Second-order Distributed Average Consensus
HO-DAC	Higher-order Distributed Average Consensus
CFO-DAC	Constant First Order Distributed Average Consensus
FT-DAC	Finite Time Distributed Average Consensus
LMS	Least Mean Square
ML	Maximum Likelihood
LLR	Log Likelihood Ratio
G-LLR	Global Log Likelihood Ratio
L-LLR	Local Log Likelihood Ratio
PDF	Probability Density Function
MAP	Maximum A Posteriori
ML	Maximum Likelihood

Chapter 1

Introduction

Due to the development of wireless sensor networks and the decreasing cost of sensor nodes, processing information that originally acquired by the nodes is often necessary. As each node only has a piece of local information, all information needs to be gathered and processed at a special node called fusion center. This method is intuitive and is called centralized signal processing.

However, fusion center is usually the most important and expensive node in the network. Once it is destroyed or failed, the whole network breaks down. In addition, as data gathering is an energy consuming process, centralized signal processing is limited due to energy constraints of sensor nodes. Moreover, there is unbalanced energy consumption between nodes with different communication loads [3][4].

Distributed signal processing has become more attractive, as it can be robust against nodes failure or topology changes [5]. However, not all the signal processing can be distributed. Many processing algorithms are still implemented by centralized method.

Distributed Average Consensus (DAC) algorithms are tools for distributed information processing. They have received significant attention recently because of their robustness and simplicity. It is widely used in many applications such as time synchronization [6], rendezvous [7], cooperative control of vehicles [8], formation control/flocking [9] and WSNs [10]. In these applications, it is often necessary that a group of nodes can agree on certain quantities.

In different applications, DAC algorithms may also need a little modification, for example, in distributed tracking of a moving target by wireless sensor networks. Suppose each sensor is observing the target coordinates but the output is corrupted by independent and identically distributed zero-mean Gaussian noise, to minimize the interference from the noise, the sensors need to take the average of all initial values. In distributed hypothesis testing, we need to find the global likelihood ratio which is equal to the product of local likelihood ratios at each sensor. Therefore, each sensor first calculates the log of the local likelihood ratio and substitutes it into the DAC algorithms. The log of global likelihood ratio is then the average multiplied by the number of nodes in the network.

In practice, DAC algorithms also face the challenges of link failure, time delay, dynamic network, asynchronous communication and other practical aspects. Therefore, a reliable solution which can face these challenges is much needed in practice. First-order DAC algorithms have been proved to be robust against topology changes and they play important roles in practice [11]. The optimization of first-order DAC algorithms in a dynamic network still attracts lots of research interests [12][13].

1.1 Motivation of Research

The research is motivated by the distributed detection of cloud using WSNs. One of the properties of sensor nodes is the limited energy capacity. Therefore, to minimize the power consumption of the sensor nodes, DAC algorithms to perform the distributed signal processing need to be optimized for faster convergence rate. In addition, the Gaussian plume model of cloud requires a modification because the sensor is using laser sensing technology. It emits a laser to illuminate the cloud and collects the backscatters to sense the cloud concentration.

1.1.1 To Find a Faster DAC Algorithm

In the distributed tracking of a moving target, when the target is highly dynamic or the sensors need to sample at a very high frequency, it requires that the DAC algorithms return the result in a short time. Thus, many efforts have been devoted to optimize the DAC algorithms.

DAC algorithms can be divided into asymptotic and non-asymptotic algorithms. For asymptotic algorithms, the optimization is to minimize the subdominate eigenvalue of a weight matrix [14][15][1]. However, these optimization of DAC algorithms are centralized methods.

For non-asymptotic DAC algorithms, such as finite-time [16] and adaptive filter DAC algorithm [17], the optimization is to minimize the number of necessary iterations before a FIR filter is estimated. Sumdaram and Hadjicostis [16] verify that there exists an FIR filter that can estimate the consensus value. Cavalcante and Mulgrew [17] follow Sundaram and Hadjicostis's work to propose an adaptive algorithm to find the filter. However, their filter estimation algorithms are not robust against topology changes. They take local values over time obtained by the first-order DAC as inputs. As a result, if the network topology changes, these filter estimation algorithms have to be reinitialized, as outdated information during the filter estimation will lead to a wrong answer.

To enable the whole system to work distributively, the DAC algorithms should be robust against topology changes and the optimization should also be distributed.

A distributed method inspired by the gossip algorithm [18] can be used to optimize the first-order DAC but it converges very slowly. The method involves triple nested distributed matrix iterations. The inner iteration has to converge to a certain range so that the iteration outside can return the right result. Thus, it is not surprising that it could not finish in a reasonable time when the network size is large.

Therefore, a distributed optimization method with less computation time is required. In addition, it is better that no cost of additional communication is required. Moreover, if the optimization algorithms and the DAC algorithms can be executed simultaneously, consensus process will not be interrupted and the optimization can be running in background to keep the optimal parameters updated in a dynamic network.

1.1.2 To Detect the Cloud Plume by Laser Sensor

The cloud here is a group of harmful particles floating in the air. To detect the cloud, it is illuminated by a laser beam emitted by a sensor and the backscatters (light reflected by particles) are collected to generate signals.

Because the laser beam penetrate the cloud, the intensity of backscatters is the integration along the laser beam in that range. Consequently, a new cloud plume model needs to be obtained by taking integration of original Gaussian plume model along the direction of the laser beam.

Specifically, in the cloud detection application, the DAC algorithms may also have a little modification. Because DAC algorithms perform the task of distributed hypothesis testing, to find the global likelihood ratio, each sensor first calculates the log of the local likelihood ratio and substitutes it into the DAC algorithms as the initial local value. DAC algorithm will then find the average. The average multiplied by the number of nodes in the network is log of global likelihood ratio.

1.2 Contributions

The main contribution of this thesis is that it proposed a distributed real-time optimization method, which could run simultaneously with constant first-order DAC algorithm. Thus, the consensus algorithms will not be interrupted by the optimization. As a result, communication cost and initialization time can be dramatically reduced compared to conventional distributed optimization. In addition, a least mean square solution is obtained to mitigate the numerical error of the optimal solution calculated by the proposed method. When using floating point number in double format and the network size is smaller than 32, the numerical error after mitigation does not dramatically decline the algorithm performance.

In addition, the proposed distributed eigenvalue estimation algorithm could be an alternative of algebra connectivity estimation [8]. By the proposed method, the times of matrix iterations can be reduced to the order of $O(n)$ and the result is obtained with sufficient numerical accuracy.

Moreover, we have proposed a generalized finite-time DAC algorithm, which does not require knowledge of network topology but will estimate the necessary consensus finding filter during the iteration. Compared to the algorithm introduced in [16], it doesn't require the re-initialization of the constant first-order DAC algorithm for several times. Therefore, total data transmission in all iterations can be reduced. However, re-initialization can be introduced to increase the accuracy of the consensus value.

Finally, a modified Gaussian plume model is presented, which is the integration of the original Gaussian plume model along the laser beam to simulate the laser penetration. This modified Gaussian plume model can only describe the mean of the cloud concentration. However, a real cloud's concentration is actually a random process. To reveal the dynamics and turbulence properties of the cloud plume, a 3D cloud animation is implemented with the help of computer graphic technology [19]. A bunch of the simulated cloud plumes are generated to test the detection algorithm.

1.2.1 Publications

"Distributed Real-time Optimization of Average Consensus", Submitted to the 9th International Wireless Communications and Mobile Computing Conference (IWCMC 2013) - Wireless Sensor Networks Symposium.

1.3 Outline

This thesis is structured as follows: Chapter 2 is the review of some asymptotic and non-asymptotic DAC algorithms, as well as some distributed signal processing methods based on DAC algorithms. In Chapter 3, a generalized finite-time DAC algorithm will be presented. In Chapter 4, a distributed real-time optimization to increase the convergence rate of asymptotic DAC algorithms is proposed. A distributed detection of cloud plume using wireless sensor networks and the DAC algorithms will be introduced in Chapter 5. Finally, Chapter 6 concludes this thesis and gives out the direction of future work.

Chapter 2

Background of DAC

2.1 Preliminary

Suppose a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A})$ is a weighted digraph (or undirected graph) consisting of a set of nodes $\mathcal{V} = \{v_1, v_2, ..., v_n\}$, a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ and a *weighted adjacency matrix* $\mathcal{A} = [a_{ij}]$. The edge $e_l = (v_i, v_j) \in \mathcal{E},\ l \sim (i, j)$ is an ordered pair of distinct nodes, associated with an element in the adjacency matrix $\mathcal{A}$, i.e. $e_l = (v_i, v_j) \in \mathcal{E} \Leftrightarrow a_{ij}$. $|\mathcal{E}|$ denotes the number of edges of the graph $\mathcal{G}$. For node v_i, its communication neighbors set is denoted by $\mathcal{N}_i = \{v_j \in \mathcal{V} | (v_i, v_j) \in \mathcal{E}\}$, and $|\mathcal{N}_i|$ denotes the number of neighbours of v_i. Assume $\forall v_i \in \mathcal{V}$, $a_{ij} = 0$ if $v_j \notin \mathcal{N}_i$ (note that $a_{ii} = 0$, as $v_i \notin \mathcal{N}_i$). Node v_i can only transmit information to nodes that belong to $\mathcal{N}_i$. During the DAC iteration, the local value at new time step is a weighted sum calculated by coefficient a_{ij}, see (2.1.7).

For each edge $e_l = (v_i, v_j) \in \mathcal{E}$, we arbitrarily choose one end of e_l to be positive and another to be negative. The *incidence matrix* $Q = Q(\mathcal{G}) \in \mathbb{R}^{n \times |\mathcal{E}|}$ is defined by

$$Q = (q_{ik}) = \begin{cases} 1 & \text{if } v_i \text{ is the positive end of } e_l \\ -1 & \text{if } v_i \text{ is the negative end of } e_l \\ 0 & \text{otherwise} \end{cases} \cdot \tag{2.1.1}$$

The *Laplacian matrix* of the graph $\mathcal{G}$, defined by $L = L(\mathcal{G}) = QQ^{\mathrm{T}}$, is a useful tool to analyse the network topology [20]. In fact L is independent of the choice of positive ends of the edges. It is a symmetric, positive semidefinite, singular matrix [21].

The *Laplacian spectrum* of a graph [22], denoted by

$$S(L) = \{\lambda_1(L), \lambda_2(L), \ldots, \lambda_n(L)\} \tag{2.1.2}$$

reflects many properties of the graph, where $\lambda_i(L)$ is the eigenvalue of L (we use $\lambda_i(A)$ to denote the eigenvalue of matrix A) and we assume they are ordered non-decreasingly so that $0 = \lambda_1(L) \leq \lambda_2(L) \leq \ldots \leq \lambda_n(L)$. The second smallest eigenvalue $\lambda_2(L)$ is called algebraic connectivity of the graph.

Suppose each node holds an initial scalar local value $x_i(0) \in \mathbb{R}$, which can be a value that is locally acquired by node representing physical quantities such as temperature, humidity, illumination, attitude, etc. The network is said to have reached a consensus if and only if $x_i = x_j$, for all nodes $v_i, v_j \in \mathcal{V}, i \neq j$. In the other words, all the nodes are in an agreement of a quantity. The common value in the consensus state is called the *consensus value.*

2.1.1 Consensus Problem on Graphs

To describe the behavior of each node or agent, suppose each node has the following dynamics

$$\dot{x}_i = f(x_i, u_i), i \in \mathcal{V} \tag{2.1.3}$$

and the graph (or network) is a system having the dynamics

$$\dot{\mathbf{x}} = F(\mathbf{x}, \mathbf{u}) \tag{2.1.4}$$

where $F(\mathbf{x}, \mathbf{u})$ is the column-wise concatenation of individual dynamics $f(x_i, u_i)$, for all nodes $i = 1, \ldots, n$. In an ad-hoc network with the mobile nodes, the topology G is switching and the system will update its $F(\mathbf{x}, \mathbf{u})$ from time to time.

The input or feedback u_i to the node's dynamic is a function of the historical states of node i and its neighbors

$$u_i = g\left(x_{j_1}, x_{j_2}, \ldots, x_{j_{m_i}}\right) \tag{2.1.5}$$

where $j_1, \ldots, j_{m_i}$ are the node indexes that belong to set $\{i\} \cup \mathcal{N}_i$. (2.1.5) is called a consensus protocol under topology G. If the network graph is not fully connected, it is said to be a distributed consensus protocol.

Definition 2.1.1. Let $\mathcal{X} : R^n \rightarrow R$ be a function of n variables of $x_1, x_2, \ldots, x_n$

and let $\mathbf{x}(0)$ denote the initial condition of the network. The $\mathcal{X}$-consensus problem is a distributed method to calculate the consensus value $\mathcal{X}(\mathbf{x}(0))$ in a graph G.

Consensus problems can be different by their consensus values. For instance, we give the definition of the average consensus $\mathcal{X}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i(0)$, maximum consensus $\mathcal{X}(\mathbf{x}) = \max(\mathbf{x})$, minimum consensus $\mathcal{X}(\mathbf{x}) = \min(\mathbf{x})$ and variance consensus $\mathcal{X}(\mathbf{x}) = \mathrm{var}(\mathbf{x})$ by their expressions, respectively. The average consensus is a special case of consensus problem, which computes the average of all initial values $\overline{x} = \mathrm{mean}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i(0)$ using a distributed system dynamics $\dot{\mathbf{x}} = F(\mathbf{x}, \mathbf{u})$ in a network G.

We are interested in the distributed solutions of the consensus problem as the network only allows an node to communicate with its neighbors. We say the protocol (2.1.5) solves the consensus problem asymptotically if and only if there exists an asymptotically stable state $x^* = \mathcal{X}(\mathbf{x})$ of system dynamics (2.1.4), which satisfyies for all $\delta > 0$, there exists a time index $t^* > 0$, such that $|x_i(t) - x^*| < \delta$ for all $t > t^*$ and $\forall i \in \mathcal{V}$.

Maximum or minimum consensus is very similar to information flooding. Each node v_i in the network just compares the local values held by itself and others in $\mathcal{N}_i$, then broadcasts the maximum or minimum values. Therefore, maximum/minimum consensus can be finished in a number of steps that equal to the diameter of the graph. For upper boundary of the diameter, see [20].

Distributed average consensus is a more challenging problem than the maximum (or minimum) consensus, because the average value is a linear combination of all the initial states of network nodes and the condition $x_i^* = \mathcal{X}(\mathbf{x})$ for all i has to be satisfied. In this chapter, we will discuss the average consensus and especially its convergence rate optimization problem, which is solved distributively by discrete-time matrix iteration. Furthermore, the variance consensus problem can be solved by two instances of average consensus, because we have the relation $\mathrm{var}(\mathbf{x}) = \mathrm{mean}(\mathbf{x} \cdot \mathbf{x}) - [\mathrm{mean}(\mathbf{x})]^2$. Thus, in the following of this work we focus on the average consensus problem and the distributed average consensus (DAC) algorithms.

2.1.2 Continuous-time vs. Discrete-time DAC

In this section, we will show the difference between continuous-time and discrete-time consensus

The dynamics of the consensus protocol that solves the average consensus problem $\mathcal{X}(\mathbf{x}) = \mathrm{mean}(\mathbf{x})$ is given by

$$\dot{x}_i(t) = u_i(t) \tag{2.1.6}$$

where $u_i(t)$ is a consensus protocol

$$u_i(t) = \sum_{j \in \mathcal{N}_i} a_{ij}(x_j - x_i) \tag{2.1.7}$$

where a_{ij} is the entry of weighted adjacency matrix $\mathcal{A}$.

The continuous-time consensus requires the nodes in a network have dynamics in a form of differential equations. Continuous-time consensus involves analog signals that are easily interfered by channel noise. Consequently, the consensus value will be a random variable with mean equals to the average of all initial values and variance proportional to the signal to noise ratio. It is shown in [23], the variance is also proportional to time t hence it is increasing in the iteration.

On the other hand, discrete-time consensus only involves the quantization error during the iteration as long as the data packages are correctly received. Nowadays, sensor nodes with digital processing unit becomes more cheaper. Consequently, discrete-time consensus algorithms are mainly discussed in the following of this thesis.

2.1.2.1 Continuous-time Consensus

The continuous-time consensus with system dynamics (2.1.6) is a linear differential equation

$$\dot{\mathbf{x}}(t) = -\mathcal{L}\mathbf{x}(t) \tag{2.1.8}$$

Solving the differential equation (2.1.8) will yield a continuous-time solution in an exponential matrix form

$$\mathbf{x}(t) = \exp(-\mathcal{L}t)\,\mathbf{x}(0) \tag{2.1.9}$$

where $\mathcal{L}$ is called the *weighted graph Laplacian* associated with network graph $\mathcal{G}$, which is defined by

$$l_{ij} = \begin{cases} \sum_{k=1,k\neq i}^{n} a_{ik} & j = i \\ -a_{ij} & j \neq i \end{cases} \tag{2.1.10}$$

For a graph with 0-1 adjacency, the weighted graph Laplacian can be denoted in another form, which is unweighted Laplacian matrix denoted by L

$$l_{ij} = \begin{cases} |\mathcal{N}_i| & j = i \\ -1 & j \in \mathcal{N}_i \\ 0 & \text{otherwise} \end{cases} \tag{2.1.11}$$

In some literature [2], they use the second definition ((2.1.11)) of the Laplacian matrix to analyze the convergence rate of DAC algorithms. It is a special case when weights a_{ij} for all edges in $\mathcal{E}$ equal to one. Therefore, to distinguish them, we denote the weighted graph Laplacian matrix and the unweighted Laplacian matrix by $\mathcal{L}$ and L, respectively.

Convergence Conditions for Continuous-time Consensus There are some important theories for continuous-time consensus convergence problem. To provide the necessary and sufficient conditions of the graph Laplacian matrix so that a convergent first-order average consensus algorithm could be carried out on the network, some results are induced from Perron-Frobenius theorem [21] and Gerschgorin's theorem [21] that gives the upper and lower boundaries of the spectral radius.

Theorem 2.1.1. *[21] Let the graph be the Laplacian matrix $\mathcal{L}$, denote the maximum node out-degree of the graph by*

$$d_{max} = \max_{1 \le i \le n} \left(\sum_{j=1, j \ne i}^{n} l_{i,j} \right) \tag{2.1.12}$$

Then, all the eigenvalues of $\mathcal{L}$ are located in the following disk,

$$|z - d_{max}| \le d_{max} \tag{2.1.13}$$

which is centered at $z = d_{max} + 0j$ on the complex plane.

Proof. [21] Based on the Gerschgorin's theorem, all the eigenvalues of $\mathcal{L}$ are located in the union of the disks.

$$|z - l_{i,i}| \le \sum_{j=1, j \ne i}^{n} |l_{i,j}|, \ 1 \le i \le n. \tag{2.1.14}$$

Since $\mathcal{A} = [a_{i,j}]$ is a non-negative matrix, by the definition of Laplacian matrix,

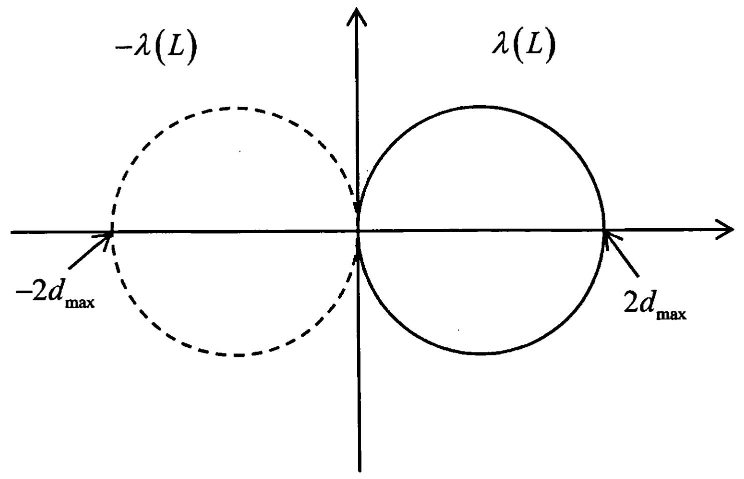


Figure 2.1: Boundary of eigenvalues of Laplacian matrix

$l_{i,j} \leq 0$ and $l_{i,i} = \sum_{k=1,k\neq i}^{n} a_{ik} \geq 0$. Therefore, let $d_i = l_{i,i}$ and

$$d_i = \sum_{j=1,j\neq i}^{n} |l_{i,j}| \tag{2.1.15}$$

the union of disks becomes

$$\bigcup_{1\leq i\leq n} \{z \in \mathbb{C} : |z - d_i| \leq d_i\} \tag{2.1.16}$$

On the other hand, all these disks are located inside the largest disk with radius d_{max}. This result ends the proof of the theorem □

Based on the Theorem 2.1.1, it is obvious all the nonzero eigenvalues of $\mathcal{L}$ have positive real parts. This immediately leads to a convergence theorem of the continuous-time consensus protocol (2.1.7). Since all the nonzero eigenvalues of $-\mathcal{L}$ located in the disk $|z + d_{max}| \leq d_{max}$, and the eigenspace associated with zero is one-dimensional, the eigenvector associated with zero eigenvalue has the form $\alpha\mathbf{1}$, i.e. $x_i = \alpha$ for all i. This result will be very helpful as the negative real part can guarantee that the system dynamic is stable and convergent, see Figure 2.1.

We will prove that the solution given in exponential matrices form (2.1.9) converges to a consensus value as $t \to \infty$ in the next.

Considering the system dynamic $x(t) = \exp(-\mathcal{L}t)\, x(0)$. Because $\exp(-\mathcal{L}t)$

is a non-negative matrix, the Perron-Frobenius theorem states that $\exp(-\mathcal{L}t)$ has a positive real eigenvalue equaling one which is also the spectral radius. Together with the Theorem 2.1.1 which implies that all the eigenvalues of $-\mathcal{L}$ have negative real part, see Figure 2.1, we immediately come to the following theorem.

Theorem 2.1.2. *[24] Assume $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A})$ is strongly a connected graph, and the graph weighted Laplacian matrix $\mathcal{L}$ is defined in (2.1.10), which has only one zero eigenvalue. Let u_r is the uniformed right eigenvector and u_l be the uniformed left eigenvector associated with the zero eigenvalue of $\mathcal{L}$, i.e. $\mathcal{L}u_r = 0$, $u_l^T\mathcal{L} = 0$. We have $u_l^T u_r = 1$ and the system dynamic*

$$x(t) = \exp(-\mathcal{L}t)\, x(0) \tag{2.1.17}$$

will have the stable state given by

$$x(t) = Kx(0) \tag{2.1.18}$$

where K is a matrix in R^n, and $K = \lim_{t\to\infty} \exp(-\mathcal{L}t) = u_r u_l^T$.

