Definitive Consensus for Distributed Data Inference

THÈSE N^{O} 5026 (2011)

PRÉSENTÉE LE 3 JUIN 2011

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

LABORATOIRE DE SYSTÈMES NON LINÉAIRES

PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Leonidas GEORGOPOULOS

acceptée sur proposition du jury:

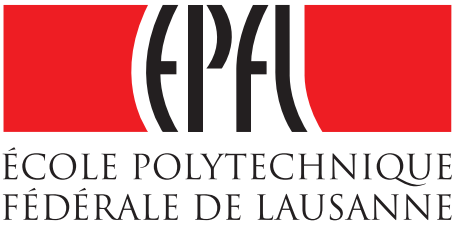
Prof. M. A. Shokrollahi, président du jury
Prof. M. Hasler, directeur de thèse
Prof. B. Hammer, rapporteur
Prof. M. Rovatsos, rapporteur
Prof. P. Thiran, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2011

DEFINITIVE CONSENSUS FOR DISTRIBUTED DATA INFERENCE

LEONIDAS GEORGOPOULOS



Doctor of Science

Laboratory of Nonlinear Systems
School of Computer and Communication Sciences
École Polytechnique Fédéral de Lausanne

April 2011

Leonidas Georgopoulos: *Definitive Consensus for Distributed Data Inference,* Doctor of Science, © April 2011

‘Διαλέγομε απ’ όλες τις πλάνες μιάν, εκείνη που περσότερο ταιριάζει με μας, και την ανακηρύχνομεν Αλήθεια.’ Συμπόσιον, Νίκος Καζαντάκης

“From all the delusions we choose one, that which fits best to ourself, and we proclaim it Truth.” Symposium, Nikos Kazantzakis

ABSTRACT

Inference from data is of key importance in many applications of informatics. The current trend in performing such a task of inference from data is to utilise machine learning algorithms. Moreover, in many applications that it is either required or is preferable to infer from the data in a distributed manner. Many practical difficulties arise from the fact that in many distributed applications we avert from transferring data or parts of it due to costs, privacy and computation considerations. Admittedly, it would be advantageous if the final knowledge, attained through distributed data inference, is common to every participating computing node.

The key in achieving the aforementioned task is the distributed average consensus algorithm or simply the consensus algorithm herein. The latter has been used in many applications. Initially the main purpose has been for the estimation of the expectation of scalar valued data distributed over a network of machines without a central node. Notably, the algorithm allows the final outcome to be the same for every participating node.

Utilising the consensus algorithm as the centre piece makes the task of distributed data inference feasible. However, there are many difficulties that hinder its direct applicability. Thus, we concentrate on the consensus algorithm with the purpose of addressing these difficulties.

There are two main concerns. First, the consensus algorithm has asymptotic convergence. Thus, we may only achieve maximum accuracy if the algorithm is left to run for a large number of iterations. Second, the accuracy attained at any iteration during the consensus algorithm is correlated with the standard deviation of the initial value distribution. The consensus algorithm is inherently imprecise at finite time and this hardens the learning process.

We solve this problem by introducing the definitive consensus algorithm. This algorithm attains maximum precision in a finite number of iterations,namely in a number of iterations equal to the diameter of the graph in a distributed and decentralised manner. Additionally, we introduce the nonlinear consensus algorithm and the adaptive consensus algorithm. These are modifications of the original consensus algorithm that allow improved precision with fewer iterations

in cases of unknown, partially known and stochastically time-varying network topologies.

The definitive consensus algorithm can be incorporated in a distributed data inference framework. We approach the problem of data inference from the perspective of machine learning. Specifically, we tailor this distributed inference framework for machine learning on a communication network with data partitioned on the participating computing nodes. Particularly, the distributed data inference framework is detailed and applied to the case of a multilayer feed forward neural network with error back-propagation. A substantial examination of its performance and its comparison with the non-distributed case, is provided.

Theoretical foundation for the definitive consensus algorithm is provided. Moreover, its superior performance is validated by numerical experiments. A brief theoretical examination of the nonlinear and the adaptive consensus algorithms is performed to justify their improved performance with respect to the original consensus algorithm. Moreover, extensive numerical simulations are given to compare the nonlinear and the adaptive algorithm with the original consensus algorithm.

The most important contributions of this research are principally the definitive consensus algorithm and the distributed data inference framework. Their combination yields a decentralised distributed process over a communication network capable for inference in agreement over the entire network.

Keywords: Consensus, Distributed, Learning, Distributed algorithms, Distributed Inference, Distributed Data, Collaborative Learning, Agreement, Nonlinear Consensus, Adaptive Consensus, Definitive Consensus, Finite time consensus, Networks, Communication networks, Neural networks, Graphs, graph Diameter, graph Radius, Groebner bases, Multilinear Polynomial systems of equations, nonlinear Optimisation, Dynamical Systems.

RESUMÉ

L'inférence à partir des données est d'une importance capitale pour nombreuses applications de l'informatique. L'utilisation des algorithmes de l'apprentissage automatique est la tendance actuelle pour l'accomplir. De plus, dans de nombreuses applications il est préférable ou requis de d'effectuer l'inférence à partir des données d'une manière distribuée. En fait, dans de nombreuses applications distribuées on aimerait éviter de communiquer les données à travers le réseau à cause des considérations de coût, de confidentialité, et de quantité des calculs. De plus, il serait avantageux si la connaissance atteinte à partir des données sera commune à tous les ordinateurs impliqués.

L'algorithme clé dans la conception d'inférence distribuée est l'algorithme du consensus moyen distribué, appelé simplement algorithme du consensus par la suite. Il y a de nombreuses des applications où cet algorithme a été utilisé. Initialement, celui-ci a été prévu pourle calcul de la moyenne d'une quantité scalaire distribuée sur un réseau d' ordinateurs qui n'ont pas de noeud central de calcul. Notons que l'algorithme fonctionne de sorte que cette moyenne est à la fin connue par chaque ordinateur du réseau.

L'utilisation de l'algorithme distribué du consensus moyen comme la pièce maîtresse rend la tâche de l'inférence à partir des données distribuées faisable. Cependant, il y a beaucoup de difficultés qui entravent de son application. Par conséquent, nous nous concentrerons sur l'algorithme distribué du consensus dans le but d'adresser ces difficultés.

Il y a deux problèmes principaux. En premier lieu, l'algorithme distribué du consensus converge asymptotiquement. Par conséquent la précision maximum est atteinte quand l'algorithme tourne pour de nombreuses itérations. En deuxième lieu, la précision atteinte durant l'exécution est corrélée avec la déviation standard de la distribution initiale des valeurs distribuées. Ainsi, l'algorithme est intrinséquement imprécis à temps fini ce qui rend le processus d' apprentissage plus difficile.

Nous résolvons ce problème en introduisant l'algorithme distribué du consensus définitif. Cet algorithme atteint la précision maximale en un nombre fini d'itérations, plus précisément en un nombre des itérations égal au diamètre du graphe qui représente le réseau de communication. Il fonctionne d' une

manière distribuée et décentralisée. De plus, nous introduisons les algorithmes du consensus non-linéaire et adaptif. Ce sont des modifications de l'algorithme du consensus original qui permettent l'amélioration de la précision en faisant moins d'itérations dans les cas de topologies inconnues, partialement connues et stochastiquement variables dans le temps.

L'algorithme du consensus définitif peut être utilisé dans le cadre de l'inférence à partir des données distribuées. Nous étudions ce problème dans la perspective de l'apprentissage automatique. Spécifiquement, nous concrétisons ce cadre de l'inférence distribuée dans le cas d'un réseau de communication des ordinateurs où les données sont distribuées aux ordinateurs. En particulier, l'inférence des données distribuées est détaillé dans le cas d' un réseau de neurones multicouche "feed-forward" avec rétro-propagation de l'erreur. Nous fournissons une analyse substantielle de sa performance ainsi que la comparaison avec le cas non-distribué.

La fondation théorétique de l'algorithme du consensus définitif est fournié. En plus, sa performance supérieure est validée par des simulations numériques. Également, une analyse sommaire des algorithmes non-linéaire et adaptif est fournie pour justifier leur performance améliorée en comparaison avec l'algorithme du consensus original. De plus, nous présentons les résultats de simulations numérique qui permettent de comparer les algorithmes non-linéaire et adaptif avec l'algorithme original.

Les contributions les plus importantes de cette thèse sont l'algorithme du consensus définitif et le cadre de l'inférence automatique à partir de données distribuées. La combinaison de ces deux algorithmes réalise le processus de l'inférence décentralisée et distribuée avec consensus sur le réseau entier.

Mots clés: le consensus, distribué, l'apprentissage, les algorithmes distribués, l'inférence distribué, les données distribuées, l'apprentissage collaboratif, l'accord, le consensus non-linéaire, le consensus adaptive, le consensus définitif, le consensus en temps fini, les réseaux, les réseaux de communication, les réseaux de neurones, le diamètre du graphe, le rayon du graphe, les bases de Groebner, les systèmes d'équations polynomiales, l'optimisation non-linéaire, les systémes dynamiques.

ACKNOWLEDGMENTS

It is rarely the case that in retrospective one can be so certain about a decision. Such a decision has been the one to commence towards a PhD under the supervision of Prof. Martin Hasler. The completion of this thesis would have been impossible without his insightful comments, numerous corrections and sincere discussions. He has surpassed all my expectations and he has aided me beyond any form of formal obligation. He has been genuinely involved with this work, and he has always been present to answer all my questions and direct me away from fallacies. I am deeply grateful to him.

I sincerely acknowledge the important contribution of David Barber who has in multiple ways enabled me to arrive at this point. Even though he had not been directly associated with this work, he has generously advised and supported me in various decisive occasions in the past years. Moreover, I would like to express my sincere gratitude to my friend Efi Kokkiopoulou for the inspiring discussions that initiated the development of this thesis.

It is of utmost importance to recognise the benefit from having received the *Scholarship for University Studies for Foreing Students* from the *State Secreteriat for Education and Research* of the *Swiss Confederation*. The reception of the aforementioned scholarship practically allowed me to commence my studies in Switzerland and subsequently studies at EPFL towards a PhD. I feel sincerely indebted to Switzerland. Moreover, it is important to acknowledge the aid of the Embassy of Switzerland in Athens for arranging many practical matters.

This work has been partially funded by the WINSOC Project, a Specific Targeted Research Project (contract number 0033914) funded by the INFSO DG of the European Commission within the RTD activities of the Thematic Priority Information Society Technologies.

CONTENTS

I BASICS 1

1 INTRODUCTION 3
- 1.1 Introduction 3
- 1.2 Preamble on the Consensus Algorithm 5
- 1.3 Purpose and Target Problems 7
- 1.4 Application Domain and Motivation 7
- 1.5 Thesis Contributions 8
- 1.6 Related Research 9
  - 1.6.1 Literature on Consensus 9
  - 1.6.2 Literature related to Distributed Data Inference 11
- 1.7 Thesis Layout 12

2 THEORY PREAMBLE 15
- 2.1 Communication Network 15
- 2.2 Graph Theory Related 16
  - 2.2.1 Graphs 16
  - 2.2.2 Elementary Graph Structures 19
  - 2.2.3 Elementary Graph Properties 20
  - 2.2.4 Classes of Graphs 24
  - 2.2.5 Types of Graphs 27
  - 2.2.6 Algebraic Graph Theory 30
- 2.3 Consensus 35
  - 2.3.1 Model of Communication 36
  - 2.3.2 The Consensus Algorithm 36
  - 2.3.3 Fixed Matrices 38
  - 2.3.4 Stochastic Matrices 41
  - 2.3.5 Weight Assignment 45
- 2.4 Groebner Bases Theory 48
  - 2.4.1 Algebraic Definitions 49
  - 2.4.2 Solving Systems Polynomials 50

II ALGORITHMS 57

3 DYNAMIC CONSENSUS 59

3.1 Introduction 59
3.2 The problem 60
3.3 The two phases of the Consensus Algorithm 62
3.3.1 Asymptotic Phase 62
3.3.2 Transient Phase 63
3.4 Nonlinear Consensus 72
3.5 Adaptive Consensus 77
3.5.1 Impact of Edge Stochasticity 77
3.5.2 Adaptive Nonlinear Consensus Algorithm 85
3.6 Summary 86
4 DEFINITIVE CONSENSUS 89
4.1 Introduction 89
4.1.1 Motivation 90
4.2 Definitive Consensus 92
4.2.1 Formulation 93
4.2.2 Existence of Solutions 94
4.2.3 Numerical Solutions 98
4.2.4 Groebner Basis Solutions 102
4.2.5 Communication Costs 106
4.3 Verification 107
4.4 Conclusions 110
5 CONSENSUS MACHINE LEARNING 113
5.1 Introduction 114
5.1.1 Problem Layout and Applicability 114
5.1.2 Related Work 116
5.2 Distributed Data Inference by Consensus 117
5.2.1 Abstract Framework 118
5.3 Extension to Definitive Consensus 129
5.4 Application of the Abstract Framework 130
5.4.1 Feed Forward Multilayer Neural Networks with Back-propagation 130
5.5 Numerical Validation 133
5.5.1 Consensus Learning Results Preamble 133
5.5.2 Comprehensive Results Consensus Learning 139
5.5.3 Definitive Consensus Machine Learning with Regularisation 145
5.6 Resume 156

III DISCUSSION 157
6 VALIDATION 159
6.1 Dynamic Consensus Performance 159
6.2 Definitive Consensus 164
6.3 Distributed Machine Learning 169
6.3.1 The 2007 TREC Public Spam Corpus 169
6.3.2 Definitive Consensus Data Inference 170
6.3.3 Summary 173
7 DISCUSSION 177
7.1 Summary 177
7.2 Application Domains 179
7.2.1 Definitive Consensus 180
7.2.2 Consensus Machine Learning 181
7.3 Future Work 183
7.4 Conclusion 184
IV APPENDICES 185
A DERIVATIONS 187
A.1 Related to (*Chapter* 3) 187
A.1.1 Derivation of expected state 187
A.1.2 Communication Cost of Unicast on a Uniform Tree 188
B ADDITIONAL NUMERICAL RESULTS 189
B.1 Definitive Consensus Example Solutions 189
B.2 Nonlinear Consensus Algorithms Comparison Results 195
C DEFINITIVE CONSENSUS SOLUTIONS 207
C.1 Star Graphs 207
C.2 Cycles 207
C.2.1 The Pentagon 207
C.2.2 The Hexagon 208
Nomenclature 212
Glossary 215
BIBLIOGRAPHY 217

LIST OF FIGURES

Figure 1 Graph Examples 17
Figure 2 Induced Graphs 18
Figure 3 Connectibity and Components ambiguity 22
Figure 4 A directed graph 25
Figure 5 A directed multi-graph with self-edges 26

Figure 6 The Transient Phase 64
Figure 7 Slow and fast components contribution 67
Figure 8 Prominent Transient Phase 69
Figure 9 Impact of convex optimisation on the transient phase 70
Figure 10 Average performance of the Nonlinear algorithm versus the Linear 75
Figure 11 Comparison of Nonlinear versus Linear 76
Figure 12 Effect of edge probability 80
Figure 13 Averaged System Verification 84
Figure 14 Impact of Stochasticity 85

Figure 15 Definitive Consensus Motivation 91
Figure 16 Definitive consensus on a medium graph 108
Figure 17 Definitive consensus on a large graph 109

Figure 18 Abstract operation of a Distributed Inference Framework 117
Figure 19 The Two Moons Dataset 134
Figure 20 Dataset and Classification output - Uniform case 135
Figure 21 Dataset and Classification output - Non-Uniform case 136
Figure 22 Overall Classification Performance with Non-uniform Partitioning 138
Figure 23 Classification Error Rate - Uniform data partitioning 139
Figure 24 Classification Error Rate - Uniform data partitioning - Contour plot 140
Figure 25 Classification Error Rate - Comparison with non-distributed - Uniform data partitioning 141

Figure 26 Classification Error Rate - Non-uniform data partitioning 142
Figure 27 Classification Error Rate - Non-uniform data partitioning - Contour plot 143
Figure 28 Classification Error Rate - Comparison with non-distributed - Non-uniform data partitioning 144
Figure 29 Classification with little regularisation $\zeta = 0.9$ - Uniform partitioning 146
Figure 30 Classification with regularisation $\zeta = 0.5$ - Uniform partitioning 147
Figure 31 Acceptable performance on the Test set during training - Uniform partitioning 148
Figure 32 Not acceptable performance on the Test set during training - Uniform partitioning 149
Figure 33 Classification Error Rate of a network with 2 hidden layers - Non-uniform data segregation - Contour plot 150
Figure 34 Good performance on the Test set during training - Uniform partitioning 151
Figure 35 Classification without differential learning and regularisation $\zeta = 0.8$ - Uniform partitioning 152
Figure 36 Classification with little regularisation $\zeta = 0.9$ - Non-uniform partitioning 154
Figure 37 Classification with regularisation $\zeta = 0.5$ - Non-uniform partitioning 155

Figure 38 Small Graphs Overall Performance 160
Figure 39 Overall errors wrt different initial standard deviation in small graphs 163
Figure 40 Definitive Consensus Example 10 Vertices Non-symmetric 165
Figure 41 Definitive Consensus Example 10 Vertices symmetric 166
Figure 42 Definitive Consensus Example 25 Vertices Non-symmetric 167
Figure 43 Definitive Consensus Example 25 Vertices symmetric 168
Figure 44 Convergence on the TREC Spam Corpus on a simple neural network 171
Figure 45 Convergence on the TREC Spam Corpus on a larger neural network 172
Figure 46 Convergence on the TREC Spam Corpus on an even larger neural network 173

Figure 47 Definitive Consensus Example 20 Vertices, Non-symmetric 190
Figure 48 Definitive Consensus Example 20 Vertices, Symmetric 191
Figure 49 Definitive Consensus Example 45 Vertices, Non-symmetric 192
Figure 50 Definitive Consensus Example 45 Vertices, Symmetric 193
Figure 51 Instantaneous errors wrt different initial standard deviation in small graphs 196
Figure 52 Overall errors wrt vertex alive probability in small graphs 197
Figure 53 Instantaneous errors wrt vertex alive probability in small graphs 198
Figure 54 Overall errors wrt to edge alive probability in small graphs 199
Figure 55 Instantaneous errors wrt to edge alive probability in small graphs 200
Figure 56 Overall errors wrt different initial standard deviation in medium graphs 201
Figure 57 Instantaneous errors wrt different initial standard deviation in medium graphs 202
Figure 58 Overall errors wrt vertex alive probability in medium graphs 203
Figure 59 Instantaneous errors wrt vertex alive probability in medium graphs 204
Figure 60 Overall errors wrt to edge alive probability in medium graphs 205
Figure 61 Instantaneous errors wrt to edge alive probability in medium graphs 206

Part I

BASICS

1 INTRODUCTION

In the preface we have briefly described the problem, our motivation, the main purpose of this thesis, provided a silhouette of the proposed solution, and the main contributions of this research. In this chapter, we embark to provide detailed arguments that justify this research, introduce the main notions in more detail, and enumerate the contributions of this work. A discussion of related research is necessitated for the reader to be persuaded for the novelties of our contributions. Finally, we provide a brief outline of the remainder of this thesis in order to guide the reader and communicate the main purpose of each chapter. Throughout this chapter we avert from using mathematical nomenclature to facilitate understanding; before we embark on more complex analysis.

1.1 INTRODUCTION

This thesis directly relates to the field of distributed information processing by consensus, or simply *information consensus* (Olfati-Saber and Murray, 2004). We extend this notion to treat a larger class of problems by introducing the distributed data inference framework. Notably, in such a setting, the distributed average consensus algorithm has a central role and its performance, and thus its finite time accuracy as well, is of key importance. The latter necessitates our research with respect to consensus. Our purpose is to improve its convergence behaviour. This results in the introduction of the so-called definitive consensus algorithm.

process consensus, and information consensus

In order to avoid misinterpratation of the term consensus, in this work it is important to distinguish it from the standard computer science nomenclature. There, consensus is related to termination criteria of a distributed set of processes such that these terminate when they are in agreement. This relates to the Byzantine generals problem and agreement of faulty processes (Lamport et al., 1982), (Lamport, 1983), (Fischer et al., 1985). Our work lies far from that field of research.

Our motivation draws initially from distributed average consensus estimation in wireless sensor networks (WSN). There a sensed physical quantity, e.g.

temperature or humidity, is measured at each wireless sensor node. Assuming a normal distribution of observations, one applies the consensus algorithm to approximate the expectation of the initial data distribution, as in (Xiao et al., 2005). This is achieved with local only communications between one-hop nodes by exchanging the associated state variable. Subsequently, a local average is computed at each node. Finally, the state at each node is updated with this new estimate. The algorithm proceeds iteratively with this two step communication - update process. The latter converges asymptotically such that all nodes in the network obtain the same approximation of the distribution of the initial values.

The most important advantage of the consensus algorithm is that eventually the average becomes available at every computing node in the WSN, without the need of a central computing node. This alleviates many problems, like routing, congestion, and computation burden while at the same time augments overall robustness of the network. However, one might object that the assumption of normality may not always hold. In contrast, we support that the consensus algorithm can be modified such that it can be applied to an augmented class of possible applications through a distributed data inference framework.

Having the latter in mind, we concentrate on determining the class of problems that can be treated with the consensus algorithm. These include the cases where due to the nature of the data, or due to computational restrictions, data has to be treated distributively. Additionally, there are numerous conjoined problems where the outcome should preferably be available at every computing node. The union of these two classes forms the class of problems which can be treated with the consensus algorithm within the distributed data inference framework. In this class, data has to be treated both distributively in order to achieve inference, and simultaneously all participating computing nodes should agree on the final outcome.

problems for distributed data inference

Particularly, problems in such a class arise from data having four principal properties. First, data is unobtainable when its collection is either impossible or undesirable. Second, data cannot be computed with a designated inference algorithm by a single machine of specific computing power. Thus, within this context it is incomputable. Third, data is intrinsically distributed; hence the collection of the data at a central node for computation would not be advantageous, given an equivalent distributed algorithm for the same task. Fourth, data is private, and collection of the data centrally would expose the data to the computing node. Inevitably, one has to infer from data distributively. We refer to this class of problems as UIDP (Unobtainable, Incomputable, Distributed,

Private). In order to treat this class of problems, we introduce a distributed data inference framework based on a variant of the consensus algorithm. Our purpose is to treat all problems, falling in the aforementioned class (UIDP), collectively.

The consensus algorithm is of key importance, in achieving many information processing tasks distributively such that the final outcome at each computing node is in agreement with the rest. The main drawback is that convergence to the average is asymptotic. Therefore, the algorithm can be very slow and imprecise in early stages. Moreover, given that the algorithm is executed for a fixed number of iterations, precision can be significantly correlated to the distribution of the initial observations.

This fact raises difficulties both in the case of contemporary applications of the consensus algorithm, e.g. WSN, and our envisioned generalisations for the treatment of the (UIDP) class of problems, as well. In order to overcome this, we shall introduce three modifications of the consensus algorithm. Principally, the definitive consensus algorithm which alleviates this problem, and achieves maximum accuracy in minimum number of iterations. The latter can be applied in the case of synchronous network communication and fully known non-varying topology. In order to treat other cases where topology is partially known, we introduce the nonlinear consensus algorithm. The later demonstrates improved precision in early stages when comparing with consensus algorithm. Finally, in cases of uncertain communication we introduce the adaptive consensus algorithm.

Let us now avert the discussion from the abstract description of the themes and problems handled to introduce the consensus algorithm, the key notion within this thesis.

1.2 PREAMBLE ON THE CONSENSUS ALGORITHM

Assume a class of machines able to store information and perform the following two functions. First, perform computation; second, communicate with another machine of the same class. Usually, memory is very small, in the range of a few variables of single precision floats.

Furthermore, suppose such machines are arbitrarily connected, and form a communication network. It is sufficient that the communication network has an associated graph that is connected. This implies that for any two machines on the network there is a sequence of hops, a route, through other machines connecting these two machines such that information on the first machine can

be transferred onto the second through this sequence of hops. Moreover, assume that each machine stores an associated state variable.

what is the consensus algorithm?