Proof. [24] Let $A = -\mathcal{L}$, and it has a Jordan form of $A = UJU^{-1}$. Then we can have $\exp(At) = U \exp(Jt)\, U^{-1}$. Because A has all its eigenvalues except a simple zero eigenvalue has negative real part, thus, as $t \to \infty$, all other Jordan block vanish, and $\exp(Jt)$ converges to a matrix with only single nonzero entry, denoted by Q. Since matrix U contains a column which associated with the zero eigenvalue of A is u_r, similarly, U^{-1} has a corresponding row equaling to u_l. By simply calculating the equation $K = UQU^{-1}$, we can show that $K = u_r u_l^T$. And the fact $U^{-1}U = I$ shows that $u_l^T u_r = 1$. This ends the proof. □

For the average consensus problem, it is obvious that all the elements in K must equal to $\frac{1}{n}$. This requires the graph Laplacian $\mathcal{L}$ satisfies the conditions: $\mathcal{L}\mathbf{1} = 0$, $\mathbf{1}^T\mathcal{L} = 0$, where $\mathbf{1} \in R^n$ is an all unity vector. And the u_r and u_l will change into vectors with equivalent constant in all its components. If they are uniformed, then $u_r = u_l = \frac{1}{\sqrt{n}}$.

2.1.2.2 Discrete-time Consensus

The discrete-time consensus with the dynamics (2.1.6) is

$$x_i(k+1) = x_i(k) + u_i(k) \tag{2.1.19}$$

For network agents having discrete-time consensus protocol, their system dynamics can be given in a matrix form

$$\mathbf{x}(k+1) = W\mathbf{x}(k) \tag{2.1.20}$$

where $W = [w_{ij}] = I - \mathcal{L}$. We say the iteration is convergent if there exists a vector denoted by $\mathbf{x}^*$, which satisfies

$$\mathbf{x}^* = W\mathbf{x}^* \tag{2.1.21}$$

Moreover,

$$\mathbf{x}(k+1) - \mathbf{x}^* = W\left[\mathbf{x}(k) - \mathbf{x}^*\right] \tag{2.1.22}$$

(2.1.21) states that $\mathbf{x}^*$ is a right eigenvector of matrix W associated with a simple eigenvalue 1. For convergence conditions and more details about discrete-time first-order DAC algorithm, see Section 2.3.1.

2.2 Categories of Distributed Consensus Average Algorithms

To introduce the DAC algorithmss, let's first take a look at the Figure 2.2. (Figure 2.2 only shows the family of discrete-time distributed consensus algorithms. For more details about continuous-time consensus algorithms, see Section 2.1.2). We may start from one of the simplest in their family: the first-order DAC algorithm introduced in Section 2.3.1. Its convergence rate is related to the spectral radius of a graph dependent matrix. So the optimization problem is to find the optimal matrix with minimum sub-dominant eigenvalue. However, global information of the graph matrix must be available. In distributed methods, this is a quite demanding condition. Without the global information, constant first-order DAC and metropolitan DAC can be the sub-optimal solution to the consensus problem [15]. The first-order DAC algorithm together with higher-order DAC algorithms (introduced in Section 2.3.2)belong to the family of the asymptotic algorithms. The motivation to develop the higher-order algorithms are the demand of a fast convergent rate. The higher-order DAC algorithms could have faster convergence rate, and no additional requirement and cost of communication is required compared with first-order DAC [1]. Therefore, higher-order DAC algorithms have applied to practical consensus protocols [25].

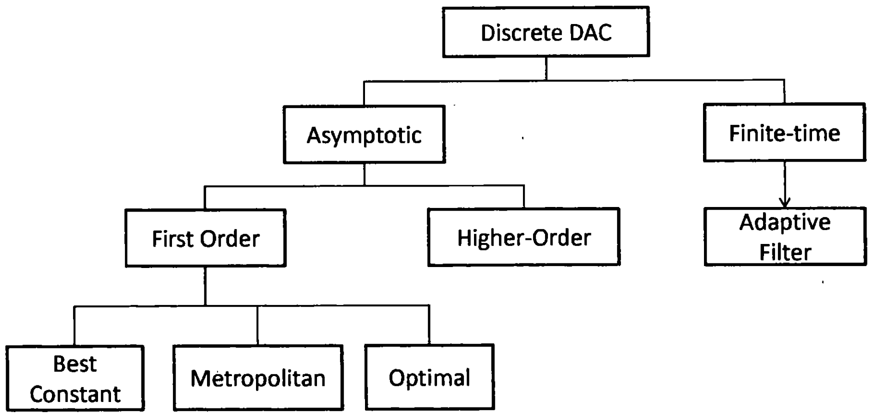


Figure 2.2: Categories of discrete-time consensus algorithms

Some of the novel methods can solve the average consensus problem in finite number of iterations. These methods are referred to as finite-time consensus algorithm [16]. It is actually a very sophisticated signal processing technique that finds the asymptotically stable equilibrium $\bar{x}$ by extrapolation, see Section 2.4. Given the sequence of local values obtained by the first-order DAC, [16] verifies that there exists a filter that can estimate the consensus value. Based on [16]'s work, [17] proposed an adaptive filter algorithm to asymptotically converge the estimated coefficients to the correct coefficients of the filter [17]. As a contribution of this thesis, a method to calculate the filter directly by inverting a Toeplitz matrix is introduced in Chapter 3.

2.3 Asymptotic Distributed Consensus Algorithms

The distributed averaging consensus problem can be solved not only by DAC algorithms, but also in many other ways, such as flooding, gossip and so on. In the flooding algorithms, each sensor maintains a table of local values which initialize by its local initial value. At each iteration of the flooding algorithm, every node exchanges the table with their neighbors. After enough steps, which is larger than the diameter of the network, every node will get all the initial values of all sensors. The gossip algorithm is asynchronous. Only one node that is selected by a random schedule wakes up and chooses another node randomly. These two nodes exchange their local values and change their local values to the average.

Due to the simplicity and robustness of asymptotic distributed consensus algorithms, it still plays an important role in the practice. They are already applied to network with a large number of nodes and proved to be robust against the topology variation [11]. In this section, we start from the first-order DAC algorithm and then expand to higher-order algorithms in order to yields a higher convergence rate.

2.3.1 Discrete First Order Distributed Consensus Algorithm

For the network $\mathcal{G}$, the first-order DAC (FO-DAC) algorithm obtains the average by the iterations given by

$$\begin{aligned} x_i(k+1) &= x_i(k) + \sum_{j\in\mathcal{N}_i} w_{ij}\left[x_j(k) - x_i(k)\right] && (2.3.1) \\ &= w_{ii}x_i(k) + \sum_{j\in\mathcal{N}_i} w_{ij}x_j(k) && (2.3.2) \end{aligned}$$

where $k = 0, 1, 2, ...$ is the time index, w_{ij} is the weight to x_j at node i, $w_{ij} = a_{ij}$, $i \neq j$ is the weight to x_j at node v_i. The local value at time $k+1$ is a weighted sum of node's local values one time k. The weight to v_i itself is $w_{ii} = 1 - \sum_{j\in\mathcal{N}_i} a_{ij}$, so that the sum of all weights equals to one, $\sum_{j=1}^{n} w_{ij} = 1$.

The iteration (2.3.1) can be written in a vector form

$$\mathbf{x}(k+1) = W\mathbf{x}(k) \qquad (2.3.3)$$

where $\mathbf{x}(k) = [x_1(k), x_2(k), \dots, x_n(k)]^T \in \mathbb{R}^n$ and $W = [w_{ij}]$ is the *weight matrix* or the Perron matrix induced by $\mathcal{G}$ [24]. We define the *initial local value vector*

$\mathbf{x}(0) \in \mathbb{R}^n$ to represent all the initial values on the network. The iteration implies that $\mathbf{x}(k) = W^k \mathbf{x}(0)$ for $k = 1, 2, \cdots$.

Let $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$ be the eigenvectors of W and $\lambda_1(W), \lambda_2(W), \ldots, \lambda_n(W)$ be the associated eigenvalues and they are ordered so that $|\lambda_1(W)| \geq |\lambda_2(W)| \geq \ldots \geq |\lambda_n(W)|$. The largest eigenvalue $\lambda_1(W)$ is called the *dominant eigenvalue* and $\mathbf{e}_1$ is called the *dominant eigenvector.* Besides, $\lambda_2(W)$ and $\mathbf{e}_2$ are called the *sub-dominant eigenvalue* and *sub-dominant eigenvector.*

Since the initial value vector is randomly chosen, the matrix W must satisfy some conditions to make sure the iteration is convergent. The problem is how to choose W so that $\forall i,\ x_i(k) \to \bar{x}$, as $k \to \infty$. Actually the convergence rate depends on the spectral radius of matrix W. The following section gives the sufficient and necessary conditions of W to achieve the average consensus.

2.3.1.1 Convergence Conditions

For the matrix iteration defined in (2.3.3), the average consensus problem is to choose the weight matrix W, so that for any initial value $\mathbf{x}(0) \in R^n$, $\mathbf{x}(k)$ converges to the vector

$$\bar{\mathbf{x}} = \left(\mathbf{1}^{\mathrm{T}}\mathbf{x}(0)/n\right)\mathbf{1} = \frac{\mathbf{1}\mathbf{1}^{\mathrm{T}}}{n}\mathbf{x}(0) \tag{2.3.4}$$

Theorem 2.3.1. *[15]For a network $\mathcal{G}$ and the associated weight matrix $W = [w_{ij}]$, whose entries $w_{ij} = 0$, if $(v_i, v_j) \notin \mathcal{E}$, $\lim_{k\to\infty} \mathbf{x}(k) = \bar{\mathbf{x}}$ if and only if matrix W satisfies*

$$\lim_{k\to\infty} W^k = \frac{\mathbf{1}\mathbf{1}^{\mathrm{T}}}{n} \tag{2.3.5}$$

which holds if and only if

$$\begin{aligned} \mathbf{1}^{\mathrm{T}}W &= \mathbf{1}^{\mathrm{T}} && (2.3.6)\\ W\mathbf{1} &= \mathbf{1} && (2.3.7)\\ \rho\left(W - \mathbf{1}\mathbf{1}^{\mathrm{T}}/n\right) &< 1 && (2.3.8) \end{aligned}$$

where vector $\mathbf{1} = [1, 1, \cdots 1,]^{\mathrm{T}} \in R^n$, $\rho(\cdot)$ denotes the spectral radius of the matrix.

(2.3.6) means that W has a left eigenvector $\mathbf{1}$ associated with the eigenvalue 1. This implies that the sum of local value vector is not changed in the iteration $\sum_{i\in\mathcal{V}} x_i(k+1) = \sum_{i\in\mathcal{V}} x_i(k)$ and the sum of each column of matrix W is equal to one. (2.3.7) shows that W is a row stochastic matrix and has an eigenvalue 1 with associated eigenvector $\mathbf{1}$. Both (2.3.6) and (2.3.7) together with the convergence condition (2.3.8) mean that W has a simple eigenvalue that equals to one, and modular of all other eigenvalues are less than one.

Lemma 2.3.1. If $\lim_{k\to\infty} W^k = \dfrac{\mathbf{1}\mathbf{1}^{\mathrm{T}}}{n}$, matrix $W - \mathbf{1}\mathbf{1}^{\mathrm{T}}/n$ shares the same eigenvalues as matrix W except the simple eigenvalue one is replaced by zero.

Proof. (2.3.6) implies that the matrix W has a left eigenvector $\mathbf{1}$ associated with eigenvalue one. (2.3.7) implies that $\mathbf{1}$ is a right eigenvector of W associated with eigenvalue one. The fact that $\lim_{k\to\infty} W^k = \dfrac{\mathbf{1}\mathbf{1}^{\mathrm{T}}}{n}$ exists if and only if there exists a matrix U and W can be Jordan decomposition as

$$W = U \begin{bmatrix} J_1 & & & \\ & J_2 & & \\ & & \ddots & \\ & & & J_m \end{bmatrix} U^{-1} \tag{2.3.9}$$

where m is the number of distinct Jordan block. J_i is the r_i dimensional Jordan block corresponding to eigenvalue λ_i, $J_1 = I_{r_1}$ is the r_i dimensional identity matrix $(0 \le r_i \le n)$ and all other Jordan blocks are convergent, i.e. $\rho(J_i) < 1, 2 \le i \le m$.

Let $u_1, u_2, \dots, u_n$ be the column of U and $v_1^T, v_2^T, \dots, v_n^T$ be row of U^{-1}. Then we have

$$\lim_{k\to\infty} W^k = U \begin{bmatrix} I_{r_1} & 0 \\ 0 & 0 \end{bmatrix} U^{-1} \tag{2.3.10}$$

$$= \sum_{i=1}^{r_1} u_i v_i^T = \frac{\mathbf{1}\mathbf{1}^{\mathrm{T}}}{n} \tag{2.3.11}$$

As the property of unitary matrix U, both u_i and v_i are sets of independent vectors, each $u_i v_i^T$ is a matrix with rank one and matrix $\sum_{i=1}^{n} u_i v_i^T$ has rank n. The sum $\sum_{i=1}^{r_1} u_i v_i^T$ must have rank r_1. Eq. (2.3.11) shows that r_1 must equal to one and $u_i v_i^T = \dfrac{\mathbf{1}\mathbf{1}^{\mathrm{T}}}{n}$, both u_i and v_i are vectors with the same constant on all components. Therefore, $W - \dfrac{\mathbf{1}\mathbf{1}^{\mathrm{T}}}{n}$ has the same Jordan decomposition as W except the Jordan block J_1 is replaced by zero, and all other Jordan blocks remain the same. This completes the proof. □

The convergence rate of the FO-DAC is related to the spectral radius of matrix $W - \dfrac{\mathbf{1}\mathbf{1}^{\mathrm{T}}}{n}$. To get the maximum convergence rate, an optimization problem to minimize the spectral radius of the matrix could be solved [15].

$$\begin{array}{ll} \text{Minimize} & \rho\left(W - \frac{\mathbf{1}\mathbf{1}^T}{n}\right) \\ \text{Subject to} & \mathbf{1}^{\mathrm{T}}W = \mathbf{1}^{\mathrm{T}},\ W\mathbf{1} = \mathbf{1} \end{array}, \tag{2.3.12}$$

But it requires knowledge of network topology to solve the problem. However, some non-optimal convergent weight matrices that satisfy the condition in (2.3.5) are given below.

2.3.1.2 Optimization of FO-DAC

Theorem 2.3.1 means one is a simple and dominant eigenvalue of W and the optimization problem is

$$\min \rho\left(W - \mathbf{1}\mathbf{1}^{\mathrm{T}}/n\right) \tag{2.3.13}$$

under the constraints in Theorem 2.3.1, where $W - \mathbf{1}\mathbf{1}^{\mathrm{T}}/n$ is a deflated matrix [26].

2.3.1.3 Constant First-order DAC

Find the optimal weight matrix for FO-DAC is not a simple task. However, a heuristic way is to set all edges symmetric and their weights equal to the constant ϵ. Thus, the weight matrix is symmetric. A special weight to a node itself is chosen to be $1 - \epsilon\,|\mathcal{N}_i|$. Therefore, the weight matrix can be defined by the Laplacian matrix

$$W = I_n - \epsilon L \tag{2.3.14}$$

where the constant ϵ is the *step length.* This kind of DAC algorithm is called the *constant FO-DAC algorithm* (CFO-DAC).

Choosing the step length From (2.3.14), the eigenvalues of W and L have relationship given by $\lambda_i(W) = 1 - \epsilon\lambda_i(L)$. Thus, we can determine the convergence range of ϵ in terms of $\lambda_i(L)$.

The constant FO-DAC algorithm is convergent, if and only if the step length ϵ is in the range $(0, 2/\lambda_n(L))$. The best constant to minimize $\rho\left(W - \mathbf{1}\mathbf{1}^{\mathrm{T}}/n\right)$ is $\epsilon_{opt,FO} = \frac{2}{\lambda_2(L)+\lambda_n(L)}$ [15].

By Gerschgorin's theorem, we have another upper boundary of $\lambda_n(L) \leq 2d_{max}$, then convergence is guaranteed if $\epsilon \in (0, 1/d_{max}]$. This maximum degree step length is usually a practical solution but very slowly convergent compared with the optimal step length.

2.3.1.4 Local Degree First Order DAC

Another option of FO-DAC is Metropolis FO-DAC (or local degree FO-DAC), where the weight matrix is determined by degree of the nodes,

$$w_{ij} = \begin{cases} \frac{1}{1+\max\{d(i),d(j)\}}, & \text{if } (i,j) \in \mathcal{E} \\ 1 - \sum_{(i,k)\in\mathcal{E}} w_{ik}, & i = j \\ 0, & \text{otherwise} \end{cases} \qquad (2.3.15)$$

where $d(i)$ and $d(j)$ are the degrees of nodes i and j.

This weight matrix would be very easy to implemented in practical, because nodes in the distributed network doesn't need to have any knowledge of the whole network topology.

2.3.2 Discrete High Order Distributed Consensus Algorithms

Higher-order DAC (HO-DAC) algorithms can be easily implemented by storing past data, which is potentially faster than FO-DCA and doesn't require additional communications. However, compared to the third-order DAC algorithm, fourth order algorithm can only achieve a negligible improvement [1]. Thus, it is not necessary to extend the order of DAC algorithms to larger than fourth.

In addition, HO-DAC can be regarded as a generalized form of DAC algorithm. When the forgetting factor is set to zero, the HO-DAC algorithms can be reduced to the CFO-DAC. Moreover, when the order is set to 2, it is reduced to second-order DAC (SO-DAC) algorithm.

In a time-invariant network $\mathcal{G}$, the M^{th} higher-order DAC algorithms have the form

$$\begin{aligned} x_i(k) &= x_i(k-1) - \epsilon \sum_{m=1}^{M} c_m(-\gamma)^{m-1}\Delta x_i(k,m) && (2.3.16) \\ \Delta x_i(k,m) &= \sum_{j\in\mathcal{N}_i} \left(x_i\left(k-m\right) - x_j\left(k-m\right)\right) && (2.3.17) \end{aligned}$$

where ϵ is a constant *step length*, M is the order of the HO-DAC, c_m is a predefined constant, $c_1 = c_2 = c_3 = 1$, $c_4 = \frac{1}{6}$ and $c_m \neq 0$ $(m > 4)$, γ is a *forgetting factor* and $|\gamma| < 1$.

After introducing the Laplacian matrix L, HO-DAC can be written as

$$\mathbf{x}(k) = (I_n - \epsilon L)\mathbf{x}(k-1) - \epsilon \sum_{m=2}^{M} c_m(-\gamma)^{m-1} L\mathbf{x}(k-m) \qquad (2.3.18)$$

assume $\forall k < 0$, $\mathbf{x}(k) = \mathbf{x}(0)$, $\mathbf{x}(0)$ is the initial local state information for node i.

2.3.2.1 Optimization Problem of HO-DAC

In the convergence analysis of higher-order DAC, we need to rewrite iteration (2.3.18) into matrix form. Therefore, we define the higher-order weight matrix $H \in \mathbb{R}^{Mn \times Mn}$

$$H = \begin{bmatrix} I_n - \epsilon L & c_2\gamma\epsilon L & \cdots & -c_M(-\gamma)^{M-1}\epsilon L \\ I_n & \mathbf{0}_{n\times n} & \cdots & \mathbf{0}_{n\times n} \\ \vdots & \ddots & & \vdots \\ \mathbf{0}_{n\times n} & \cdots & I_n & \mathbf{0}_{n\times n} \end{bmatrix} \tag{2.3.19}$$

and the matrix $J \in \mathbb{R}^{Mn \times Mn}$ to deflate the matrix H

$$J = \begin{bmatrix} \mathbf{K} & \mathbf{0}_{n\times n} & \cdots & \mathbf{0}_{n\times n} \\ \mathbf{K} & \mathbf{0}_{n\times n} & \cdots & \mathbf{0}_{n\times n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{K} & \mathbf{0}_{n\times n} & \cdots & \mathbf{0}_{n\times n} \end{bmatrix} \tag{2.3.20}$$

where $\mathbf{K} = \left(\frac{1}{n}\right)\mathbf{1}\mathbf{1}^T$, and $\mathbf{0}_{n\times n}$ denotes the $n \times n$ all-zero matrix.

Note that the deflated matrix $H - J$ has the same eigenvalue as H, except the eigenvalue $\lambda_1(H) = 1$ is replaced by zero. Therefore, the high-order DAC optimization problem can be formulated by

$$\begin{aligned} \text{Minimize} \quad & \rho(H - J) \\ \text{Subject to} \quad & \epsilon, \gamma \in R \end{aligned} \tag{2.3.21}$$

The high-order DAC algorithms with initial condition $\mathbf{x}(-M+1) =, \ldots, = \mathbf{x}(-1) = \mathbf{x}(0)$ is convergent if and only if $\rho(H - J) < 1$ [1].

2.3.2.2 Solve the Optimization Problem of HO-DAC

As shown in (2.3.21), the convergence rate maximization for high order DAC algorithms can be cast into a spectral radius minimization problem.

The spectral radius is related to eigenvalues of Laplacian matrix. In the optimization problem of HO-DAC, eigenvalues of the associated graph Laplacian matrix or weight matrix must be known.

Finding the solution is not easy because we have to solve the high-order polynomial to calculate eigenvalues.

For example, the high-order polynomial of the eigenvalues of $H - J$ when $M = 3$, $c_1 = 1$ and $c_2 = 1$ is

$$f(\lambda) = \lambda^3 - (1 - \epsilon\lambda_i(L))\lambda^2 - \gamma\epsilon\lambda_i(L)\lambda + \gamma^2\epsilon\lambda_i(L) = 0 \qquad (2.3.22)$$

where $\lambda_i(L)$ is the i^{th} smallest eigenvalue of the Laplacian Matrix [1].

Since each $\lambda_i(L)$, $i = 1, 2, \cdots, n$ with generate one equation which has M roots, there are totally $M \times n$ eigenvalues for $H - J$. Thus, (2.3.21) can be written into minimize the maximum absolute value of all the eigenvalues of $H - J$.

$$\begin{array}{ll} \text{Minimize} & \max\{|\lambda_i(H - J)|\},\ i = 1, 2, \cdots, n \\ \text{Subject to} & \epsilon, \gamma \in R \end{array}, \qquad (2.3.23)$$

The convergence rate optimization problem (2.3.23) of HO-DAC in undirected network can be graphically illustrated by Figure 2.3, where the surface of $\max\{|\lambda_i(H - J)|\}$, $i = 1, 2, \cdots, n$ is plotted and the optimal solution $(\epsilon_{opt}, \gamma_{opt})$ is marked with a star. We denote the optimized spectral radius by $\rho_{opt} = \min\rho\{(H - J)\}$.

Since the eigenvalues of Laplacian matrix are network topology dependent, if $M > 2$, problem (2.3.23) is more challenging as it requires the whole Laplacian spectrum to solve a high-order polynomial to obtain $\rho(H - J)$ and find the global minimum on the surface of $\rho(H - J)$ [1]. The problem (2.3.23) may not have a unique analytical solution. In this case, the problem will be solved numerically.