The discrete time distributed average consensus algorithm, or just consensus algorithm, is an iterative distributed algorithm. It allows the computation of the average of the state variables, distributed on the machines in the communication network, without routing, and in a completely distributed and decentralised manner. The latter is achieved in the following manner.

Each machine is able to communicate directly on the network with at least one other machine, i.e. a neighbour. In the first step of the algorithm, each machine communicates with all its neighbours and performs two actions. Firstly, it sends its own state, and thereafter retrieves the states of its neighbouring machines. The manner that this communication happens, i.e. packet transmission protocols and communication frames, is not within the scope of this research. However, we mention that there are more than just a few protocols to allow this type of communication in an ad-hoc, non routed network.

Secondly, in the next step of the algorithm, the machine computes a local weighted average of the values retrieved from its neighbouring machines along with its own. Subsequently, the state variable of the machine is updated with this local weighted average. This two step process, communicate and update, converges asymptotically to the sample average of the initial value distribution for every participating machine. A nice presentation of the algorithm can be found in (Lynch, 1996). A thorough examination of the algorithm's asymptotic behaviour is given in (*Chapter* 2).

Given that the aforementioned machines perform finite precision arithmetic, we can affirm the following two. First, that after a large, but finite number of iterations, the algorithm will attain maximum precision. Thus, at any machine the absolute of the difference of the state variable from the average will not decrease in subsequent iterations. Second, that the absolute of the difference of a machine from any other in the network will not decrease any further. In that case, the machines are in agreement and we say that they are at consensus.

The principal disadvantage of the algorithm is its asymptotic convergence. The latter implies that many iterations are needed. These are costly in terms of communication, energy, and time. Moreover, another predicament deduced from the fact that states of any two machines may not by identical; instead, these vary near the average. Even when the algorithm has reached agreement, the states among different machines may present differences in the least significant digit. However, the algorithm's significant advantage of non-centralised completely

distributed ad-hoc computation capability over a set of machines fuels this research with the purpose of ameliorating these problems.

1.3 PURPOSE AND TARGET PROBLEMS

The principal purpose of the research, presented within this thesis, is the creation of algorithms based on the aforementioned consensus algorithm that retain its advantages, ameliorate its deficiencies, and at the same time solve the following problems:

Problem 1.1. *Suppose that the topology of the communication network is completely known. Then, minimise the number of update iterations needed to reach consensus up to precision.*

Problem 1.2. *Infer from data belonging to the UIDP class in a completely distributed fashion over a communication network.*

Moreover, in order to extend the applicability of the distributed data inference framework, the following problems are considered, as well.

Problem 1.3. *Maximise the accuracy of the consensus algorithm while minimising the number of iterations in all three cases of completely known, partially known, and unknown topology.*

Problem 1.4. *Maximise the accuracy of the algorithm while minimising the number of iterations in the case of uncertain communications.*

1.4 APPLICATION DOMAIN AND MOTIVATION

In order to capture the interest of the reader, we provide an introductory discussion of possible applications of this work. This may also serve as further justification of our motives in the research field of information consensus. A broader discussion of envisaged applications is given in (*Section* 7.2).

The consensus algorithms, developed within this thesis, can be primarily applied to any field where the linear average consensus algorithm is applicable. There are some additional minor demands on memory; however, even for huge networks, these should not surpass the kilobyte limit. Some of the algorithms need floating point arithmetic which can be usually taken care with look-up tables, in systems with limited processing power; something that would

require more memory. However, with the advent of modern computer systems and low-cost, low-energy processors these matters are without great practical importance. Hence, such memory demands should not be regarded as an important drawback. Summarising a few of the contemporary applications of the consensus algorithm, one should include WSN, distributed computing, parallel computing, multi-agent coordination and collaboration.

The application of the distributed data inference framework has greater demands computationally. However, the computational burden is primarily due to the machine learning algorithm being employed. Therefore, this predicament is not a result of the distributed inference algorithm but depends on the learning algorithm of choice. The machine learning algorithm can be chosen per case to correspond with the computing power available at each machine on the network.

The distributed data inference can be possibly applied in a number of applications where inference from complex patterns is required, such that environmental monitoring, data mining on large datasets, social network mining, clinical data mining, banking data mining. Some examples of envisaged application are provided in (*Section* 7.2).

1.5 THESIS CONTRIBUTIONS

Our contribution in the field of information consensus is twofold. Firstly, we add to the research area related to the distributed average consensus algorithm, which has been primarily initiated from the work of (Tsitsiklis, 1984). Secondly, we introduce the framework for distributed data inference by consensus. Currently, machine learning is principally employed to perform the task of inference. We follow this trend in our work, as well.

The principal contribution is the definitive consensus algorithm, accompanied by a scheme to determine the coefficients of the weighted edges. This algorithm is both theoretically sound and numerically verified. The second major addition is the distributed data inference framework and its application in the case of a multilayer feed-forward neural network with back-propagation.

However, the definitive consensus algorithm cannot be applied when the communication network is huge or its topology partially known. Moreover, the case of uncertain communication cannot be treated. In order to handle these deficiencies, we explore the convergence behaviour of the consensus algorithm. This results in another important contribution. Specifically, we introduce the notions of the transient and the asymptotic phase. Something that leads to the

claim that the modulation of the edge weights, throughout the process, may improve the rate of convergence. This can be best performed by varying the weights with respect to state. The development of the nonlinear average consensus algorithm is founded on these observations. Finally, following the same research direction, we attempt to treat the case of uncertain communications with the adaptive consensus algorithm.

1.6 RELATED RESEARCH

This thesis relates directly with current ongoing research in the field of distributed average consensus algorithms and the area of ad-hoc distributed data inference. An overall description of past and current research, in the aforementioned fields, is provided within this section. Moreover, the differences between the approaches found in the literature and our work are highlighted.

1.6.1 *Literature on Consensus*

applications of consensus

The applications of distributed average consensus are mainly targeted in wireless sensor networks, parallel computation and multi-agent coordination. One should refer to (Bertsekas and Tsitsiklis, 1989) and (Tsitsiklis et al., 1986), for the application of the consensus algorithm in distributed computing. Also, the method in (Hendrickson and Kolda, 2000) may be combined with the consensus algorithm in a distributed setting. Even though the method is not explicitly outlined for the case of distributed computation, it provides nevertheless an abstract framework.

In the case of WSNs the related literature is rapidly increasing in the past few years. Particularly, the research described within this thesis has been partially funded by the WINSOC project, refer to Acknowledgments. The project has been a EU initiative with main purpose of applying decentralised algorithms for environmental monitoring; something that demonstrates a larger interest in this field of research. An application of the algorithm in the case of landslide detection, within the context of the WINSOC project, can be found in (Ramesh et al., 2009). Another application in the case of fire detection has been presented in (Khadivi and Hasler, 2010). Additional research in the field of WSNs can be found in (Zhao et al., 2003), (Gupta et al., 2005). Numerous applications of the algorithm are also found in the field of multi-agent coordination and indicative

current work may be found in (Blondel et al., 2005), (Olfati-Saber, 2006), (Ren et al., 2008), (Dimarogonas and Kyriakopoulos, 2008), (Tanner et al., 2007).

Our research has been mainly theoretical, and to our knowledge, none of the algorithms or mathematical proofs contained within this thesis can be found in the literature, unless specifically cited. However, a large part of our work is based on previous research, and the algorithms developed can be considered as extensions or variants of the consensus algorithm. For completeness' sake, we mention the related theoretical work.

initial work on consensus

The distributed average consensus algorithm has been initially mentioned in (Tsitsiklis, 1984). The reader should consult pages 121-140 of the aforementioned thesis for a thorough theoretical presentation of the algorithm. A preceding description of the algorithm is given in (Borkar and Varaiya, 1982). A rather abstract approach of the communication-update process can be found in (Geanakopolos, 1982).

A discussion of the algorithm in the specific perspective of WSNs is given in (Lynch, 1996). Moreover, a rather detailed review and analysis is given in (Olfati-saber et al., 2007) for both continuous and discrete time systems. Specifically, we consider only the discrete time case; a comprehensive discussion of the algorithm for discrete time systems is given in (*Section* 2.3).

As already mentioned, the problem of consensus in scalar values, or information consensus, may be treated with the distributed average consensus algorithm. However, the number of iterations until convergence can be large due to the algorithm's asymptotic nature; something that deteriorates as the size of the associated communication network increases. The convergence time can be improved by adjusting the weights in the weighted average associated with the retrieved states from the neighbouring machines. In this such an approach, one tries to minimise an associated utility function. In the work of (Xiao and Boyd, 2003) and (Xiao et al., 2007), it has been demonstrated that the maximisation of the spectral gap of the weight matrix, associated with the network, results in a convex problem. Optimising the weights (Boyd and Vandenberghe, 2004) results in maximum asymptotic speed of convergence, as demonstrated in (Xiao et al., 2007). Details, related to the problem's graph theoretic notions, are provided in (*Section* 2.2).

convergence improvements for the consensus algorithm

The case of partial knowledge of topology can be handled in a non-optimal manner by using the so called Metropolis-Hastings or local-degree weights (Xiao and Boyd, 2003). Constant edge weights may be used in case of unknown topology (Xiao and Boyd, 2003). The latter, require knowledge of the degree

of each node on the network. Moreover, with a simple framework based on these weights, one is able to handle failing links. This is due to the fact that the node's own degree is always known. Another reference, involving time-varying weights, has been made in (Olshevsky and Tsitsiklis, 2006).

Our work relates to the aforementioned research. Particularly, one finds at the core of our research the problem of selecting the weights at each step such that time to convergence is reduced. Specifically, our work is differentiated from the work of (Xiao and Boyd, 2003); mainly for the fact that the weights are modulated during the process, with the purpose of improving the rate of convergence during the duration of the process. There, this is suggested for the purpose of treating uncertain communications. However, we similarly vary the weights with the purpose of improving the rate of convergence.

time-varying weights

Similar attempts are made in (Olfati-Saber and Murray, 2004), where the notion of dynamic graphs is introduced, and a proof of convergence is given for the case of strongly connected balanced digraphs. We have shown this in (Georgopoulos and Hasler, 2009b) for any undirected graph. Additional research with the purpose of treating the problem of failing links with time-varying weights may be found in (Gupta et al., 2005), (Moreau, 2005).

In our work, we relate the weight modulation with the state variable instead of the number of failed communication links, as in the aforementioned literature. Moreover, we distinguish the existence of two phases, the asymptotic and the transient. Furthermore, we recognise the impact of the latter, in the number of iterations to convergence. Concluding, we employ time varying weights for both reducing the number of iterations to convergence and the treatment of uncertainties in communications. Specifically, the reduction of iterations to convergence, is performed with the definitive, and the nonlinear consensus algorithm. The case of uncertain communications with failing links can be improved by the adaptive consensus algorithm.

1.6.2 *Literature related to Distributed Data Inference*

The most closely related research area is the field of distributed optimisation, (Rabbat and Nowak, 2004). The latter can be considered a specific case of distributed computation, as in (Bertsekas, 1996). The past few years a growing interest in the field has lead to a number of publications, as in the case of distributed gradient estimation through sub-gradients by consensus (Johansson et al., 2008). Also, a distributed Newton method employing the consensus

distributed optimization

algorithm has been introduced in (Jadbabaie et al., 2009). Additional work on distributed optimisation may be found in (Nedic et al., 2010), (Cavalcante et al., 2009). The common factor in the aforementioned work is that a local gradient is associated with the state of each machine in the network. Subsequently, by application of the consensus algorithm, it is made possible to retrieve an estimate of the gradient at each participating machine in the entire network.

distributed machine learning and optimisation differences

However, the interest of researchers in distributed optimisation is different. An optimisation differs from a learning problem; the generative function of the data is known a priori. Instead, in standard machine learning jargon, it is implied that data is generated from an unknown generative function. In the simplest case, one is aware of the class of generative functions but this still poses a hard problem. Its treatment is usually attempted with parametric methods; these result in an optimisation problem. However, in the general case, the generative function of the data is unknown. The first successful machine learning algorithm to treat the problem in its general form was the multilayer neural network with back-propagation. There no assumptions are made for the model of the data, and it has been theoretically supported that a neural network with one hidden layer can learn any problem (Bishop, 1996). Alas, this theoretical result does not guarantee the performance of the network in practice. Concluding, the aforementioned methods in the literature of distributed optimisation are the tools that may be employed to perform distributed data inference, in the same sense that machine learning algorithms utilise non-distributed optimisation to infer from data. Yet, machine learning is not considered the same as optimisation.

distributed machine learning literature

Particularly, in the field of distributed machine learning there is some ongoing research. This is mainly focused in the case of distributed or parallel support vector machines, (Ang et al., 2008), (Navia-Vazquez et al., 2006), (Lu et al., 2008). Another notable attempt in modifying the EM-algorithm for distributed computation by information consensus is found in (Kowalczyk and Vlassis, 2005). However, these studies are specific to each of the learning algorithms. In contrast, our work may be applied to larger class of machine learning algorithms.

1.7 THESIS LAYOUT

This thesis is organised in three parts, the Basics, the Algorithms and the Discussion. The first part has the purpose of introducing the reader to the theme of this thesis and provide the necessary theoretical foundation. Specifically, in (*Chapter* 1) we describe the problem, the basic notions, and the foreseen

solutions while avoiding any mathematical formulation. Then, in (*Chapter* 2), we shall give the necessary theoretical foundation and introduce all the necessary mathematical formulation.

In part ii we present the algorithms. Particularly, in (*Chapter* 3) the nonlinear and the adaptive consensus algorithms are presented. Then, in (*Chapter* 4) we present the definitive consensus algorithm which is the major contribution of this research. In (*Chapter* 5) we present the second major contribution, which is the distributed data inference framework. This is applied in the case of distributed machine learning, specifically for the multilayer feed-forward neural networks with back-propagation.

The next part, part iii, has the purpose of persuading the reader of the truthfulness and the practicality of our claims. For this reason, we present in (*Chapter* 6) the results of extensive numerical experiments for all the algorithms presented in this thesis. Finally, in (*Chapter* 7), we summarise the thesis, provide a discussion on the possible application domains of this work, and detail possible future directions.

2

THEORY

Within this chapter, we illustrate the theoretical arrangement that is necessary for the comprehension of successive chapters. Additionally, this should serve as reference for notions subsequently mentioned . The subjects developed within this chapter are *Graph theory*, *Consensus*, *Groebner Bases*. The acquainted reader of these subjects may advance to (*Chapter* 3) and refer here as needed. We assume the reader to be knowledgeable in linear algebra and dynamical systems theory.

2.1 COMMUNICATION NETWORK

We commence by considering a number of machines able to compute and communicate in an arbitrary manner. The term communication permits the complete process of emission of a packet of digital information from a machine A and its reception from another machine B without the intermediation of a third machine C. Machines A and B are said to be directly connected, but for simplicity's sake we just refer to as connected; such machines are also called neighbours. We do not consider particularly the manner that this communication is achieved in detail. Hence, the incorporated communication protocols are of no interest within the scope of this research. In terms of the OSI layered model for communications, our work resides in the application layer (number 7).

Moreover, we assume that all machines in the network are capable to complete a round of communications with all their neighbours within a given time frame. Communications outside this time frame are considered dropped; these are going to be indicated as link failures, later on. Therefore, matters of delayed communications evade the purpose of this research. This time frame is considered to be the basic time unit. The sequence of communications, with respect to this time unit, defines a time sequence for the system of networked machines or simply time. The time index shall be denoted as t herein. As direct consequence of this assumption, the communications shall be considered synchronous. Even though the specific communication protocols, incorporated

to achieve these communications, can be asynchronous, have delays, use routing, and treat congestion, in our perspective these are of no concern.

This network of connected machines gives rise to a topology at each time unit. Such a topology can be well described with a graph depicting the connections as edges and the machines as vertices for each time instantiation. Graphs are of key importance for the advancement of the analysis performed herein, and we provide subsequently an introduction on graph theoretical matters of interest.

2.2 GRAPH THEORY RELATED

The literature on the subject is vast. The reader may consult the following introductory (Biggs, 1974),(Godsil and Royle, 2001) and advanced (Hatcher, 2001) material for additional details on the subject. We illustrate some the notions, found in the literature, hereafter.

2.2.1 *Graphs*

Graphs are often used to indicate the relation of a group objects in an large area of science such as biology, informatics, economics, chemistry, social sciences and physics. In computer science, graphs have been initially related to the representation of state machines and automata. In communication sciences, graphs are vastly employed to represent networked machines. Herein as well, graphs are adopted to describe the topology of the communication network arising from the operation of the consensus algorithm.

Definition 2.1 (Graph)**.** *A graph $\mathcal{G}$ is a pair of sets, the vertex set $\mathcal{V}$ and the edge set $\mathcal{E}$, along with an incidence function γ; one writes $\mathcal{G}(\mathcal{V}, \mathcal{E}, \gamma)$.*

The essential part of a graph is the vertex set, which is the representation of the objects of which the relations are being modelled with the edges. The vertex set is necessarily not empty.

Definition 2.2 (Vertex Set)**.** *A vertex set $\mathcal{V}$ is a finite non empty set with its elements called vertices.*

The vertices' labels can be unordered. However, an ordered labelling eases the discussion. Herein vertices are labelled as v_i where $i \in \{1, 2, \ldots, n\}$ and n is the number of elements in $\mathcal{V}$, i.e. the cardinality of the vertex set. One also writes

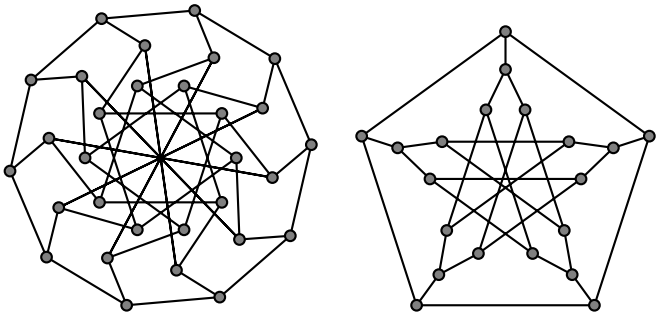


Figure 1: **Graph Examples**. The vertices are denoted by grey dots and edges are the lines connecting them. **Left**: Tutte's 8-cage. **Right**: Largest 3-regular of diameter 3.

$\mathcal{V}(\mathcal{G})$ to denote the vertex set of the graph $\mathcal{G}$, e.g. $\mathcal{V}(\mathcal{G}_k)$ denotes the vertex set of graph $\mathcal{G}_k$.

In order to model the relations between the objects included in the vertex set, the edge set of the graph is defined. Each edge in the set signifies a relation between two vertices.

Definition 2.3 (Edge Set)**.** *An edge set* E *is a finite set of tuples of vertices; the tuples are called edges.*

The edge set can be an unlabelled unordered set, possibly empty. In order to facilitate the discussion, the elements of the edge set are denoted e_k and $k \in \{1, 2, \ldots m\}$ with m denoting the cardinality of $\mathcal{E}$. As in the case of the vertex set, we write $\mathcal{E}(\mathcal{G})$ to signify the edge set of a graph $\mathcal{G}$.

Conventionally, vertices are graphically depicted as points and the edges as lines connecting the associated vertices. In fact, the naming of the edges and vertices draws from this intuitive graphical representation of relations between the objects. A couple of example graphs are depicted in (*fig.*1).

The incidence function maps each edge e_k to a pair of vertices $\{v_i, v_j\}$ where $i \in \{1, 2, \ldots n\}$ and $j \in \{1, 2, \ldots n\}$, denoting a relation of objects. Generally, an edge may be mapped to an tuple of vertices, called multi-edges. Conventionally, an edge is a pair of vertices. Particularly, within the scope of our research, multi-edges evade the purpose of modelling the the communication network's topology, arising from the operation of the consensus algorithm. We only consider edges between pairs of not necessarily distinct vertices. In the latter case, i.e. the edge is $\{v_i, v_i\}$, these are called self-edges or self-loops.

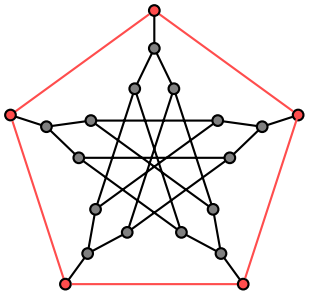


Figure 2: **Induced Graphs**The red edges and vertices depict a subgraph which is both vertex and edge induced

Definition 2.4 (Incidence function)**.** *The incidence function γ of the graph $\mathcal{G}$ is a map $\gamma : \mathcal{E} \to \mathcal{V} \times \mathcal{V}$.*

Notation can be simplified by omitting the incidence function, and define the edge set as the set of associated vertices. We shall follow this convention from here on.

Definition 2.5 (Edge Set)**.** *An edge set $\mathcal{E}$ is a finite unordered set of pairs of vertices, called edges.*

We can redefine the graph using this simplified notion of an edge set. The incidence function may also be omitted and write $\mathcal{G}(\mathcal{V}, \mathcal{E})$. Usually, this subsequent simplified definition of a graph is used.

Definition 2.6 (Graph)**.** *A graph* G *is a pair of sets, the vertex and the edge set, denoted $\mathcal{G}(\mathcal{V}, \mathcal{E})$ where the vertex set* V *is a finite non empty set and the edge set $\mathcal{E}$ is set of pairs of the elements of $\mathcal{V}$.*

2.2.1.1 *Subgraphs*

Suppose two graphs such that the vertex and the edge set of the first are subsets of the second. Then, the first is a subgraph of the latter. Particularly, given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and another graph $\mathcal{G}_s$ satisfying $\mathcal{V}(\mathcal{G}_s) \subseteq \mathcal{V}(\mathcal{G})$ and $\mathcal{E}(\mathcal{G}_s) \subseteq \mathcal{E}(\mathcal{G})$, then $\mathcal{G}_s$ is a subgraph of $\mathcal{G}$.

An important class of subgraphs are the so-called induced subgraphs. An induced subgraph can be edge induced or vertex induced. In an induced subgraph $\mathcal{G}^*$, the inducing set is a subset of the corresponding set of $\mathcal{G}$ and the induced set of $\mathcal{G}^*$ includes those elements of the conjoined set of $\mathcal{G}$ which are adjacent to the elements of the inducing set.

Definition 2.7 (Vertex induced subgraph)**.** *A vertex induced subgraph $\mathcal{G}(\mathcal{V}^*)$ of a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ has a vertex set which is a subset of the vertex set of $\mathcal{G}$, satisfying $\mathcal{V}^* \subseteq \mathcal{V}$, and an edge set $\mathcal{E}^*$ that includes all the edges associated with the elements in $\mathcal{V}^*$.*

Hence a vertex induced subgraph has fewer vertices and all the edges incident on these vertices from the edge set of the original graph. The edge induced subgraphs are likewise defined.

Definition 2.8 (Edge induced subgraph)**.** *An edge induced subgraph $\mathcal{G}(\mathcal{E}^*)$ of a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ has an edge set $\mathcal{E}^*$ which is a subset of the edge set of $\mathcal{G}$, i.e. satisfying $\mathcal{E}^* \subseteq \mathcal{E}$, and a vertex set $\mathcal{V}^*$ that includes all the vertices associated with the elements in $\mathcal{V}^*$.*

A subgraph that spans the entire graph is a spanning subgraph. Specifically, given some $\mathcal{G}(\mathcal{V}, \mathcal{E})$, a spanning subgraph $\mathcal{G}$ has the same vertex set $\mathcal{V}$ with $\mathcal{G}$.

2.2.2 *Elementary Graph Structures*

Examining a graph, one trivially observes that these are rich in structure. Nevertheless, apart from the basic elements of a graph, vertex and edge, there are structures that recur in almost any graph. These exhibit specific characteristics, can be easily defined and bear specific importance for the represented network.

In fact, there are many research areas where graphs are used as a model. The number of such structures, their presence or absence, and or their concentration allow to perform important abstractions related to each specific application. Moreover, in graph theory their existence, absence and quantification are of great importance.

The simplest structure identified on a graph is a walk. The latter is a sequence of vertices and edges. Based on a walk another two structures are identifiable, the path and the trail. The path is a walk where each vertex is found only once. The trail is a walk where each edge is found only once.

Another important structure on a graph is a closed walk, also called a cycle. The latter is a closed trail without duplicate vertices. A cycle cannot be found in all types of graphs. A cycle that visits every vertex in a graph is a *Hamiltonian path*. A graph with a *Hamiltonian path* is a so-called *Hamiltonian graph*. A summary of the basic structures of a graph is given below.

Definition 2.9 (Basic graph properties). *A subgraph of a graph $\mathcal{G}$ that exhibits some abstract property is a structure of the graph $\mathcal{G}$. The following basic structures are defined.*

- *A **walk** from v_1 to v_k is an interlacing sequence of adjacent vertices and edges $v_1, \{v_1, v_{j_1}\}, \{v_{j_1}, v_{j_2}\}, \ldots, \{v_{j_{i-1}}, v_{j_i}\}, \{v_{j_i}, v_{j_{i+1}}\}, \ldots, v_k$*
- *A **path** is a walk without any duplicate vertices*
- *A **trail** is a walk without any duplicate edges*
- *A **cycle** is a closed trail, $v_1 = v_k$, without any other duplicate vertices except v_1.*

For sake of simplicity, $p\{v_i, \ldots, v_j\}$ and $c\{v_i, \ldots, v_j\}$ shall denote a path and a cycle respectively from vertex v_i to v_j. Similar notation, $w\{v_i, \ldots, v_j\}$ and $t\{v_i, \ldots, v_j\}$, will be used where needed for walks and trails respectively.

2.2.3 *Elementary Graph Properties*

The elementary properties of a graph are the cardinalities of its vertex and edge sets. The *connectivity* and the *connectedness* of a graph are typically considered the two most important macroscopic properties of a graph. Literature related to connectedness can be found in (Erdos and Renyi, 1961), (Erdos and Renyi, 1959).

Definition 2.10 (Connectedness). *Let an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, then the connectedness of a graph $c(\mathcal{G})$ is the minimum number of edges needed to remove for the graph to become disconnected.*

A graph is said to be connected if there is a walk from any vertex to any other vertex in the graph.

Definition 2.11 (Connected Graph). *A graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is connected if and only if for any two vertices $v_i, v_j \in \mathcal{V}$ exists a walk $\{v_i, \ldots, v_j\}$. Such a graph is called a connected graph.*

Distinguishing between connected and disconnected graphs is of great importance, particularly in the case of networks with stochastic communication. Moreover, a graph where any two vertices are adjacent is said to be fully connected.

Definition 2.12 (Fully Connected Graph). *A graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is fully connected if and only if for any two vertices $v_i, v_j \in \mathcal{V}$ exists a walk with only one edge, $\{v_i, \{v_i, v_j\}, v_j\}$.*

In a disconnected graph, a unique decomposition into maximal connected components exists. These are the disconnected components. Particularly, a disconnected component is a disjointed connected subgraph. Hence, there is no walk between a vertex belonging in the component and another vertex not belonging to the subgraph. Apart from knowing if the graph is connected or not, one often wants to know the number disconnected components in the graph.

Components of a graph are abstract structures

It is advantageous to generalise the notion of a disconnected component to that of a component. Conceptually, the notion is abstract and remains difficult to define. Intuitively, component of a graph can be considered any subgraph where the connectivity within the subgraph is larger than the connectivity of the entire graph. Connectivity, even though it can be conceptually understood, still remains an abstract notion. We proceed to the clarification of the term, and for this purpose *Menger's theorem* is included. The theorem can be found in (Godsil and Royle, 2001).

Theorem 2.1 (Menger's Theorem)**.** *A graph $\mathcal{G}$ is* k*-connected if and only if for each pair of vertices v_i and v_j are* k *in number paths $\{v_i, v_j\}$ which have only v_i and v_j common.*

Connectivity of a graph and k*-connectedness*

Otherwise put, considering any pair of vertices $\{v_i, v_j\}$, a k-connected graph has k distinct paths $p\{v_i, \ldots, v_j\}$. Obviously, k-connected graphs are also $(k-1)$-connected as well. This observation directly leads to the definition of connectivity.

Definition 2.13 (Graph Connectivity)**.** *The connectivity of a graph $\kappa(\mathcal{G})$ is the maximum value of* k *for which a graph is* k*-connected.*

In other words, connectivity is the minimum of the number of distinct paths over all pairs of vertices $\{v_i, v_j\}$, in the graph.

In fact, connectivity, as defined, tells nothing about the number of components in a graph. However, connectivity can be incorporated to determine the components of a graph, which can be achieved by considering the difference in connectivity between subgraphs and the connectivity of the graph itself. Nevertheless, the resulting components might be unnatural.

This can be further understood by examining the graphs in (*fig.*3). There, any subgraph has smaller connectivity than the corresponding graph. However, one should not be able to distinguish undoubtedly components by simple observation.

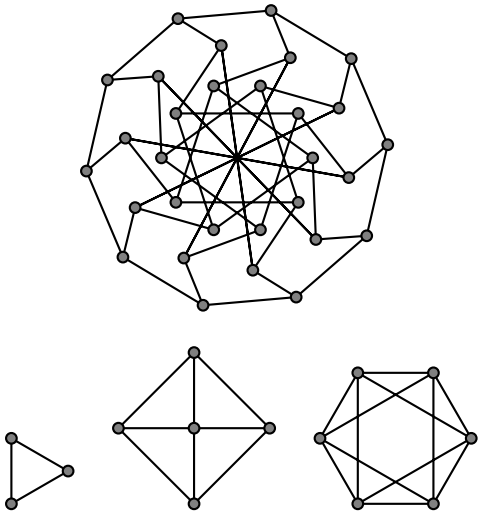


Figure 3: **Connectibity and Components ambiguity**. Connectivity cannot always be employed to distinguish components. The top graph is a uniform graph with $\kappa(\mathcal{G}) = 3$. Components are difficult to identify in a natural manner

The dual problem of defining a component and determining components of a graph, remains abstract. In the literature, many attempts have been made to define quantities for this purpose. Specifically, this problem connects with graph clustering, also known as grouping, partitioning and segregating. The latter remains an open problem. Connectivity does not fully capture the notion of components in a graph even though they are closely related. Nevertheless, the terms, connectivity and graph components are going to be employed in subsequent discussion as they can be commonly conceived. Connectivity in the perspective of algebraic graph theory is discussed in (*Section* 2.2.6). We concentrate on other basic properties of a graph for the remainder of this section.

graph distance function

A very basic property defined on a graph is the distance function. Even though it does not capture a conceptual notion related to the entire graph, it allows the definition of other properties.

Any function defined on the graph which has the properties of a metric is a graph distance function. Distances may be between vertices or edges, but distance are usually defined between vertices. Commonly, a distance function is defined with respect to a path between two vertices. In some cases a continuum can be defined on the graph along with a continuous function on that continuum.

This function can validly be considered as a distance function. Cases can be found in the literature where distance functions are not metrics. Usually, this occurs because these functions are not symmetric.

The shortest path between vertices is in many cases an appropriate choice for the definition of a distance. Specifically, the shortest path is a path, possibly not unique, that consists of the least number of elements; that is vertices, edges or both.

Definition 2.14 (Shortest Path). *Let the shortest path $\pi\{v_i, v_j\}$ (or π_{ij} for short) between any two vertices $v_i, v_j \in \mathcal{V}$ be the path $p\{v_i, \dots, v_j\}$ with the smallest cardinality, i.e. number of elements.*

$$\pi\{v_i, v_j\} = \arg\min_{v_i, v_j} |p\{v_i, \dots, v_j\}|$$

Summarising, within the scope of this research a distance function is implied to be defined with respect to the shortest path. The definition follows.

Definition 2.15 (Distance). *Assume a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and let two vertices v_i and v_j with $i \neq j$. The number of edges of the shortest path between vertices v_i and v_j defines their distance $d(v_i, v_j)$. Let $d(v_i, v_j) = \infty$ if and only if there is no path between v_i and v_j.*

It can be easily verified that (*Definition* 2.15) satisfies

$$d(v_i, v_j) \geqslant 0 \text{ with } d(v_i, v_j) = 0 \text{ iff } v_i = v_j$$
$$d(v_i, v_j) = d(v_j, v_i)$$
$$d(v_i, v_j) \leqslant d(v_i, v_h) + d(v_h, v_j)$$

and therefore it is a metric. An important property of all graphs is their diameter. The latter is defined as the length of the longest shortest path, or simply the maximum distance between any two vertices on the graph,

$$d(\mathcal{G}) = \max_{i,j} d(v_i, v_j) \tag{2.1}$$

where $d(\mathcal{G})$ denotes the diameter of the graph. The diameter has an important role for our research. As it shall be illustrated in (*Chapter* 5), consensus cannot be reached in less than $d(\mathcal{G})$ communication steps, i.e. the diameter of the graph.

2.2.4 *Classes of Graphs*

Unlike vertices, which are a rather dull construct, edges can be found in many different types. Graphs can be classified with respect to the types of edges defined on the graph. The edges can be undirected, directed, multiple, and loops. The latter, may be present in all three classes of graphs, undirected, directed and multi-graphs. Thus, we can define three classes of graph depending on the edge type found on them, undirected, directed and multi-graphs. Their presentation follows, subsequently.

2.2.4.1 *Undirected Graphs*

The undirected graph is the simplest type of graph. There, any two connected vertices are linked with only one edge. The definition follows.

Definition 2.16 (Simple Graph without loops)**.** *An undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a graph having an edge set $\mathcal{E}$ satisfying:*

1. *The pairs $\{v_i, v_j\}$ in the edge set $\mathcal{E}$ are unique.*
2. *The pairs $\{v_i, v_j\}$ and $\{v_j, v_i\}$ are considered identical $\forall i, j$.*
3. *There is no pair $\{v_i, v_j\} \in \mathcal{E}$ such that $i = j$ $\forall i, j$.*

Clarifying, an undirected graph does not have duplicate edges. These do not have a direction and they can be equivalently signified by writing either $\{v_i, v_j\}$ or $\{v_j, v_i\}$ for some i and j. Commonly, in the definition there are no edges connecting a vertex with itself. However, omitting the last constraint we can define a so-called undirected graph with self-edges, also called loops. We do not make distinction among the two in the text and normally it should be assumed that loops may be present. It shall be noted otherwise.

Definition 2.17 (Simple Graph)**.** *An undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a graph having an edge set $\mathcal{E}$ satisfying all the following conditions.*

1. *The pairs $\{v_i, v_j\}$ in the edge set $\mathcal{E}$ are unique.*
2. *The pairs $\{v_i, v_j\}$ and $\{v_j, v_i\}$ are considered identical $\forall i, j$.*

Undirected graphs without loops are an appropriate model for the case of synchronous bidirectional communication in networks. However, local computations or communications, e.g. the loopback interface, cannot be represented with an undirected graph. In such a case, an undirected graph with self-edges is appropriate for the description of the network's operation.

2.2.4.2 *Directed Graphs*

A directed graph or digraph is a graph where edges have direction. An edge is either incoming to a vertex or outgoing.

Definition 2.18 (Directed Graph, without loops)**.** *A directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ has an edge set $\mathcal{E}$ satisfying the following conditions.*

1. *The pairs $\{v_i, v_j\}$ in the edge set are unique.*
2. *There is no pair $\{v_i, v_j\}$ such that $i = j$ $\forall i, j$.*

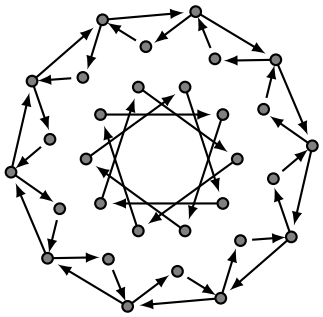


Figure 4: **A directed graph**.

The first condition designates that for each pair of adjacent vertices there can be only one edge per direction. The second condition implies that there cannot be an edge connecting a vertex with itself. As in the case of an undirected graph we can define self-edges. However, it does not make sense to define directed edges onto ones self. Hence, loops $\{v_i, v_i\}$ should always be considered as undirected.

Definition 2.19 (Directed Graph)**.** *A directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ has an edge set $\mathcal{E}$ with unique pairs $\{v_i, v_j\}$.*

Directed graphs are appropriate to describe communication in networks where transmission and reception may not be performed between any two machines. Again, the self-edge may represent a local computation, local communication, or just a local state update.

Directed graphs are connected if the associated undirected graphs, obtained by transforming each directed edge to an undirected, is connected. A digraph is said to be strongly connected when there exists a path connecting any two vertices.

2.2.4.3 *Multi-graphs*

A directed multi-graph has both directed edges and self-edges. For each pair of vertices there can be multiple edges defined.

Definition 2.20 (Multi-graph)**.** *A multigraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ has an edge set with at least two edges connecting the same two vertices v_i and v_j.*

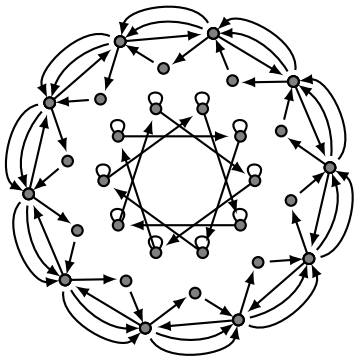


Figure 5: **A directed multi-graph with self-edges**.

In accordance with the simple graph definition (*Definition* 2.6), the edge set may include multiples of some pairs $\{v_i, v_j\}$. Conceptually, this poses no difficulties. However, one might want for some purpose to discriminate among these multiple edges. There are two ways to tackle this. One either uses an additional label of the multiple edges or uses the complex definition (*Definition* 2.1) which is inclusive of the incidence function. Hence, only the latter needs to be defined. However, it would be convenient to avoid the definition of the incidence function. Below, such a definition of a multi-graph is given.

Definition 2.21 (Directed Multi-graph)**.** *Let $\mathcal{G}$ be a directed multi-graph with vertex set $\mathcal{V}$ and an edge set $\mathcal{E}$. Let the edge set be a triple $\{\{v_i, v_j\}, k\}$ where $k \in \{1, 2, \ldots, n_{ij}\}$ and n_{ij} is the multiplicity of the elements of $\mathcal{E}$ that contain the pair $\{v_i, v_j\}$, signifying a directed edge from vertex v_i to v_j. Moreover, let $\{\{v_i, v_j\}, k\}$ be unique.*

Illustrating, we have modified the definition of the edge set and introduced triplets instead of pairs. The third index introduced is a numbering of the multiple edges, going from one up to the multiplicity of each pair $\{v_i, v_j\}$.

2.2.5 *Types of Graphs*

Four specific types of graphs are of interest to us, trees, random graphs, random geometric graphs, and small world graphs. The first, because it is related to the definitive consensus algorithm, presented in (*Chapter* 4). Random graphs are employed to research the properties of large graphs. These can also be used as a model of stochastic communication. Random geometric graphs are an appropriate model in the case of random placement on a surface of wireless sensor nodes. Finally, small world networks are commonly employed to model computer networks and social networks, e.g. the Internet. We proceed in their definition and the presentation of their most important properties.

2.2.5.1 *Trees*

Intuitively a tree is graph having a root, branches and leaves. The vertices are ordered hierarchically from the root to the leaves. Specifically, there is a root vertex which is adjacent directly to at least another two vertices. There, are also leaves which are adjacent with exactly one other vertex. Every leave is connected to the root node through a unique path. Therefore, in a tree there are no cycles. The latter is the definitive property for trees.

Definition 2.22 (Tree)**.** *A maximal undirected connected graph without cycles is a tree.*

Moreover, a tree has $n-1$ edges, where n is the number of vertices. In fact any connected graph with $n-1$ edges is a tree and therefore has no cycles. A connected graph with n edges has exactly one circle and is called an unicyclic graph.

In many cases, it is important to find a tree in a graph that spans the entire vertex set of the original graph. These are called spanning trees and are very important, especially in applications that involve routing, e.g. in computer networks or transportation networks. Spanning trees bear importance in our work as well.

Definition 2.23 (Spanning Tree)**.** *Given a graph $\mathcal{G}$, a spanning tree of $\mathcal{G}$ is a spanning subgraph of $\mathcal{G}$ that is a tree.*

The number of spanning trees in a graph is given by *Kirchhoff's Matrix-Tree Theorem*. The theorem simply states that the number of spanning trees in a graph is the product of the non-zero eigenvalues of the *Laplacian* matrix divided

by the cardinality of the vertex set. The definition of the *Laplacian* matrix is given in (*Section* 2.2.6.4).

Theorem 2.2 (Kirchoff's Matrix Tree Theorem)**.** *Let a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with cardinality $|\mathcal{V}| = n$. Let $\mathcal{T}(\mathcal{G})$ denote the number of spanning trees in $\mathcal{G}$ and $\lambda_i(\mathbf{L})$ denote the eigenvalues of the Laplacian matrix. Assume, that the eigenvalues for $i = k, k+1, \ldots, n$ are non-zero. The equation holds:*

$$\mathcal{T}(\mathcal{G}) = \frac{1}{n} \prod_{i=k}^{n} \lambda_i(\mathbf{L}) \tag{2.2}$$

2.2.5.2 *Random Graphs*

Erdos and Renyi introduced a random generating process of a graph in an effort to capture the emerging structural properties in large graphs. The so-called *random graphs* describe allegedely a "typical" graph of a certain vertex set size, (Durrett, 2006).

The generating process of an Erdos-Renyi random graph is as follows. Assume that there are n vertices in the vertex set. Then, one picks at random s edges from the $n(n-1)/2$ possible edges in the fully connected graph. This process can be slightly modified by assigning a probability $p \in (0, 1)$ for the presence of each edge on the graph. Obviously, the second method is inclusive of the first for $p = 2s/n(n-1)$. The generating process of random graphs shall be employed in the theoretical model for the case of communication with link failures.

The most important property of a random graph is the emergence of the giant component. The latter, aids our intuition of the convergence in the case of stochastic communications.

Let the probability of the edges be $p = c/n$ with $c > 1$. We remark that there is a constant $\theta(c) > 0$ such that for n large the largest component has near $\theta(c)n$ vertices and the second largest component has size in the order of $\log(n)$, see (Durrett, 2006), (Erdos and Renyi, 1959). In the case that $c < 1$, then for large n the graph has many small components and the largest component has order of size $\log(n)$. These conjectures are supported by the theorem below, found in (Erdos and Rényi, 1959)

Theorem 2.3. *Let $\mathcal{G}_{n,m}$ be a random graph with n possible vertices and m edges. Let $P_k(n, m(c))$ denote the probability of the greatest connected component of $\mathcal{G}_{n,m}$ consisting of $n-k$ points. Then we have*

$$\lim_{n\to\infty} P_k(n, m(c)) = \frac{e^{-2kc-e^{-2c}}}{k!} \tag{2.3}$$

where $m(c) = \left[\frac{1}{2} n \log n + cn\right]$.

Hence, the probability that there exists a component with less than $n-1$ vertices is almost zero when $c > 1$. In contrast, for $c < 1$ this is larger than zero.

Apart from random graphs, other categories of graphs are employed to model various types of real networks. Such a model for computer and social networks are the *small world* networks. Random placement of wireless sensor nodes can be modelled with *random geometric* graphs. We introduce these two graphs below. The reader could consult (Wang and Chen, 2003) for an illustration of the subject.

2.2.5.3 *Random Geometric Graphs*

The construction process of a random geometric graph consists of selecting a square region, e.g. $[0,1]^2$, and uniformly distributing points on it. Then by defining a range $r > 0$, edges can be assigned between those nodes that have distances smaller than r. The resulting graph might not be connected.

These types of graphs are a good representation for a topology emerging from the random placement of wireless sensor nodes that form a communication network. This model is going to be used extensively for our numerical simulations, see (*Chapter* 6).

The emergence of the giant component occurs also in the case of random geometric graphs. However, in comparison to random graphs, where the giant component occurs when the *"mean vertex degree exceeds the critical value 1"* for n large, in this case this happens when the mean vertex degree is finite, (Penrose, 2003).

2.2.5.4 *Small World Graphs*

Small world graphs emerge often in social or community networks, like friendship, author and actor networks. In those cases, the random graph model does not entirely capture the result nor the generative process can be represented with the Erdos-Renyi generating process.

The small world network has two principal properties. First, it is a connected network. Second, it has the small world property. The latter is somehow vaguely defined in the literature, see (Wang and Chen, 2003), (Watts and Strogatz, 1998). One could intuitively state that the small-world property is that large networks retain a surprisingly small diameter or "'*remain small while large*".

The construction of a network with the small world property has been pioneered by (Watts and Strogatz, 1998). The process is as follows. First, construct a ring network with n vertices and m/n edges per vertex. Then, each edge is rewired with some probability p connecting two vertices at random. Another construction process is described in (Newman and Watts, 1999). There, the edges are not rewired at random but some edges are added between long distance pairs with some probability. The number of shortcuts is drawn from a *Poisson distribution* with mean $np/2$.

Within this thesis we are going to utilise the second process (Newman and Watts, 1999) for a number of simulations in (*Chapter* 6).

2.2.6 *Algebraic Graph Theory*

The combinatorial foundation of a graph found in (*Section* 2.2) can be extended algebraically. The definition of the adjacency, the incidence and the *Laplacian* matrix are based on accounting the relations of edges and vertices. The properties of these matrices, and especially their spectral properties, lead to an algebraic approach to the examination of graphs that describes well many macroscopic properties of the system represented by the graph. Particularly, the convergence of the consensus algorithm, under a dynamical system approach, is well described with the eigenvalues and the eigenvectors of the Laplacian matrix, (*Section* 2.3).

2.2.6.1 *The Adjacency Matrix*

The adjacency matrix accounts which vertices are connected to each vertex. Hence, it captures the information of the vertex-vertex relation; the definition follows.

Definition 2.24 (Adjacency Matrix)**.** *Suppose a graph* $\mathcal{G}(\mathcal{V}, \mathcal{E})$*. Let the adjacency matrix* $\mathbf{A} \in \mathbb{R}^{n\times n}$ *be the matrix of elements* $[\mathbf{A}]_{ij} = a_{ij}$ *with* $a_{ij} = 1$ *if the edge* $\{v_i, v_j\} \in \mathcal{E}$ *and* $a_{ij} = 0$ *otherwise.*

The adjacency matrix is also known as connectivity matrix or communications matrix; we prefer the term adjacency matrix.

In the case of undirected graphs without self-edges, the adjacency matrix is a symmetric matrix with all the elements on the diagonal equal to zero. Considering self-edges in undirected graphs then there are elements on the diagonal. In

the case of a directed graph with self-edges, the matrix is asymmetric. The case of a multi-graph cannot be represented by the adjacency matrix.

2.2.6.2 *The Incidence Matrix*

The incidence matrix accounts the relation of edges with vertices, i.e. the vertex-edge relation. Specifically, the elements of the matrix record the connection of an vertex with the edges it is connected.

Definition 2.25 (Incidence Matrix, Simple Graph)**.** *Assume an undirected graph* $\mathcal{G}(\mathcal{V}, \mathcal{E}, \gamma)$*. Let the incidence matrix be a matrix of elements* $[\mathbf{Q}]_{ij} = q_{ij}$ *with* $q_{ij} = 1$ *and* $q_{mj} = -1$ *if* $e_j \in \mathcal{E}$ *with* $\{v_i, v_m\} = \gamma(e_j)$ *otherwise* $q_{ij} = 0$*.*

Other definitions of the incidence matrix for undirected graphs are also present in the literature. Specifically, the elements are strictly positive. The elements of the matrix q_{ij} and q_{mj} are both signified with 1. Instead we chose positive and negative values indicating direction because in that manner this can be directly extend to digraphs. In the case of a digraph the definition is only slightly different. Particularly, we have to explicitly assign -1 for the outgoing and 1 for the incident edges.