2.3.2.3 Solutions for Second-order DAC Optimization

If $M = 2$, HO-DAC reduces to the second-order DAC (SO-DAC) algorithm. In this case, the problem (2.3.23) does have an analytical solution [2]. The convergence region for second-order DAC in undirected network which satisfies $\rho(H - J) < 1$ is $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, where

$$\mathcal{R}_1 = \left\{ \frac{-1}{\epsilon\lambda_n(L)} < \gamma < 1, 0 < \epsilon < \frac{1}{\lambda_n(L)} \right\} \qquad (2.3.24)$$

$$\mathcal{R}_2 = \left\{ \frac{-1}{\epsilon\lambda_n(L)} < \gamma < \frac{2}{\epsilon\lambda_n(L)} - 1, \frac{1}{\lambda_n(L)} \leq \epsilon < \frac{3}{\lambda_n(L)} \right\} \qquad (2.3.25)$$

Figure 2.4 graphically illustrates these region $\mathcal{R}_1$ and $\mathcal{R}_2$. The eigenvalues of **H** corresponding to eigenvalue of $\lambda_i(L)$ is denoted by $\lambda_{i'}(H)$ and $\lambda_{i''}(H)$, which

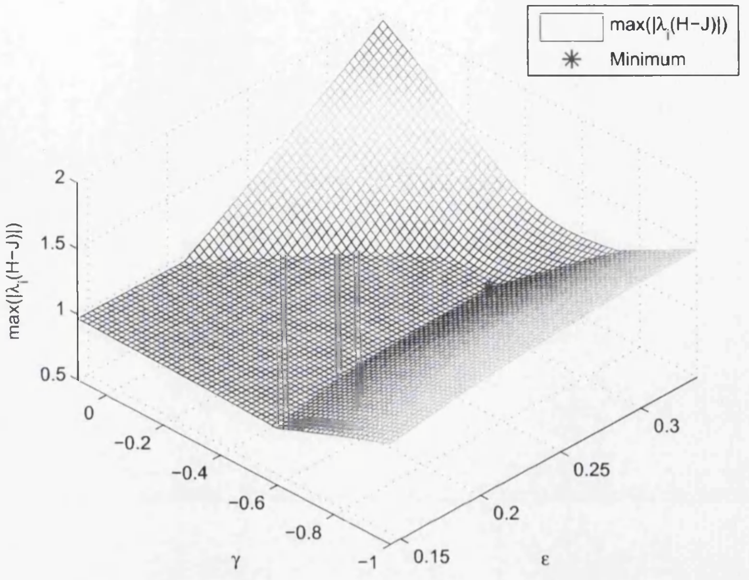


Figure 2.3: Illustration of convergence rate optimization [1].

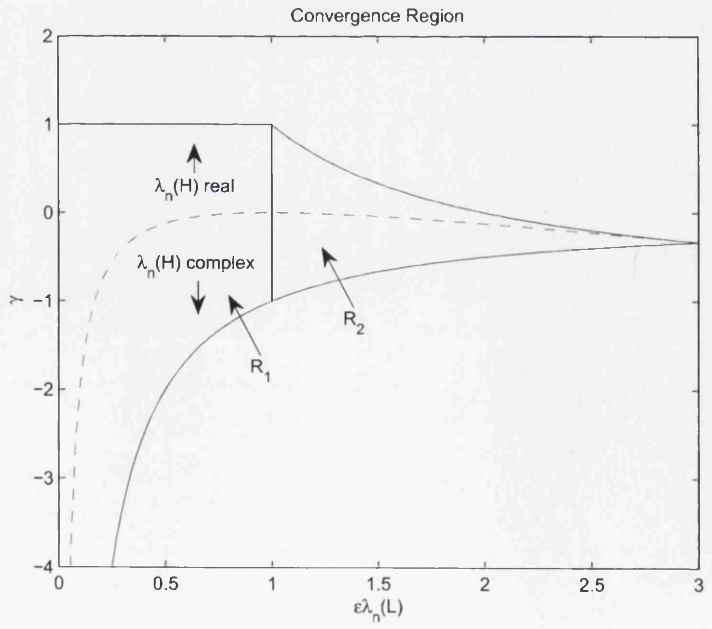


Figure 2.4: Convergence region for second-order DAC [2].

are

$$\begin{aligned}\lambda_{i'}(H) &= \frac{1}{2}\left[1-\epsilon\lambda_i(L)+\sqrt{(1-\epsilon\lambda_i(L))^2+4\gamma\epsilon\lambda_i(L)}\right],\\ \lambda_{i''}(H) &= \frac{1}{2}\left[1-\epsilon\lambda_i(L)-\sqrt{(1-\epsilon\lambda_i(L))^2+4\gamma\epsilon\lambda_i(L)}\right].\end{aligned} \tag{2.3.26}$$

Since $\lambda_2(L) \leq \cdots \leq \lambda_n(L)$, in the convergence region of the second-order DAC algorithm, the optimization problem (2.3.23) can be equivalent to

$$\begin{array}{ll}\text{Minimize} & \max\{|\lambda_{2'}(H)|, |\lambda_{2''}(H)|, |\lambda_{n'}(H)|, |\lambda_{n''}(H)|\} \\ \text{Subject to} & \epsilon, \gamma \in R\end{array}, \tag{2.3.27}$$

Finding the optimal solution needs consideration of different combinations of $\lambda_{2'}(H), \lambda_{2''}(H), \lambda_{n'}(H), \lambda_{n''}(H)$ when they are real value or complex value. However, for a connected network and in the convergence region, $\lambda_{2'}(H), \lambda_{2''}(H)$ are real and $\lambda_{n'}(H), \lambda_{n''}(H)$ are complex values. The minimum is achieved when the following equation satisfies

$$|\lambda_{2'}(H)| = |\lambda_{n'}(H)| = |\lambda_{n''}(H)| \tag{2.3.28}$$

Thus, by solving this equation, we have the optimal solution for second-order DAC algorithm.

$$\epsilon_{opt,SO} = \frac{3\lambda_n(L)+\lambda_2(L)}{\lambda_n(L)\left[\lambda_n(L)+3\lambda_2(L)\right]} \tag{2.3.29}$$

$$\gamma_{opt,SO} = -\frac{\left[\lambda_n(L)-\lambda_2(L)\right]^2}{\left[\lambda_n(L)+3\lambda_2(L)\right]\left[3\lambda_n(L)+\lambda_2(L)\right]} \tag{2.3.30}$$

which only requires the second smallest and the largest eigenvalues of L. These parameters could be flooded to all nodes before the algorithm starts.

Besides these results, it needs to point out that the optimal solutions $\epsilon_{opt,SO}$ and $\gamma_{opt,SO}$ have the following relationship

$$\gamma_{opt,SO} = \frac{\left[1-\epsilon_{opt,SO}\lambda_n(L)\right]^2}{-4\epsilon_{opt,SO}\lambda_n(L)} \tag{2.3.31}$$

which implies the discriminant under the radical sign in (2.3.26) is equal to zero and $\lambda_{n'}(H), \lambda_{n''}(H)$ are all real values. Note the dashed curve in Figure 2.4 separates the regions where $\lambda_{n'}(H)$ and $\lambda_{n''}(H)$ are real and complex. The optimal solution is always located on this curve.

2.3.3 Simulation and Algorithms Performance

To test the performance of high-order DAC algorithms with different orders, 1000 random networks are generated and a simulation is carry out on each network. The algorithm's performance is evaluated by the average spectral radius and average mean square error, shown in Figure 2.6.

2.3.3.1 Random Network Generation

The following is a method of network generation when the wireless sensor nodes are distributed. The methods of network generation and communication ranges are illustrated in Figure 2.5b.

First, a certain number of nodes are randomly and uniformly distributed in a unit square. Second, connect any two nodes if they satisfy certain communication constraints. Finally, each sensor chooses a random local initial value that is uniformly distributed in a certain range.

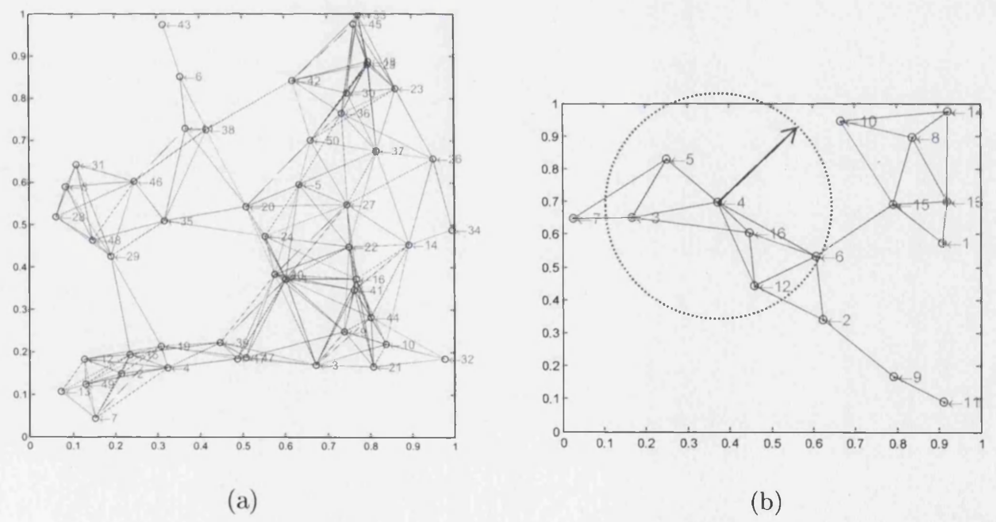


Figure 2.5: Randomly generated networks (a) 50 nodes and 200 edges (b) 16 nodes and radius constraint $R = 0.3$

There are two cases to generate the links between nodes.

Case 1. Consider the graph show in Figure 2.5a, which has 50 nodes and 200 edges. The number of nodes and edges are fixed; 50 nodes are randomly and uniformly distributed in the unit square; Any two nodes are connected if their edge is in the list of 200 shortest edges.

Case 2. The network in Figure 2.5b is generated as follows: nodes are randomly and uniformly distribute in the unit square ; connect any two nodes if their distance is less than the communication radius constraint R.

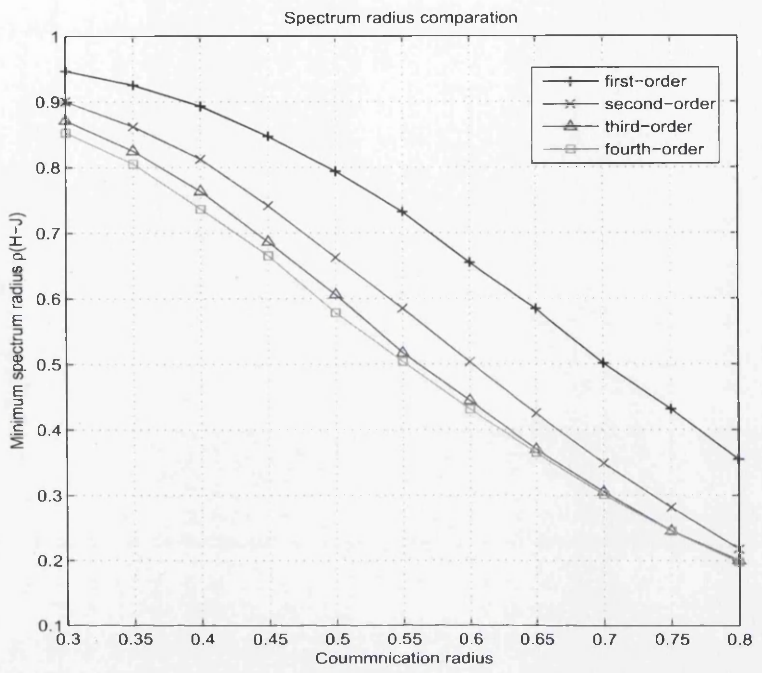
2.3.3.2 Performance Comparison for Asymptotic DAC

The performance of DAC algorithms with different orders are compared by the optimal spectral radius $\rho_{opt} = \min \rho \{(\mathbf{H} - \mathbf{J})\}$. The spectral radius has the relationship with convergence rate given by $r_{opt} = -log\,(\rho_{opt})$. Figure 2.6a shows the optimal spectral radius for DAC algorithms with different communication radius constraints. Y-axis is corresponding to the minimum spectral radius and x-axis is corresponding to the radius constraint. For each instance of DAC algorithm, the minimum on the surface $\max\{|\lambda_i\,(\mathbf{H} - \mathbf{J})|\}$ obtained by numerical searching. Each curve is the average obtained by 1000 instances of DAC initialized with random local value vectors and random networks.

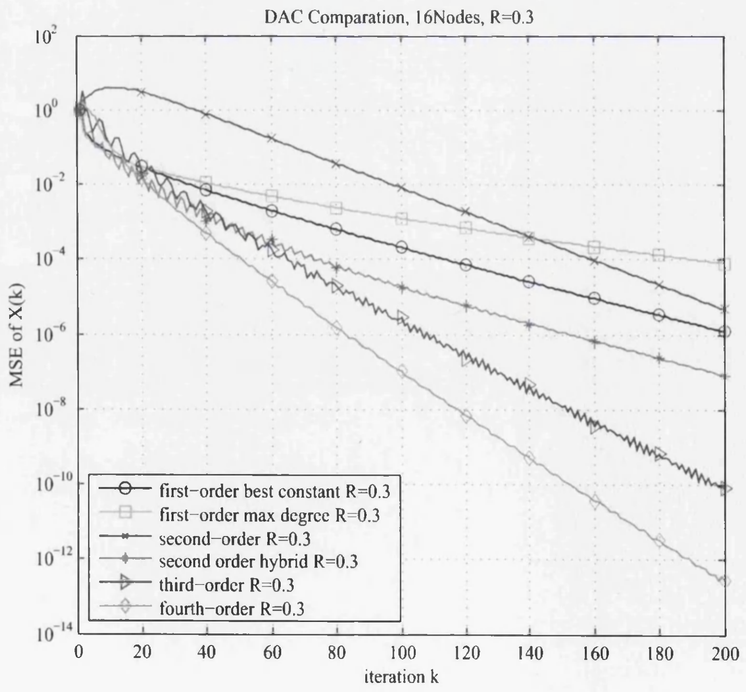
In another point of view, Figure 2.6b plots the convergence behavior of local value vector of high-order DAC algorithms together with first-order DAC in terms of mean square error (MSE) on random network. The MSE is defined by

$$MSE(k) = \frac{1}{n} \sum_{i=1}^{n} |x_i(k) - \bar{x}|^2 \tag{2.3.32}$$

which actually is the Euclidean distance between current local vector $\mathbf{x}_i(k)$ and the global average. By observing the gradient of curves, it is apparent that higher-order DAC algorithmss have larger convergence rate. However, there are negligible improvement for the fourth order DAC compared to the third order one. Furthermore, high-order DAC algorithms have a MSE overshoot at the beginning. This phenomenon happens especially when the communication radius is small. Second-order DAC algorithm does have a faster convergence rate than first order algorithm as the slop of the curve is steeper. But it becomes worse as it converges to error tolerance 10^{-6} more later than the first order algorithm. A hybrid algorithm is proposed to overcome this disadvantage. Its step size is equal to the first-order DAC step size and forgetting factor is equal to second-order DAC forgetting factor.



(a)



(b)

Figure 2.6: Comparison of DAC algorithms (a) compared by spectral radius (b) compared by MSE over time.

2.4 Finite-Time DAC Algorithm

Finite-Time Distributed Consensus Algorithm (FT-DAC) is an non-asymptotic algorithm. It tries to find the consensus value after the FO-DAC algorithm is iterated for a certain number of times. This is based on the assumption that after a certain number of iterations, local values would contain sufficient information to estimate the consensus value.

Definition 2.4.1. Given the network $\mathcal{G}$ and initial local value vector $\mathbf{x}(0)$, it is said the algorithm achieves a finite-time consensus if it solves a consensus problem, and there exists a time t^* and the consensus value $\bar{x}$ such that $x_i(t) = \bar{x}$, for all time $t > t^*$.

Motivated by the works in [27], we try to decompose the local value vector into a form which reveals an important property of its iteration. Therefore we propose a filtering technique to estimate the consensus value.

2.4.1 Find the Consensus Value by Linear Filter

In the network that adopts first-order DAC algorithm, each node has a number of consecutive local values obtained after serveral iterations. There exists a linear filter, if each node passes its consecutive local values to this filter, the output after is the global average of the initial values over the network.

Definition 2.4.2. The linear filter $\mathbf{h} \in R^d$, whose output $\hat{x}_i(k) = \mathbf{h} * x_i(k)$ is the consensus value $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i(0)$ of the network, is called the *consensus finding filter,* where local values $x_i(k)$ are obtained by (2.3.1), $k > d$.

Theorem 2.4.1. *[27] There exists a linear filter defined by $\mathbf{h} \in R^d$, such that by passing through local values $x_i(k)$* obtained by (2.3.1), *the output $\hat{x}_i(k) = \mathbf{h} * x_i(k)$ is the consensus value $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i(0)$ of the network after k step $(k \geqslant m)$, where m* is the number of *distinct and nonzero eigenvalues of the weight matrix W and W satisfies the convergence condition* (2.3.5).

Proof. To avoid complicated analysis, suppose an undirected network with n nodes, where duplex communication is possible in each link. Therefore, the associated weight matrix $W \in \mathbf{R}^{n \times n}$ is symmetric and diagonalizable. Thus, there are n linear independent eigenvectors of the weight matrix. Let $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$ be the eigenvectors of W and $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the associated eigenvalues. They are ordered so that $1 = |\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$. λ_1 is called the dominant eigenvalue and $\mathbf{e}_1$ is associated dominant eigenvector. Since all the eigenvectors consist a and $\mathbf{b}_i(d) = [\beta_{i1}, \beta_{i2}, \cdots \beta_{id}]^{\mathrm{T}}$, then we have the following equation satisfied

$$\mathbf{y}_i(k,d) = \mathbf{V}(k,d)\mathbf{b}_i(d) \tag{2.4.7}$$

To obtain the consensus value $\bar{x}$ for node i, we need to take sufficient samples of $x_i(k)$ and solve (2.4.7), where d should at least be equal to or larger than m, which is the number of distinct and nonzero eigenvalues of weight matrix W

Let $A(k,m) = \mathbf{V}^{-1}(k,m)$ and the first row of $A(k,m)$ is given by

$$\mathbf{h} = [A_{11}(k,m), A_{12}(k,m), \ldots, A_{1m}(k,m)]^{\mathrm{T}} \tag{2.4.8}$$

we have

$$\beta_{i1} = \mathbf{h}^{\mathrm{T}}\mathbf{y}_i(k,m) \tag{2.4.9}$$

(2.4.9) shows that the consensus value can be calculated by the filter $\mathbf{h}$. This ends the proof. □

Lemma 2.4.1. *The sum all coefficients $\{h_0, \ldots, h_{m-2}, h_{m-1}\}$ is equal to one, i.e. $\sum_{j=0}^{m-1} h_j = 1$.*

Proof. Since $\bar{x} = \mathbf{h} * \mathbf{y}_i(k,m)$ satisfied as $k \to \infty$, when the iteration converges, we have $\bar{x} + h_{m-1}\bar{x} + \ldots + h_1\bar{x} + h_0\bar{x} = \bar{x}$. Then, canceling the value $\bar{x}$ from the equation will obtain the result. □

It is worth noting that Vandermonde matrix is related to a polynomial interpolation problem and can be easily inverted in terms of Lagrange basis polynomials [28]. Due to this reason, this method can be treated as an extrapolation method to find the consensus value at infinity. At the same time, we only need to find out the first coefficient β_{i1} in the distributed averaging. Therefore, only the elements in the corresponding row of $\mathbf{V}^{-1}(k,m)$ need to be found. This approach can save lots of computation time in the inverting of Vandermonde matrix.

Since simulation results show that $\mathbf{h}$ only depends the associated Vandermonde matrix $\mathbf{V}(k,m)$ which is independent of the nodes initial values and time index k. Therefore, each node in the network could find the consensus value at any time $k \geqslant m$ by passing a number of consecutive local values to this filter. In addition, all nodes in this network may share the same filter, which means all the filters have the same impulse response. However, such a consensus finding filter is not unique. As an example, a number of filters which have different impulse responses or filter lengths are found by different samples of $x_i(k)$. Each node could choose its own, but filter length determines how many time-steps before a node could find the consensus value.

Suppose we take d samples of $x_i(k)$, where $d > m$. Because the Vandermonde matrix $\mathbf{V}(k,d) \in R^{d \times m}$ is non-square, we introduce the Moore-Penrose pseudo inverse to find the least mean square solution.

Let

$$A(k,d) = \mathbf{V}^{+}(k,d) \tag{2.4.10}$$

where $^+$ denotes the Moore-Penrose pseudo inverse [29]. The first row of $A(k,d)$ is given by

$$\mathbf{h}' = [A_{11}(k,d), A_{12}(k,d), \ldots, A_{1d}(k,d)]^{\mathrm{T}} \tag{2.4.11}$$

Still, the value $\beta_{i1} = \mathbf{h}'^{\mathrm{T}} \mathbf{y}_i(k,m)$ is an accurate estimation of consensus value. Therefore, another consensus finding filter is obtained.

Due to the multiplication of the consensus finding filter, the set of filter is defined by

$$\{\mathbf{h} \in R^d | \forall d \geqslant m,\ \mathbf{h}^{\mathrm{T}} \mathbf{y}_i(k,d) = \bar{x}\} \tag{2.4.12}$$

However, the shortest filter has its length equaling to m, which means node can only have the consensus value after m steps.

In Section 3.3, the performance of this algorithm is shown by comparing it with the first-order DAC algorithm using optimal weight matrix in [15].

2.4.2 Inverse of the Vandermonde Matrix

The Vandermonde matrix has its applications in some problems like polynomial fitting, reconstruction of distributions from their moments, and so on. Solving Vandermonde matrix is related to a polynomial interpolation problem and can be easily inverted in terms of Lagrange basis polynomials. It can be very difficult to invert in other way if the size of the matrix is large, as the Vandermonde matrix is notoriously ill-conditioned by its nature. It is a good way to always work on the problems related to Vandermonde matrix in double precision or higher.

Let V_m be the Vandermonde's matrix of order m given by

$$V_m = \begin{bmatrix} \lambda_1 & \lambda_2 & \cdots & \lambda_m \\ \lambda_1^2 & \lambda_2^2 & \cdots & \lambda_m^2 \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_1^m & \lambda_2^m & \cdots & \lambda_m^m \end{bmatrix} \tag{2.4.13}$$

Then the equation

$$\begin{bmatrix} \lambda_1 & \lambda_2 & \cdots & \lambda_m \\ \lambda_1^2 & \lambda_2^2 & \cdots & \lambda_m^2 \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_1^m & \lambda_2^m & \cdots & \lambda_m^m \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \tag{2.4.14}$$

is related to the problem of moments: Given the values of all λ_i, find the unknown coefficients b_i, so that they match the given values y_i of the first m moments.

Its inverse is closely related to Lagrange's polynomial interpolation formula.

Let the polynomial of degree m be defined by

$$P_j(\lambda) = \prod_{\substack{i=1 \\ i \neq j}}^{m} \frac{\lambda - \lambda_i}{\lambda_j - \lambda_i} = \sum_{k=1}^{m} b_{jk} \lambda^{k-1} \tag{2.4.15}$$

The polynomial $P_j(\lambda)$ is a function of λ specially designed so that it is equal to zero at all λ_i except $i = j$ and takes on a value of one at $\lambda = \lambda_j$. In other words,

$$P_j(\lambda_i) = \delta_{ij} = \sum_{k=1}^{m} b_{jk} \lambda_i^{k-1}$$

where $\delta_{ij} = 1$ when $i = j$. The equation says that b_{jk} is exactly the inverse of the matrix V_m, with the subscript k as the column index.

To drive the analytical expression of b_{jk} and make it as easy as possible, let's define some intermediate result. Define the polynomial $q_j(\lambda)$ and work out its coefficients

$$\begin{aligned} q_j(\lambda) &= \frac{\prod_{i=1}^{m} (\lambda - \lambda_i)}{(\lambda - \lambda_j)} = \prod_{i=1, i \neq j}^{m} (\lambda - \lambda_i) \\ &= c_{j,m} \lambda^{m-1} + c_{j,m-1} \lambda^{m-2} + \ldots + c_{j,2} \lambda + c_{j,1} \end{aligned} \tag{2.4.16}$$

Examining the polynomial (2.4.15) and polynomial (2.4.16), we have

$$b_{jk} = \frac{c_{j,k}}{q_j\left(\lambda_j\right)}$$

Therefore, the solution of (2.4.14) is just the inverse of Vandermonde matrix time the vector on the right.

$$\beta_j = \sum_{k=1}^{m} b_{jk} y_k$$

If we only need to calculate the consensus value, as explained in Section 2.4.1 only the elements in the corresponding row of inverse of Vandermonde's matrix need to be found. The computation saving can be enormous by this approach.

2.5 Consensus Based Signal Processing

Sensor networks have a variety of applications such as surveillance, environment monitoring and collaborative signal processing. As the advantages of reliability, survivability, and increased range of coverage, there is an increasing interest in employing multiple distributed sensors for these applications [5]. A fundamental problem in sensor networks is to process spatially distributed information using a scalable algorithm [30].

Generally, there are two options for multiple sensors signal processing: First option is centralized signal processing. This requires the network contains a fusion center, and all sensor's information being transmitted to the central processor.

The Second option is distributed signal processing. Distributed average consensus (DAC) algorithms are tools for distributed information processing. It has received significant attention recently because of its robustness and simplicity. In this section, we will introduce two distributed signal processing methods based on consensus algorithms.

2.5.1 Data Fusion and Decision Making

In this section, we will consider a distributed detection problem in wireless sensor networks without the fusion center. A consensus based approach of distributed data fusion and decision making is introduced.

In centralized data fusion and decision making, a hypothesis test based on the ML (Maximum Likelihood), MAP (Maximum a Posteriori) or Bayesian decision rule will be carried out at the fusion center. Therefore, we intend to carry out the hypothesis test in a distributed manner.

Considering a binary hypothesis testing problem with the following two hypotheses

1. target is absent
2. target is present.

Suppose each sensor acquires a scale value from its sensing area, its signal has the following form

$$x_l = \begin{cases} \mu_{l,0} + n_l & \text{if no target present} \\ \mu_{l,1} + n_l & \text{if target present} \end{cases} \tag{2.5.1}$$

where $\mu_{l,m}$ is the mean value of x_l depending on hypothesis m and n_l is the noise of x_l. The prior probability of these two hypotheses is denoted by $P(H_m) = P_m, m = 0, 1$.

To make a declaration whether the target present or not, based on classical hypothesis test theory, the global log likelihood ratio (G-LLR) test is given by

$$LLR(x_1, ..., x_L) = \log \frac{f(x_1, ..., x_L|H_1)}{f(x_1, ..., x_L|H_0)} \underset{H_0}{\overset{H_1}{\gtrless}} \log \frac{P(H_o)}{P(H_1)} \tag{2.5.2}$$

where $f(x_1, ..., x_L|H_m)$ is the likelihood function of hypothesis H_m.

If the sensor signals are independent from one to another. Therefore, we have $f(x_1, ..., x_L|H_m) = f(x_1|H_m) \cdot \ldots \cdot f(x_L|H_m)$ and

$$LLR(\mathbf{x}) \;=\; \log \frac{f(x_1|H_1) \cdot \ldots \cdot f(x_L|H_1)}{f(x_1|H_0) \cdot \ldots \cdot f(x_L|H_0)} = \sum_{i=1}^{L} LLR(x_i) \tag{2.5.3}$$

According to (2.5.3), G-LLR is the sum of local log likelihood ratio (L-LLR) and can be calculated by distributed average consensus (DAC) algorithms. First, each sensor node i only calculates its L-LLR individually based on x_i. Then, all the sensors update their L-LLRs in the DAC iteration until they converge to a common value. Once the DAC algorithms converges, G-LLR is obtained by multiplying the average value with the number of sensors in the network.

LLR Calculation When Noises Are Correlated

The detection noise in sensor signal are Gaussian white noise and can be treated as independent. However, in real environment the sensor signal may also interfered by other aspect, which may cause correlated sensor signal fluctuation. One of the examples can be found in Chapter 5.

If the noises in sensor's signals are not independent from one to another, some expressions need to be derived to calculate G-LLR by DAC algorithms.

Let the sensors observation, the joint Gaussian white noises and the mean of sensor observation in a vector form be given by

$$\mathbf{x} = [x_1, \ldots, x_L]^{\mathrm{T}}$$

$$\mathbf{n} = [n_1, \ldots, n_L]^{\mathrm{T}} \sim \mathcal{N}(0, \Sigma)$$

$$\mathbf{u}_m = [\mu_{1,m}, \ldots, \mu_{L,m}]^{\mathrm{T}}, \; m = 0, 1. \tag{2.5.4}$$

where the noises is the joint Gaussian white noise denoted by $\mathbf{n} \sim \mathcal{N}(0, \Sigma)$.

Thus, the likelihood function is a joint Gaussian function $f(x_1, ..., x_L | H_m) = \frac{1}{(2\pi)^{L/2} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{u}_m)^T \Sigma^{-1} (\mathbf{x} - \mathbf{u}_m)\right)$ and G-LLR becomes

$$\begin{aligned} LLR(\mathbf{x}) &= \left(\mathbf{u}_1^{\mathrm{T}} - \mathbf{u}_0^{\mathrm{T}}\right) \boldsymbol{\Sigma}^{-1} \mathbf{x} + \frac{1}{2}\left(\mathbf{u}_0^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} \mathbf{u}_0 - \mathbf{u}_1^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} \mathbf{u}_1\right) \\ &= \sum_{l=1}^{L} w_l x_l + C \end{aligned} \tag{2.5.5}$$

where w_l is the l^{th} component of $\left(\mathbf{u}_1^{\mathrm{T}} - \mathbf{u}_0^{\mathrm{T}}\right) \boldsymbol{\Sigma}^{-1}$, C is the last term of the equation. (2.5.5) means G-LLR can be a weighted sum of sensor's observation plus C.

As a result, G-LLR can be caculated by DAC algorithms, provided that each sensor knows the weight w_l and C. Actually, the constant C changes the threshold of the hypothesis testing and can be subtract from both side of the hypothesis testing equation. Therefore, we have

$$\sum_{l=1}^{L} w_l x_l \underset{H_0}{\overset{H_1}{\gtrless}} \frac{P(H_o)}{P(H_1)} - C$$

where the average values $\frac{1}{L} \sum_{i=1}^{L} w_l x_l$ is obtained by distributed average consensus (DAC) algorithms. Then each sensor multiply the average i with the number of sensors in the network to get $\sum_{l=1}^{L} w_l x_l$.

In Section 5.4, how to calculate G-LLR by DAC algorithms on more generalized conditions will be presented. The sensor signals will be correlated and mixed with joint Gaussian white noises. In addition, the signal fluctuation noises are not only correlated but also depends on the existence of target.

2.5.2 Network Information Flooding

The conventional information flooding is actually done by copying information to all other nodes. Each node maintains a table of all nodes values in the network, initialized with its own value, and exchanges the tables of their own with those from their neighbors in each step. After a certain number of time steps which is equal to the diameter of the network, each node will obtain values of all nodes. distributed averaging can also be implemented by this way.

However, copying information and forwarding to all other nodes takes too much resources for the networks. If a networks has n nodes, the conventional flooding will need at least $(n-1)$ copies for each piece of information. And this estimation doesn't consider the cost in transmitting the information to the destination.

We intend to develop an algorithm to accomplish the information flooding, but didn't copy all the information to other nodes, so that the communication cost can be dramatically reduced. However, The difficulties of this kinds of distributed signal processing is that any individual node only knows partial information on the whole network. And information exchange is only allowed between neighbors.

Based on FT-DAC algorithm we introduced in Section 2.4, each node could decompose the local value vector by a linear combination of eigenvalues and eigenvectors and can calculate the coefficients before each term. Actually, with some signal processing techniques, there are more information in the local values sequence that each node can extract. These may also be applied to wireless sensor networks as they could be carried out distributively.

Therefore, in this section we propose a novel network information flooding technique based on consensus algorithms. It doesn't require copying information for so many times, instead, it transmits information by broadcasting. The advantage of this method is that it can save the costs in copying and transmitting information.

Suppose the network weight matrix W satisfies the same condition (2.3.5). Recall the initial local value decomposition given by

$$\mathbf{x}(0) = \sum_{j=1}^{n} \alpha_j \mathbf{e}_j \tag{2.5.6}$$

which shows that the initial local value vector can be defined by a set of coefficients and a new basis defined by the eigenvectors. In addition, the eigenvectors $\mathbf{e}_i$ only depends on the weight matrix. Therefore, once the coefficients α_i and weight matrix are available, the initial values vector $\mathbf{x}(0)$ can be calculated. It is possible to exchange information between nodes in the network without copying information to all other nodes.

To show how this method can be carried out distributively, let the sample vector $\mathbf{y}_i(k,d) \in R^d$ be defined by the history of $x_i(k)$

$$\mathbf{y}_i(k,d) = [x_i(k), x_i(k-1), \ldots x_i(k-d+1)]^{\mathrm{T}} \tag{2.5.7}$$

and the coefficients vector $\mathbf{a} = [\alpha_1, \alpha_2, \ldots, \alpha_n]^T$. If we involve the eigenvectors and redefine (2.4.7) by

$$\mathbf{y}_i(k,d) = \mathbf{V}_i(k,d) diag\left(\left[\mathbf{e}_1^T\mathbf{u}_i, \mathbf{e}_2^T\mathbf{u}_i, \dots, \mathbf{e}_n^T\mathbf{u}_i\right]\right)\mathbf{a} \qquad (2.5.8)$$

where $\mathbf{u}_i$ is the unit vector with all zeros except the i^{th} component is one, $\mathbf{e}_j^T\mathbf{u}_i$ means the i^{th} component of $\mathbf{e}_j$. Solving the above equation will obtain the coefficients α_j.

The consensus based information flooding is ideally suitable for time-invariant network. Because this method requires a node knows all the eigenvectors and eigenvalues of the network before it can estimate initial values of other nodes. It can be initialized by flooding all weight coefficients to all nodes in the network. However, instead of flooding a table of local values, flooding the weight matrix is only performed at the initialization stage for one time. The proposed method could have lower cost both in computation and communication, if the topology changes at a very low frequency.

2.6 Summary

In this chapter, we first introduced the preliminary in Section 2.1, where consensus problem on network is presented as well as the difference and connection between continuous-time and discrete-time DAC algorithms. The whole family of DAC algorithms and their categories are shown in Section 2.2.

In Section 2.3, it states that optimized higher order DAC (its spectral radius of weight matrix are minimized) will have better convergence rate than the lower order one. However, the algorithm complexity and computational cost will increase by introducing more orders. There are negligible improvement for the fourth order DAC compared to the third order one.

The convergence rates of HO-DAC algorithms depend on eigenvalues of graph Laplacian matrix $\lambda_i(L)$ which is topology dependent. Therefore, the optimizations of HO-DAC larger than second order don't have unique analytic solution.

In Section 2.4, it is shown that the finite-time DAC algorithm in an invariant network can find the average in finite number of iterations, and local value vector $\mathbf{x}(k)$ can be represented by a linear combination of a new normal basis defined by eigenvectors of W. Furthermore, the weight matrix W doesn't need to be convergent.

Some distributed signal processing based on consensus algorithms, such as data fusion and decision making, information flooding were introduced in Section 2.5.

Chapter 3

Finite-time DAC on Generalized Conditions

The finite-time DAC algorithm can calculate the consensus value by a linear combination of local values in the past, as shown in Section 2.4.1 Although a distributed algorithm to calculate the coefficients of the linear combination has been introduced in [16], it requires the FO-DAC algorithm being executed for several instances initialized with a set of linear independent vectors. As mentioned before, the finite-time DAC is not reliable to topology changes. If the method has to run multiple re-initializations of original FO-DAC algorithm, it may take more risks of being terminated by topology changes. Alternatively, the original consensus algorithm can run these instances in parallel with a set of independent initial values. However, the data transmission at each iteration increases.

Here we propose a generalized finite-time DAC algorithm, which does not require knowledge of network topology and multiple re-initializations of the original consensus algorithm. In the future research, the finite-time DAC algorithm probably will be generalized to non-symmetrical network. Before introducing the algorithm, there are two important linear filters need to be introduced, as the algorithm is based on the relationship of these filters. The first is the FIR filter to estimate the consensus value which has been already introduced in Section 2.4.1. The second one is introduced in the next Section 3.1.

3.1 Linear Predictor for Local Value

In observing the convergence curve of each node's local value sequence, we come to the idea that the sequence must obey some rules as it converges. In Section 2.4.1, it is shown that local value vector can be decomposed in terms of eigenvalues and eigenvectors. Based on this fact, a consensus value estimation method by inverting the Vandermonde matrix is proposed.

Furthermore, we have the following theorem to reveal another important property of FO-DCA.

Theorem 3.1.1. *Suppose an undirected graph $\mathcal{G}$ with associated weight matrix $W \in \mathbb{R}^{n \times n}$ which satisfies the conditions in Theorem 2.3.1. For the DAC iteration $\mathbf{x}(k+1) = W\mathbf{x}(k)$, local value $x_i(k)$ at node v_i is equal to a linear combination of local values of itself in m previous time steps, i.e. $x_i(k) = -a_{m-1}x_i(k-1) - \ldots - a_1 x_i(k-m+1) - a_0 x_i(k-m)$, where for any $k \geq m$ and m is a certain number.*

Proof. The minimal polynomial of W is given by

$$\begin{aligned} p(\lambda) &= \prod_{i=1}^{r} (\lambda - \lambda_i)^{r_i} = 0 \\ &= \lambda^m + a_{m-1}\lambda^{m-1} + \ldots + a_1\lambda + a_0 \end{aligned} \tag{3.1.1}$$

where $m = \sum_{i=1}^{r} r_i$. Since $p(W) = \mathbf{0}_{n \times n}$, we have,

$$W^m + a_{m-1}W^{m-1} + \ldots + a_1 W + a_0 I = \mathbf{0}_{n \times n}$$

Multiplying both sides with $\mathbf{x}(0)$ we obtain

$$\mathbf{x}(k) + a_{m-1}\mathbf{x}(k-1) + \ldots + a_0\mathbf{x}(k-m) = \mathbf{0}_{n \times n} \tag{3.1.2}$$

For any node $v_i \in \mathcal{V}$, its local value at time k is just the i^{th} component of $\mathbf{x}(k)$. After a little evolution, we have

$$x_i(k) = -a_{m-1}x_i(k-1) - \ldots - a_0 x_i(k-m) \tag{3.1.3}$$

In other words, $x_i(k)$ can be predicted by an FIR filter given by $[-a_r, -a_{r-1}, \ldots, -a_0]$, if we pass a number of consecutive local values in the past through that filter, after the time index $k > m$. In addition, all the nodes in the network can share the same coefficients, when the minimal polynomial of W is not changed. □

Lemma 3.1.1. *The sum of coefficients $a_{m-1}, \ldots, a_1, a_0$ is equal to -1.*

Proof. Since (3.1.2) is satisfied as $k \to \infty$, we have $\bar{\mathbf{x}} + a_{m-1}\bar{\mathbf{x}} + \ldots + a_1\bar{\mathbf{x}} + a_0\bar{\mathbf{x}} = \mathbf{0}$. Then, canceling the vector $\bar{\mathbf{x}}$ will obtain $\sum_{j=0}^{m-1} a_j = -1$. □

Given the local value vector sequence obtained by FO-DCA, one may instantly comes to the idea of applying an adaptive filter algorithm to estimate the set of coefficients. For example the Kalman filter algorithm. One benefit of the adaptive filter algorithm is that when the network topology is changed, the filter could adaptively change its coefficient during the iteration.

Find the Coefficients of Linear Predictor

After running FO-DAC algorithm for a number of steps, node v_i could list a number of equations by its available local values. Once they are sufficient equations to construct a matrix, which is a Toeplitz matrix, we can have an important result in the following.

Let $T_i = T_i(k, D_i) \in \mathbb{R}^{D_i \times D_i}$ be a function at v_i which outputs a Toeplitz matrix with size D_i with $x_i(k)$ as diagonal entries.

$T_i(k, D_i) =$

$$\begin{bmatrix} x_i(k) & x_i(k-1) & \dots & x_i(k-D_i+1) \\ x_i(k+1) & x_i(k) & \cdots & x_i(k-D_i+2) \\ \vdots & \vdots & \ddots & \vdots \\ x_i(k+D_i-2) & x_i(k+D_i-3) & \cdots & x_i(k-2) \\ x_i(k+D_i-1) & x_i(k+D_i-2) & \cdots & x_i(k) \end{bmatrix} \tag{3.1.4}$$

Similarly, define the function $\mathbf{y}_i = \mathbf{y}_i(k, D_i) = [x_i(k+1), x_i(k+2), \dots, x_i(k+D_i)]^T \in \mathbb{R}^{D_i}$ which outputs the local value history at v_i from time $k+1$ to $k+D_i$. Then, let $\mathbf{a}_i(D_i) = [a_{i,D_i-1}, \dots, a_{i,1}, a_{i,0}]^T \in \mathbb{R}^{D_i}$ be a vector of length D_i representing the coefficients in the minimal polynomial calculated at v_i. Then, we have the solution of these equations given by

$$\mathbf{a}_i(D_i) = -T_i^{-1}(k, D_i)\,\mathbf{y}_i(k, D_i) \tag{3.1.5}$$

The size of the Toeplitz block D_i, should be less or equal to m so that (3.1.5) has unique solution. To solve (3.1.5), node i needs to have sufficient local values.

Toeplitz matrix is a special type of matrix and can be inverted by some algorithms in the polynomial time of order D_i^2, rather than the order of D_i^3 in general case (for example by LU decomposition), which is an enormous computational saving [28]. Levinson developed a recursive algorithm to solve the system quickly if the Toeplitz matrix is symmetric.

One interesting advantage of Levinson's method is it can also apply to the case when weight matrix is non-symmetric. The fact that the method can be generalized to the non-symmetric case is stated in the texts [31]. Therefore, We will invert this Toeplitz matrix using Levinson's method.

It totally needs $2D_i + 2$ local values to construct the Toeplitz matrix and (3.1.5), all the local values can be obtained from one instance of FO-DCA after $2D_i + 1$ iterations. By comparing to the original finite-time consensus algorithm [16] who requires D_i^2 iterations of FO-DCA in total, the proposed method has improvement in reduced number of matrix iterations.

In Section 3.2, we will show how to obtain the consensus finding filter by the linear predictor.

3.2 Convert Linear Predictor to Consensus Finding Filter

Once there are sufficient local values, node i can solve the equation and obtain the set of coefficients $\mathbf{a}$ and construct the polynomial

$$p_i(\lambda) = \lambda^m + a_{m-1}\lambda^{m-1} + \ldots + a_1\lambda + a_0$$

If matrix W satisfies the condition in Theorem 2.3.1 there is a simple eigenvalue equaling to one. As shown in Section 2.4.1 we can let the first eigenvalue $\lambda_1 = 1$, and it will be located in the first row of Vandermonde's matrix. Therefore, the consensus finding filter is given by the first row of inverse of Vandermonde's matrix.

Examining the Lagrange's polynomial interpolation formula (2.4.15) we can rewrite the consensus finding filter in terms of a_k.

To make the expression simple, we may define another polynomial (Also see (2.4.16))

$$q_i(\lambda) = \frac{p_i(\lambda)}{\lambda - 1} = c_m\lambda^{m-1} + c_{m-1}\lambda^{m-2} + \ldots + c_2\lambda + c_1$$

where $c_k = 1 + \sum_{j=k+1}^{m} a_j$. The consensus finding filter can be given by the coefficient of $q_i(\lambda)$.

$$\mathbf{h} = \frac{[c_m, c_{m-1}, \ldots, c_2]}{\sum_{k=1}^{m} c_k} \tag{3.2.1}$$

This indicates the possibility of using the adaptive filter to estimate the consensus value after a certain number of iterations.

One interesting property of the consensus finding filter is that it can be found by this method when the weight matrix is non-symmetric. It can be demonstrated

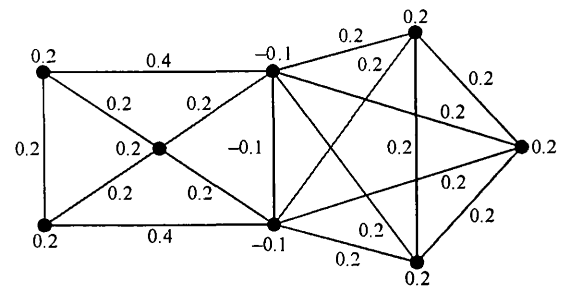


Figure 3.1: Graph with optimal weights which maximize convergence rate

by finding the inverse of a confluent Vandermonde matrix. In examining the expression of the inverse of confluent Vandermonde matrix, we note that the corresponding row where the dominant eigenvalueλ_1 is located are actually the coefficients of a consensus finding filter. The length of the filter is not required to be the same. It may be not equal to the number of distinct and nonzero eigenvalues m, but less or equal to the sum of all orders in minimal polynomial of weight matrix $\sum_{i=1}^{m} r_i$. However, the relationship of local value predictor and consensus finding filter still holds.

3.3 Simulation

Consider the graph from [15] (see Figure 3.1), the weight matrix W corresponding to this graph is symmetric and has eigenvalues $\lambda(W) = \{1, 0.6, 0.4, 0, 0, 0, -0.4, -0.6\}$. The time index k can be chosen large enough so that there are sufficient number of local values to estimate the filter coefficients. For example, there are 5 distinct and nonzero eigenvalues of W, so we choose the time index $k = 5$ and $d = 5$ which is the minimum filter length.