Definition 2.26 (Incidence Matrix)**.** *Assume a directed graph* $\mathcal{G}(\mathcal{V}, \mathcal{E}, \gamma)$*. Let the incidence matrix* $\mathbf{Q} \in \mathbb{R}^{n \times m}$ *be a matrix of elements* $[\mathbf{Q}]_{ij} = q_{ij}$ *with* $q_{ij} = 1$ *if* $e_j \in \mathcal{E}$ *with* $\gamma(e_j) = \{v_i, v_k\}$*,* $q_{ij} = -1$ *if* $e_j \in \mathcal{E}$ *with* $\gamma(e_j) = \{v_k, v_i\}$ *for some* $k \in \{1, 2, \ldots, n\}$*, otherwise* $q_{ij} = 0$ *where* $n = |\mathcal{V}|$ *and* $m = |\mathcal{E}|$*.*

$$[\mathbf{Q}]_{ij} = \begin{cases} 1 & e_j \in \mathcal{E} \quad \gamma(e_j) = \{v_i, v_k\} \\ -1 & e_j \in \mathcal{E} \quad \gamma(e_j) = \{v_k, v_i\} \\ 0 & e_j \notin \mathcal{E} \end{cases}$$

This definition includes as well the cases of directed, multi-graphs, and simple graphs. The case of a simple graph's incidence matrix is included in the aforementioned definition for directed graphs by replacing each undirected edge with a couple of directed edges, one for each direction. However, we chose to define it explicitly in order to present the matter to the reader in an intuitive manner, instead of leaving this to be vaguely implied.

2.2.6.3 *The Weight Matrix*

In the case of the adjacency matrix the presence or absence of an edge between two vertices is marked is noted with 1 and 0, respectively. Instead in the case of

the weight matrix the a connection's strength or importance can be related to a coefficient; the absence of a connection should be marked with 0.

In some cases the coefficients, defined on the edges, may resemble transition probabilities or graph traversal probabilities, as in probabilistic modelling. In other cases the weights may determine information flow throughput. Therefore, their meaning is specific to the application at hand. In our perspective we avoid to strictly define their meaning. Instead, letting these coefficients be abstractly related to an edge allows to disconnect the discussion from a bottom-up approach and let us develop a top-down method for their determination. Hence, the weight matrix can be defined as the matrix of coefficients associated with each separate edge, as in the case of the adjacency matrix.

Definition 2.27 (Weight Matrix)**.** *Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the weight matrix $\mathbf{W} \in \mathbb{C}^{n \times n}$ is defined as*

$$[\mathbf{W}]_{ij} = \begin{cases} w_{ij} & \text{iff} \quad \{v_i, v_j\} \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

where $w_{ij} \in \mathbb{C}$ and $n = |\mathcal{V}|$ the cardinality of the vertex set.

In contrast with the definition commonly given by many authors, we do not restrict the edge weights to the positive real domain. Complex coefficients and negative weights do not have an intuitive meaning like transition probabilities. It is for this reason that these are overlooked in the literature. However, as we are going to see in (*Chapter* 4) their usage provides many advantages. Moreover, this justifies our decision to disconnect the definition from the meaning of the edge weights.

2.2.6.4 *The Laplacian Matrix*

The *Laplacian* matrix has an established position in the literature. This is mainly due to the fact that its spectrum relates to many properties of natural systems which can be described by a graph.

The term *Laplacian* matrix draws from the study of vibrations of membranes, initiated from the paper of (Kac, 1966). A discrete approximation of the membrane with a set of points leads to a graph. The vertical displacement of the membrane is an eigenfunction of the *Laplacian* operator of the drum. There, the *Laplacian* matrix is a discrete approximation of the drum. Initially, the *Laplacian* matrix was introduced by Kirchoff in view of the matrix tree theorem, (*Theorem*

2.2). It has been evident throughout the years that the *Laplacian* matrix bares significance in many fields of research.

The *Laplacian* matrix is retrieved from the difference of the degree matrix minus the adjacency matrix, $\mathbf{L} = \mathbf{D} - \mathbf{A}$. The matrix $\mathbf{D}$ is a diagonal matrix with the vertex degrees across the diagonal. Hence, the degree of the i^{th} vertex is the element $\mathbf{D}_{ii}$. The degree of a vertex is the number of adjacent edges. In the case of digraphs and multi-graphs the in-degree and out-degree is defined as the number of incoming and outgoing edges respectively.

The *Laplacian* matrix of an undirected graph is equal with the square product of the incidence matrix.

$$\mathbf{L} = \mathbf{Q}\mathbf{Q}^T \tag{2.4}$$

The latter will prove to be a rather useful equality. The *Laplacian* has row and column sums equal to zero. The *Laplacian* matrix for a digraph is difficult to define. Usually, it is suggested that the in-degree or the out-degree should be used, as seem fitting.

The weighted *Laplacian* matrix can also be defined in a similar manner. The latter is the difference of the unity matrix minus the weight matrix $\mathbf{L} = \mathbf{I} - \mathbf{W}$. In the case of an undirected weighted graph, the equality $\mathbf{L} = \mathbf{Q}diag(w)\mathbf{Q}^T$ holds, where $diag(w)$ is a diagonal matrix with the edge weights on the diagonal.

2.2.6.5 *Spectral properties of Graph Matrices*

Every graph has an associated unique *Laplacian* matrix but the reverse does not hold. The spectrum of the *Laplacian* is very important for the study of the consensus algorithm's convergence.

The eigenvalues of the *Laplacian* and the weight matrix are associated with the formula:

$$\lambda_i(\mathbf{L}) = 1 - \lambda_i(\mathbf{W}) \tag{2.5}$$

where $\lambda_i(\mathbf{x})$ is the i^{th} eigenvalue of a matrix $\mathbf{x}$.

The *Laplacian* of an undirected graph is a symmetric positive semi-definite matrix. This follows from (*eq.*2.4)

$$\mathbf{L}\mathbf{x} = \lambda\mathbf{x} \Rightarrow$$
$$\lambda\mathbf{x}^T\mathbf{x} = \mathbf{x}^T\mathbf{L}\mathbf{x} = \mathbf{x}^T\mathbf{Q}\mathbf{Q}^T\mathbf{x} = \|(\mathbf{Q}^T\mathbf{x})\|^2 \geqslant 0$$

where λ is an eigenvalue of the *Laplacian* matrix. Similar results can be retrieved for the weighted *Laplacian* of an undirected graph,

$$\begin{aligned} &\mathbf{L}\mathbf{x} = \lambda\mathbf{x} \Rightarrow \\ &\lambda\mathbf{x}^*\mathbf{x} = \mathbf{x}^*\mathbf{Q}\mathrm{diag}(w)\mathbf{Q}^T\mathbf{x} = \sum_i w_i[(\mathbf{Q}^T\mathbf{x})^2]_i \end{aligned} \tag{2.6}$$

where i is an index of the edges. Hence, the positiveness of the eigenvalues is determined from the positive edge weights. In this case, positive edge weights guarantee that the eigenvalues are positive semidefinite. This is also supported from the *Perron-Frobenius* theorem below.

Theorem 2.4 (Perron-Frobenius)**.** *Let a matrix* $\mathbf{x} \in \mathbb{R}^{n\times n}$ *such that* $[\mathbf{x}]_{ij} > 0$, $\forall i, j \in \{1, 2, \ldots, n\}$. *Then* $\mathbf{x}$ *has a non-negative eigenvalue* $\rho > 0$*, called the perron eigenvalue, such that* $0 < |\lambda_i| < \rho$, $\forall i \in \{1, 2, \ldots, n-1\}$ *. The associated eigenvector with the perron eigenvalue* ρ *is entry-wise non-negative, i.e.* $x_j > 0$, $\forall j \in \{1, 2, \ldots, n\}$*, called the Perron eigenvector. All other eigenvectors are not strictly positive, i.e. any other eigenvector has at least one negative, zero, or complex element.*

The theorem cannot be applied in general to any weight matrix. Since it is expected that there shall be zero elements, marking non-connected pairs of vertices. In (*Corollary* 2.1), (*Theorem* 2.4) is extended to include matrices that are not strictly positive but contain at least one zero element.

Corollary 2.1. *Let a non-negative matrix* $\mathbf{x} \in \mathbb{R}^{n\times n}$*, then* $\mathbf{x}$ *has a non-negative eigenvalue* $\rho \geqslant 0$*, such that* $|\lambda_i| \leqslant \rho$, $\forall i \in \{1, 2, \ldots, n-1\}$ *. There is an associated eigenvector* $\mathbf{x}$ *with the perron eigenvalue* ρ *that is non-negative* $\mathbf{x} \geqslant \mathbf{0}$*.*

where $\mathbf{x} \geqslant 0$ implies element-wise comparison $x_i \geqslant 0$ $\forall i \in \{1, 2, \ldots, n\}$. Arbitrary selection of positive real weights guarantees that the eigenvalues of the weight matrix are going to be smaller in modulus than a positive eigenvalue. According (*eq.*2.6) the eigenvalues of the *Laplacian* shall also be positive. The eigenvalues of the weight matrix are found in $\lambda(\mathbf{W}) \in [-\rho, \rho]$ when $\mathbf{W}$ is *Hermitian*. Otherwise, these are upper bounded by the *Perron* eigenvalue from (*Corollary* 2.1). Thus, the eigenvalues of the *Laplacian* are $\lambda(\mathbf{L}) \in [0, 1 + \rho]$.

The eigenvalues of the *Laplacian* matrix of $\mathcal{G}$ an undirected graph are bounded and ordered as

$$0 = \lambda_1(\mathbf{L}) \leqslant \lambda_2(\mathbf{L}) \leqslant \ldots \leqslant \lambda_n(\mathbf{L})$$

with $\lambda_n(\mathbf{L}) = \rho(\mathbf{L})$. Since row sums are always 0, it follows that $\lambda_1 = 0$, and the associated eigenvector is the unity vector $\mathbf{1} = (1, 1, \ldots, 1)^T$. Furthermore,

the number of the connected components in $\mathcal{G}$ is equal to the number of zero eigenvalues. Therefore, a connected graph has an associated *Laplacian* with a unique zero eigenvalue.

Remark 2.1. *A graph $\mathcal{G}$ is connected if and only if the associated Laplacian matrix has a unique eigenvalue equal to zero. Therefore the associated spectrum is*

$$0 = \lambda_1(\mathbf{L}) < \lambda_2(\mathbf{L}) \leqslant \ldots \leqslant \lambda_n(\mathbf{L}) \tag{2.7}$$

In the case of weight matrices with complex and possibly negative elements we have to result in *Gerschgorin's circle theorem*. We include this here for ease of reference.

Theorem 2.5 (Gerschgorin's Circle Theorem)**.** *Let X be a square complex matrix. Around every point on the complex plain corresponding to an element $[\mathbf{x}]_{ii}$ on the diagonal of the matrix, we draw a circle with radius the sum of the norm of the other elements on the same row $r_i = \sum_{j=1, j\neq i}^{n} |\mathbf{x}_{ij}|$, called Gerschgorin discs. Every eigenvalue of A lies in one of these Gerschgorin discs. Hence the eigenvalues lie in the union of the* Gerschgorin discs*.*

$$\bigcup_{i=1}^{n} \{z \in \mathbb{C} : |z - [\mathbf{x}]_{ii}| < r_i\}$$

Drawing from (*Theorem* 2.5), the following result can be obtained for the eigenvalues of a digraph (Godsil and Royle, 2001).

Theorem 2.6. *Let $\mathbf{X} \in \mathbb{C}^{n\times n}$ associated with a weighted digraph $\mathcal{G}$, then $\mathbf{x}$ has n eigenvalues that lie in the union of the cycles*

$$\bigcup_{c} \left\{ z \in \mathbb{C} : \prod_{c} |z - x_{ii}| \leqslant \prod_{c} r_i \right\}$$

where $r_i = \sum_{j=1, j\neq i}^{n} |\mathbf{x}_{ij}|$ and $\bigcup_c$ implies the union over all cycles c of $\mathcal{G}$ and $\prod_c$ is the product of the elements corresponding to the vertices along c.

2.3 CONSENSUS

The consensus algorithm has been introduced in (*Section* 1.2). Here we present a detailed mathematical presentation of the matter. The algorithm is designed to be executed in a distributed and decentralised manner on a set of machines. These machines can perform some simple local computations and are able to communicate arbitrarily. What needs to be noticed is that this communication is expected to be synchronous.

2.3.1 *Model of Communication*

Suppose a set of machines, possibly different, but for simplicity's sake, we can assume them to be identical; at least in their basic manifestation for computation and communication. This does not lead to loss of generality. The connection topology between them can be described with a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. The vertex set $\mathcal{V}$ is in fact a labelling of the machines. Assume that in a given a time frame the machines can complete all necessary communications, and that given an identical time frame in the future, these communications can similarly recur. These persistent connections are represented by the edges in $\mathcal{E}$.

If we assume communications to be bidirectional, then an undirected graph is an appropriate choice of model. In contrast, if the edge coefficients are assigned differently for the incoming and outgoing edges, the associated graph with the network is a weighted digraph. In case that bidirectional communications cannot be guaranteed, the resulting graph should necessarily be conceived as a digraph.

The case of failing links can also be easily represented. In that case, the assumption is that communication between these links cannot be guaranteed to have been completed within the predefined time frame. This case can be modelled with a digraph with a presence probability associated with each edge.

2.3.2 *The Consensus Algorithm*

Suppose that every vertex in the graph has an associated scalar value. The purpose of the algorithm is to compute the mean over these scalar values, and make it available at each vertex. These tasks is achieved by the consensus algorithm by local synchronous communication between connected vertices and computation of local weighted averages.

The linear consensus algorithm (Lynch, 1996) consists of a simple vertex-local update equation.

$$x_i(t+1) = \sum_j w_{ij} x_j(t) \tag{2.8}$$

where $x_i \in \mathbb{R}$ is the state variable defined on the vertex, and $w_{ij} \in \mathbb{R}_+$ are the coefficients associated with the edges of the graph. Specifically, $w_{ij} \neq 0$ when vertices v_i and v_j are connected by an edge on the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and $w_{ij} = 0$

otherwise. These coefficients are such that the coherence of the local estimates is guaranteed.

The global update equation can be readily retrieved in matrix form. Hence, (*eq.*2.8) becomes,

$$\mathbf{x}(t+1) = \mathbf{W}\mathbf{x}(t) \tag{2.9}$$

where the states associated with each vertex constitute now the elements of a vector $\mathbf{x} \in \mathbb{R}^n$ associated with the vertex set $\mathcal{V}$. The elements of $\mathbf{W} \in \mathbb{R}^{n\times n}$ are the coefficients on the edges, $[\mathbf{W}]_{ij} = w_{ij}$. The process is presented in (*Algo.*2.1) where t is the iteration index, and q is the number of iterations executed until

Algorithm 2.1 Consensus Algorithm, $\mathfrak{C}(\mathbf{W}, \mathbf{x}, q)$

1: Execute the while loop for every i^{th} machine simultaneously
2: **for** $t = 1$ to q **do**
3: $\quad x_i \leftarrow \sum_{j=1}^{n} [\mathbf{W}]_{ij} x_j$
4: **end for**

termination of the algorithm. The convergence of the algorithm depends solely on the selection of these coefficients. Sufficient conditions for convergence are

$$\mathbf{W}^T = \mathbf{W} \tag{2.10}$$

$$\mathbf{W}\mathbf{1} = \mathbf{1} \tag{2.11}$$

$$\rho(\mathbf{W} - \mathbf{1}\mathbf{1}^T/n) < 1 \tag{2.12}$$

where $\mathbf{1} = (1, 1, 1, \ldots, 1)^T \in \mathbb{R}^n$, and $\rho(\cdot)$ denotes the spectral radius. Due to the presence of the eigenvector $\mathbf{1}$ with eigenvalue 1, the dynamical system has an infinity of fixed points of the form $\mathbf{x}_* = \alpha\mathbf{1}$ where $\alpha \in \mathbb{R}$. Asymptotic convergence of the solutions of (*eq.*2.9) to a fixed point is enforced due to condition $\rho(\mathbf{W} - \mathbf{1}\mathbf{1}^T/n) < 1$ and that fixed point under conditions in (*eq.*2.10), (*eq.*2.11), (*eq.*2.12) is such that $\alpha = \mathbf{1}^T\mathbf{x}(0)/n$, i.e. α is the arithmetic mean of the initial states. The reader is directed to (Xiao et al., 2007) for further details on the subject.

The conditions can be enforced easily when the topology of the graph is known a priori. In fact, the problem of selecting the coefficients of the matrix $\mathbf{W}$ in order to minimise $\rho(\mathbf{W} - \mathbf{1}\mathbf{1}^T/n)$ has been shown to be convex (Xiao et al., 2007). In cases of unknown topology there is a number of coefficient assignment schemes (Xiao and Boyd, 2003) that guarantee that the unknown matrix $\mathbf{W}$ will

satisfy the conditions in (*eq.*2.10), (*eq.*2.11), (*eq.*2.12). These matters are detailed below in (*Section* 2.3.3).

Practically, we cannot execute the algorithm for an infinite number of iterations. The number of iterations executed, denoted by $q \in \mathbb{Z}^+$, affects the precision of estimation of the mean and the level of agreement. Moreover, depending on the graph and the initial variance of the state vector $\mathbf{x}$, maximum arithmetic precision supported by a specific machine will be attained, after a large number of iterations, i.e. for q large.

2.3.3 *Fixed Matrices*

There is a number of issues related to the consensus algorithm, among the most prominent being the selection of the edge coefficients. These affect the convergence and the speed of convergence of the algorithm. Another important matter is the case of failing communication links between machines. In the literature, the current trend is to set the coefficients a priori. This results in a fixed weight matrix $\mathbf{W}$. We proceed in detailing the different cases which arise with respect to the communication model.

2.3.3.1 *Symmetric*

In the case of bidirectional communication, we can select the edge weights such that $w_{ij} = w_{ji}$. This results in a symmetric weight matrix; thus the condition in (*eq.*2.10) is automatically satisfied. When the other two conditions are satisfied, (*eq.*2.11), (*eq.*2.12) from the selection of the edge weights, the system will converge to the mean of the values initially distributed over the nodes in the network.

A few remarks can be made about these weight matrices. The matrix is symmetric but not necessarily *Hermitian*. The weight assignment may be such that the conditions for consensus are satisfied but the matrix is not diagonalisable.

In case that the matrix is real $\mathbf{W} \in \mathbb{R}^{n\times n}$, then it is necessarily *Hermitian*, normal and diagonalisable. Thus it has a full set of distinct eigenvalues and a full set of orthonormal eigenvectors $\mathbf{U}^{-1} = \mathbf{U}^T$,

$$\mathbf{W} = \mathbf{U}\Lambda\mathbf{U}^T \tag{2.13}$$

where the columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{W}$. The eigenvectors of $\mathbf{W}$ form a basis of $\mathbb{R}^{n\times n}$.

In case that $\mathbf{W}$ is definitely complex, i.e. it has at least one non-real element, it cannot be *Hermitian*, instead we say that it is complex symmetric. Unfortunately,

we cannot guarantee that it is diagonalisable. However, if $\mathbf{W}$ has a full set of distinct ,possibly complex, eigenvalues, then it also has a full set of orthogonal eigenvalues as in

$$\mathbf{W} = \mathbf{U}\Lambda\mathbf{U}^{*} \tag{2.14}$$

Furthermore, the weight matrix has an eigenvalue equal to 1 due to (*eq.*2.11).

When $\mathbf{W} \in \mathbb{C}^{n\times n}$ is not diagonalisable, then it is fairly trivial to show that the system diverges when there are two eigenvalues equal in modulus to 1. This is mainly due to the additional binomial terms in the relative *Jordan block* that would cause the state $\mathbf{x}(t)$ not to converge as $t \to \infty$. The proof is included in (*Chapter* A). Hence, the eigenvalues of the weight matrix are bounded such that $|\lambda_i(\mathbf{W})| \in [0, 1)$, $\forall i \in \{2, 3, \dots, n\}$ and $\lambda_1(\mathbf{W}) = 1$.

In the case that $\mathbf{W} \in \mathbb{C}^{n\times n}$ and diagonalisable, it has a full set of distinct eigenvalues, possibly complex, and the eigenvectors are orthogonal. Moreover, these eigenvalues satisfy $|\lambda_i(\mathbf{W})| \in [0, 1)$, $\forall i \in \{2, 3, \dots, n\}$ and $\lambda_1(\mathbf{W}) = 1$. The presence of two eigenvalues equal to 1 and -1, respectively, would imply that there are two eigenvectors $\mathbf{u}_1 = \mathbf{1}$ and $\mathbf{u}_{-1}$, respectively; which is excluded due to (*eq.*2.11).

Applying (*eq.*2.5), we arrive at the conclusion that the eigenvalues of $\mathbf{L}(\mathcal{G})$, where $\mathcal{G}(\mathcal{V}, \mathcal{E})$ in an undirected weighted graph with possibly complex coefficients, have to be such that $|\lambda_i(\mathbf{L})| \in [0, 2)$, $\forall i \in \{1, 2, \dots, n\}$. Moreover, due to the fact that the one eigenvalue $\lambda_1(\mathbf{W}) = 1$ is unique, the zero eigenvalue $\lambda_1(\mathbf{L}) = 0$ is unique, as well. We remind that the number of connected components are equal to the number of zero eigenvalues of the *Laplacian* matrix. Therefore, $\mathcal{G}$ has to be a connected graph something that is intuitively expected. Obviously, the consensus algorithm cannot convey information among disconnected components. Therefore, these cannot be in consensus unless the initial state $\mathbf{x}$ is already such. A summary of the properties of $\mathbf{W}$ when $\mathbf{W} = \mathbf{W}^{\mathrm{T}}$ is given below.

Corollary 2.2. *Let a connected weighted undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, its weight matrix $\mathbf{W}$, and state vector $\mathbf{x} \in \mathbb{R}^n$ associated with $\mathcal{V}$. Let $\mathbf{W}$ satisfy $\mathbf{W} = \mathbf{W}^{\mathrm{T}}$, $\mathbf{W}\mathbf{1} = \mathbf{1}$, $\rho(\mathbf{W} - \mathbf{1}\mathbf{1}^{\mathrm{T}}/n) < 1$. The following predicates are true for $\mathbf{W}$:*

1. *If $\mathbf{W} \in \mathbb{R}^{n\times n}$, then the following statements are true:*
   - $\mathbf{W}$ *is Hermitian*
   - $\mathbf{W}$ *has a full set of distinct real eigenvalues*
   - $\mathbf{W}$ *is diagonalisable as* $\mathbf{W} = \mathbf{U}\Lambda\mathbf{U}^{\mathrm{T}}$

- *It has a full set of eigenvectors which are orthogonal*
- *Its eigenvectors form a basis in* $\mathbb{R}^{n\times n}$
- *There is a unique eigenvalue equal to 1*
- *The eigenvalues are bounded such that* $\lambda_i(\mathbf{W}) \in (-1, 1]$

2. *If* $\mathbf{W} \in \mathbb{C}^{n\times n}$ *and there is at least one element that is not purely real then the following statements hold:*
   - $\mathbf{W}$ *is definitely not Hermitian*
   - *There is a unique eigenvalue* $\lambda_1(\mathbf{W}) = 1$
   - *The eigenvalue equal to 1 has multiplicity 1*
   - *The eigenvalues are bounded such that* $|\lambda_i(\mathbf{W})| \in [0, 1)$, $\forall i \in \{2, 3, \ldots, n\}$
3. *If* $\mathbf{W} \in \mathbb{C}^{n\times n}$ *and it has a full set of eigenvalues then the statements hold:*
   - $\mathbf{W}$ *is definitely not Hermitian*
   - $\mathbf{W}$ *is diagonalisable as* $\mathbf{W} = \mathbf{U}\Lambda\mathbf{U}^*$
   - *It has a full set of eigenvectors which are orthogonal*
   - *There is a unique eigenvalue* $\lambda_1(\mathbf{W}) = 1$
   - *The eigenvalue equal to 1 has multiplicity 1*
   - *The eigenvalues are bounded such that* $|\lambda_i(\mathbf{W})| \in [0, 1)$, $\forall i \in \{2, 3, \ldots, n\}$

The associated discrete time dynamical system $\mathbf{x}(t+1) = \mathbf{W}\mathbf{x}(t)$ *attains average consensus as* $t \to \infty$*. However, maximum machine precision will be achieved, after a finite number of iterations of the consensus algorithm.*

2.3.3.2 *Asymmetric*

In the case that bidirectional communications cannot be guaranteed or that for some reason we want to assign different edge weights for the incoming and outgoing edges, then the resulting weight matrix will not be symmetric. The conditions laid out above have to be modified as follows.

$$\mathbf{1}^T\mathbf{W} = \mathbf{1}^T \tag{2.15}$$

$$\mathbf{W}\mathbf{1} = \mathbf{1} \tag{2.16}$$

$$\rho(\mathbf{W} - \mathbf{1}\mathbf{1}^T/n) < 1 \tag{2.17}$$

The left eigenvalue $\mathbf{1}^T$ guarantees that the mean is preserved and the right eigenvalue $\mathbf{1}$ that the discrete time dynamical system reaches consensus.