Based on (3.2.1) the consensus finding filter is given by

$$\mathbf{h} = [1.8601,\ 0,\ -0.9673,\ 0,\ 0.1071]^{\mathrm{T}}$$

For any random generated $\mathbf{x}(0) \in R^n$, node values vector $\mathbf{x}(k)$ is updated by the iteration (2.3.3). At the same time each node passes its local values through filter $\mathbf{h}$. Filter output is given by $\hat{x}_i(k) = \mathbf{h}(k) * x_i(k)$. Figure 3.2 is presented to compare the convergence performance of first-order DAC (FO-DAC) algorithm with optimal matrix and the finite-time consensus algorithm. The performance is evaluated by the mean square error (MSE), defined by $\mathrm{MSE}_{\text{FO-DAC}}(k) = \sum_{i\in\mathcal{N}} E[|x_i(k) - \bar{x}|^2]$, $\mathrm{MSE}_{\text{filter}}(k) = \sum_{i\in\mathcal{N}} E[|\alpha_i(k) - \bar{x}|^2]$ respectively, where $\bar{x} = (1/n)\sum_{i\in\mathcal{N}} x_i(0)$. The result shows that the consensus finding filter cal-

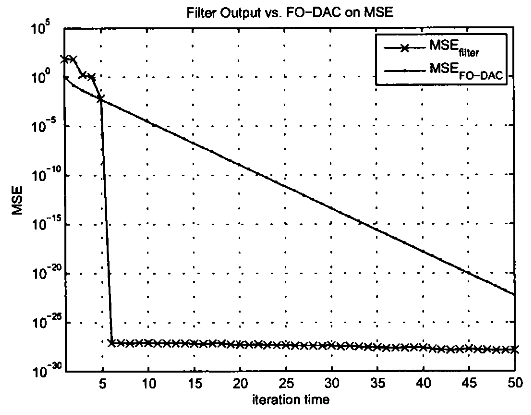


Figure 3.2: Performance of the first order iteration with optimal matrix vs. consensus finding filter algorithm

culates the consensus value after a finite number of iterations and MSE drops dramatically to the quantization error at the same time.

3.4 Summary

In this section, we introduced the finite-time DAC on generalized condition. Compared to the previous version of finite-time DAC, the improvements are: it doesn't require the eigenvalues to be known prior to the algorithm. However, the number of iterations can not be less than a certain limit, because the minimum number of iterations depends on the weight matrix, which is the same as the previous finite-time DAC. With this improvement, some further algorithm could be driven. They will be introduced in Chapter 4.

Chapter 4

Real-time Optimization of DAC

Distributed average consensus (DAC) algorithms utilize matrix iteration to find the dominant eigenvector. To minimize the required number of iterations, the algorithms need to be optimized. However, this optimization needs the knowledge of network topology, which is very hard to obtain for an individual agent in distributed networks. Thus, optimal step length and forgetting factor need to be calculated offline and forwarded to every agent. To solve this problem, we propose a distributed real-time optimization technique so that each node can estimate these optimal parameters individually. In addition, the method is based on constant first-order DAC itself, so it will not stop the consensus process. The result shows that a numerical error due to quantization would exist in the distributed solution. It will increase as the network becomes larger. Thus, a numerical technique is introduced to mitigate the error. The estimated parameters after mitigation do not obviously decline the performance of higher-order DAC when network size is smaller than a threshold.

4.1 Introduction

As introduced in 1.1.1, DAC algorithms can be divided into asymptotic and non-asymptotic algorithms. However, the current optimization of asymptotic DAC algorithms are centralized methods. The distributed method inspired by the gossip algorithm [18] converges very slowly. Non-asymptotic DAC algorithms, such as finite-time [16] and adaptive filter DAC algorithm [17], they find the average in finite number of iterations but not robust against topology changes.

After investigating these problems, we intend to find a distributed optimization method for the constant first-order DAC and higher-order DAC algorithms to enable the whole system to work distributively. Because in centralized optimization methods, optimal parameters of these algorithms are only related to the eigenvalues of Laplacian matrix of the network [1], if we could estimate these eigenvalues in a distributed manner, these centralized methods could be carried out distributively.

Consequently, a distributed eigenvalue estimation algorithm is proposed in this chapter. In contrast to other distributed algorithms [32][33][8], initialization of the proposed algorithm is actually the first-order DAC itself. Therefore, first-order DAC will not be interrupted during the optimization and algorithm complexity and communication cost can be dramatically reduced.

However, the distributed solution has a numerical error due to quantization, which may decline the algorithm performance. Therefore, a least mean square solution is obtained to mitigate the numerical error. When using the floating point number in double format and the network size is smaller than 32, the numerical error after mitigation does not dramatically decline the performance and the proposed method is applicable.

The rest of this chapter is structured as follows. First, the distributed real-time optimization of DAC will be given in Section 4.2. Second, the mitigation of numerical error will be proposed in Section 4.3. Third, the algorithm complexity will be analysed in Section 4.4. Fourth, in Section 4.5, the performance of DAC using the distributed real-time optimization will be analysed and compared with the centralized one. Finally, the conclusion will be given in Section 4.6.

4.2 Real-time Optimization of DAC

Traditional optimization of HO-DAC or CFO-DAC requires a centralized node to gather information of the Laplacian matrix [1]. Without the spectrum of Laplacian matrix, each node could only choose a non-optimal point (ϵ, γ) in the boundary of the convergence region.

In Section 2.3.2, it is shown that the optimal parameters $\epsilon_{opt}, \gamma_{opt}$ of HO-DAC are only related to $\lambda_i(L)$. To enable the distributed optimization, the key is to estimate these eigenvalues in a distributed manner.

In fact, there are some decentralized techniques [8][32][33] to estimate the eigenvalues. However, they are not designed for DAC algorithms and will involve complex and costly initialization. In addition, as they are also matrix iteration algorithms, DAC algorithms have to be stopped during the eigenvalue estimation.

In contrast, the distributed real-time optimization and CFO-DAC can be running simultaneously and algorithm complexity and communication cost can be dramatically reduced. Because the initialization of the proposed algorithm is to store a number of local values obtained by CFO-DAC, the distributed system can still work using non-optimal parameters at the very beginning just after the deployment. After a number of iterations of CFO-DAC, these eigenvalues could be estimated and better parameters could be used in the next iterations.

4.2.1 Find the Characteristic Polynomial

Because $\lambda_i(W)$ is the root of the characteristic polynomial $p(\lambda) = \prod_{i=1}^{m}(\lambda - \lambda_i)^{r_i} = \lambda^D + a_{D-1}\lambda^{D-1} + \ldots + a_0 = 0$, the distributed eigenvalues estimation is subject to the calculation of the coefficients $\{a_j\}$.

To calculate $\{a_j\}$, we need to use the Theorem 3.1.1 introduced in Section 3.1. If more local values are available, node v_i could list a number of equations similar to (3.1.3). Once they are sufficient to construct a matrix, the coefficients $\{a_j\}$ could be estimated.

Define the function $\mathbf{y}_i = \mathbf{y}_i(k, D_i) = [x_i(k+1), x_i(k+2), \ldots, x_i(k+D_i)]^T \in \mathbb{R}^{D_i}$ which outputs a *local value history vector* of v_i from time $k+1$ to $k+D_i$. Then, define a function $T_i = T_i(k, D_i) \in \mathbb{R}^{D_i \times D_i}$ that outputs a Toeplitz matrix with $x_i(k)$ on the diagonal. $T_i(k, D_i) =$

$$\begin{bmatrix} x_i(k) & x_i(k-1) & \ldots & x_i(k-D_i+1) \\ x_i(k+1) & x_i(k) & \cdots & x_i(k-D_i+2) \\ \vdots & \vdots & \ddots & \vdots \\ x_i(k+D_i-1) & x_i(k+D_i-2) & \cdots & x_i(k) \end{bmatrix} \tag{4.2.1}$$

Besides, let $\mathbf{a}_i(D_i) = [a_{i,D_i-1}, \ldots, a_{i,1}, a_{i,0}]^T \in \mathbb{R}^{D_i}$ be a vector to store the coefficients calculated at v_i. Finally, we have the solution of $\mathbf{a}_i(D_i)$ given by

$$\mathbf{a}_i(D_i) = -T_i^{-1}(k, D_i)\,\mathbf{y}_i(k, D_i)\,. \tag{4.2.2}$$

Toeplitz matrix is a special type of matrix and can be inverted by Levinson's algorithms in the polynomial time of order D_i^2, rather than the order of D_i^3 in general case (for example by LU decomposition) [28].

4.2.2 Estimated Eigenvalues of Laplacian Matrix

After node v_i could calculate the coefficients vector $\mathbf{a}_i(D_i)$, it will construct a local polynomial $p_i(\lambda)$ and find the roots of the polynomial. Then, the local eigenvalues spectrum of W at v_i is obtained, which is defined by $\hat{S}_i(W) = \left\{\hat{\lambda}_j^{(i)}(W)\right\} =$

$\{\lambda | p_i(\lambda) = 0\}$, $j = 1, \ldots, D_i$. Because $W = I_n - \epsilon L$, an estimated Laplacian spectrum at v_i could be obtained, which is denoted by $\hat{S}_i(L) = \left\{\hat{\lambda}_j^{(i)}(L)\right\}$, where $\hat{\lambda}_j^{(i)}(L) = \frac{1-\hat{\lambda}_j^{(i)}(W)}{\epsilon}$, $j = 1, 2. \ldots, D_i$.

4.2.3 Eigenvalues Missing In Local Spectrum

In simulation, some eigenvalues are missing in some of the local eigenvalue spectrums. The reason is because sometimes the Toeplitz matrix T_i with the original size D will lose rank, so that the solution is not unique. Node v_i could only build a smaller Toeplitz matrix with size D_i ($D_i \leq D \leq n$). As a result, the local polynomial $p_i(\lambda)$ will be reduced to D_i coefficients.

However, the following theorem in [16] assure that the roots of local polynomial is the same roots of minimal polynomial of W.

Theorem 4.2.1. *[16] The local polynomial $p_i(\lambda)$ of node v_i divides the minimal polynomial $p(\lambda)$ of W for all $1 \leq i \leq n$. (see [16] for proof)*

Every node may recover the complete eigenvalue spectrum of W by exchanging its local eigenvalue spectrum with its neighbors. To assure that no eigenvalue is missing in the final eigenvalue spectrum, we have the following theorem that any eigenvalue $\lambda_l(W)$ must be estimated by at least one node in the network.

Theorem 4.2.2. *Suppose a graph $\mathcal{G}$ whose associated weight matrix W satisfies the convergence condition, and initial value vector $\mathbf{x}(0) \in \mathbb{R}^n$ is chosen randomly. If all nodes estimate the eigenvalue of W by the proposed method using local values that are available, any eigenvalue $\lambda_l(W)$ could be estimated by at least one node in the network.*

Proof. The proof can be generalized to non-diagnosable matrix with the help of Jordan decomposition of W.

$$\begin{aligned} W^k &= UJ^kU^{-1} \\ &= U \begin{bmatrix} J_1^k & & \\ & \ddots & \\ & & J_{\hat{m}}^k \end{bmatrix} U^{-1} \end{aligned}$$

where $\hat{m}$ is the number of Jordan blocks. Therefore, $\mathbf{x}(k) = UJ^kU^{-1}\mathbf{x}(0)$. □

Assume that all nodes miss an eigenvalue λ_l in their estimated eigenvalue spectrums, $l = 1, 2, \ldots, \hat{m}$. This means all the terms contain λ_l are multiplied by zero. Since $\mathbf{x}(0)$ is chosen randomly in $\mathbb{R}^n$, we can only have $W^k = UJ^kU^{-1}$ with all terms involving λ_l are equal to zero, thus

$$U \begin{bmatrix} \mathbf{0} & & \\ & J_l^k & \\ & & \mathbf{0} \end{bmatrix} U^{-1} = \mathbf{0}_{n\times n} \tag{4.2.3}$$

Since $J_l^k \neq \mathbf{0}$ and the matrix U is consisted of linear independent vectors, the above equation (4.2.3) can not be valid. Thus, the assumption is not true and $\lambda_l(W)$ could be estimated by at least one node in the network.

4.3 Mitigation of Numerical Error of Eigenvalues

Due to the limited accuracy of the floating point number, the solution obtained by the proposed method has a numerical error. To mitigate the error, one of the options is to use floating number with more bits. However, communication cost is increased in each iteration.

As floating point number in double format is available on most computers, we apply our numerical mitigation on this basis.

The idea is to have more equations similar to (3.1.3) and involve the Moore-Penrose pseudo-inverse to find the least mean square solution. Thus, Toeplitz matrix in (4.2.2) should be replaced by a matrix constructed by some other way, whose height is larger than width. The local value history vector $\mathbf{y}_i(k, D)$ on the left of (4.2.2) is also expanded accordingly.

There are two ways to expand the matrix. First, the new matrix can be built by concatenating Toeplitz matrix T_i and $T_j, v_j \in \mathcal{N}_i$ along the column. As x_j is available for node v_i, this improvement will not increase the communication cost.

Second, more than one instances of CFO-DAC could be carried out to obtain more useful local samples. Let $\mathbf{x}_1(0), \mathbf{x}_2(0), \ldots, \mathbf{x}_N(0)$ denote N different and independent initial local value vectors. Each one of them is used to reinitialize an instance of CFO-DAC and each instance of CFO-DAC is iterated for at least $2n+2$ steps. During this initialization, each node v_i will store the local values obtained by each instance of CFO-DAC.

To construct the new matrix, first let $T_{i,s} = T_{i,s}(D_i - 1, D_i)$ be the Toeplitz matrix of v_i at s instance of CFO-DAC, with $x_{i,s}(D_i - 1)$ on the diagonal, where $x_{i,s}$ is the local value of node v_i at s instance of CFO-DAC. Second, concatenating

$T_{i,s}$ and $T_{j,s}, \forall v_j \in \mathcal{N}_i$ will obtain $M_{i,s}$.

$$M_{i,s} = \left[T_{i,s}^T, T_{j_1,s}^T, \ldots, T_{j_{|\mathcal{N}_i|},s}^T\right]^T \tag{4.3.1}$$

where $s = 1, ..., N$. Finally, concatenating the matrix $M_{i,s}$ will construct another matrix,

$$\tilde{M}_i = \left[M_{i,1}^T, M_{i,2}^T, \ldots, M_{i,N}^T\right]^T. \tag{4.3.2}$$

The width of $\tilde{M}_{i,s}$, denoted by D_i, is the maximum integer so that the matrix $\tilde{M}_{i,s}$ has full column rank.

On the other hand, let $\mathbf{y}_{i,s}= \mathbf{y}_{i,s}\left(D_i - 1, D_i\right)= \left[x_{i,s}\left(D_i\right), x_{i,s}\left(D_i + 1\right), \ldots, x_{i,s}\left(2D_i\right)\right]^T$ be the local value history vector of v_i at s instance of DAC. First, concatenating $y_{i,s}$ and $y_{j,s}, v_j \in \mathcal{N}_i$ will obtain

$$\mathbf{q}_{i,s} = \left[\mathbf{y}_{i,s}^T, \mathbf{y}_{j_1,s}^T, \mathbf{y}_{j_2,s}^T, \ldots, \mathbf{y}_{j_{|\mathcal{N}_i|},s}^T\right]^T. \tag{4.3.3}$$

Second, concatenating $\mathbf{q}_{i,s}$ will obtain

$$\tilde{\mathbf{q}}_i = \left[\mathbf{q}_{i,1}^T, \mathbf{q}_{i,2}^T, \ldots, \mathbf{q}_{i,N}^T\right]^T. \tag{4.3.4}$$

The vector $\tilde{\mathbf{q}}_i$ is constructed by this way so that each local value is one time step later than the first local value in each row of matrix $\tilde{M}_i$.

Therefore, the coefficient vector can be obtained by inverting the new matrix $\tilde{M}_i$

$$\mathbf{a}_i = -\tilde{M}_i^+ \tilde{\mathbf{q}}_i \tag{4.3.5}$$

where $^+$ denotes the Moore-Penrose pseudoinverse. Simulation result indicates that, the more rows in $\tilde{M}_i$, the more accurate the solution could be. However, increasing the height of $\tilde{M}_i$ after a limit $n\left|\mathcal{N}_i\right|$ can not obtain a more accurate result.

4.4 Analysis of Algorithm Complexity

This section is to compare the algorithm complexity of the existing and proposed distributed optimization for DAC.

To solve the problem (2.3.12), Xiao [15] proposed two centralized methods: interior-point method and subgradient method to minimize $\rho\left(W - \mathbf{1}\mathbf{1}^T/n\right)$. Let n be the network size and m be the number of edges. The interior-point method is iterative and usually finds the optimal solution in $20n \sim 80n$ step, at a cost of

Algorithm 4.1 Distributively find the Optimal Matrix for FO-DAC.

```
• Initialization: Initialize vector w with some feasible entries, for example
  the maximum degree weights.
• Repeat for $t \geq 1$
    – $W = I - Q\mathbf{w}^{(t)}Q^T$.
    – Repeat for $s \geq 1$ to find subgradient $g^{(t)} \in \mathbb{R}^{|\mathcal{E}|}$
        * $u^{(s+1)} = Wu^{(s)}$
        * $u^{(s+1)} = u^{(s+1)} - \text{Avg}\left(u_i^{(s+1)}\right)$
        * $u^{(s+1)} = u^{(s+1)} / \left\|u^{(s+1)}\right\|, s = s + 1$
    – Until $\left\|u^{(s)} - u_r\right\|_2 \leq \epsilon_{sub}$.
    – $g_l = \begin{cases} -(u_i - u_j)^2 & \text{if } \rho\left(W - \frac{\mathbf{1}\mathbf{1}^T}{n}\right) = \lambda_2(W) \\ (u_i - u_j)^2 & \text{if } \rho\left(W - \frac{\mathbf{1}\mathbf{1}^T}{n}\right) = \lambda_n(W) \end{cases}$
      $l \sim (i, j) \in \mathcal{E}$
    – $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \beta_k \frac{g^{(t)}}{\left\|g^{(t)}\right\|}, t = t + 1$.
```

$(10/3)n^3 + (1/3)m^3$ flops per steps.

Compared to interior-point method, the subgradient method can be computed locally but relatively slow. [18] proposed a distributed subgradient method to minimize the sub-dominate eigenvalue of a probability matrix. This method can be used to optimize the first-order DAC after a little modification. The modified method is given in Algorithm 4.1, where the function Avg $\left(u^{(s)}\right) = \frac{1}{n}\sum_{i=1}^{n}\left(u_i^{(s)}\right)$ is implemented by DAC. Given the error tolerance ϵ_{ave}, the DAC algorithm has to be iterated at least $T_{ave} = O\left(n^2 log\left(\frac{1}{\epsilon_{ave}}\right)\right)$ times [34], so that the local value vector can converge to the range $\left\|\mathbf{x}\left(T_{ave}\right) - \bar{\mathbf{x}}\right\| \leq \epsilon_{ave}$. The second loop to calculate sub-dominant eigenvector u_r is also similar to the first iteration. It needs to be executed for a number of times to obtain a result within the error tolerance ϵ_{sub} [18]. Furthermore, the third iteration, which is the subgradient algorithm, takes enormous steps to converge and there is no simple stopping criterion to guarantee a certain level of optimality [15]. Therefore, with a conservative estimation, the times of matrix iteration might be more than $O\left(n^4\right)$.

Compare to the enormous number of matrix iterations in Algorithm 4.1, there is a significant time reduction by the proposed optimization if problem (2.3.12) reduces to find the best constant. When higher accuracy is needed, we execute n instances of CFO-DAC and the number of matrix iterations is in the order of n^2.

Furthermore, if the network topology doesn't change after the optimization, the estimated eigenvalues can drive to an estimated solution $(\hat{\epsilon}, \hat{\gamma})$ for HO-DAC algorithms, which are faster than the optimal FO-DAC.

4.5 Simulation of Eigenvalue Estimation

The simulation is coded by Matlab and taken by the following steps. First, n nodes are uniformly distributed in an unit square and a link is established between any two nodes if their distance is less than a threshold r. To ensure the generated graph is connected with high possibility, r is chosen as $\sqrt{\log_{10}(n)/n}$ [35]. Besides, assume each link is symmetric so that the network graph is undirected.

Second, on the random generated network, one or more instances of CFO-DAC are executed. Local values obtained in each DAC instance are stored in local memory of each node. The step length is a constant shared by all nodes and chosen in the convergence range.

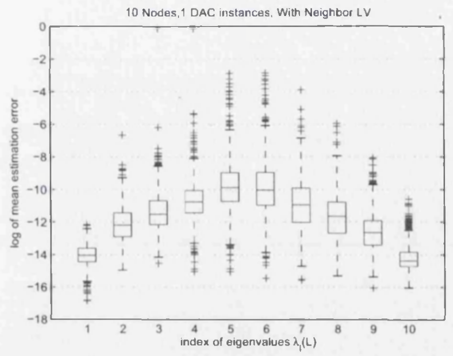
Third, once sufficient number of local values are obtained, the eigenvalue estimation algorithm is executed and local Laplacian spectrum is obtained at each node.

Finally, the performance of eigenvalue estimation is evaluated by the estimation errors. Before that, each Laplacian eigenvalue $\lambda_j(L)$ is matched with only one eigenvalue $\hat{\lambda}_k^{(i)}(L)$, if the distance $\left|\hat{\lambda}_k^{(i)}(L) - \lambda_j(L)\right|$ is the minimum for all eigenvalues in the estimated local Laplacian spectrum. Let the minimum distance be $e_{i,j} = \min_{l=1,\ldots,D_i} \left|\hat{\lambda}_l^{(i)}(L) - \lambda_j(L)\right|$. The mean estimation error of $\lambda_j(L)$ is $\frac{1}{n}\sum_{i=1,\ldots n} e_{i,j}$.

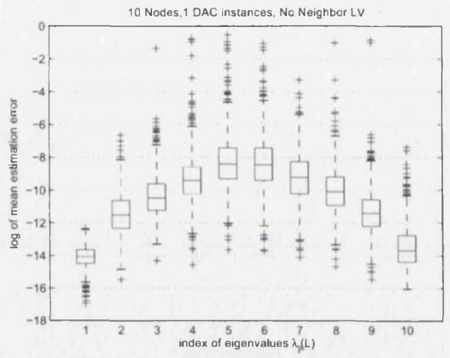
In Figure 4.1, we use the box plot to graphically illustrate the performance of eigenvalue estimation. The distribution of log mean estimation errors are obtained from 1000 simulations.