What is mentioned in (*Corollary* 2.2) for the case that $\mathbf{W}$ is real (1), holds in this case as well, when the weight matrix $\mathbf{W}$ is *Hermitian*. The contrasting case has to be considered. We examine the cases that $\mathbf{W} \in \mathbb{R}^{n\times n}$ and $\mathbf{W} \in \mathbb{C}^{n\times n}$ are not *Hermitian*. The theory of dynamical systems provides a sound basis for this.

Given an asymmetric weight matrix, complex or not, that has a full set of eigenvalues, then these have to be bounded as $|\lambda_i(\mathbf{W})| \in [0, 1)$, $\forall i \in \{2, 3, \ldots, n\}$ and a unique eigenvalue $\lambda_1(\mathbf{W}) = 1$ must exist. If $\mathbf{W}$ does not have a full set of eigenvalues, then it is sufficient that $\lambda_1(\mathbf{W}) = 1$ is simple and there is no other eigenvalue such that $|\lambda_i(\mathbf{W})| = 1$.

2.3.4 *Stochastic Matrices*

The case that communication is performed over unreliable links can be modelled with a random graph. One only needs to define a probability for each edge in the graph. This type of approach fits the case that communications have different reliability for each direction, as well. In the case of a digraph, one should define a different probability for incoming and outgoing edges.

2.3.4.1 *Symmetric Link Failures*

Assume a weighted undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. In the general case, we assume that each edge $\{v_i, v_j\} \in \mathcal{E}$ is present with probability p_{ij} and that the presence of one edge in the graph is independent of the presence of another edge. We also assume that communication links fail in both directions simultaneously. Consequently, we assign probabilities only for $i > j$, and enforce $w_{ij} = w_{ji}$. Thereafter, a stochastic weight matrix $\mathbb{W}$ can be defined, having as elements the random variables defined below.

$$[\mathbb{W}]_{ij} = \begin{cases} w_{ij} & i < j \quad \text{with probability } p_{ij} \\ 0 & i < j \quad \text{with probability } 1 - p_{ij} \\ w_{ij} = w_{ji} & i > j \\ 1 - \sum_{k=1}^{n} w_{ik} & i = j \end{cases} \tag{2.18}$$

However, such a matrix has row sums smaller or equal to 1. In order to guarantee convergence of the consensus algorithm the weight on the self-loop w_{ii} can be

adjusted to guarantee column and row sums equal to 1. The resulting matrix, included below, satisfies (*eq.*2.11).

$$[\mathbb{W}]_{ij} = \begin{cases} w_{ij} & i < j \quad \text{with probability } p_{ij} \\ 0 & i < j \quad \text{with probability } 1 - p_{ij} \\ w_{ij} = w_{ji} & i > j \\ 1 - \sum_{k=1}^{n} [\mathbb{W}]_{ik} & i = j \end{cases} \tag{2.19}$$

A less worrisome assumption is to consider every edge on the graph having equal presence probability $0 < p < 1$. The diagonal elements of the weight matrix can be adjusted such that the row and column sums remain equal to 1; therefore the matrix remains doubly stochastic. Thus, we may define a stochastic weight matrix $\mathbb{W}$ where the elements are random variables as shown below.

$$[\mathbb{W}]_{ij} = \begin{cases} w_{ij} & i < j \quad \text{with probability } p \\ 0 & i < j \quad \text{with probability } 1 - p \\ w_{ij} = w_{ji} & i > j \\ 1 - \sum_{k=1}^{n} [\mathbb{W}]_{ik} & i = j \end{cases} \tag{2.20}$$

Consequently, the corresponding weighted graph *Laplacian* is $\mathbb{L} = \mathbf{I} - \mathbb{W}$.

The random graph described by the weight matrix (*eq.*2.20) is an appropriate model for a wireless network where communication links fail with the same probability and link failures are bidirectional. We explore how this model affects the convergence of the consensus algorithm (*Algo.*2.1).

Convergence under symmetric link failures

Let us define a stochastic dynamical system by replacing the system defined in (*eq.*2.8) with the following,

$$\mathbf{x}(t+1) = \mathbb{W}(t)\mathbf{x}(t) \tag{2.21}$$

where $\mathbb{W}(t)$ are independently chosen instances of the random matrix. We prove that for almost all sequences $\mathbb{W}(t), t = 1, 2, \ldots$ $\mathbf{x}(t)$ converges to $\frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x}$ as $t \to \infty$.

Theorem 2.7. *Assume a random graph $\tilde{\mathcal{G}}(\mathcal{V}, \tilde{\mathcal{E}})$ associated with a connected undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where if $e_i \in \mathcal{E}$ then $e_i \in \tilde{\mathcal{E}}$ with probability $p \in (0, 1)$. Suppose, $\mathbf{W}$ and $\mathbb{W}$ are respectively the weight matrices corresponding to $\mathcal{G}$ and $\tilde{\mathcal{G}}$ as in (eq.2.20). If $\mathbf{W}$ satisfies conditions in (eq.2.10), (eq.2.11),and (eq.2.12) then the solution of the time dependent dynamical system (eq.2.20) converges for almost all sequences $\mathbb{W}(t), t = 1, 2, \ldots$ $\mathbf{x}(t)$ as $\mathbf{x}(t) \underset{t\to\infty}{\longrightarrow} \frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x}(0)$*

Proof. The subspace span$\{\mathbf{1}\}$ is invariant under all instances of $\mathbb{W}$, and the stochastic weight matrix is symmetric $\mathbb{W}(t) = \mathbb{W}(t)^T$, $\forall t \in \{1, 2, \ldots\}$. Thus, it holds that $(\mathbb{W}(t) - \mathbf{1}\mathbf{1}^T/n)(\prod_{k=t-1}^{0} \mathbb{W}(k) - \mathbf{1}\mathbf{1}^T/n) = \prod_{k=t}^{0} \mathbb{W}(k) - \mathbf{1}\mathbf{1}^T/n$. We are lead to

$$\|\prod_{k=t}^{0} \mathbb{W}(k) - \mathbf{1}\mathbf{1}^T/n)\| = \prod_{k=t}^{0} \|\mathbb{W}(k) - \mathbf{1}\mathbf{1}^T/n\|$$

where $\|\cdot\|$ denotes a matrix norm. However we are interested in the spectral radius. A similar result can be retrieved for the spectral radius by employing Gelfand's formula $\rho(\mathbf{A}) = \lim_{k\to\infty} \|\mathbf{A}^k\|^{1/k}$

$$\begin{aligned}
\prod_{k=t}^{0} \rho(\mathbb{W}(k) - \mathbf{1}\mathbf{1}^T/n) &= \prod_{k=t}^{0} \lim_{m\to\infty} \|(\mathbb{W}(k) - \mathbf{1}\mathbf{1}^T/n)^m\|^{1/m} \\
&= \lim_{m\to\infty} \prod_{k=t}^{0} \|(\mathbb{W}(k) - \mathbf{1}\mathbf{1}^T/n)^m\|^{1/m} \\
&= \lim_{m\to\infty} \left(\prod_{k=t}^{0} \|(\mathbb{W}(k) - \mathbf{1}\mathbf{1}^T/n)^m\| \right)^{1/m} \\
&= \lim_{m\to\infty} \left(\|\prod_{k=t}^{0} (\mathbb{W}(k) - \mathbf{1}\mathbf{1}^T/n)^m\| \right)^{1/m} \\
&= \lim_{m\to\infty} \left(\|(\prod_{k=t}^{0} \mathbb{W}(k) - \mathbf{1}\mathbf{1}^T/n)^m\| \right)^{1/m}
\end{aligned}$$

which leads to the identity.

$$\rho(\prod_{k=t}^{0} \mathbb{W}(k) - \mathbf{1}\mathbf{1}^T/n) = \prod_{k=t}^{0} \rho(\mathbb{W}(k) - \mathbf{1}\mathbf{1}^T/n) \tag{2.22}$$

where $\rho(\cdot)$ denotes the spectral norm.

Since the eigenvalues of the *Laplacian* are non-increasing when an edge is removed, then for all instances of $\mathbb{W}$ the eigenvalues satisfy $1 = \lambda_1(\mathbb{W}) \geqslant \lambda_2(\mathbb{W}) \geqslant \ldots \geqslant \lambda_n(\mathbb{W}) > -1$, where $\lambda_n(\mathbb{W})$ is not smaller than the corresponding eigenvalue $\lambda_n(\mathbf{W})$ of $\mathbf{W}$. Therefore, it holds that

$$\rho(\mathbb{W}(t) - \mathbf{1}\mathbf{1}^T/n) \leqslant 1 \tag{2.23}$$

with equality only when $\lambda_2 = 1$. Such a case is when the corresponding graph $\tilde{\mathcal{G}}(t)$ is not connected. Furthermore, there is only a finite number of possible

instances $\tilde{\mathcal{G}}$ and for a finite number of possible connected graphs. Since each of these connected graphs satisfies (*eq.*2.23), then there is some constant $0 < \gamma < 1$ such that for any connected graph, it holds that

$$\rho(\mathbb{W}(t) - \mathbf{1}\mathbf{1}^T/n) \leqslant \gamma \tag{2.24}$$

One writes for the state of any sequence of $\mathbb{W}(t)$ in (*eq.*2.20).

$$\begin{aligned}
&\|\mathbf{x}(t) - \mu\mathbf{1}\| = \|\mathbf{x}(t) - \mathbf{1}\mathbf{1}^T\mathbf{x}(0)/n\| \\
&= \|\mathbb{W}(t)\mathbb{W}(t-1)\dots\mathbb{W}(0)\mathbf{x}(0) - \mathbf{1}\mathbf{1}^T\mathbf{x}(0)/n\| \\
&\leqslant \rho(\mathbb{W}(t)\mathbb{W}(t-1)\dots\mathbb{W}(0) - \mathbf{1}\mathbf{1}^T/n)\|\mathbf{x}(0)\| \\
&\leqslant \prod_{k=t}^{0} \rho(\mathbb{W}(k) - \mathbf{1}\mathbf{1}^T/n)\|\mathbf{x}(0)\|
\end{aligned} \tag{2.25}$$

If for infinitely many t the graph is connected, there are infinitely many inequalities $\rho(\mathbb{W})(t) - \mathbf{1}\mathbf{1}^T/n) \leqslant \gamma$. The remaining terms satisfying $\rho(\mathbb{W}(t) - \mathbf{1}\mathbf{1}^T/n) = 1$. Hence, according to (*eq.*2.25) this converges $\rho(\mathbf{x}(t) - \mu\mathbf{1}) \to 0$ as $t \to \infty$.

The set of sequences of graphs where only finitely many are connected has probability o. Indeed, in this case, for any sequence there is a time t_0 such that all graphs for $t > t_0$ are disconnected. The probability of a graph to be disconnected is smaller or equal to $(1-p)^{|\mathcal{E}|}$, where $|\mathcal{E}|$ is the number of edges. In fact, the original graph is by assumption connected and its probability is $p^{|\mathcal{E}|}$ as an instance. Therefore the probability that the graphs at time $t_0 + 1, t_0 + 2, \dots, t_0 + m$ are all disconnected is bounded by $(1 - p^{|\mathcal{E}|})^m$. Thus the probability that all graphs chosen after time t_0 are disconnected is o, and this is true for all t_0. □

The theorem holds in the case that the link presence probabilities are different, as well. Then, the argument for infinitely many t that the graph is connected is still true. Just consider that we can build an equivalent problem by selecting a common link presence probability such that $p < p_{ij}$, $\forall i, j$. Therefore, the probability that there are no connected graphs should vanish as $t \to \infty$ and is bounded from above with $(1 - p^{|\mathcal{E}|})^m$. Therefore, the theorem can be shown to hold in that case as well.

2.3.4.2 *Asymmetric Link Failures*

The case of uncertain communications where link failures are directional can be represented with a directed random graph $\tilde{\mathcal{G}}(\mathcal{V}, \tilde{\mathcal{E}})$. Naturally, each edge

$\{v_i, v_j\} \in \mathcal{E}$ can be associated with a coefficient and a link presence probability p_{ij}. This results in asymmetric random weight matrix $\mathbb{W}$.

$$[\mathbb{W}]_{ij} = \begin{cases} w_{ij} & i \neq j \quad \text{with probability } p_{ij} \\ 0 & i \neq j \quad \text{with probability } 1 - p_{ij} \\ 1 - \sum_{k=1}^{n} [\mathbb{W}]_{ik} & i = j \end{cases} \tag{2.26}$$

As with the symmetric case we may select a global probability $p \in (0, 1)$ for each link.

$$[\mathbb{W}]_{ij} = \begin{cases} w_{ij} & i \neq j \quad \text{with probability } p \\ 0 & i \neq j \quad \text{with probability } 1 - p \\ 1 - \sum_{k=1}^{n} [\mathbb{W}]_{ik} & i = j \end{cases} \tag{2.27}$$

(*Theorem* 2.7) may be extended in this case as well. What one has to guarantee, in order to achieve average consensus, is that row and column sums equal to one, as in (*eq.*2.15), (*eq.*2.16) and (*eq.*2.17). Thus, the weight matrices $\mathbb{W}(t)$ shall be invariant with respect to $\mathbf{1}\mathbf{1}^T/n$, $\forall t$. The proof in (*Theorem* 2.7) does not change much, since the spectral radius remains bounded due to (*Theorem* 2.5).

2.3.5 *Weight Assignment*

Until this point, it has not been addressed the manner how the coefficients have to be set to satisfy the conditions for convergence of average consensus. We illustrate below how this can be performed, according to current methods.

2.3.5.1 *Max degree*

The degree of the vertices can be employed to define an heuristic for the weight assignment. The amount of information required on graph topology is global but the full knowledge of graph topology is not required. Particularly, the degrees of all the vertices must be known, i.e. only the diagonal of the adjacency matrix is needed. The weight coefficients are set as the inverse of the maximum degree among all vertices on the graph.

$$\alpha = \frac{1}{\max_{i \in \mathcal{V}}\{d_i\}} \tag{2.28}$$

$$w_{ij} = \alpha \tag{2.29}$$

$$w_{ii} = 1 - d_i \alpha \tag{2.30}$$

The latter can be transformed to the matrix equation below

$$\mathbf{W} = \mathbf{I} - \alpha \mathbf{L}$$

where $\mathbf{L}$ is the non weighted *Laplacian* matrix. The selection of α is dictated from a well known heuristic for bounding the spectral radius of $\mathbf{L}$

$$\lambda_2(\mathbf{L}) \leqslant \max_{\{v_i, v_j\} \in \mathcal{E}} \{d_i + d_j\} \tag{2.31}$$

where d_i and d_j are the degrees of vertices v_i and v_j, respectively, adjacent to edge $\{v_i, v_j\}$.

2.3.5.2 *Metropolis-Hastings*

The relation in (*eq.*2.31) provides another method for assigning the weights on the edges. The weight of an edge is set as the inverse of the maximum degree of two connected vertices. The definition follows.

$$w_{ij} = \frac{1}{\max\{d_j, d_i\}} \tag{2.32}$$

$$w_{ii} = 1 - \sum_{j=1, j \neq i}^{n} w_{ij} \tag{2.33}$$

The information on graph topology required is only second order knowledge of degrees, i.e the vertex degree and the degree of the neighbouring vertices. This method may be applied both globally and distributively. The weights guarantee convergence of the consensus algorithm (*Algo.*2.1) if and only if the graph is not bipartite. The weights are mentioned as local degree weights in (Xiao and Boyd, 2003).

2.3.5.3 *Subgraph weights*

Drawing from the definition of the *MH*-weights, we can generalise as follows.

$$w_{ij} = \frac{1}{\max\{f(\mathcal{G}_i), f(\mathcal{G}_j)\}} \tag{2.34}$$

$$w_{ii} = 1 - \sum_{j \in \beta_i} w_{ij} \tag{2.35}$$

Where in (*eq.*2.34) $\mathcal{G}_i$ is a vertex induced subgraph $\mathcal{G}$ inclusive of all the vertices $v_k \in \mathcal{V}$ such that there is a path from v_k to v_i that is not inclusive of v_j. Similarly $\mathcal{G}_j$ is defined. The function $f(\mathcal{G})$ is a map $f : \mathcal{G} \to \mathbb{N}$. In that sense, it can be a function accounting the edges, the number of paths, or the number of trees in the subgraph. Such functions result in a symmetric weight matrix, and condition of having row sums equal to one has to be enforced. This is achieved by appropriately setting the weights on the diagonal as $w_{ii} = 1 - \sum_{i \neq j} w_{ij}$. The number of all shortest paths in the subgraph $\mathcal{G}_i$ and $\mathcal{G}_j$ can be chosen. Conditions for convergence in (*eq.*2.10), (*eq.*2.11), (*eq.*2.12) can be enforced with:

$$\mathbf{W} \to \mathbf{W}/\rho(\mathbf{W})$$

$$[\mathbf{W}]_{ii} = 1 - \sum_{j=1, j\neq i}^{n} [\mathbf{W}]_{ij}$$

In fact, any abstract function can be selected in (*eq.*2.34) as long as the resulting weight matrix satisfies convergence conditions. An interesting choice is node betweenness centrality , introduced in (Freeman, 1977) and (Anthonisse, 1971) . Also, the connection graph stability measure can be incorporated as well, see (Belykh et al., 2004) and (Khadivi and Hasler, 2010). The definitions are included below for completeness.

Definition 2.28 (Betweenness Centrality)**.** *Let a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and*

$$C_B(u) = \sum_{i \neq u \neq t, t \in \mathcal{V}} \frac{\sigma_{st}(u)}{\sigma_{st}} \tag{2.36}$$

be the betweenness centrality of a vertex where $u \in \mathcal{V}$ is some vertex, $\sigma_{st}(u)$ is the number of shortest paths $\pi\{s, \ldots, u, \ldots, t\}$, i.e. starting from vertex i and ending at vertex j, $\sigma_{st} = \sum_{\{u \neq s \neq t\}} \sigma_{st}(u)$ the total number of shortest paths from vertex s to t.

Definition 2.29 (Connection Graph Stability, CGS)**.** *Let a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and let a collection of paths $\mathcal{C}(\mathcal{G})$ built by arbitrarily selecting one path $p\{v_s, \ldots, v_t\}$ between any two vertices $v_s, v_t \in \mathcal{V}$, let be the k^{th}, for all possible pairs. Given $\mathcal{C}(\mathcal{G})$, the CGS for each edge $e_{st} \in \mathcal{E}$ is defined:*

$$\Theta_k(\mathcal{C}(\mathcal{G})) = \frac{1}{2} \sum_{i=1, j=1}^{n} \phi_{ij}(e_{st}) |p\{i, \ldots, j\}|$$

where

$$\phi_{ij}(e_{st}) = \begin{cases} 1 & e_{st} \in p\{i, \ldots, j\} \\ 0 & \textit{otherwise} \end{cases}$$

Maximum CGS can be used directly to assign the weights since it arguably captures the importance of an edge.

2.3.5.4 *Weights by Convex Optimisation*

Instead of setting the weights based on some property of the graph, like the degree, one can find the appropriate weight coefficients with convex optimisation.

The dynamical system (*eq.*2.8) converges to the average of the initial state as

$$\mathbf{x}_i(t) \underset{t\to\infty}{\longrightarrow} \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i(0),\ \forall\, i \in \{1, 2, \dots n\}$$

with exponential speed of convergence. Since the eigenvalues should satisfy:

$$1 = \lambda_1 > \lambda_2 \geqslant \lambda_3 \dots \geqslant \lambda_n > -1$$

the exponent is time-dependent, and is determined by the initial state and the eigenvalues of $\mathbf{W}$. Therefore the exponential speed of convergence is at least

$$|\log \min\{1-\lambda_2, 1+\lambda_n\}| \tag{2.37}$$

The latter is as well the asymptotic exponential speed of convergence as $t \to \infty$. Under the current weight assignment scheme the purpose is to maximise the asymptotic rate of convergence.

This minimisation problem is convex with respect to the weights and thus can be solved in polynomial time. Extensive analysis of the method is presented in (Xiao et al., 2007), (Boyd and Vandenberghe, 2004).

2.4 GROEBNER BASES THEORY

The methods illustrated in (*Section* 2.3.5) result in symmetric weight matrices. Moreover, the weight assignment scheme presented in (*Section* 2.3.5.4) maximises only the asymptotic speed of convergence. We are going to see in (*Chapter* 3) that by varying timely the weight coefficients with respect to the state $\mathbf{x}$ is advantageous. Furthermore, in (*Chapter* 4), we are going to show that selecting appropriately time varying weight coefficients allows to minimise the duration of the consensus algorithm. The determination of these coefficient yields a set of multivariate polynomial equations. The necessary theoretical background, to understand the difficulties of such problems, is provided within this section. We illustrate very basic notions of Groebner bases theory. A concise study of the subject can be found in (Adams and Loustaunau, 1994).

2.4.1 *Algebraic Definitions*

Definitions of necessary notions are given below. These can be found in many introductory texts to algebra. However, for completeness and ease of reference, we have decided to included them herein.

Definition 2.30 (Field)**.** *A field is a set* F *together with two binary operations on* F*, called addition and multiplication, which are denoted with* $+$ *and* $\cdot$ *respectively and satisfying the following properties, for all* $a, b, c \in F$*:*

1. $a + (b + c) = (a + b) + c$ *(associativity of addition)*
2. $a + b = b + a$ *(commutativity of addition)*
3. $a + 0 = a$ *for some element* $0 \in F$ *(existence of zero element)*
4. $a + (-a) = 0$ *for some element* $-a \in F$ *(existence of additive inverses)*
5. $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ *(associativity of multiplication)*
6. $a \cdot b = b \cdot a$ *(commutativity of multiplication)*
7. $a \cdot 1 = a$ *for some element* $1 \in F$*, with* $1 \neq 0$ *(existence of unity element)*
8. *If* $a \neq 0$*, then* $a \cdot a^{-1} = 1$ *for some element* $a^{-1} \in F$ *(existence of multiplicative inverses)*
9. $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ *(distributive property)*

Equivalently, a field is a commutative ring F *with identity such that:*

- $1 \neq 0$
- *If* $a \in F$*, and* $a \neq 0$*, then there exists* $b \in F$ *with* $a \cdot b = 1$*.*

Definition 2.31 (Ring)**.** *A* ring *is a set* R *together with two binary operations, denoted* $+ : R \times R \longrightarrow R$ *and* $\cdot : R \times R \longrightarrow R$*, such that*

1. $(a + b) + c = a + (b + c)$ *and* $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ *for all* $a, b, c \in R$ *(associative law)*
2. $a + b = b + a$ *for all* $a, b \in R$ *(commutative law)*
3. *There exists an element* $0 \in R$ *such that* $a + 0 = a$ *for all* $a \in R$ *(additive identity)*

4. *For all* $a \in R$, *there exists* $b \in R$ *such that* $a + b = 0$ *(additive inverse)*

5. $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ *and* $(a + b) \cdot c = (a \cdot c) + (b \cdot c)$ *for all* $a, b, c \in R$ *(distributive law)*

Definition 2.32 (Ideal)**.** *Let* R *be a ring. A* left ideal *(resp.,* right ideal*)* I *of* R *is a nonempty subset* $I \subset R$ *such that:*

- $a + b \in I$ *for all* $a, b \in I$
- $r \cdot a \in I$ *(resp.* $a \cdot r \in I$*) for all* $a \in I$ *and* $r \in R$

A two-sided ideal *is a left ideal* I *which is also a right ideal. If* R *is a commutative ring, then these three notions of ideal are equivalent. Usually, the word "ideal" by itself means two-sided ideal.*

Definition 2.33 (Closure)**.** *The* closure $\overline{A}$ *of a subset* A *of a topological space* X *is the intersection of all closed sets containing* A*.*

2.4.2 *Solving Systems Polynomials*

We are interested in determining the solution of a system of nonlinear multivariate polynomials. The related theory aids us in justifying the existence of solutions. Specifically, let a set of polynomials

$$\mathcal{S} = \{f_i | f_i \in k[x_1, x_2, \ldots, x_n], i = 1, 2, \ldots, s\}$$

where $k[x_1, x_2, \ldots, x_n]$ is a so-called *k-vector space* with the set

$$\mathbb{T}^n = \{\boldsymbol{x}^{\mathbf{b}} = x_1^{b_1} x_2^{b_2} \ldots x_n^{b_n} | b_i \in \mathbb{N}, i = 1, 2, \ldots, n\}$$

as a basis. The kernel of $\mathcal{S}$ is its solution set, and defines its ideal $\mathcal{I}$ which is a finite generating set of $\mathcal{S}$. The solution set is where the evaluation of the functions, defined by the polynomials in $\mathcal{I}$, are found to be equal to zero. Simply stated, the solution through substitutions and algebraic operations produces a possibly empty set of polynomials, an ideal. We denote an ideal defined by a set of polynomials $\mathcal{S}$ as $\mathcal{I} =< \mathcal{S} >$ or explicitly $\mathcal{I} =< f_1, f_2, \ldots, f_s >$. Such an ideal is a set consisting of all polynomial linear combinations.

$$< \mathcal{S} >= \{\sum_{i=1}^{s} h_i f_i | f_i \in \mathcal{S}, h_i \in k[x_1, x_2, \ldots, x_n] \forall i \in \{1, 2, \ldots, s\}\}$$

The matter is mathematically immense, and a thorough discussion is avoided. The existence of a finite solution set is guaranteed by the *Hilbert Basis Theorem*. A finite solution set is not be mistaken with the case that the system of equations has a finite number of points that it is equal to zero.