Simulation results show that taking local values from neighbors has better performance in lower estimation errors. Excluding local values of neighbors will create more outliers and increase the estimation error. In addition, the estimation errors will decrease if more instances of CFO-DAC are taken. On the other hand, the estimation errors will increase as the network size becomes larger. However, the numerical errors of $\lambda_2(L)$ and $\lambda_n(L)$ increase very slowly compared to other eigenvalues in the spectrum. Even their outliers in Figure 4.1 have estimated error lower than 10^{-8}.

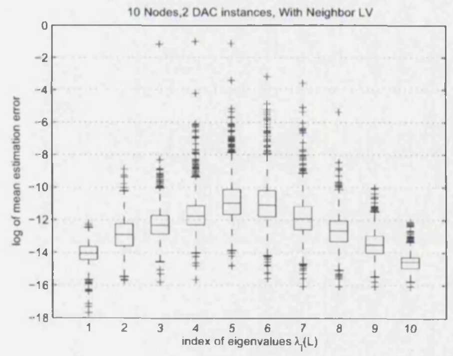
To see how these estimation errors take effect on the performance of DAC, we conducted another simulation where estimated parameters $(\hat{\epsilon}, \hat{\gamma})$ are used to



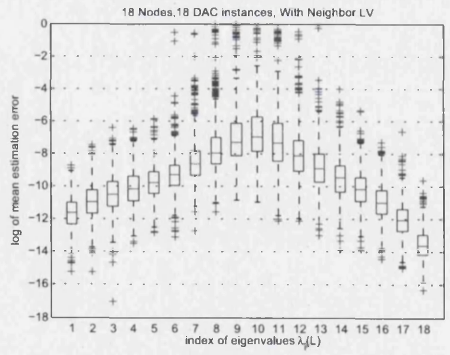
(a) 10 nodes, 1 DAC instance, with neighbor local values



(b) 10 nodes, 1 DAC instance, no neighbor local values



(c) 10 nodes, 2 DAC instance, with neighbor local values



(d) 18 nodes, 18 DAC instance, with neighbor local values

Figure 4.1: Box plot of log mean estimation error of eigenvalues, with/without local value of neighbors.

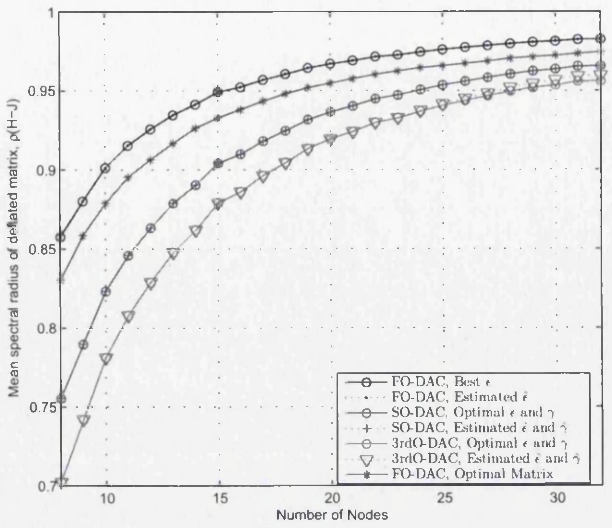


Figure 4.2: Mean of spectral radius of CFO-DAC and HO-DAC, using optimal parameters and estimated parameters.

construct a suboptimal higher-order weight matrix $\hat{H}$. The spectral radius of $\left(\hat{H} - J\right)$ is plotted and compared with the optimal one in Figure 4.2.

As shown in Figure 4.2, CFO-DAC or SO-DAC using estimated parameters has similar spectral radius as the one using optimal parameters. It seems that the numerical errors of $\lambda_2(L)$ and $\lambda_n(L)$ do not decline their performances dramatically. For third-order DAC algorithm, estimated errors of eigenvalues don't have disastrous impact on the performance. The spectral radius goes slightly upper than the optimal one after the network size is larger than 25. It seems that estimating other eigenvalues with low accuracy is not a critical problem. However, the parameters $\hat{\epsilon}$ and $\hat{\gamma}$ might not be located in the convergence region if the network size is larger than 32. The simulation of fourth-order DAC for even larger network up to 40 nodes, reports several divergent cases.

4.6 Summary

In this chapter, we introduced a distributed method to estimate the optimal parameters for DAC algorithms. However, numerical errors of these parameters due to quantization can decline the algorithm performance. Especially for DAC algorithms with order larger than second, they are more sensitive to the errors. To mitigate this effect, we introduce a numerical technique to find the least mean square solution. After mitigation, the numerical errors of estimated parameters slightly declines the performance of first-order DAC and second-order DAC. For the third-order DAC, estimated parameters by local Laplacian spectrum is still convergent and the algorithm is faster than second order. However, if floating point number in double format is used and the network size is larger than 32 nodes, the numerical errors will be too large even after mitigation. These findings indicate that the proposed method is applicable to optimize higher order DAC algorithms when network size is small. Otherwise, we should decrease the order of DAC or increase the accuracy of the floating point number. The second-order consensus algorithm could be a compromise as it requires fewer parameters, converges faster than first-order DAC and maintains convergence reliability. In the future, we are intending to investigate the effect of link failure and other practical aspects while applying the proposed method to a distributed system.

Chapter 5

Distributed Cloud Detection

In this chapter, the DAC algorithms will be applied to the remote detection of cloud.

The introduction of the cloud detection is shown in Section 5.1, followed by the the system model in Section 5.2. Gaussian plume model is widely used in cloud concentration modeling. However, the sensors is using laser remote sensing technology. In addition, the cloud concentration of a plume is actually a random process, which the Gaussian plume model can not capture. Therefore, the Gaussian plume model needs a little modification. Furthermore, a 3D cloud animation is implemented to get data of cloud concentration. The modified models are presented in Section 5.3.

Section 5.4 illustrates the techniques of distributed detection of cloud and sensor observation model. Expectation-maximization algorithm is adopted to build Gaussian mixture model for sensor observation from background in the training stage, followed by the detection stage. The decision of cloud existence is made based on a log likelihood ratio test, which is performed by each sensor with the help of DAC algorithms. Finally, the detection performance is simulated and presneted in Section 5.5. To get more real cloud data, 3D fluid animation techniques with turbulence flow are used. It shows the multiple sensor detection will be more reliable in the noisy environment.

5.1 Introduction

A cloud is a group of liquid or solid particles floating in the air. Sometimes it contains harmful or dangerous particles so that it needs to be observed and tracked. According to their types of targets, the detection can be categorized into smoke/gas/aerosol detection. The concepts are similar, but the types of sensors to perform the detection are various. For example, sensors are divided into contact detection and remote sensor, based on their sensing methods.

Smoke and gas detection sensors are very common in daily life, for example, the fire alarm sensors inside our buildings, which are contact sensing devices. When smoke or gas agent contacts with the sensing element, the chemical or physical reactions change the electrical characteristics of the sensing element. Then the alarm is set off if the agent's concentration is larger than a threshold.

Smoke detection based on video and image processing is possible., which also attracts a lot of research interests. For example, in the wildfire video surveillance, people try to change the human based surveillance into automatic smoke/fire detection. These intelligent algorithms extract smoke's features based on video signal, and then classify the areas in the video frames as smoke or non-smoke.

Sometimes, the agent does not directly contact with the sensing element. The device pumps in some air sample and illuminates it with multiple wavelengths to get the absorption spectrum, which is normally done by an infrared spectroscopy. After that, the concentration and type of the agent can be identified when prior knowledge of this agent is given.

When sensors are not able to contact with the cloud in the sky, laser technology enables the remote sensing of cloud's properties. The concept of remote sensing is very close to spectroscopy. When the cloud is illuminated by a laser beam, the particles absorb the energy and emit fluorescent light, as well as "reflect" light back to the source (referred as backscatter). These lights are collected by the sensor. Then, microprocessor in sensor node could identify the received signal, and make a declaration of cloud. The back-scattered light wavelength is identical to the transmitted light [36], and the magnitude of the back-scattered light at the given range depends on the back-scatter coefficient of scatterers and the extinction coefficients of the scatterers along the path at that range [37]. The "fingerprint" of the fluorescent light can be an evidence of the species of particles in that cloud [38].

In battlefield applications, aerosol detection may also relate to bio-aerosol detection [39]. As the bio-aerosol released by a biological weapon is extremely dangerous to any biological unit in that area. It is necessary to detect and discriminate it as soon as possible. The detection and discrimination can be done by a LIDAR (Light Detection And Ranging) detector. Similar to the spectroscopy, it illuminates the bio-aerosol and collects back-scatters by a telescope. Therefore, the agent could be discriminated according to its spectrum. In research simulation, the bio-aerosol is often created by spread bacillus subtilis spores in the air. However, the reflection/extinction coefficients and absorption spectrum are closely related to the species of particles and different from one agent to another. To estimate the concentration of an aerosol based on received signals requires prior knowledge of the wavelength-dependent backscatter coefficients.

There are many challenges in outdoor environment for this method. First, in real environment, the received signals may be interfered by background radiation, noise and moving object. For example, the background radiation (sunshine or cloud reflection) will ruin the LIDAR signals. But it can be compensated by increasing the laser power, adding optical filter in front of the telescope and tracking the background radiation level. Second, the interference of moving objects in the sky like birds and balloons will have a very high reflection coefficient compared with gases/aerosol. Third, sensors are energy limited and vulnerable to intruders and they often malfunction or become unreliable. Fourth, the cloud plume is a diffusive target whose special concentration is random and the distributions are different from one place to another. The model have to capture the special variation of concentration of cloud [40]. Finally, a LIDAR detector with discrimination ability is very expensive so that it often performs passive detection and discrimination. Active detection of the bio-aerosol is taken by some cheaper sensors distributed in the battlefield. So this comes to our problem: detecting the bio-aerosol by unreliable sensors distributed in the environment with noise and interference.

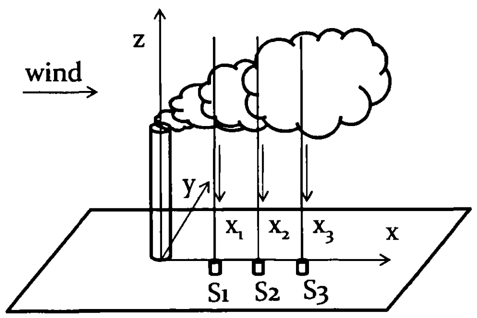
Signal can be processed by distributed processing method such as distributed average consensus (DAC) algorithms. The word consensus means each node would reach an agreement on the declaration of the target after the algorithm. In addition, each node only broadcast its local value until the algorithm converges [15]. This method can save much energy for nodes that heavily loaded in the data gathering.

5.2 Background and System Model

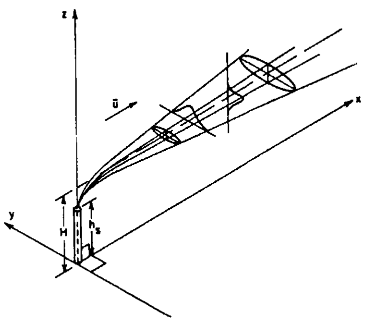
This section shows the system model, the sensor observation model and Gaussian plume model.

5.2.1 Cloud Detection Scenario

In the cloud detection scenario shown by Figure 5.1a, a source produces cloud in a fixed position with fixed power. A wind with time invariant velocity blows in parallel with the ground, which brings the cloud particles to the positive direction



(a) Cloud detection scenario



(b) Gaussian plume model.

Figure 5.1: Cloud detection system model

of the x-axis. With these parameters available, the Gaussian plume model [41][42] is used to describe the cloud concentrate at any position (x, y, z). On the ground, multiple sensors (numbered from 1 to L) aim to the plume perpendicularly to the ground plane and don't change their positions. The positions were chosen so that the laser beams can penetrate into the plume and backscatters are observed by the laser receivers.

5.2.2 Gaussian Plume Model of Diffusive Cloud

This section introduces the Gaussian plume model mentioned in some state-of-art applications. The Gaussian plume model is widely used in cloud plume concentration modeling since 1970's, which can describe the mean value of the cloud concentration at any position [43]. If we observe the cloud plume for a long time and take the average, the result is Gaussian plume model. However, in a real cloud plume, the concentration is actually a random process.

[40] gives a model of the concentration values C of pollutants to be emitted by point instantaneous source at height H, described by the normal (Gaussian) distribution

$$\begin{aligned} C(x,y,z,t) &= \frac{Q}{(2\pi)^{3/2}\sigma_x\sigma_y\sigma_z}\exp\left(\frac{-(x-ut)^2}{2\sigma_x^2}\right)\exp\left(\frac{-(y-vt)^2}{2\sigma_y^2}\right)\ldots && (5.2.1)\\ &\left(\exp\left(\frac{-(z-H-wt)^2}{2\sigma_z^2}\right)+\exp\left(\frac{-(z+H-wt)^2}{2\sigma_z^2}\right)\right) && (5.2.2)\end{aligned}$$

where t is the time, Q the source emission power, u, v, w are the orthogonal components of wind velocity, $\sigma_x, \sigma_y, \sigma_z$ are the horizontal and vertical dispersions, H the source height.

To describe the cloud emitted by a continuous point source, an integration form $t = 0$ to ∞ is taken for (5.2.1). So the model for continuous source is not related to the time t. The model after integration is called Gaussian plume model. To make things easy, assume the wind velocity components $v = 0, w = 0$. The Gaussian plume model changes into

$$C(x, y, z, H) = \int_0^\infty C(x, y, z, t)dt \quad (5.2.3)$$

$$= \frac{Q}{2\pi u \sigma_y \sigma_z} \exp\left(\frac{-y^2}{2\sigma_y^2}\right)\left(\exp\left(\frac{-(z-H)^2}{2\sigma_z^2}\right) + \exp\left(\frac{-(z+H)^2}{2\sigma_z^2}\right)\right) \quad (5.2.4)$$

where $\sigma_y = ax^b$, $\sigma_z = cx^d$, a, b, c, d are the coefficients which relate to the atmospheric stability.

The Gaussian plume model describes the mean concentration of cloud plume. However, especially designed for remote laser sensing, some other models are introduced in Section 5.3.

5.3 Modified Gaussian Plume Model For Laser Detection

The Gaussian plume model can describe the mean value of the cloud concentration. However, the cloud concentration of a real cloud plume is actually a random process, which can not captured by Gaussian plume model. In this section, first some other models are derived based on the diffusive equations to obtain the modified Gaussian plume model especially for laser detection. Second, with the 3D animation technology [19], simulated cloud plume is implemented to show its fluid dynamics and turbulence properties. In addition, a bunch of cloud plumes are generated for testing and detection.

5.3.1 Integration of Cloud Along Laser Beam

Because laser emitted by optical sensor is penetrating the cloud, the received signal at each sensor can be simplified to be proportional to the integration of concentration along the line of laser. Therefore, the Gaussian plume model needs some modifications, based on the same assumption.

Similar to the heat diffusion model, the cloud's diffusive behavior can be described by the 3-dimensional partial differential equation.

$$\frac{\partial C}{\partial t} = D_x \frac{\partial^2 C}{\partial^2 x} + D_y \frac{\partial^2 C}{\partial^2 y} + D_z \frac{\partial^2 C}{\partial^2 z} \tag{5.3.1}$$

where C is cloud concentration, and D_x, D_y, D_z are the diffusion coefficients along three axis, respectively.

This equation indicates that the rate of density change is proportional to the curvature of cloud concentration. The density increases where curvature is positive and decreases where it is negative. If the cloud is released instantaneously at a single point, the spatial distribution will be a 3-dimensional normal distribution.

In consideration of the isotropic diffusion case, the diffusion equation can be simplified, which means $D_x = D_y = D_z = D$. Assume a point instantaneous source, located at the origin starts to release cloud at time $t = 0$, the solution to (5.3.1) is

$$C(x, y, z, t) = \frac{Q}{(4\pi Dt)^{3/2}} exp\left(-\frac{x^2 + y^2 + z^2}{4Dt}\right) \tag{5.3.2}$$

where Q is the power of the point source. This solution can be verified by taking partial derivative for both sides.

In addition, if the surrounding air is assumed to be moving towards the positive direction of x-coordinate in a constant velocity u. The model changes into

$$C(x, y, z, t) = \frac{Q}{(4\pi Dt)^{3/2}} \cdot exp\left(\frac{(x - ut)^2 + y^2 + z^2}{4Dt}\right) \tag{5.3.3}$$

For point and continuous source at origin, an integration from $t = 0$ to T is taken to find the concentration distribution. If $T \to \infty$, the concentration model for continuous source evolves into

$$C(x, y, z) \quad = \quad \int_0^\infty C(x, y, z, t)dt \tag{5.3.4}$$

This integration is very hard to find without the help of computer, as the denominator in (5.3.3) contains $t^{\frac{3}{2}}$. Some later research of atmospheric diffusion has found the analytical integration [41].

As the laser penetrates the cloud, the received signal at each sensor can be simplified to be proportional to the integration of concentration along the line of laser. Therefore, the received signal

$$S(x, y) = \int_{-\infty}^{\infty} C(x, y, z)dz = \int_{-\infty}^{\infty} \int_0^\infty C(x, y, z, t)dtdz \tag{5.3.5}$$

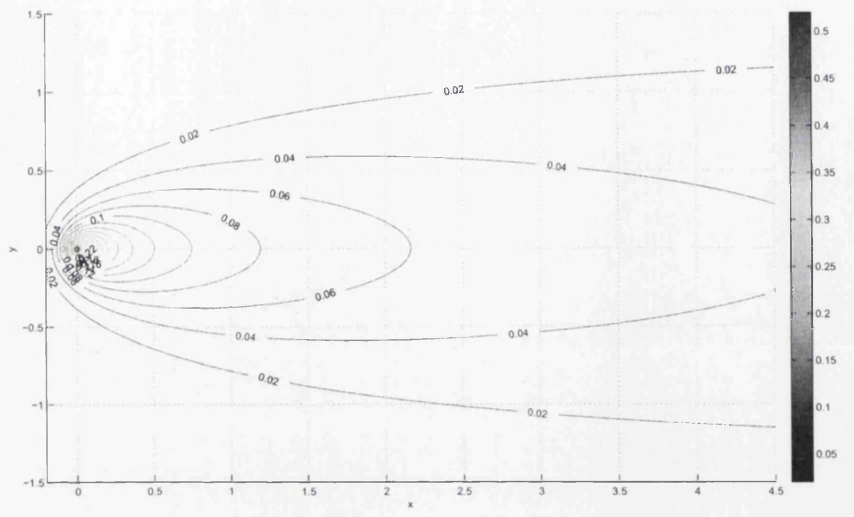


Figure 5.2: 2-D Gaussian plume model modified for laser detection

Since $\int_{-\infty}^{\infty}\int_0^{\infty}|C(x,y,z,t)|\,dtdz<\infty$, the two integrations can be swapped. thus, we have

$$\int_{-\infty}^{\infty}\int_0^{\infty}C(x,y,z,t)dtdz=\int_0^{\infty}\int_{-\infty}^{\infty}C(x,y,z,t)dzdt \tag{5.3.6}$$

Therefore, the finial model of received signal at point (x,y) can be given by

$$S(x,y)=\frac{Q}{2\pi D}\cdot exp\left(\frac{xu}{2D}\right)\cdot\int_0^{\infty}\frac{1}{\sqrt{\tau^2+1}}\cos(\frac{u\tau\sqrt{x^2+y^2}}{2D})d\tau \tag{5.3.7}$$

or

$$S(x,y)=\frac{Q}{2\pi D}\cdot exp\left(\frac{xu}{2D}\right)\cdot K_0\left(\frac{u\sqrt{x^2+y^2}}{2D}\right) \tag{5.3.8}$$

where $K_0(z)$ is the special case of modified Bessel function of the second kind $K_n(z)$. $K_0(z)$ is simplified to

$$K_0(z)=\int_0^{\infty}\cos(z\cdot\sinh\tau)d\tau=\int_0^{\infty}\frac{\cos(z\cdot\tau)}{\sqrt{\tau^2+1}}d\tau \tag{5.3.9}$$

The concentration distribution is shown in Figure 5.2.

5.3.2 Simulated Cloud by Fluid Dynamics

In 3D fluid simulation [19], the cloud model is a 3D space which is divided into tiny cells where fluid dynamics and wind turbulence is considered. The cloud plume will be more like real with the help of this technology.

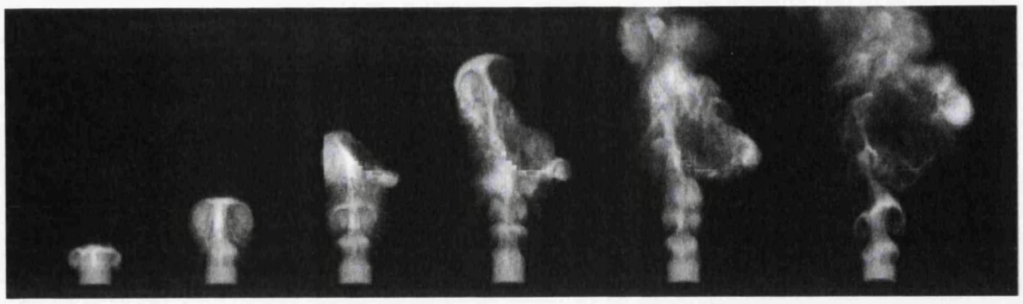


Figure 5.3: Cloud simulation frame sequence. 192 × 256 pixels in each frame

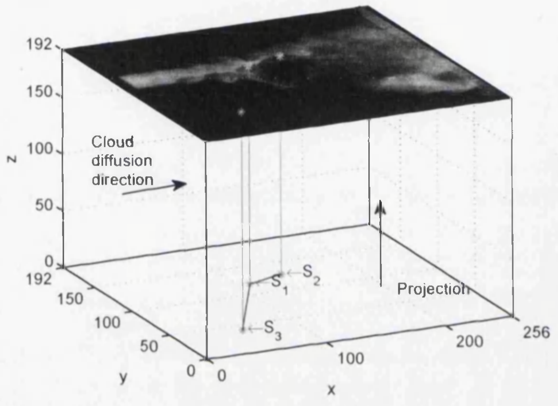


Figure 5.4: Cloud concentration and Sensor's observation. Images frame are obtained by 3D raw data projection

The Figure 5.3 shows the frame sequence obtained by the fluid simulation. Each frame image is obtained by taking integration of the 3D raw data along z-axis, as shown in Figure 5.4. This is to simulate the effect of laser beam penetration through cloud. As the received light magnitude is the integration of backscatter along the laser beam, pixels value in the frame image is proportional to the magnitude of received light. In this simulation, the pixel values in these frames are normalized by dividing them with the maximum pixel value in these frames.

5.4 Distributed Cloud Detection Based on DAC Algorithms

In the applications of environment surveillance and monitoring, sensor networks performs the data gathering of spatially distributed sources, and the collaborative signal processing. Processing the local acquired signals using a scalable algorithm is a fundamental problem in a sensor network [30].

Cloud declaration is normally made through a hypothesis test. When new observation is acquired, the decision of cloud existence is made based on the ML or MAP decision rule [5], which need the probability density function of all the sensor's observation and calculate the log likelihood ratio.