Theorem 2.8 (Hilbert Basis Theorem)**.** *In the ring* $k[x_1, x_2, \ldots, x_n]$ *the following two propositions always hold:*

1. *Given any ideal* $\mathcal{I} \in k[x_1, x_2, \ldots, x_n]$*, then there exists a possibly empty finite set of polynomials* $\{f_1, f_2, \ldots, f_s\}$ *such that* $\mathcal{I} =< f_1, f_2, \ldots, f_s >$ *and the set is empty if and only if* $\mathcal{I}$ *is null.*

2. *Let* $\mathcal{I}_1 \subseteq \mathcal{I}_2 \subseteq \ldots \mathcal{I}_n \subseteq \ldots$ *be an ascending chain of ideals of* $k[x_1, x_2, \ldots, x_n]$*, then there exists a finite* N *such that the chain terminates,* $\mathcal{I}_N = \mathcal{I}_{N+1} = \mathcal{I}_{N+2} = \ldots$.

The theorem tells us that the solution set of the ideal, which is its generating set as well, is finite. Therefore, we can describe algebraically the solution of the system of equations. However, the evaluation of the solutions may not be a finite set of points, instead it can be a set of lines, curves, manifolds, and so on. Therefore, the theorem considers only the existence of such sets, and states that a system of polynomial equations is consistent if that set describes an ideal which is not null.

The process of determining the solution set is similar to Gaussian elimination. The operations of addition, subtraction, multiplication and division are employed. The first three are easy to comprehend. However, in order to define polynomial division, one needs to generalise the notion of division for natural numbers. Particularly, dividing two numbers a with b, implies determining the number of times that b has to be added to itself such that the result is equal to a. Specifically, the quotient q is that number, that we say it is multiplied with b, and the remainder r is added to retrieve a.

This simple thought can be extended to the field of univariate polynomials. Given two polynomials f_1 and f_2 we wish to divide f_1 with f_2. This can be done if we are able to find another couple of polynomials q (quotient) and r (remainder) such that $f_1 = qf_2 + r$. In general, this can be performed by sequentially cancelling out the leading terms of f_1 with terms found in f_2.

Division of multivariate polynomials

Multivariate polynomials are extensions of univariate polynomials. The notions of addition, subtraction, and multiplication are also easy to define and

comprehend. However, division of multivariate polynomials bears some difficulties and has to be treated explicitly. The general idea is the same as in the case of univariate polynomials. Given two polynomials one finally seeks the quotient and the remainder. The process is more or less the same as with univariate polynomials. However, now one has to define a term ordering of the variables since the leading term cannot be uniquely defined. This poses a problem in determining which term can be divided with another. In fact, we cannot tell intuitively which is a larger term, e.g. $x_1^2x_2$ or $x_1x_2^2$. In contrast, in natural number we are aware that $5 > 3$, and in univariate polynomials that $x^6 \geqslant x^3$.

Definition 2.34 (Term order). *Let* $\mathbb{T}^n = \{\mathbf{x}^b | b_i \in \mathbb{N}, i = 1, 2, \ldots, n\}$ *be the set of power products with* $k[x_1, x_2, \ldots, x_n]$ *an associated field. A term order satisfies*

1. *for any two terms* $\mathbf{x}^a, \mathbf{x}^b$ *exactly one relation* $(<, >, =)$ *holds*
2. $1 < \mathbf{x}^b, \forall \mathbf{x}^b \in \mathbb{T}^n$ *and* $\mathbf{x}^b \neq 1$
3. *If* $\mathbf{x}^a < \mathbf{x}^b$ *then* $\mathbf{x}^a\mathbf{x}^s < \mathbf{x}^b\mathbf{x}^s$, $\forall \mathbf{x}^s \in \mathbb{T}^n$

In clarification of nomenclature, the term monomial and power product are frequently used in the literature without distinction. Herein we distinguish among the two in that the power product is a finite multiplicative product of variables raised to a power, e.g. $x_1x_2^7x_3^2$, and the monomial is a finite multiplicative product of variables raised to some power and numbers , e.g. $6x_13x_2^7x_3^2$. The product of these numbers is the coefficient of the monomial. A term can be either a monomial or a power product.

Remark 2.2 (Well-ordering). *Any term order on* $\mathbb{T}^n$ *is a well-ordering. Hence, there is no infinite descending chain* $\mathbf{x}^{a_1} > \mathbf{x}^{a_2} > \mathbf{x}^{a_3} > \ldots$ *in* $\mathbb{T}^n$.

This remark is much important because it guarantees that a term order defines finite chains of power products. This further supports that all term orders are equivalent for determining the existence of solutions. This can be intuitively understood from (*Theorem* 2.8) since it would imply that different term orders lead to different solution sets that describe different ideals. That contradicts the uniqueness of the ideal implied in (*Theorem* 2.8). However, different term orders do not require the same effort for the retrieval of solutions. This is related to the algorithm employed to perform this as well. The algorithms to perform the task, of which we are aware of, are based on the notion of multivariate polynomial division, which is defined below.

Definition 2.35 (Multivariate Polynomial Division)**.** *Suppose a polynomial* f *defined over a ring* $k[x_1, x_2, \dots x_n]$ *where* $x_i \in \mathbb{K}$, $\forall i = 1, 2, \dots, n$ *and* $\mathbb{K}$ *is a field. Additionally, assume a set of polynomials* $F = \{f_1, f_2, \dots f_n\}$ *over* k *as well. Then one divides* f *by* F *to determine polynomials* $r \neq 0$ *and* u_j *over* k *with* $j = 1, 2, \dots t$, $t \in \mathbb{Z}_+$, $t \leqslant n$ *such that* $f = \sum_{j=1}^{t} u_j f_j + r$ *and* r *is not divisible by* $f_i \in F \forall i$.

The process of performing the division is called reduction, it is denoted $f \xrightarrow{F} r$, and we say f is reduced modulo F to r. There is a well known algorithm for performing this kind of division of polynomials but it is not necessary for our purpose and consequently it is omitted. The reader is kindly directed to (Adams and Loustaunau, 1994) to cultivate his interest in the matter.

According to theory, for a given set of polynomial equations $S = \{f_i | i = 1, 2, \dots, s\}$ in $k[x_1, x_2, \dots, x_n]$ there is a set of polynomials $\mathbf{G} = \{g_1, g_2, \dots, g_l\}$ in $k[x_1, x_2, \dots, x_n]$ such that for any i then any set of polynomials in S is reduced to null through $\mathbf{G}$, $f_i \xrightarrow{\mathbf{G}} 0$. We say that $\mathbf{G}$ is a *Groebner basis*. *Groebner bases*

Definition 2.36 (Groebner Basis)**.** *Let* $\mathcal{I}$ *a non-empty ideal of* $k[x_1, x_2, \dots x_n]$. *A Groebner basis* $\mathbf{G} = \{g_1, g_2, \dots, g_l\}$ *is a set of non zero polynomials in* $k[x_1, x_2, \dots x_n]$ *such that any polynomial* $f \in \mathcal{I}$ *is reduced through subsequent divisions with polynomials* g_i $g_i \in \mathbf{G}$ *to null. We write* $f \xrightarrow{\mathbf{G}}_{+} 0$.

Where $\longrightarrow_+$ implies subsequent appropriate divisions of the elements of $\mathcal{I}$ with the elements $\mathbf{G}$ such that the ideal $\mathcal{I}$ is reduced to an empty set.

Sadly, *Groebner bases* are not unique. Even though evaluation of equivalent *Groebner basis* will lead to the same result, they can be troublesome theoretically. Particularly, in the case of determining finiteness and existence of a solution set. Enforcing a few conditions on the polynomials allows to obtain uniqueness.

Definition 2.37 (Minimal Groebner Basis)**.** *A Groebner basis is called minimal if for all* i, $\mathrm{lc}(g_i) = 1$ *and for all* $i \neq j$, $\mathrm{lp}(g_i)$ *does not divide* $\mathrm{lp}(g_j)$.

Where $\mathrm{lp}()$ and $\mathrm{lc}()$ are the leading power product and leading coefficient, respectively.

Minimal Groebner bases are not unique. However, these can be transformed into a unique form which is also minimal to form a reduced Groebner Base.

Definition 2.38 (Reduced Groebner Basis)**.** *A Groebner basis* $\mathbf{G} = \{g_1, g_2, \dots, g_l$ *is called reduced if, for all* i, $\mathrm{lc}(g_i) = 1$ *and* g_i *is reduced with respect to* $\mathbf{G} - \{g_i\}$. *That is, for all* i, *no non-zero term in* g_i *is divisible by any* $\mathrm{lp}(g_i)$ *for any* $j \neq i$.

Simply put, a reduced *Groebner basis* is a minimal **G** such that the polynomials in **G** are not divisible by other polynomials in **G**.

The existence of a *Groebner basis* and additionally of a solution set is supported from the theorem below.

Theorem 2.9 (Weak Hilbert Zero Point)**.** *Let $\mathcal{I}$ be an ideal contained in $k[x_1, x_2, \dots x_n]$. Then the varieties (solution set) is empty $V(\mathcal{I})_k = \emptyset$ if and only if $\mathcal{I} = k[x_1, x_2, \dots x_n]$*

.

The theorem states that an ideal without a solution set is the whole ring. Intuitively thinking, this is so since its kernel is empty. The theorem below makes a connection of the theoretical result in (*Theorem* 2.9) with the properties that the Groebner base of a finite ideal must have. The theorems are given without proof but are known to hold. Please consult (Adams and Loustaunau, 1994).

Theorem 2.10. *The following statements are equivalent.*

1. *The variety V_k is finite*
2. *For each $i = 1, 2, \dots, n$ there exists $j \in \{1, 2, \dots, k\}$ such that for some $g_j \in \mathbf{G}$ the leading power product is univariate, $lp(g_j) = x_i^{\nu}$.*

getting the solutions of a polynomial set of equations

The method for solving a polynomial system of equations can be summarised as a simple four step process. Firstly, select a term order. Secondly, retrieve a *Groebner basis* by employing *Buchberger's Algorithm* or any other appropriate algorithm (Faugère, 1999), (Buchberger, 1998). Thirdly, obtain a minimal *Groebner basis*. Fourthly, obtain the reduced *Groebner basis* from the minimal. This process is guaranteed from *Buchberger's theorem*. The algorithms for the computation of a *Groebner basis* are a generalisation of the *Gaussian elimination* for linear polynomials with obvious representations. An excellent illustration of the matter can be found in (Sturmfels, 2005). Finally, specific solution points can be obtained by solving the simplest equation in the *Groebner basis* and subsequently substituting and solving in equations with higher term orders.

Theorem 2.11 (Buchberger)**.** *Fix a term order. Then every non-zero ideal has a unique reduced Groebner basis with respect to this term order.*

Examination of the terms in the polynomials of the reduced *Groebner basis* allows to determine if the system of polynomial equation has no solution, a finite, or an infinite solution set. Particularly, if the reduced Groebner base is

$\mathbf{G} = \{1\}$ then the system has no solution. In contrast, when the leading power product is univariate, then the solution set is finite, otherwise it is infinite.

Corollary 2.3 (Solution of polynomial systems)**.** *Let a set of polynomials* $\mathcal{S} = \{f_1, f_2, \ldots, f_s\}$*. The solution set of* $< f_i = 0 >$ *for* $i = 1, 2, \ldots, s$ *is:*

1. *void when* $\mathbf{G} = 1$*.*
2. *finite when for each polynomial* $g_i \in \mathbf{G}$ *the leading power product* $\mathrm{lp}(g_i) = x^{\nu}$ *is univariate.*
3. *infinite otherwise.*

where $\mathbf{G}$ *is the reduced Groebner basis of* $\mathcal{S}$*.*

Given that the solution set is null, then by division with $\mathbf{G}$ the set of polynomials $\mathcal{S}$ can be reduced to 0. This implies that the polynomials in $\mathcal{S}$ are linearly independent. Therefore,

$$\sum_{i=1}^{s} h_i f_i = 1 \tag{2.38}$$

for some h_i, $i = 1, 2, \ldots, s$. We are lead to the following remark.

Remark 2.3. *Given a set of polynomials* $\mathcal{S} = \{f_1, f_2, \ldots, f_s\}$*, then the associated system of polynomial equations* $< f_i = 0 >$ *has an empty set of solutions if and only if there does not exist a set of polynomials* $\mathcal{H} = \{h_1, h_2, \ldots, h_s\}$ *such that:*

$$\sum_{i=1}^{s} h_i f_i = 1$$

Summarising, we have illustrated a method for determining the solution set of a system of polynomial equations. *Groebner basis* theory is of central importance for this purpose. In this section, we have given a primer on the subject . However, the matter is immense and complicated. Solutions to a system of polynomials can be obtained by computing the *Groebner basis*, which can be achieved with *Buchberger's algorithm*. The latter has in best case polynomial time complexity, and in worst case exponential, with respect to the number of variables. However, in many cases, we are interested in determining if a system has a null, finite, or infinite set of solutions. Having retrieved a reduced *Groebner basis*, then we need only examine the leading power product. However, due to an immense computational effort required, the method is impractical in many cases. Nevertheless, the knowledge of such a result affirms the existence of a solution, which is of great theoretical importance.

Part II

ALGORITHMS

3

DYNAMIC CONSENSUS

Within this chapter we acknowledge the main issue with the consensus algorithm; its slow rate of convergence. This hinders its applicability. Our main interest within this chapter is the performance of the consensus algorithm under reliable and stochastic communications. Specifically, but not necessarily limited to, we concentrate on the case that topology may be unknown or partially known. We examine the convergence of the algorithm by taking into account the impact of all the eigenvalues of the weight matrix. Our analysis recognises the existence of two distinct phases, the transient and the asymptotic. Based on this observation, we introduce a modification of the update equation of the consensus algorithm, the nonlinear consensus algorithm. The latter allegedly has better performance. Numerical simulations are presented to support this claim. Furthermore, the case of unreliable link communication is of interest. The adaptive consensus algorithm is introduced to compensate for the effects of link and node failures. These two algorithms are improvements of the consensus algorithm, and can be considered mainly in cases that the definitive consensus algorithm (*Chapter* 4) cannot be applied.

3.1 INTRODUCTION

Selecting the weights to maximise the speed of convergence in the asymptotic phase has been examined in (*Section* 2.3.5.4). A similar approach of the transient phase would pose a nonlinear optimisation problem, which is a difficult task, due to its high dimensionality. We circumvent this problem by introducing a modulation of the weights with respect to state differences of adjacent vertices. This enhances the speed of convergence during the transient phase. The nonlinear average consensus algorithm, based on that principle of operation, is presented in (*Section* 3.4).

Furthermore, we consider the case of bidirectional link failures in a communication network. Such failures hinder the convergence of the algorithm, and may shift the final value from the average. However, this effect is ameliorated, if one can implement the algorithm in such a manner that the weights are

adjusted according to (*eq.*2.19). With the purpose of enhancing the speed of convergence, we propose the so-called adaptive consensus algorithm, presented in (*Section* 3.5) which is based on the same principle as the nonlinear average consensus algorithm. This is achieved with adaptation of the weights based on the one-step and two-step time difference of the vertices' states by modulation of the parameters of the nonlinear consensus algorithm.

A theoretical discussion is provided, and a validation of the algorithm's efficacy is made by numerical simulations. Comparisons are performed against the linear consensus algorithm (*Algo.*2.1). Extensive validation results for these algorithms are presented in (*Chapter* 6). Herein, the necessary simulation plots are provided to persuade for the validity of our claims.

3.2 THE PROBLEM

The original version of the average consensus algorithm is slow in larger networks. This has been partially improved by selecting edge weights such that the spectral gap of the associated *Laplacian* matrix is maximised. Consequently, the asymptotic speed of convergence is maximised. The latter is given as

Rate of convergence

$$R = \sup_{\mathbf{x}(0)\neq\frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x}} \lim_{t\to\infty} \left(\frac{\|\mathbf{x}(t) - \frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x}\|^2}{\|\mathbf{x}(0) - \frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x}\|^2} \right)^{1/t} \tag{3.1}$$

which can be determined from the eigenvalues of the weight matrix.

$$R = \max_{i\neq 1} \lambda_i(\mathbf{W})^2 \tag{3.2}$$

However, the aforementioned approach does not maximise the rate of convergence throughout the entire process of the algorithm. We consider the stepwise rate of convergence

$$r(t) = \sup_{\mathbf{x}(t)\neq\frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x}} \frac{(\mathbf{x}(t+1) - \frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x})^2}{(\mathbf{x}(t) - \frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x})^2} \tag{3.3}$$

It holds that $\lim_{t\to\infty} r(t) = R$, (Xiao and Boyd, 2003).

In fact, the asymptotic rate of convergence is the minimum speed that the algorithm may attain throughout the process. Hence, the algorithm can have larger speed of convergence in early stages. Therefore, selecting weights that maximise the asymptotic rate of convergence does not necessarily imply that the

rate of convergence is maximised as well throughout the entire process. Slower rate of convergence shall result in lower precision in the determination of the average at some given iteration.

This effect becomes even more evident as the graph increases in size. There, the state space is of higher dimension and the contribution of each eigenvalue, apart from $\lambda_i = \arg\max_{i \neq 1} \lambda_i(\mathbf{W})^2$, bears importance.

One could perform an optimisation in order to maximise the stepwise speed of convergence along every direction. However, such an optimisation is nonlinear and the space of indeterminants can be highly-dimensional. Alas, a very difficult problem to solve, that may as well be ill-posed. Such a process would become infeasible for prevalent computers as the size of the network increases. Instead a workaround is proposed, that is to modulate the weights along the process in an appropriate manner.

The eigenvalues of the weight matrix, and consequently of the *Laplacian* matrix as well, determine the rate of convergence. Obviously, there are eigenvectors whose contribution to the rate of convergence is higher than others. Ideally, it would suffice to modulate the weights in such a fashion that the eigenvector of the *Laplacian* matrix with the fastest eigenvalue is aligned with the state vector at each time step. Alas, this is a perhaps unsolvable problem. Instead we consider modulating the weights such that when the state vector is far from consensus, then most of the eigenvalues of the *Laplacian* are concentrated near values contributing to high speed of convergence. Even if this results in slower asymptotic speed of convergence, the total contribution of the rest of the eigenvalues is larger than $r(t)$ in early stages. Therefore, we benefit by having a larger rate of convergence early on in the process. Later on, as the state approaches the consensus eigenspace, i.e. the span of $\mathbf{1}$, it suffices to modulate the weights back to the values that maximise the asymptotic rate of convergence.

Experience shows that, in the case of the original consensus algorithm, there might be a tradeoff between maximum initial and asymptotic rate of convergence. However, that research direction has not been followed further within this thesis, and remains open for future investigation. We examine the two phases of the consensus algorithm subsequently.

We start from the fact that the consensus algorithm has two phases. In the first phase, the so called transient phase, all the eigenvectors of the associated dynamical system

$$\mathbf{x}(t+1) = \mathbf{W}\mathbf{x}(t)$$

contribute to the rate of convergence. In the second phase only one component is active, the asymptotic. The asymptotic phase is easier to understand and is subsequently presented.

3.3.1 *Asymptotic Phase*

The dynamical system in (*eq.*2.9), also restated above, representing the operation of the consensus algorithm (*Algo.*2.1) will converge to the average of the initial state as

$$x_i(t) \underset{t\to\infty}{\longrightarrow} \frac{1}{n}\sum_{i=1}^{n} x_i(0),\ \forall\, i \in \{1, 2, \dots n\} \tag{3.4}$$

with exponential speed of convergence if conditions (*eq.*2.10) to (*eq.*2.12) are satisfied. The exponent is time-dependent, and it is determined by the initial state and the eigenvalues of $\mathbf{W}$. The eigenvalues satisfy:

$$1 = \lambda_1(\mathbf{W}) > \lambda_2(\mathbf{W}) \geqslant \lambda_3(\mathbf{W}) \dots \geqslant \lambda_n(\mathbf{W}) > -1 \tag{3.5}$$

Therefore the speed of convergence is at least:

$$R = \max\{1 - \lambda_2(\mathbf{W}), 1 + \lambda_n(\mathbf{W})\} \tag{3.6}$$

This is as well the asymptotic exponential speed of convergence as $t \to \infty$.The eigenspace spanned by the eigenvector corresponding to the eigenvalue λ_m such that

$$m = \arg \max_{i\in\{2,n\}} \{|\lambda_i(\mathbf{W})|\} \tag{3.7}$$

is the slowest eigenspace. We distinguish the eigenspace corresponding to $\lambda_m(\mathbf{W})$ from the rest, and name the corresponding eigenspace $\mathbf{u}_m$.

This implies that the other components are exponentially faster. That is, the projection of the initial state onto the corresponding eigenspaces vanishes much

faster than $\mathbf{u}_m$. The time interval during which the projection of the state onto the eigenvector corresponding to $\lambda_m(\mathbf{W})$ is larger than the rest, it is called asymptotic phase. The latter is dominated by $\mathbf{u}_m$. Theoretically, this extends from some finite time instance to infinity. Though, in machines performing finite precision arithmetic, this extends from some iteration until system precision has been reached. The asymptotic phase is preceded by the so called transient phase. The length of the latter is determined by the iteration that the asymptotic starts. We proceed in its examination in the next section.

3.3.2 *Transient Phase*

The consensus algorithm determines after convergence the average, at each participating machine, of the scalar initially associated with the each machine's state.

$$\mu = \frac{1}{n}\mathbf{x}^T\mathbf{1} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

In many applications, it is assumed that the initial state is sampled from some normally distributed quantity. Hence, the average is the maximum likelihood estimate of the expectation for the elements of the initial state vector. However, in many applications, the estimate does not have to be extremely precise. Thus, the consensus algorithm is stopped long before convergence has maximum precision has been achieved for the specific computer. Moreover, in many applications, execution of the consensus algorithm for a large number of iterations implies the utilisation of an enormous amount of time, energy and other related costs.

The current trend and most prominent approach in selecting the weights is to utilise convex optimisation, as in (*Section* 2.3.5.4). These weights maximise the asymptotic convergence rate, hence the minimum speed of convergence throughout the execution of the algorithm. This is usually attained in later stages of the algorithm, and there are no guarantees about the speed of convergence in earlier stages. The finite time speed is larger or equal to the asymptotic rate. However, for a given graph, it cannot be maximised with the aforementioned method. Thus, it may not be the best choice to maximise the asymptotic convergence rate. This observation motivates us to examine specifically the initial transient dynamics.

In (*fig.*6), we provide an example, where given the same graph the rate of convergence is different for two weight assignments throughout the execution

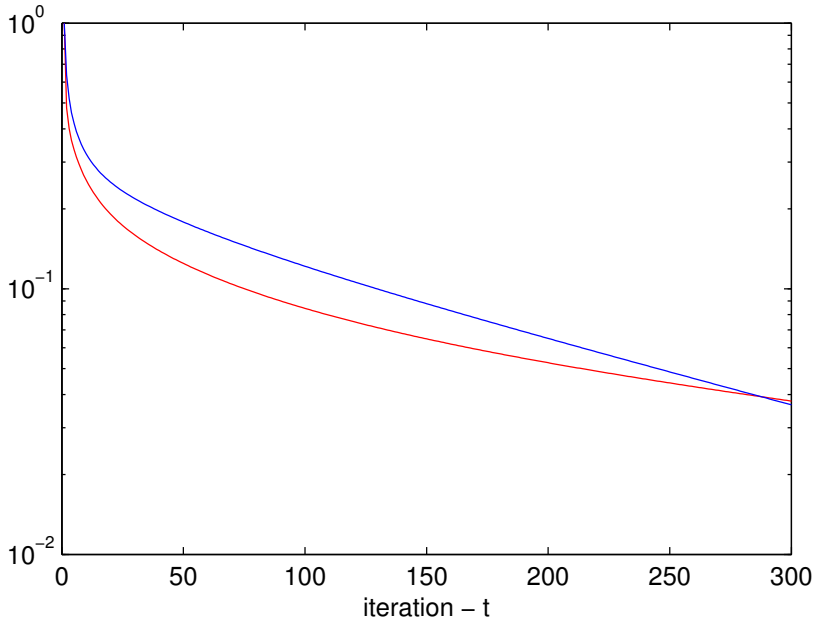


Figure 6: **The Transient Phase**. The plot was generated by executing the consensus algorithm with 1000 different initial states. The states were sampled from a normal distribution with standard deviation 1 and 0 mean. A geometric graph was randomly generated to have $|\mathcal{V}| = 100$ and $|\mathcal{E}| = 350$. The quantity shown is the mean standard deviation over all 1000 different executions of the algorithm. Two different weighting schemes are compared. The blue line illustrates the realisations of the algorithm with the *Metropolis-Hastings* weights. The red indicates the case where weights have been assigned by convex optimisation. The *Metropolis-Hastings* weights have better performance on average early on, until two lines intersect, near the end.

of the consensus algorithm. Selecting different weights than those assigned by maximising the asymptotic rate of convergence, results in having higher speed of convergence and precision, early in the process. The two realisations of the consensus algorithm intercept at a much later stage. We examine the mechanics of this phenomenon hereafter.

We commence our analysis from the simplest case where the eigenvalues of the weight matrix $\mathbf{W}$ are $1 = \lambda_1(\mathbf{W}) > \lambda_2(\mathbf{W}) \geqslant \lambda_3(\mathbf{W}) \ldots \geqslant \lambda_n(\mathbf{W}) > -1$ and $|1 - \lambda_i(\mathbf{W})| > |1 - \lambda_2(\mathbf{W})|$, $\forall i \in \{3, 4 \ldots, n\}$. This covers a larger part of weighting graph assignments and simplifies the discussion. The other case, which might be a bit more difficult to construct, will be considered throughout the text. Initially, the exponential speed of convergence is expected to be much

higher than its asymptotic value. This is due to the fact that the components corresponding to the eigenvalues $1-\lambda_i(\mathbf{W})$, where $i \in \{3,4,\dots,n\}$, diminish rapidly. Therefore, in early stages of the process these eigenvalues contribute and essentially determine the speed of convergence. Their impact vanishes later on. Finally, only the spectral gap $1-\lambda_m(\mathbf{W})$, where m is given by equation (*eq.*3.7), is important as $t \to \infty$. This transient effect is rather evident in larger networks where the number of the eigenvalues is large and many of them are close to λ_n and λ_2. Therefore, in these networks, the collapse of the projections of the state onto the eigenvectors requires more iterations to complete.

Assume that the initial state of each vertex is sampled from a normal distribution $x_i(0) \sim \mathcal{N}(\mu, \sigma^2)$, $\forall i \in \{1,2,\dots n\}$. Therefore the initial state on the graph is sampled from a normal distribution $\mathbf{x} \sim \mathcal{N}(\mu\mathbf{1}, \sigma^2\mathbf{I})$. The weight matrix is a diagonalisable matrix according to:

$$\mathbf{W} = \mathbf{U}\Lambda\mathbf{U}^{-1}$$

Suppose that $\mathbf{W}$ is a symmetric matrix, according to (*Section* 2.10). Thus, its eigenvectors form an orthonormal basis $\mathbf{U}^T\mathbf{U} = \mathbf{I}$. Hence, the state $\mathbf{x}$ can be decomposed as $\mathbf{y} = \mathbf{U}^T\mathbf{x}$. Subsequently, the components follow a normal distribution,

$$\mathbf{y}(0) \sim \mathcal{N}(\mathbf{U}^T\mu\mathbf{1}, \sigma^2\mathbf{I})$$

which leads to,

$$\mathbf{y}(0) \sim \mathcal{N}(\mathbf{u}_n\mu/c, \sigma^2\mathbf{I})$$

where $\mathbf{u}_1 = \{1,0,0\dots,0\}$ and $c = 1/\sqrt{n}$ is a normalisation constant on $\mathbf{U}$. It holds that

$$\|\mathbf{x}(t) - \mathbf{1}\mathbf{1}^T\mathbf{x}(0)/n\|^2 = \|(\mathbf{W} - \mathbf{1}\mathbf{1}^T/n)^t\mathbf{x}(0)\|^2$$

where $\|\cdot\|$ denotes the L_2 norm. Let define the deviation from consensus as the quantity

$$\Phi(t) = \frac{1}{n-1}\|(\mathbf{W} - \mathbf{1}\mathbf{1}^T/n)^t\mathbf{x}(0)\|^2 \tag{3.8}$$

which is in fact the mean square deviation from consensus of the dynamical system. Since $\mathbf{y}(t+1) = \Lambda\mathbf{y}(t)$, then one can readily show that the deviation

from consensus of the dynamical system in (*eq.*2.8), given some initial state $\mathbf{x}(0)$, is given by

$$\Phi(t) = \frac{1}{n-1}\mathbf{y}(0)^T\tilde{\Lambda}^{2t}\mathbf{y}(0) \tag{3.9}$$

where $\mathbf{y}(0) = \mathbf{U}^T\mathbf{x}(0)$ and $\tilde{\Lambda} = diag\{0, \lambda_2, \lambda_3, \ldots, \lambda_n\}$. Thereafter, its expectation, given a normal random state vector at time-step (t), may be obtained by

$$E[\Phi(t)] = \frac{1}{n-1}Tr[\tilde{\Lambda}^{2t}]E[\mathbf{y}(0)^T\mathbf{y}(0)]$$

which leads to

$$E[\Phi(t)] = \frac{1}{n-1}\sum_{i=2}^{n}\lambda_i^{2t}\sigma^2 \tag{3.10}$$

Examining the latter equation, in early stages the contribution of the components related to fast eigenvalues is larger than those related to slow eigenvalues. Given that the weight matrix $\mathbf{W}$ is known, there can be a critical time that separates the transient from the asymptotic phase. This can be recovered from,

$$\sum_{i\in\alpha}\lambda_i(\mathbf{W})^{2t} = (n - |\alpha| - 1)\lambda_m(\mathbf{W})^{2t} \tag{3.11}$$

where α is the set of indices for some small ϵ such that $\alpha = \{\, i \in \{2, 3, \ldots, n\} : |\lambda_i| < |\lambda_m(\mathbf{W})| - \epsilon \,\}$, where we remind that $\lambda_m(\mathbf{W})$ refers to the slowest eigenvalue. The purpose of ϵ is to allow to include in the summation those eigenvalues that are relatively close in modulus to $\lambda_m(\mathbf{W})$. These in practice contribute to the asymptotic phase due to the errors introduced by finite precision arithmetic.

These claims can be readily verified by examination of (*fig.*7). There, we have plotted the contribution of the slow components in the expected deviation from consensus $E[\Phi_m(t)]$ versus the fast components $E[\Phi_\alpha(t)]$.

$$E[\Phi_m(t)] = \frac{n - |\alpha| - 1}{n-1}\lambda_m(\mathbf{W})^{2t} \tag{3.12}$$

$$E[\Phi_\alpha(t)] = \frac{1}{n-1}\sum_{i\in\alpha}\lambda_i(\mathbf{W})^{2t} \tag{3.13}$$

The critical time, of the transition between the transient and the asymptotic phase, is evident. In (*fig.*7), even though the presence of both phases is identifiable, the transient phase lasts only a few iterations, and its impact is small. An example where the transient phase is larger is given in the subsequent figure.

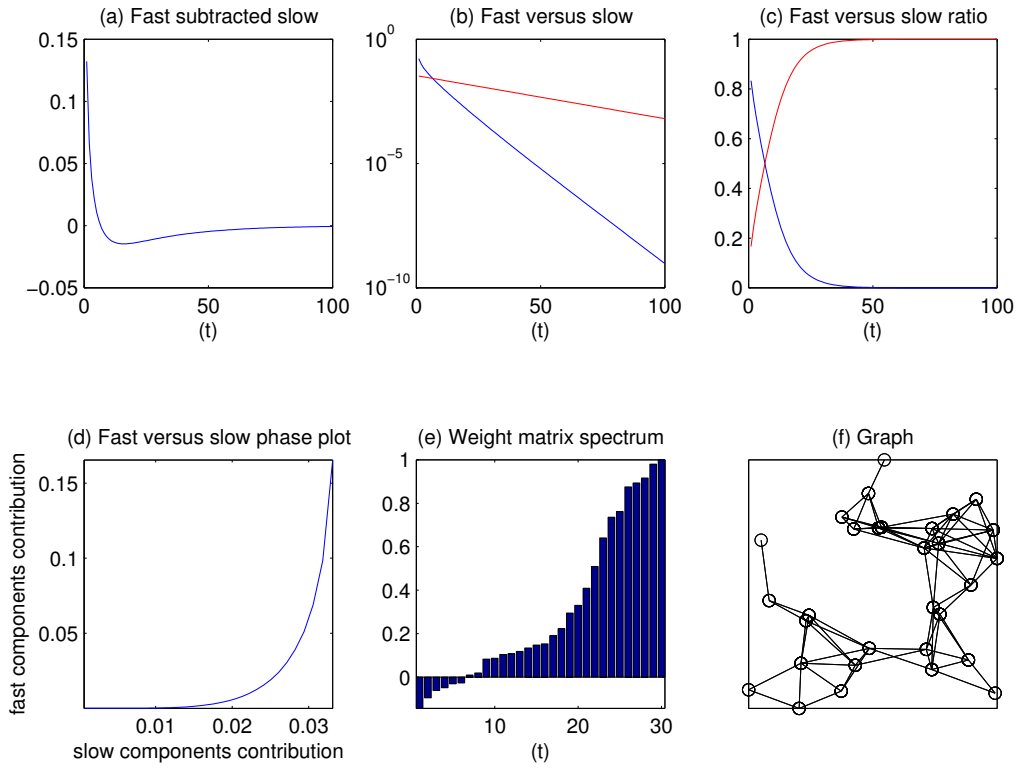


Figure 7: **Contribution comparison of slow and fast components**. The contributions to the expected mean deviation from consensus are compared with $\epsilon = 0.1$. Evidence of the transient phase is given for the depicted weighted graph. The weights have been set with respect to *Metropolis-Hastings* weighting scheme, (*Section* 2.3.5.2). **(a)** The contributions of the fast minus the slow components. The negative values indicate that after $t = 10$ the slow component dominates convergence. **(b)** The Red line is the contribution of the slow component. The Blue line is the contribution of the fast component. The two intersect at about $t = 10$. **(c)** The ratio of contributions of slow and fasts components with respect to the total is shown, $\frac{E[\Phi_m(t)]}{E[\Phi(t)]}$ and $\frac{E[\Phi_\alpha(t)]}{E[\Phi(t)]}$, red is slow and blue is fast, respectively. **(d)** The contribution of the fast plotted against the slow components' contribution. The curve is convex, illustrating the impact of the fast components. The larger the curvature the larger the impact of the transient phase. **(e)** The spectrum of the weight matrix. **(f)** The graph.

There the same graph has been weighted with a different scheme. In (*fig.*7), the weights have been set according to the *Metropolis-Hastings* weight scheme (*Section* 2.3.5.2), whereas the weights in (*fig.*8) have been selected in accordance with the maximum degree method, (*Section* 2.3.5.1).

figures' explanation

All figures follow the same layout. The top three graphs compare the contribution (*eq.*3.10) of the components corresponding to the fast and slow eigenvalues as in, (*eq.*3.13) and (*eq.*3.12), respectively. There, going from left to right, the first plot (a) is the difference $E[\Phi_\alpha(t)] - E[\Phi_m(t)]$. Negative values indicate that the slow components are expected to contribute more in the deviation from consensus. In contrast, positive values indicate that fast components contribute more. The *x-axis* indicates the time-step (t) of the algorithm.

The second plot (b) illustrates the contributions of the fast $E[\Phi(t)]_\alpha$ and the slow $E[\Phi(t)]_m$ components with red and blue lines, respectively. The point where the two lines intersect designates the end of the transient phase. Nevertheless, the fast components can still affect the speed of convergence during the asymptotic phase, as in (*fig.*8). However, their contribution is much smaller and diminishes rapidly.

The third plot (c) at each graph illustrates the partition of contribution of each of the fast and slow components in respect to the total. In fact the quantities plotted are $E[\Phi_\alpha(t)]/E(\Phi(t)]$, blue line, and $E[\Phi_m(t)]/E(\Phi(t)]$, red line.

At the bottom part of (*fig.*7), (*fig.*8) and (*fig.*9), the phase plot (d) of fast versus slow illustrates the impact of the transient phase. The curvature indicates that the transient phase dominates. Specifically, the larger the curvature, the larger is the impact of the slow components, and consequently of the transient phase. This is evident by comparing plot (d) in (*fig.*9) and (*fig.*8).

The next two plots (e) and (f) are the spectrum of the weight matrix and the graph itself. The same graph was utilised among all three figures. It is important to observe, by comparing again (*fig.*8) with (*fig.*9), that with a spectrum that spans evenly the entire range $(-1, 1]$, the transient phase is suppressed and the asymptotic phase is more important.

observations on components' contribution

Overall, in these three figures (*fig.*7), (*fig.*8), and (*fig.*9) the impact of the so called fast and slow components is illustrated, with respect to different weighting schemes for the same graph. The fast and slow components are primarily involved in the emergence of the transient and asymptotic phases, respectively. The two phases have a conjoined relation. The fast components may be suppressed when the weights are selected in such a manner that the minimum speed of convergence is maximised, that is the slow components

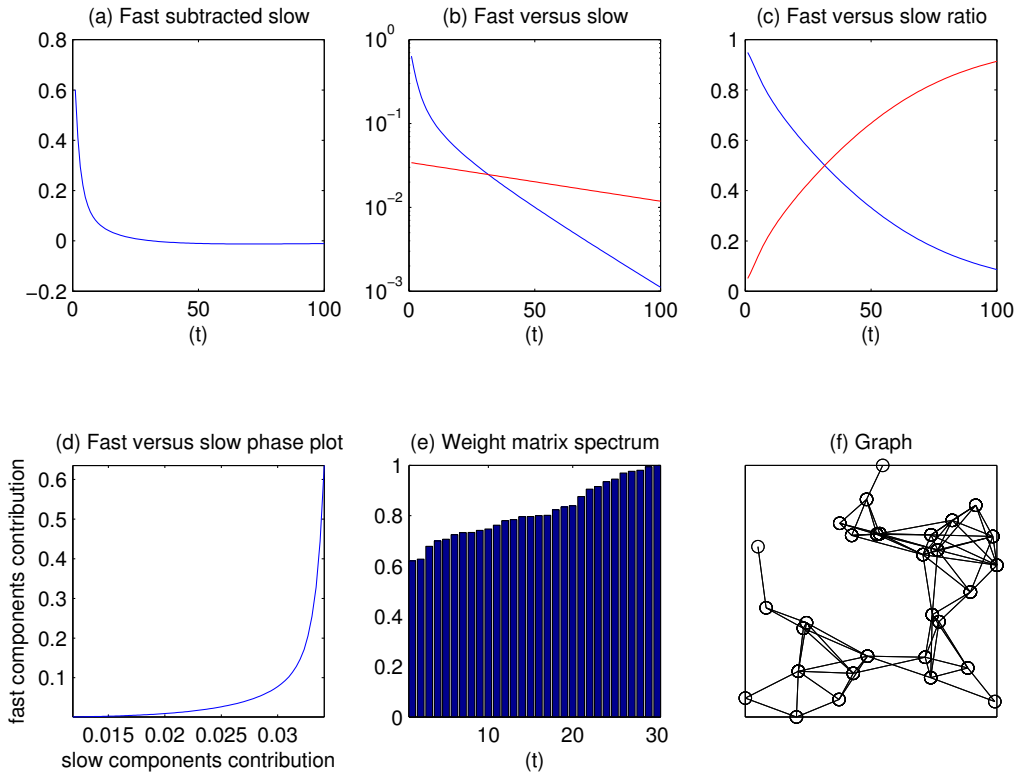


Figure 8: **Prominent transient phase**. Evidence of the transient phase is given for the depicted weighted graph with $\epsilon = 0.1$. The weights have been set with respect to *Maximum degree* weighting scheme, (*Section* 2.3.5.1). **(a)** The contributions of the fast subtracted the slow components. After about $t = 30$ the value becomes slightly negative, and the slow component dominates convergence. **(b)** The Red line is the slow components and blue is fast. **(c)** The ratio of each component with respect to the total is depicted, using the same color code. **(d)** The ratio of contribution of the fast versus the slow components. The larger the curvature the larger the impact of the transient phase. **(e)** The spectrum of the weight matrix. **(f)** The graph.

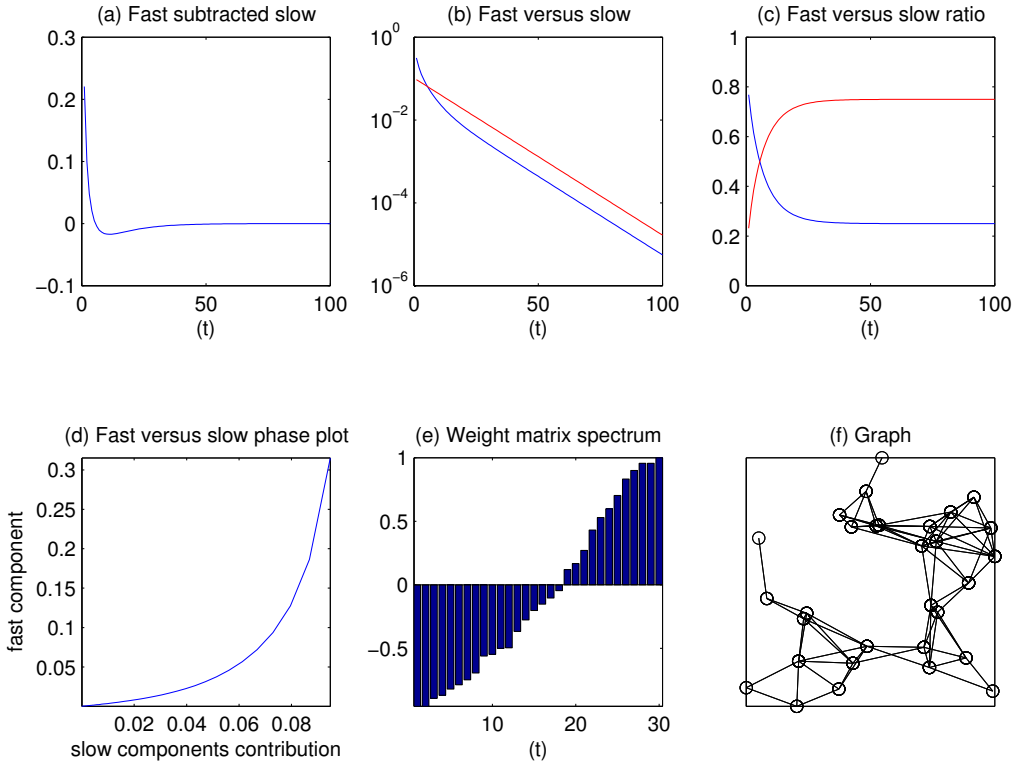


Figure 9: **Impact of convex optimisation onto transient phase**. Convex optimisation enhances slow components. Evidence of the transient phase is given for the depicted weighted graph **(f)**, having $\epsilon = 0.1$. The weights have been set by *convex optimisation*, (*Section* 2.3.5.4). **(a)** The contributions of the fast subtracted the slow components are shown. After about $t = 5$ slow components dominates convergence. The **(b)** red line designates contribution of the slow components and blue of the fast. **(c)** The ratio of the two components, slow and fast, with respect to the total is depicted. Likewise, the red depicts the slow and blue the fast. **(d)** The ratio of contribution of the fast versus the slow components is shown. The larger the curvature, the larger is the impact of the transient phase. Here the transient phase is suppressed. **(e)** The spectrum of the weight matrix. **(f)** The graph.

become as fast as possible given a specific graph. However, one can assign the weights in such a fashion that fast components become faster, as in (*fig.*6). In that manner, the iterations needed to achieve some level of precision can be less than in the case where the asymptotic phase has been favoured. In many cases, the required precision by the application may be such that it can be attained in the transient phase, given that the weights have been set appropriately.

The transient phase is more important in larger networks. There, convergence requires more iterations, and the transient phase has larger duration. Additionally, the impact of the total energy and other costs, related to the number of communications, is much larger. This is due the fact that the number of communications per iteration, with respect to the number of vertices, increases at least with $2n$ for trees and with $\binom{n}{2}p$ in random graphs, where p is the edge presence probability. In such cases, it is of interest to achieve maximum precision as fast as possible. An example has been given already in (*fig.*14). Assume that an application requires precision to the first decimal digit, then the algorithm will need about 100 iterations less, in this case, if we select the *Metropolis-Hastings* (*Section* 2.3.5.2) weights, in comparison to having had set the weights by *Convex optimisation* (*Section* 2.3.5.4). This might be the scenario in many applications. Therefore, we would like to favour having better precision as soon as possible. That requires considering the stepwise speed of convergnece instead of the asymptotic.

However, as already noted before (*Section* 3.2), this is a difficult nonlinear optimisation problem. Instead, we propose the nonlinear consensus algorithm (*Section* 3.4), which aims at achieving a similar result online. The algorithm achieves better performance than the linear consensus algorithm on average throughout the entire process.

Summarising, in this section two principal ideas are communicated. Firstly, the weights assigned on the graph shall result in different speed of convergence throughout the process. The speed of convergence has a minimum value which can be maximised by appropriately assigning the weights. This however does not guarantee that the stepwise speed of convergence is maximum for the given graph, throughout the process. Secondly, there are two distinct phases of the algorithm, the transient and the asymptotic. These are present in any graph and given any initial state vector, except those where the initial state vector is proportional to the eigenvector with eigenvalue equal to $|\lambda_m(\mathbf{W})|$, i.e. the slow component. Therefore, the selection of the weights can be such that the maximum precision is attained in the least number of steps, or such that a

specific precision is reached in the minimum number of iterations. The first case can be posed as an easy convex optimisation problem. The second is a non-linear optimisation problem which is difficult to solve. An alternative approach is needed which is presented subsequently.

3.4 NONLINEAR CONSENSUS

Our intention is to have high exponential speed in both the transient and the asymptotic phase by using a nonlinear function. The trick is to leave the system with weights at the asymptotic phase that maximise the minimum rate of convergence given the graph while modulating them appropriately during the transient phase.

The proposed local update rule can be summarised in

$$x_i(t+1) = x_i(t) + \sum_j w_{ij} f(v_{ij}(t)) \tag{3.14}$$

Nonlinear update protocol

where $v_{ij} = x_j - x_i$ and the nonlinear function in the summation can be any C^1 function which has

$$f(0) = 0,\ f(-u) = -f(u),\ \frac{df}{du} > 0$$

This family of functions for consensus has been mentioned in (Saber and Murray, 2003). However, in contrast to this work, their study is focused on the continuous time case, and does not acknowledge the advantages of this function in relation to the transient phase, thus overlooking its importance. Moreover, we provide a theorem that suffices for the convergence of this discrete time non-linear dynamical system (Georgopoulos and Hasler, 2009a).