Generally, there are two options for multiple sensors signal processing.

Firstly, it is the centralized processing. This requires the network containing a fusion center, and all sensor's data needs to be transmitted to the fusion center and processed. At the same time, a hypothesis test based on the ML (Maximum Likelihood), MAP (Maximum A Posteriori) or Bayesian decision rule will be carried out at the fusion center. Normally the global log likelihood ratio (G-LLR) is calculated and compared with a threshold to make the decision. Besides, an optimal data fusion scheme is proposed in [5], the decision is made by an optimal linear combination of local decisions of all sensors.

Secondly, the global LLR can be calculated without a fusion center by distributed signal processing. In the consideration of reliability, survivability, and range of coverage, there is an increasing interest in employing multiple sensors for these applications [5].

In this section, we will consider a distributed detection problem in wireless sensor network without the fusion center. Then, there will be an introduction about consensus based approach in the distributed data fusion and decision making, in the case of each sensor acquires a scale value of an unknown parameter. However, [44] discussed the case when each sensor acquires a vector of unknown parameters and the signal is mixed with joint Gaussian white noise, and proposed a more sophisticated data fusion scheme. We will show it later.

The detection method should be separated into two stages, training and detection. The training stage is to build the probability density function (PDF) for background radiation and cloud reflection signals (modeled by a Gaussian mixture model) based on their own observation distributively.

5.4.1 Received Signal Model

As we know, the cloud concentration variation is caused by turbulence flow, which is a random process which brings the cloud particles more far away than the molecular motion. Therefore, the associated sensor's detection at a given position is a random process with mean and variance.

Suppose the sensor's observation has the following form:

$$x_l = \begin{cases} \mu_{l,0} + n_{l,0} & \text{if no cloud exists} \\ \mu_{l,1} + n_{l,1} & \text{if cloud exists} \end{cases} \tag{5.4.1}$$

where $\mu_{l,m}$ is the mean value of x_l depending on hypothesis m; $n_{l,m}$ is the noise of x_l depending on hypothesis m and $n_{l,m} \sim \mathcal{N}(0, \sigma_m)$. Actually, $\mu_{l,0}$ and $\sigma_{l,0}$ denote the mean and variance of background noise. Besides, we have $\mu_{l,1} = \mu_{l,0} + \mu_t$, $n_{l,1} = n_{l,0} + n_t$, where μ_t and n_t denote the signal of mean and fluctuation of cloud concentration due to wind turbulence. These parameters is estimated or calculated expectation-maximization algorithm in the training stage.

This received signal model doesn't consider interference of moving objects or be covered by obstacles. To deal with this problem, sensors may need to build the Gaussian mixture model, which was introduced in [45]. Expectation-Maximization(EM) algorithm [46] can be adopted to build the Gaussian mixture mode.

5.4.2 Hypothesis Testing

Considering the binary hypothesis testing problem with the following two hypotheses

1. target is absent.

2. target is present.

The prior probability of these two hypotheses is denoted by $P(H_m) = P_m = \frac{1}{2}, m = 0, 1$.

Suppose the observation of all sensors $x_1, ..., x_L$ is available (for example, gathered by a fusion center), we can have the global log likelihood ratio (G-LLR) test given by

$$LLR(x_1, ..., x_L) = log \frac{f(x_1, ..., x_L|H_1)}{f(x_1, ..., x_L|H_0)} \underset{H_0}{\overset{H_1}{\gtrless}} \log \frac{P(H_o)}{P(H_1)} \tag{5.4.2}$$

where $f(x_1, ..., x_L|H_m)$ is the likelihood function of H_m.

To drive and simplify the expression of global log likelihood ratio, sensor's observation is described by the model in Section 5.4.1. Let

$$\mathbf{x} = [x_1, \ldots, x_L]^{\mathrm{T}} \tag{5.4.3}$$

$$\mathbf{n}_m = [n_{1,m}, \ldots, n_{L,m}]^{\mathrm{T}} \tag{5.4.4}$$

$$\mathbf{u}_m = [\mu_{1,m}, \ldots, \mu_{L,m}]^{\mathrm{T}}, \; m = 0, 1. \tag{5.4.5}$$

If $\mathbf{n}_m \sim \mathcal{N}(0, \Sigma_m)$, G-LLR is given by

$$\begin{aligned} LLR(\mathbf{x}) &= \frac{1}{2}(\mathbf{x}-\mathbf{u}_1)^{\mathrm{T}}\left(\Sigma_0^{-1}-\Sigma_1^{-1}\right)(\mathbf{x}-\mathbf{u}_0) + \frac{1}{2}log\left(\frac{|\Sigma_0|}{|\Sigma_1|}\right) & (5.4.6)\\ &= \left(\mathbf{u}_1^{\mathrm{T}}\Sigma_1^{-1} - \mathbf{u}_0^{\mathrm{T}}\Sigma_0^{-1}\right)\mathbf{x} + \frac{1}{2}\left[log\,|\Sigma_0| - log\,|\Sigma_1|\right] - \\ &\quad \frac{1}{2}\left(\mathbf{u}_1^{\mathrm{T}}\Sigma_1^{-1}\mathbf{u}_1 - \mathbf{u}_0^{\mathrm{T}}\Sigma_0^{-1}\mathbf{u}_0\right) - \frac{1}{2}\left[\mathbf{x}^{\mathrm{T}}\left(\Sigma_1^{-1}-\Sigma_0^{-1}\right)\mathbf{x}\right] & (5.4.7)\\ &= \sum_{l=1}^{L} w_l x_l + C - \frac{1}{2}\left[\mathbf{x}^{\mathrm{T}}\left(\Sigma_1^{-1}-\Sigma_0^{-1}\right)\mathbf{x}\right], & (5.4.8) \end{aligned}$$

where w_l denotes the lth component of $\mathbf{u}_1^{\mathrm{T}}\Sigma_1^{-1} - \mathbf{u}_0^{\mathrm{T}}\Sigma_0^{-1}$, and $C = \frac{1}{2}\left[log\left(|\Sigma_0|\right) - log(|\Sigma_1|)\right] - \frac{1}{2}\left(\mathbf{u}_1^{\mathrm{T}}\Sigma_1^{-1}\mathbf{u}_1 - \mathbf{u}_0^{\mathrm{T}}\Sigma_0^{-1}\mathbf{u}_0\right)$.

5.4.3 Distributed Calculation of G-LLR

A common approximation is to assume $\Sigma_1 = diag\left(\sigma_{1,1}^2, \sigma_{2,1}^2, \ldots, \sigma_{L,1}^2\right)$ and $\Sigma_0 = \sigma_0^2 I$, (5.4.8) can be reduced to

$$LLR(\mathbf{x}) = \sum_{l=1}^{L} w_l x_l + C - \frac{1}{2}\sum_{l=1}^{L}\left(\sigma_{1,1}^{-2} - \sigma_0^{-2}\right)x_l^2 \tag{5.4.9}$$

which can be calculated by two instances of DAC algorithms. Thus, the observation of sensors is assumed to be independent from one to another. Sensors will build the Gaussian signal model separately.

Provided that each sensor knows the weight w_l,$\sigma_{1,1}^2$ and σ_0^2, (5.4.9) states that G-LLR is equal to the weighted sum of x_l and x_l^2 plus the constant C. Therefore, the distributed average consensus (DAC) algorithms can be used to calculate it. Actually, the constant C changes the threshold of the hypothesis testing and can be subtracted from both sides of hypothesis testing equation. Therefore, we modify the hypothesis testing into

$$LLR(\mathbf{x}) = \sum_{l=1}^{L} w_l x_l - \frac{1}{2}\sum_{l=1}^{L}\left(\sigma_{1,1}^{-2} - \sigma_0^{-2}\right)x_l^2 \underset{H_0}{\overset{H_1}{\gtrless}} \frac{P(H_o)}{P(H_1)} - C \tag{5.4.10}$$

where w_l is the l^{th} component of $\left(\mathbf{u}_1^{\mathrm{T}}\Sigma_1^{-1} - \mathbf{u}_0^{\mathrm{T}}\Sigma_0^{-1}\right)$.

The algorithm is as follows: First, each sensor calculates its local LLR individually; Then, each sensors updates its local LLR in the DAC iterations until it converges to the consensus value; Finally, after the algorithm converges, the global LLR is obtained by multiplying the average local LLRs with the number of sensors in the network. When G-LLR is available, cloud declaration could be made based on the ML or MAP decision rule.

Challenges of Distributed Calculation of G-LLR

In a network without fusion center, it is desirable to find $LLR(\mathbf{x})$ in a distributed manner. However, the quadratic form in (5.4.8) contains high-order components of $\mathbf{x}$. To calculate the global LLR, $\mathbf{\Sigma}_1^{-1}$, Σ_0^{-1}, $\mathbf{u}_1$ and $\mathbf{u}_0$ should be known. Specifically, the entries of $\left(\mathbf{\Sigma}_1^{-1} - \Sigma_0^{-1}\right)$ should be known to nodes.

In a centralized method, $\mathbf{\Sigma}_1$ and $\mathbf{u}_1$ can be obtained by expectation-maximization (EM) algorithm. But it seems not possible to calculate $\mathbf{\Sigma}_1^{-1}$ without global information.

When no cloud exists, the sensor's signal is only caused by atmospheric backscatter and noise. It is described by joint Gaussian distribution $\mathcal{N}(\mathbf{u}_0, \Sigma_0)$. If these distributions are independent and identical, we have

$$\mathbf{\Sigma}_0 = \sigma_0^2 I. \tag{5.4.11}$$

On the contrary, $\mathbf{\Sigma}_1$ can't be written in the same form. At first, the cloud concentration fluctuating may be different from one sensor to another. Second, the fluctuating of cloud concentration is correlated, especially for the sensors close to each other. As shown in Figure 5.7a, the correlation is obvious when sensor's observation is shifted with the right time delay. The correlation is the major challenge to invert $\mathbf{\Sigma}_1$.

To make it possible to calculate G-LLR using DAC algorithm, we assume that correlations only exist between sensors that are located very close to each other. If node i and node j are not neighbors, their signals have low correlation and are approximately not correlated. Therefore, each node can only store the entries of $\left(\mathbf{\Sigma}_1^{-1} - \Sigma_0^{-1}\right)$ that related to itself and its neighbors. The last term in (5.4.8) can be written into

$$\frac{1}{2}\left[\mathbf{x}^{\mathrm{T}}\left(\mathbf{\Sigma}_1^{-1} - \Sigma_0^{-1}\right)\mathbf{x}\right] = \frac{1}{2}\sum_{i=1}^{L}\sum_{j=1}^{L} c_{ij}x_i x_j \tag{5.4.12}$$

where $[c_{ij}] = \left(\mathbf{\Sigma}_1^{-1} - \Sigma_0^{-1}\right)$, $c_{ij} = 0$, if $v_i \notin \mathcal{N}_i$.

We can find the value of (5.4.8) by the following algorithm:

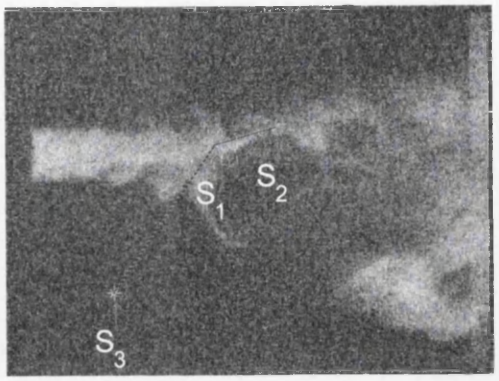


Figure 5.5: Sensor's observation impaired by Gaussian white noise

1. Assume for a node i, it stores the value x_i and c_{ij}. As the matrix is symmetric and the entries $c_{ij} = c_{ji}$, node i sends x_i and receives x_j from all node j in the neighbors set $\mathcal{N}_i$ to compute the value

$$v_i = \sum_{j \in \mathcal{N}_i} c_{ij} x_i x_j + c_{ii} x_i^2 \tag{5.4.13}$$

2. Initialize a DAC algorithms with local values v_i until they converge to the average $\bar{v} = \frac{1}{L} \sum_{i=1}^{L} v_i$.

3. Start another DAC algorithm to find $\bar{u} = \sum_{l=1}^{L} w_l x_l$.

4. G-LLR is equal to $\bar{u} + \bar{v} + C$ multiplied by the number of sensors in the network.

5.5 Simulation

The distributed cloud detection simulation consists of two stages, training and detection. In the training stage, sensors are trained to build the Gaussian mixture models, which are important to calculate the L-LLR. After the training, when a new observation comes, all sensors calculate L-LLRs and take them into DAC iterations to obtain G-LLR. The decision of cloud existence can be made distributively once the algorithm converges.

The 3D cloud animation is simulated to generate a bunch of cloud plumes. If we observe the cloud for a long time and take the average, the result is Gaussian plume model.

The cloud animation is simulated several times to generate enough data, which is divided into two groups for both training and testing of system performance.

Because the turbulence flow is a random process, the data generated each time is different from the others in the cloud concentration distribution, as well as the cloud particles moving track. This provides a very practical testing environment for the system.

Some other considerations for this simulation are that sensors are randomly distributed in the sensing area and sensor's detections are impaired by Gaussian white noise, which will be introduced in the following.

5.5.1 Sensor's Observation

Gaussian white noise impairs the received signal when sensor observes the cloud concentration. And the source of noise includes: (i) the external noise, arising from the incidence of radiation at the detector both from laser scattering and from the background; and (ii) the internal noise, arising from fluctuations in the detector dark current and thermal noise in the detector load resistor [37]. These noise is additive, and the overall noise is treated as a Gaussian white noise denoted by $\mathcal{N}(u_0, \sigma_0)$. In this simulation, these parameters are chosen as $u_0 = 0.3$ and $\sigma_0 = 0.1$, shown in Figure 5.5.

Again, sensors are randomly distributed in the sensing area with the uniform distribution. The sensing area in this simulation is defined by pixels in the frame which has $x > D$, where D is the distance from the cloud source on the downwind direction. After they are distributed, to get ready for the DAC algorithm, all the nodes will automatically connect their neighbors to obtain a network. DAC algorithm is used to obtain G-LLR without copying L-LLRs to all sensor nodes in the network. Once G-LLR is available, cloud declaration could be made based on the ML or MAP decision rule.

Here we give an example, three sensors are distributed as shown in Figure 5.5. They build a Gaussian mixture model by processing the training data. Here we choose $K = 1$. It can be 2 or more, depending on how many components of the noise in the environment. Then, the testing data generated by another cloud animation is passed through all the sensors frame by frame. As shown in Figure 5.6, L-LLRs for the two sensors and G-LLR is given. Before frame 47, all the sensors have no contact with cloud, and only noise is presented in each sensor. Only after sensor S_1 has its cloud contact, G-LLR raises to the first stage. After S_2 has its contact of cloud at frame 63, G-LLR is increased to an even high level, which is larger than the threshold and a strong evidence of the cloud existence. Because S_3 has no chance to contact with the cloud, its L-LLR is actually less sensitive to the sensors signal. Thus, its L-LLR has very low contribution to

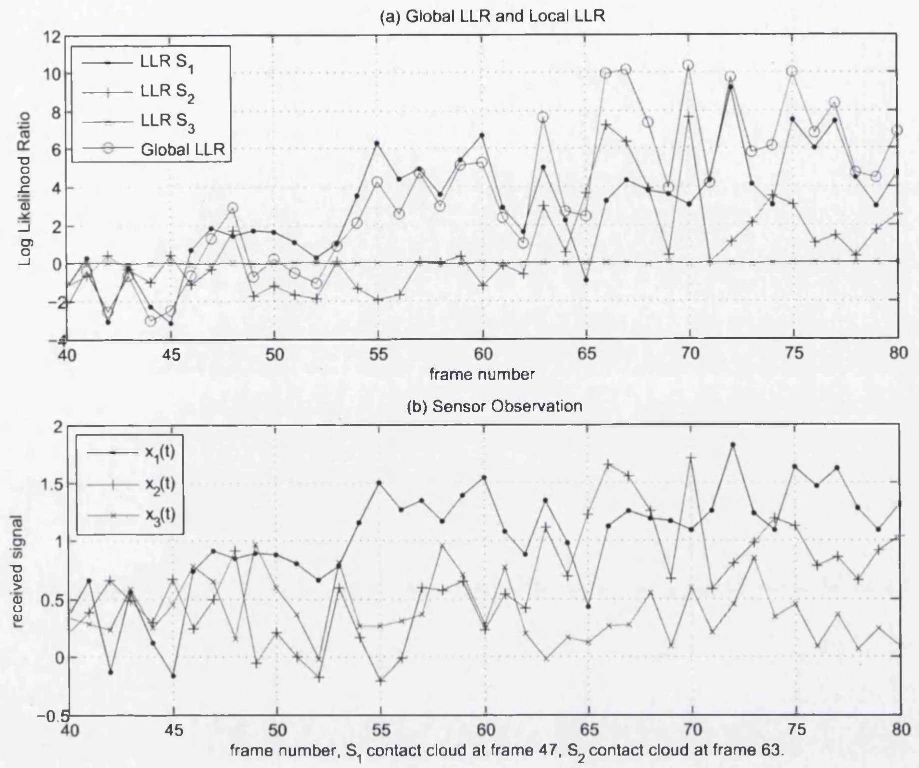


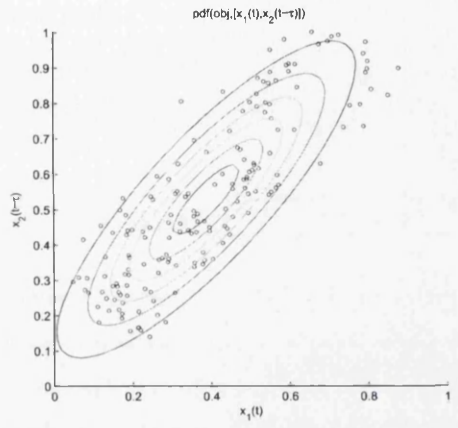
Figure 5.6: (a) Global LLR vs. Local LLRs and (b) Sensor's Observation (Only frame 40 to 80 are shown)

G-LLR.

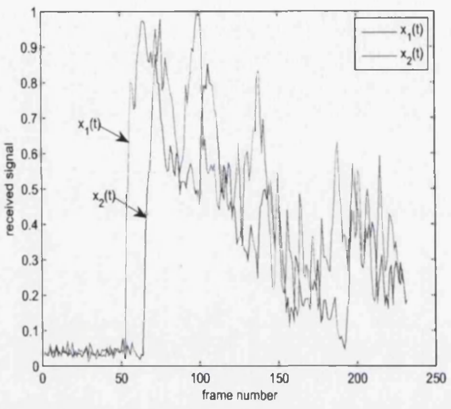
Another interesting thing is that, if two sensors $S1$ and $S2$ are put close to each other, especially when they are very nearly located on the cloud particle's moving path, their observation have high correlation. For example in Figure 5.5 and Figure 5.7a. The time delay τ is simply calculated by local wind velocity and sensor's distance. The cross-correlation can be another feature of the moving cloud.

5.5.2 Performance of Cloud Detection Sensor Network

Simulation is running for hundreds of times to give the average performance for different number of sensors in the network. The sensor's positions are chosen randomly in the area where $x > 128$ (x is the index of the pixel). Figure 5.8 gives the relative operating characteristic (Also known as a ROC curve) of the detection system with different sensors numbers. The curve is represented by plotting the fraction of true detection out of the cloud exist vs. the fraction of false alarm out of the cloud not exist. When there is only one sensor in operation, a special case



(a) The distribution of tuples $[x_1(t), x_2(t-\tau)]$, τ is the time delay



(b) Two sensors observation $x_1(t), x_2(t)$

Figure 5.7: Correlation of sensor's observation

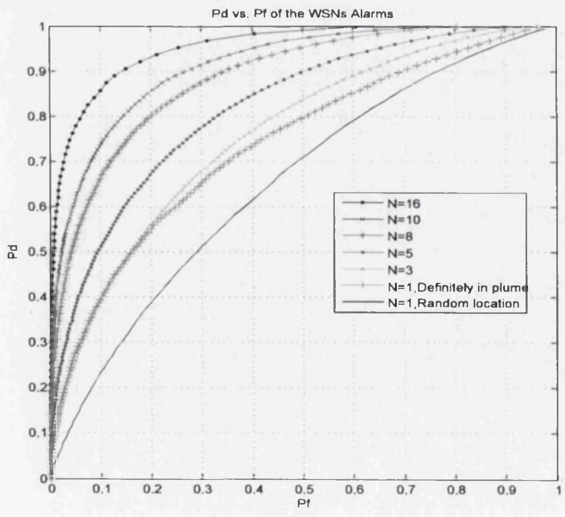


Figure 5.8: Detection relative operating characteristic for different sensor numbers

is that the position is chosen by hand to make sure the sensor can contact with the cloud plume. With this assurance, the one sensor detecting system could have performance close to the three sensor detecting system randomly distributed. The advantage of using distributed detection with multiple sensors is obvious. As the detecting system with more sensors is more reliable to noise which leads to a higher performance.

5.6 Summary

In this chapter, we introduced a distributed detection method using wireless sensor networks and the DAC algorithms. First, the hypothesis testing based on the ML or MAP decision rule is introduced. In the outdoor environment, signal of an individual sensor might be corrupted by Gaussian noise or moving object with high reflection coefficient, which might raise the false alarm in high possibility. Because multiple sensors detection has better performance, it was adopted in the cloud detection and the expectation-maximization algorithm is used to build the joint Gaussian model of background noise and cloud backscatter. Thus, interferences to a few sensors in the network are less likely to raise the false alarm. Second, by assuming sensors signals are independent, global log likelihood ratio is the average of local log likelihood ratio. The global log likelihood ratio in the hypothesis testing is calculated by the DAC algorithms. Each sensor calculates the local log likelihood ratio and substitutes it into the DAC iterations as the initial local value. The global log likelihood ratio is available to each sensor once the DAC algorithms converges.

Chapter 6

Conclusion and Future Work

This section gives a summary of this thesis and direction for future research.

6.1 Conclusion

In this thesis, at first asymptotic and non-asymptotic DAC algorithms are reviewed. The asymptotic algorithmsfirst-order DAC are robust against topology changes and their optimization needs to know the network topology, which is very difficult to obtain for any individual node and can not change during the optimization. Non-asymptotic algorithms can find the average faster in a time-invariant network. They use a finite impulse response filter to estimate the consensus value. However, the filter estimation is not reliable and will be interrupted once the topology is changed, because outdated information will lead to a wrong answer of the filter. Therefore, to choose the suitable DAC algorithm, it depends on the network properties.

Second, in Chapter 3, a generalized finite-time DAC algorithm is presented. Compared to the previous version of finite-time DAC, the number of iteration can be reduced to $2n$, where n is the number of nodes in the network. In addition, the eigenvalues of the associated weight matrix are not required prior to the algorithm. Actually, the number of iterations can be further reduced to the number of distinct eigenvalues of the associated weight matrix.

Third, in Chapter 4, we proposed a distributed real-time optimization method to increase the convergence rate of asymptotic DAC algorithms. As stated in Chapter 4, the optimal parameters are only related to eigenvalues of the Laplacian matrix associated to the network. Therefore, the key of the optimization algorithm is to distributively estimate the eigenvalues. However, numerical errors of these parameters due to quantization can decline the algorithm performance.

To mitigate this effect, we introduce a numerical technique to find a least mean square solution. If floating point number in double format is used and the network size is smaller than 32 nodes, the numerical errors of estimated parameters after mitigation will only slightly decline the performance of higher-order DAC. first-order DACsecond-order DACthird-order DAC Otherwise, the numerical errors will be too large even after mitigation. Generally, parameters optimized for the old network are not optimal for the new network or even not located in the convergence region. So the DAC algorithm must be reinitialized with convergent parameters to continue the consensus process. Once the network topology is stable for a certain time, the optimal parameters are estimated again.

Finally, we introduced a distributed detection of cloud plume using wireless sensor networks and the DAC algorithms in Chapter 5. First, the hypothesis testing based on the ML or MAP decision rule is introduced. In the outdoor environment, signal of an individual sensor might be corrupted by Gaussian noise or interfered by moving object. Therefore, single sensor detection is unreliable and might raise the false alarm very often. Multiple sensors detection is adopted to improve the performance of cloud detection. Thus, interferences to a few sensors in the network are less likely to raise the false alarm. Second, if we assume sensors signals are independent, the global log likelihood ratio in the hypothesis testing can be calculated by the DAC algorithms.

6.2 Future Work

In this thesis, we introduced several DAC algorithms. It seems to be impossible to find an algorithm robust against topology changes and faster than finite-time DAC in a distributed and dynamic network. It is very demanding for any individual node to know the network topology.

Therefore, further research could be carried out to optimize the existing algorithms and make some modifications according to the applications.

First, the distributed real-time optimization to DAC can be applied on a distributed system. Before that, a consensus protocol should be developed and implemented. We are supposed to deal with some of the problems in practice, such as link failure, channel noise, time-delay and asynchronous communication.

Second, the distributed real-time optimization to DAC might be able to applied in a dynamic network. As stated in Chapter 4, the optimal parameters is only related to eigenvalues of the Laplacian matrix associated to the network. Therefore, optimized parameters for the old network are usually not optimal or not located in the convergence region for the new network. However, simulation results show that constant first-order DAC algorithm and second-order DAC algorithm can maintain the convergence in most of the time, even old parameters are used. Some research should be carried out to find out the conditions to maintain the convergence.

Third, to avoid complex analysis, the network graph is assumed to be symmetric in the analysis. However, some algorithms such as the finite-time DAC algorithm and the eigenvalue estimation algorithm, can be generalized to the case when the network in unsymmetrical. Therefore, the distributed real-time optimization might be able to be generalized the unsymmetrical network.

Finally, for the application of cloud detection, the correlation of sensor signals can be used to improve the performance of the cloud detection. If sensors are located in short distance in the plume, their detections are correlated. In addition, interference due to Gaussian noise or moving object has very low correlation between sensors. Therefore, correlation can be an important property of the cloud plume. Besides, a modified Gaussian plume model need to be developed to capture the the correlation and random properties of the cloud of the concentration at different position. mean value. In addition, the parameters of cloud plume should be should be treated as unknown random variable, such as the position the diffusion coefficient, wind speed and so on. The DAC algorithms need to be modified to capture these changes. At the same time sensors need to be able to estimate these parameters from their observation.

Appendix A

Graph and Matrix Theory Review

In this section, some basic concepts of the graph theory and matrix theory will be introduced. They are used in the analysis of convergence or performance of consensus algorithms. Because consensus algorithms actually relates to a matrix iteration, it is necessary to introduce some of these theorems. For full information about matrix theory, see [21], and the work [20] states more details about Laplacian matrix. However, some useful properties of Laplacian matrix will to be introduced here.

A.1 Basic Concepts

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A})$ be a graph with n nodes. The in-degree and out-degree of node i are defined by:

$$D_{in}(i) = \sum_{i=1}^{n} a_{i,j} \tag{A.1.1}$$

$$D_{out}(i) = \sum_{j=1}^{n} a_{i,j} \tag{A.1.2}$$

where $a_{i,j}$ is the elements of matrix $\mathcal{A}$. This definition states that in-degree of node i is the i^{th} column sum of matrix $\mathcal{A}$ and the out-degree of node i is the i^{th} row sum of matrix $\mathcal{A}$. And the graph Laplacian matrix $\mathcal{L}$ induced by the $\mathcal{G}$ is the same as defined before, see (2.1.10). In addition, we can find the relationship of $\mathcal{L}$ and $\mathcal{A}$.

$$\mathcal{L} = \Delta - \mathcal{A} \tag{A.1.3}$$

where Δ is a diagonal matrix $\Delta = [\Delta_{i,j}]$, $\Delta_{i,j} = 0$ for all $i \neq j$ and $\Delta_{ii} = D_{out}(i)$.

Note that we assume the diagonal elements $a_{i,i}$ of matrix $\mathcal{A}$ equal to zero for all i. Thus, the Laplacian matrix is only dependent on the off-diagonal elements of $\mathcal{A}$. Moreover, if we assume matrix $\mathcal{A}$ is non-negative, we can benefit from the properties of non-negative matrix, and use them in optimization of convergence rate of consensus algorithm.

A.2 Irreducibility and Strong Connected Graph

For an undirected graph, the consensus state $\mathbf{x}^*$ can be achieved if and only if the graph is connected. (Note that the consensus is a stable state of the system dynamic. For the average consensus problem the consensus state is a state where all the network nodes converge to the global average.) But for the directed graph, the consensus state can be achieved if and only if the graph is strongly connected.

A directed graph is strongly connected if and only if for any two distinct nodes i and j, there exists a path that follows the direction of the edges and connects i and j on the digraph.

Definition A.2.1. For $n > 1$, an $n \times n$ matrix $A \in R^{n \times n}$ is reducible if there exists an $n \times n$ permutation matrix P such that PAP^T is in block upper triangular form.

$$PAP^T = \begin{bmatrix} A_{1,1} & A_{1,2,} \\ O & A_{2,2} \end{bmatrix} \tag{A.2.1}$$

where $A_{1,1}$ is a $r \times r$ submatrix and $A_{2,2}$ is an $(n-r) \times (n-r)$ submatrix, and O is a null matrix, $1 \leq r < n$. If no such a permutation matrix exists, matrix A is irreducible. If $n = 1$, then A is reducible if $A = 0$, and irreducible otherwise.

The relationship of the irreducible property of matrix A and the strong connected property of directed graph $\mathcal{G}(A)$ is stated by the following theorem.

Theorem A.2.1. *An $n \times n$ complex matrix $A \in C^{n \times n}$ is irreducible if and only if its directed graph $\mathcal{G}(A)$ is strongly connected. [21]*

The proof of this theorem is obvious. If a graph is strongly connected, all the off diagonal elements of graph matrix A cannot be vanished by matrix permutation. Therefore, matrix A doesn't exists the block upper triangular form as given in Def. A.2.1.

A.3 Spectral Radius of a Matrix

Spectral radius of a matrix is one of the basic concepts in the matrix iteration theory. It is defined by the largest eigenvalue of the matrix. The matrix iteration is very useful in many applications, denoted by $A, A^2, A^3, \ldots$. The power sequence is said to be convergent, if and only if $\lim_{k\to\infty} A^k = O$, where O is a zero matrix with all zero entries. The following theorem states that the convergent property is strongly connected with the spectral radius.

Theorem A.3.1. *[21] If $A \in C^{n\times n}$ is an $n \times n$ complex matrix, then A is convergent if and only if $\rho(A) < 1$.*

Proof. The proof uses the Jordan form of a matrix. For any matrix $A \in C^{n\times n}$, there exists a nonsingular $n \times n$ matrix T, such that A reduces into the Jordan normal form

$$T^{-1}AT = J = \begin{bmatrix} J_1 & & O \\ & J_2 & \\ O & & \ddots \; J_m \end{bmatrix} \tag{A.3.1}$$

where each Jordan block J_i is a $r_i \times r_i$ submatrix in the form

$$J_i = \begin{bmatrix} \lambda_i & 1 & & & \\ & \lambda_i & 1 & & \\ & & \lambda_i & \ddots & \\ & & & \ddots & 1 \\ & & & & \lambda_i \end{bmatrix} \tag{A.3.2}$$

Thus, matrices J and A are similar and have the same eigenvalues $\lambda_i, i = 1, \ldots m$

A direct computation of the power iteration of matrix A will give us the following equation

$$A^k = TJ^kT^{-1} = T \begin{bmatrix} J_1^k & & O \\ & J_2^k & \\ O & & \ddots \; J_m^k \end{bmatrix} T^{-1}$$

because the property of Jordan block, the power of each Jordan block will have the form

$$J_i^2 = \begin{bmatrix} \lambda_i^2 & 2\lambda_i & 1 & & \\ & \lambda_i^2 & 2\lambda_i & \ddots & \\ & & \lambda_i^2 & \ddots & 1 \\ & & & \ddots & 2\lambda_i \\ & & & & \lambda_i^2 \end{bmatrix}, \text{if } r_i \geq 3$$

and more generally, let $J_i^k = [c_{m,n}(i,k)]$, $1 \le m,\, n \le r_i$, and it has

$$c_{m,n}(i,k) = \begin{cases} 0 & n < m \\ \begin{pmatrix} k \\ n-m \end{pmatrix} \lambda_i^{k-n+m} & m \le n \le \min(r_i, k+m) \\ 0 & k+m < n < r_i \end{cases}$$

Since $\rho(A) < 1$, and matrix J shares the same eigenvalue with A, $|\lambda_i| < 1$. This leads to $\lim_{k\to\infty} c_{m,n}(i,k) = 0$, for all $1 \le m \le r_i, 1 \le n \le r_i$, so that the each Jordan block is convergent. Therefore, the matrix iteration $A^k = TJ^kT^{-1}$ is also convergent. This completes the proof. □

We give the proof of Theorem A.3.1 here as it will be very useful in the proof of an convergence conditions theorem for distributed consensus algorithms, see Section 2.3.1. At the same time, the Jordan normal form weight matrix $W^k = TJ^kT^{-1}$ gives the local value vector $\mathbf{x}(k) = W^k\mathbf{x}(0)$ another expression in terms of eigenvalues and eigenvectors, which reflects the basic ideas of the finite-time consensus algorithms. This will be introduced in Section 2.4.

A.4 Gerschgorin's Theorem

The calculation of eigenvalues of a matrix A involves determination of the matrix $\lambda I - A$ and solving a high order polynomial equation. In some situations, for example, when the matrix dimension is very large, it is very difficult to determine the spectral radius precisely. However, the following theorem of Gerschgorin provides an upper bound of the spectral radius.

Theorem A.4.1. *[47] Let $A = (a_{i,j})$ be an arbitrary $n \times n$ matrix. Denote the*

$$d_i = \sum_{j=1, j\ne i}^{n} |a_{i,j}| \tag{A.4.1}$$

then all the eigenvalues of matrix A are lie in the union of the disks.

$$|z - a_{i,i}| \le d_i,\ 1 \le i \le n. \tag{A.4.2}$$

The theorem is well-known, so the proof is omit here.

The above theorem immediately gives the result of

Recall the problem of finding the bounds for the spectral radius, the Perron-Frobenius theorem provides the nontrivial lower-bound of $\rho(A)$. In Theorem A.4.1 the nontrivial upper bound of $\rho(A)$ is found. Together with these two results, we can have a conclusion of the spectral radius of a non-negative and irreducible matrix given in the following.

Lemma A.5.1. *[21] If $A = [a_{i,j}]$ is an $n \times n$ non-negative and irreducible matrix, then either*

$$\sum_{j=1}^{n} a_{i,j} = \rho(A), \textit{for all } 1 \leq i \leq n, \tag{A.5.1}$$

or

$$\min_{1 \leq i \leq n} \left(\sum_{j=1}^{n} a_{i,j} \right) < \rho(A) < \max_{1 \leq i \leq n} \left(\sum_{j=1}^{n} a_{i,j} \right). \tag{A.5.2}$$

Theorem A.5.2. *[21] Let $A = [a_{i,j}]$ be an $n \times n$ non-negative and irreducible matrix, for any $\mathbf{x} > \mathbf{0}$, either*

$$\min_{1 \leq i \leq n} \left(\frac{\sum_{j=1}^{n} a_{i,j} x_j}{x_i} \right) < \rho(A) < \max_{1 \leq i \leq n} \left(\frac{\sum_{j=1}^{n} a_{i,j} x_j}{x_i} \right) \tag{A.5.3}$$

or

$$\frac{\sum_{j=1}^{n} a_{i,j} x_j}{x_i} = \rho(A), \textit{ for all } 1 \leq i \leq n. \tag{A.5.4}$$

Moreover,

$$\max_{\mathbf{x} \in P} \min_{1 \leq i \leq n} \left(\frac{\sum_{j=1}^{n} a_{i,j} x_j}{x_i} \right) = \rho(A) = \min_{\mathbf{x} \in P} \max_{1 \leq i \leq n} \left(\frac{\sum_{j=1}^{n} a_{i,j} x_j}{x_i} \right) \tag{A.5.5}$$

The equality is valid if we choose the $\mathbf{x}$ equal to the positive eigenvector $e > 0$ corresponding to the eigenvalue $\rho(A)$. The method shown above will be applicable, because it provides us both the upper bounds and lower bounds for the spectral radius of a non-negative and irreducible matrix, by a simple algorithm without calculating the determination of $\lambda I - A$.

A.6 Diagonalizable Matrix & Symmetric Matrix

The Jordan normal form of weight matrix $W^k = TJ^kT^{-1}$ gives the local value vector $\mathbf{x}(k) = W^k\mathbf{x}(0)$ an analytical expression in terms of eigenvalues and eigenvectors. Moreover, if the matrix W is symmetric, the expressions of $\mathbf{x}(k)$ can be simplified. Then, some algorithms such as the finite-time consensus, can be implemented more easily. In addition, under the the assumption of symmetric

Bibliography

[1] G. Xiong and S. Kishore, "Linear high-order distributed average consensus algorithm in wireless sensor networks," *EURASIP J. Adv. Signal Process*, vol. 2010, pp. 31:1–31:6, February 2010.

[2] ——, "Discrete-time second-order distributed consensus time synchronization algorithm for wireless sensor networks," *EURASIP J. Wirel. Commun. Netw.*, vol. 2009, pp. 1:1–1:12, January 2009.

[3] R. Cristescu, B. Beferull-Lozano, and M. Vetterli, "On network correlated data gathering," in *Proc. INFOCOM 2004. Twenty-third AnnualJoint Conf. of the IEEE Computer and Communications Societies*, vol. 4, 2004, pp. 2571–2582.

[4] K. Yuen, B. Liang, and L. Baochun, "A distributed framework for correlated data gathering in sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 57, no. 1, pp. 578–593, 2008.

[5] Z. Chair and P. K. Varshney, "Optimal data fusion in multiple sensor detection systems," *IEEE Transactions on Aerospace and Electronic Systems*, no. 1, pp. 98–101, 1986.

[6] L. Schenato and F. Fiorentin, "Average timesynch: A consensus-based protocol for clock synchronization in wireless sensor networks," *Automatica*, vol. 47, no. 9, pp. 1878 – 1886, 2011.

[7] W. Ren, H. Chao, W. Bourgeous, N. Sorensen, and Y. Chen, "Experimental validation of consensus algorithms for multivehicle cooperative control," *Control Systems Technology, IEEE Transactions on*, vol. 16, no. 4, pp. 745 –752, july 2008.

[8] P. Yang, R. Freeman, G. Gordon, K. Lynch, S. Srinivasa, and R. Sukthankar, "Decentralized estimation and control of graph connectivity for mobile sensor networks," *Automatica*, vol. 46, no. 2, pp. 390 – 396, 2010.

[9] R. Olfati-Saber and P. Jalalkamali, "Coupled distributed estimation and control for mobile sensor networks," *Automatic Control, IEEE Transactions on*, vol. 57, no. 10, pp. 2609–2614, 2012.

[10] S. O. H. F. D. P. Hlinka, O. and M. Rupp, "Likelihood consensus and its application to distributed particle filtering," *IEEE Transactions on Signal Processing*, vol. 60, no. 8, pp. 4334 –4349, aug. 2012.

[11] W. Ren, R. W. Beard, and E. M. Atkins, "Information consensus in multivehicle cooperative control," *IEEE Control Systems Magazine*, vol. 27, no. 2, pp. 71–82, 2007.

[12] D. JakoveticÌĄ and, J. Xavier, and J. Moura, "Weight optimization for consensus algorithms with correlated switching topology," *Signal Processing, IEEE Transactions on*, vol. 58, no. 7, pp. 3788 –3801, july 2010.

[13] A. Nedic, A. Ozdaglar, and P. Parrilo, "Constrained consensus and optimization in multi-agent networks," *Automatic Control, IEEE Transactions on*, vol. 55, no. 4, pp. 922 –938, april 2010.

[14] C. Asensio-Marco and B. Beferull-Lozano, "Network topology optimization for accelerating consensus algorithms under power constraints," in *Distributed Computing in Sensor Systems (DCOSS), 2012 IEEE 8th International Conference on*, may 2012, pp. 224 –229.

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

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

[17] R. L. G. Cavalcante and B. Mulgrew, "Adaptive filter algorithms for accelerated discrete-time consensus," *IEEE Transactions on Signal Processing*, vol. 58, no. 3, pp. 1049–1058, 2010.

[18] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2508 – 2530, june 2006.

[19] S. He, H.-C. Wong, and U.-H. Wong, "An efficient adaptive vortex particle method for real-time smoke simulation," in *Computer-Aided Design and*

Computer Graphics (CAD/Graphics), 2011 12th International Conference on, sept. 2011, pp. 317 –324.

[20] Russell and Merris, "Laplacian matrices of graphs: a survey," *Linear Algebra and its Applications*, vol. 197-198, no. 0, pp. 143 – 176, 1994.

[21] R. Varga, *Matrix Iterative Analysis*. Springer, 2010, vol. 27.

[22] K. Das, "The laplacian spectrum of a graph," *Computers & Mathematics with Applications*, vol. 48, no. 5-6, pp. 715 – 724, 2004.

[23] S. Kar and J. Moura, "Distributed consensus algorithms in sensor networks with imperfect communication: Link failures and channel noise," *IEEE Transactions on Signal Processing*, vol. 57, no. 1, pp. 355 –369, jan. 2009.

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

[25] W. Yu, G. Chen, W. Ren, J. Kurths, and W. X. Zheng, "Distributed higher order consensus protocols in multiagent dynamical systems," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 58, no. 8, pp. 1924 –1932, aug. 2011.

[26] L. Mackey, "Deflation methods for sparse pca," *Advances in neural information processing systems*, vol. 21, pp. 1017–1024, 2009.

[27] E. Kokiopoulou and P. Frossard, "Accelerating distributed consensus using extrapolation," *Signal Processing Letters, IEEE*, vol. 14, no. 10, pp. 665 –668, oct. 2007.

[28] W. Prass, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge University Press, 2007.

[29] R. Piziak and P. Odell, *Matrix Theory: From Generalized Inverses to Jordan Form*, ser. Pure and applied mathematics. Chapman & Hall/CRC, 2007.

[30] R. Olfati-Saber and J. Shamma, "Consensus filters for sensor networks and distributed sensor fusion," in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, dec. 2005, pp. 6698 – 6703.

[31] E. Robinson and S. Treitel, *Geophysical Signal Analysis.* Soc of Exploration Geophysicists, 2000.

[32] D. Kempe and F. McSherry, "A decentralized algorithm for spectral analysis," *Journal of Computer and System Sciences*, vol. 74, no. 1, pp. 70 – 83, 2008.

[33] M. Franceschelli, A. Gasparri, A. Giua, and C. Seatzu, "Decentralized laplacian eigenvalues estimation for networked multi-agent systems," in *CDC/CCC 2009. Proceedings of the 48th IEEE Conference on Decision and Control*, dec. 2009, pp. 2717 –2722.

[34] J. Zhou and Q. Wang, "Convergence speed in distributed consensus over dynamically switching random networks," *Automatica*, vol. 45, no. 6, pp. 1455–1461, 2009.

[35] W. Li, H. Dai, and Y. Zhang, "Location-aided fast distributed consensus in wireless networks," *IEEE Transactions on Information Theory*, vol. 56, no. 12, pp. 6208–6227, 2010.

[36] (2011, Sep) Lidar. Wikipedia. [Online]. Available: http://en.wikipedia.org/wiki/LIDAR

[37] P.M.Hamilton, "The application of a pulsed-light rangefinder (lidar) to the study of chimney plumes," *Royal Society of London Philosophical Transactions Series A*, vol. 265, pp. 153–172, Nov. 1969.

[38] J. Simard, G. Roy, P. Mathieu, V. Larochelle, J. McFee, and J. Ho, "Standoff sensing of bioaerosols using intensified range-gated spectral analysis of laser-induced fluorescence," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 42, no. 4, pp. 865–874, 2004.

[39] J. Bufton, "Development of the lidar controlled-airspace scanner for bioaerosol detection," in *Lasers and Electro-Optics, 2007. CLEO 2007. Conference on*, may 2007, p. 1.

[40] N.Kh. and Arystanbekova, "Application of gaussian plume models for air pollution simulation at instantaneous emissions," *Mathematics and Computers in Simulation*, vol. 67, no. 4Ũ5, pp. 451 – 458, 2004.

[41] J.-S. Lin and L. M. Hildemann, "Analytical solutions of the atmospheric diffusion equation with multiple sources and height-dependent wind speed

and eddy diffusivities," *Atmospheric Environment*, vol. 30, no. 2, pp. 239 – 254, 1996.

[42] (2011, Sep) "gaussian plume model java applet". [Online]. Available: http://www.geos.ed.ac.uk/abs/research/micromet/java/plume.html

[43] L. J. Shieh, P. K. Halpern, B. A. Clemens, H. H. Wang, and F. F. Abraham, "Air quality diffusion model; application to new york city," *IBM Journal of Research and Development*, vol. 16, no. 2, pp. 162 –170, march 1972.

[44] L. Xiao, S. Boyd, and S. Lall, "A scheme for robust distributed sensor fusion based on average consensus," in *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on.* Ieee, 2005, pp. 63–70.

[45] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Proc. IEEE Computer Society Conf Computer Vision and Pattern Recognition*, vol. 2, 1999.

[46] T. Moon, "The expectation-maximization algorithm," *Signal Processing Magazine, IEEE*, vol. 13, no. 6, pp. 47 –60, nov 1996.

[47] R. Horn and C. Johnson, *Matrix Analysis.* Cambridge Univ Pr, 1990.