Theorem 3.1. *Suppose that* $\mathbf{W} \in \mathbb{R}^{n\times n}$ *is a doubly stochastic matrix. Let* f *be an odd, increasing scalar function* $f : \mathbb{R} \to \mathbb{R}$*, with a bounded first order derivative* $0 < \frac{df}{du} \leqslant 1$*. Assume* $\rho(\mathbf{W} - \mathbf{1}\mathbf{1}^T/n) < 1$*, then the evolution of the discrete dynamical system* $\mathbf{x}(t+1) = \mathcal{A}(\mathbf{x}(t))\mathbf{x}(t)$ *converges according to:*

$$x_i(t) \underset{t\to\infty}{\longrightarrow} \frac{1}{n}\sum_{i=1}^{n} x_i(0),\ \forall i \in \{1, 2, \dots n\}$$

Proof. Equation (*eq.*3.14) can be written

$$x_i(t+1) = x_i(t) + \sum_{j\in\beta_i} w_{ij}\frac{f(v_{ij}(t))}{v_{ij}(t)} v_{ij}(t)$$

as long as $x_j(t) \neq x_i(t)$.

Define the matrix $[\mathcal{A}]_{ij} = w_{ij}\frac{f(v_{ij})}{v_{ij}}[\mathbf{A}]_{ij}$, $\forall i \neq j$ and $[\mathcal{A}]_{ii} = 1 - \sum_{j=1,j\neq i}^{n}[\mathcal{A}]_{ij}$, where $\mathbf{A}$ is the adjacency matrix. When $v_{ij} = 0$ the matrix element is defined as $[\mathcal{A}]_{ij} = \lim_{v_{ij}\to 0} w_{ij}\frac{f(v_{ij})}{v_{ij}}$.

The matrix $\mathcal{A}$ is a function of $f(v_{ij})$. Since v_{ij} is a function of $\mathbf{x}$, then $\mathcal{A}$ is a function of $\mathbf{x}$ as well. Subsequently, we are led to the following global update equation.

$$\mathbf{x}(t+1) = \mathcal{A}(\mathbf{x}(t))\mathbf{x}(t) \tag{3.15}$$

The matrices $\mathcal{A}$ and $\mathbf{W}$ can be written $\mathcal{A}(\mathbf{x}) = \mathbf{I} - \mathcal{L}(\mathbf{x})$ and $\mathbf{W} = \mathbf{I} - \mathbf{L}$. Where $\mathbf{L}$ is the weighted graph *Laplacian*, Godsil and Royle (2001), and $\mathcal{L}(\mathbf{x})$ the corresponding weighted graph *Laplacian* of the nonlinear system. The time index is omitted for simplicity where it is not necessary.

Let μ_k and λ_k be the eigenvalues of $\mathbf{L}(\mathbf{x})$ and $\mathbf{L}$, respectively. Therefore the eigenvalues of the two matrices, $\mathcal{A}$ and $\mathbf{A}$, are $1-\mu_k$ and $1-\lambda_k$, respectively. Due to the fact that these matrices are symmetric, their eigenvalues are ordered as $\lambda_1 \leqslant \lambda_2 \leqslant, \ldots, \lambda_n$ and $\mu_1 \leqslant \mu_2 \leqslant, \ldots \mu_n$. One can see that $\mathcal{A}$ is symmetric by considering:

$$\begin{aligned}
[\mathcal{A}]_{ij} &= w_{ij}\frac{f(v_{ij})}{v_{ij}}[\mathbf{A}]_{ij} = w_{ji}\frac{f(v_{ij})}{v_{ij}}[\mathbf{A}]_{ji} \\
&= w_{ji}\frac{f(-v_{ji})}{-v_{ji}}[\mathbf{A}]_{ji} = w_{ji}\frac{f(v_{ji})}{v_{ji}}[\mathbf{A}]_{ji} \\
&= [\mathcal{A}]_{ji}
\end{aligned}$$

The eigenvalues of λ_k and μ_k are increasing functions of $[\mathbf{W}]_{ij}$, $[\mathcal{A}]_{ij}$ respectively. This follows from

$$\mathbf{x}^T\mathbf{L}\mathbf{x} = 2\sum_{i=1}^{n}\sum_{j=1}^{n}[\mathbf{W}]_{ij}(x_j - x_i)^2$$

Hence $\mathbf{x}^T\mathbf{L}\mathbf{x}$ is a positive increasing function of w_{ij}. Therefore the *Courant-Fischer* theorem holds for $\mathcal{L}(\mathbf{x})$. This extends to $\mathcal{A}$, therefore we need only the conditions on $\mathbf{W}$ to hold, (*eq.*2.10), (*eq.*2.11), and (*eq.*2.12). The first has already been shown. The second is trivial since by definition the diagonal elements $[\mathcal{A}]_{ii}$ are such that $\mathcal{A}_{ii} = 1 - \sum_{j=1,j\neq i}^{n}[\mathcal{A}]_{ij}$. The last condition $\rho(\mathcal{A}(\mathbf{x}) - \frac{\mathbf{1}\mathbf{1}^T}{n}) < 1$, also holds. Indeed, $mu_n \leqslant \lambda_n$, therefore $1-\mu_n \geqslant 1-\lambda_n$ and $1-\mu_2 < 1$ because the graph remains connected. □

A function that satisfies the conditions of theorem (*Theorem* 3.1), and modulates the weights appropriately is

$$f(v) = \tanh(\theta v)\kappa \tag{3.16}$$

where parameters $\{\theta, \kappa\} \in \mathbb{R}_+$ are assigned such that

$$\kappa\theta \leqslant 1 \tag{3.17}$$

We are going to verify in (*Chapter* 6) by simulation that this function performs better than the linear consensus algorithm. We provide (*fig.*10) within this section as evidence of our claims. There we have simulated the nonlinear consensus algorithm and the consensus algorithm for 1000 different initial states, all sampled from the same initial distribution with mean 10 and standard deviation 1, i.e. $\mathbf{x} \sim \mathcal{N}(10\mathbf{1}, \mathbf{I})$. The standard deviation of the state is an appropriate measure for the level of agreement in the network. The average of the state's standard deviation from the arithmetic mean for each realisation, over all the 100 executions, is shown in (*fig.*6). The weights assigned on the edges were obtained with convex optimisation, as in (*Section* 2.3.5.4), and the algorithm parameters where $\kappa = 1$ and $\theta = 1$.

By examination of the plots in (*fig.*10) two conclusions may be deduced. First, that the nonlinear consensus algorithm outperforms the consensus algorithm throughout the entire process. This is due to the modulation of the weights with respect to the state, which results in increased speed of convergence during the transient phase. Subsequently, in the asymptotic phase, the speed of convergence is maximised, since the modulated weights converge to the preassigned weights, assuming that it holds $\frac{df}{dx}(0) = 1$ for the given function. In this case, these weights are such that the speed of convergence is optimal during the asymptotic phase for the given graph.

One arrives at the second conclusion by comparison of the performance plots of the two example graphs (*fig.*10). The left graph consists of 30 vertices, whereas the graph of the right consists of 100 vertices. Our claim that the impact of the fast components is more evident in larger graphs is verified by comparing the slopes. The difference in performance of the larger graph between the execution of the nonlinear and the original consensus algorithm is roughly 10 times the corresponding performance difference for the smaller graph.

Evidence of better performance

We have also executed the following experiment to examine the performance of the algorithm in small and large graphs. We have generated 10-tuples of random geometric graphs with $[10, 20, \ldots, 400]$ vertices and average vertex degree of 1.3 .

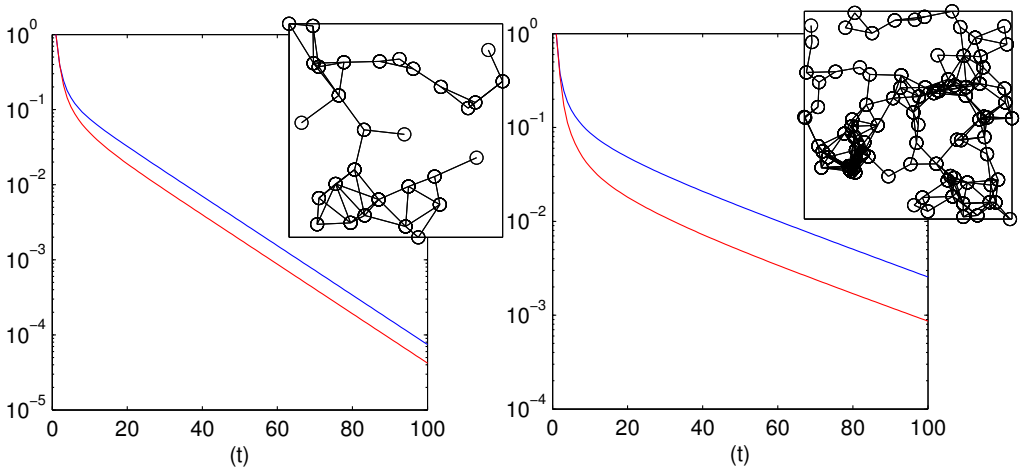


Figure 10: **Average performance of the Nonlinear algorithm versus the Linear**. The average standard deviation of the state is plotted for 1000 different initial states sampled from a normal distribution with standard deviation 1 and mean value 10. Red line indicates nonlinear average consensus algorithm. Blue indicates the consensus algorithm. **Left** plot is a graph of 30 vertices and about 50 edges. **Right** plot is a graph of 100 vertices and about 170 edges. The nonlinear algorithm performs better on average by taking advantage of both the transient and the asymptotic phase. The nonlinear consensus algorithm has been executed with parameters $\kappa = \theta = 1$.

Thereafter, we simulated the nonlinear (*eq.*3.14) and linear protocols (*eq.*2.8) of the consensus algorithm (*Algo.*2.1) for each graph, for 100 different initial states sampled from a normal distribution $\mathbf{x} \sim \mathcal{N}(10\mathbf{1}, \mathbf{I})$. For each 10-tuple we have computed the average of the standard deviation at each time-step of the process. The nonlinear consensus algorithm parameters had been set to $\kappa = \theta = 1$. The resulting data-set was fitted with a cubic smoothing spline which is depicted in (*fig.*11). The smoothness parameter was 10^{-5}.

In (*fig.*11) two concepts are illustrated. First, the benefit in precision by incorporating the nonlinear consensus algorithm in random geometric graphs. Justifiably, the importance of profiting from the transient phase by performing weight modulation is evident. Second, the improved performance in comparison to the linear consensus algorithm in larger graphs. Therefore, we claim that the transient phase is more important as the number of vertices increases in random

geometric graphs. This work has been presented in (Georgopoulos and Hasler, 2009a).

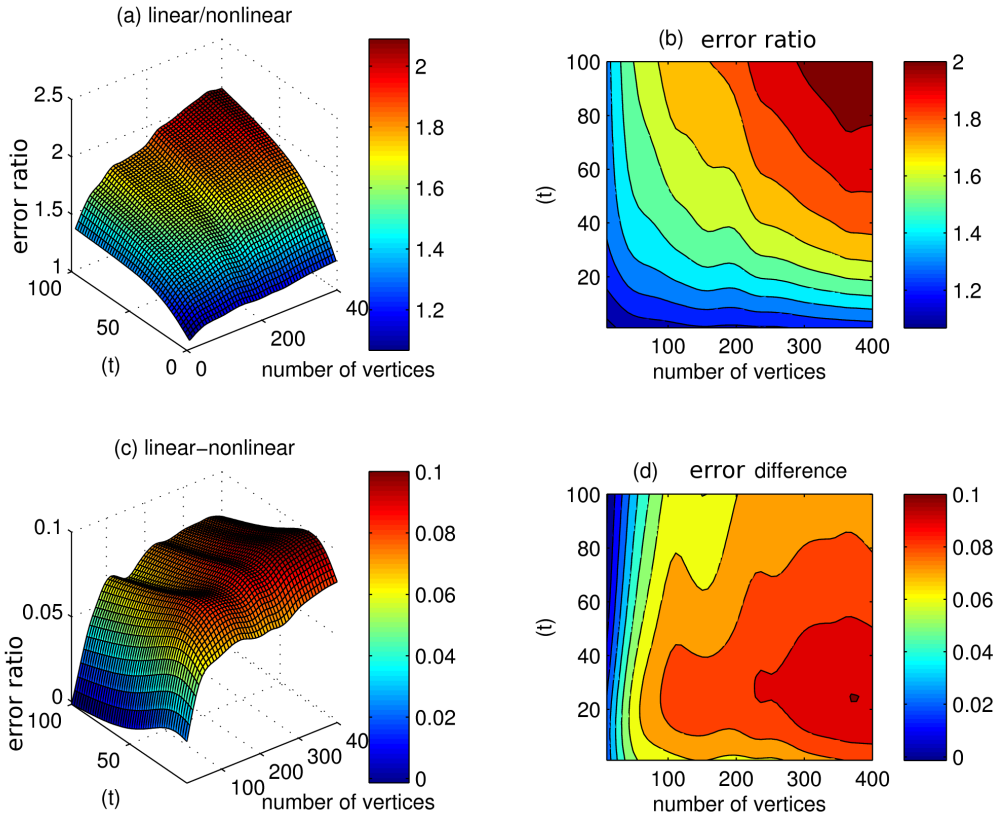


Figure 11: **Comparison Nonlinear versus Linear**. **(a)** The ratio of the standard deviation of the state of the linear consensus algorithm over the linear is depicted. The linear algorithm has up to twice the standard deviation when compared to the nonlinear algorithm.. **(b)** The contour graph of (a). Better performance in larger graphs is evident for the nonlinear consensus algorithm. **(c)** The difference of the standard deviation of the state of the linear consensus algorithm from the nonlinear case is illustrated. **(d)** Similarly the contour graph of (c). The nonlinear consensus algorithm parameters had been $\kappa = \theta = 1$ and ϵ_i was adapted with a typical momentum like policy. The surfaces displayed have been fitted a cubic smoothing spline. The smoothness parameter was 10^{-5}.

Communications cannot be guaranteed in many applications of the consensus algorithm. We discuss this case here, for the purpose of understanding how the consensus algorithm is affected by edge variability. Based on our observations we introduce the adaptive consensus algorithm, which may perform better than the nonlinear and the linear consensus algorithms in case of symmetric link failures.

In such cases a model of stochastic link failures (*eq.*2.19) is appropriate, as presented in (*Section* 2.3.4.1). The convergence of such a stochastic dynamical system is supported by (*Theorem* 2.7). The theorem can also be extended to the case of asymmetric link failures. However, the constraints on the second largest eigenvalue should be guaranteed at each step of the algorithm.

In view of the findings in (*Section* 3.3.2) and (*Section* 3.4), there are two principal matters related to the execution of the consensus algorithm under stochastic communications. First, the impact of stochastic communications on the transient and asymptotic phases. Second, the manner that the nonlinear consensus algorithm be employed to benefit from the transient effect, given the fact that the transient phase bears great importance for the level of precision attained throughout the entire process. These matters are examined within this section.

3.5.1 *Impact of Edge Stochasticity*

This discussion focuses on the case of symmetric link failures. We build upon two facts about the dynamical system in (*eq.*2.8). First, that the asymptotic convergence rate is R. Second, that the rate of convergence during the transient phase is determined by the ensemble of eigenvalues of the weight matrix. The smaller the modulus of these, the faster the convergence rate during the transient. This can be verified by inspecting (*eq.*3.10).

The following claims are made for the stochastic dynamical system case (*Section* 2.3.4.1). First, that the expected rate of convergence is smaller than the corresponding non-stochastic graph with the same weight assignments, in both the asymptotic and the transient phase. Second, and most importantly that the transient phase appears to last longer but having smaller contribution in the reduction of the deviation from consensus. Finally, we claim that this effect is more pronounced for smaller edge alive probabilities p.

The stochastic dynamical system evolves in respect to the model

$$\mathbf{x}(t+1) = \prod_{k=t}^{0} \mathbb{W}(k)\mathbf{x}(0) \tag{3.18}$$

where $\mathbb{W}$ is a realisation of the stochastic weight matrix under (*eq.*2.19) at the k^{th} step of the algorithm's execution. There we assume of a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with self-loops, and suppose that the edges of $\mathcal{G}$ are present with some probability $p \in (0, 1)$ at each iteration. The self-loops are assumed to be always present, i.e. $p = 1$. Finally, the realisations of the stochastic weight matrix are in fact subgraphs of $\mathcal{G}$.

One should notice that $\mathbf{x}(t+1)$ is dependent on the previous state $\mathbf{x}(t)$, and therefore on the sequence of weight matrices as well. In fact, we are interested on determining the expected rate of convergence at a time t, given some initial state $\mathbf{x}(0)$. However, this is a difficult task, because the numerator and the denominator are dependent. Nevertheless, our claims can be supported by postulating on the expected state.

$$\mathbb{E}[\mathbf{x}(t+1)|\mathbf{x}(0)] = (\mathbf{I} - p\mathbf{L})^{t+1}\mathbf{x}(0) \tag{3.19}$$

The derivation may be found in (*Appendix* A). By inspection of (*eq.*3.19), we are led to the conclusion that the edge probability's effect is to scale the *Laplacian*.

The eigenvalues of the expected weight matrix are $\lambda(\mathbb{E}[\mathbb{W}]) = 1 - p\lambda(\mathbf{L})$. Therefore, the effect of p onto the eigenvalues of the expected weight matrix is such that for $\lambda_i(\mathbf{L}) \in \left[0, \frac{2}{p+1}\right)$ the eigenvalues increase, i.e. $|1 - \lambda_i(\mathbf{L})| > |1 - p\lambda_i(\mathbf{L})|$. On the other hand for $\lambda_i(\mathbf{L}) \in \left(\frac{2}{p+1}, 2\right)$, the eigenvalues decrease. Hence, how the rate of convergence changes depends on p in a complex manner.

$$\begin{aligned}
|1 - \lambda_i(\mathbf{L})| &< |1 - p\lambda_i(\mathbf{L})| && \text{, if } \lambda_i(\mathbf{L}) \in \left[0, \frac{2}{p+1}\right) \\
|1 - \lambda_i(\mathbf{L})| &> |1 - p\lambda_i(\mathbf{L})| && \text{, if } \lambda_i(\mathbf{L}) \in \left[\frac{2}{p+1}, 2\right) \\
|1 - \lambda_i(\mathbf{L})| &= |1 - p\lambda_i(\mathbf{L})| && \text{, if } \lambda_i(\mathbf{L}) = \frac{2}{p+1}
\end{aligned}$$

Therefore, when the weight matrix has all its eigenvalues in the region $[0, 1]$, the behaviour of the stochastic dynamical system is simple. That is the convergence rate is decreased throughout all phases monotonically, irrespective of the non-stochastic weight matrix assignment. That is because the eigenvalues

increase. Considering the case where the eigenvalues are spread in $[0, 2)$, the effect in the transient phase can be rather complex.

In the case where the asymptotic convergence rate is optimal, the asymptotic rate is reduced or remains the same, as result of edge stochasticity. This due to the fact that the weights are obtained by minimising the norm $\|\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{n}\|$ and are globally optimal, where $\mathbf{W}$ is the non-stochastic weight matrix. This holds because the aforementioned optimisation is convex. Removing an edge is equivalent to setting the corresponding edge weight to 0. Therefore, having a larger rate of convergence implies that there is another point on the parameter space such that the norm $\|\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{n}\|$ is smaller. This implies a second global optimal, which contradicts that the objective function is convex. However, the asymptotic rate may increase under failing links for other weight assignments. That is when the weight matrix assignment and p are such that

$$|1 - p\lambda_n(\mathbf{L})| < |1 - p\lambda_2(\mathbf{L})|$$

which also implies that $|1 - \lambda_n(\mathbf{L})| < |1 - \lambda_2(\mathbf{L})|$. However, this is a case that has to be specifically engineered, but does not occur for the weight assignment schemes that we have described. In these other cases where $|1 - \lambda_n(\mathbf{L})| \geqslant |1 - \lambda_2(\mathbf{L})|$, the asymptotic convergence rate decreases or remains the same.

The effect on the transient phase is largely dependent on the number of eigenvalues of the *Laplacian* which reside in each of the regions $[0, 2/(p+1))$ and $(2/(p+1), 2)$. In order to gain better understanding of this, we have plotted $|1 - p\lambda_i(\mathbf{L})| - |1 - \lambda(\mathbf{L})|$ in (*fig.*12). The eigenvalues that contribute in the transient phase, i.e. the fast eigenvalues, are found around 1. Whereas, the slow eigenvalues are near 0 and 2, near the vertical edges of the figure.

We can distinguish two distinct cases. The rest can be conceived as compound cases of these two. The first being when the edge presence probability p is small. The second is for p being large. In the first case, the resulting behaviour is quite intuitive. All the eigenvalues of the expected weight matrix increase in absolute value, and therefore the stochastic dynamical system is expected to have a reduced rate of convergence in all phases. Furthermore, the transient phase is affected more, since in the region around 1 the eigenvalues of $\mathbb{E}[\mathbb{W}]$ increase, which holds for larger values of p as well.

The other case is displayed on the top side of (*fig.*12). There, some eigenvalues increase while the rest decrease. In most cases, the weighted graph is assumed to have eigenvalues distributed over the entire region under the weighting schemes that we consider. Therefore, the system should exhibit slower rate of

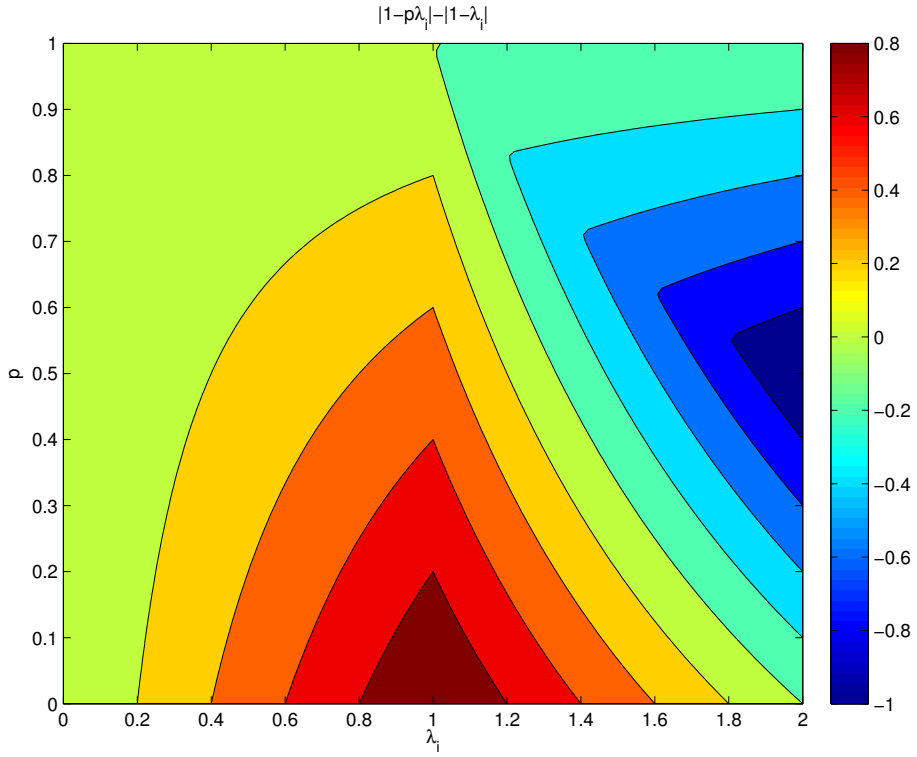


Figure 12: **Effect of edge probability**. The difference of the magnitute of the eigenvalues of $\mathbf{W}$ and $\mathbb{E}[\mathbf{W}]$ is shown with respect to the corresponding eigenvalue $\lambda_i(\mathbf{L})$ and p is displayed. There is a hyperbola $\lambda_i(\mathbf{L}) = \frac{2}{p+1}$ where there is no change. Eigenvalues found on left side of the hyperbola will decrease and on the right side will increase upon effect of failing edges. As a consequence the rate of convergence decreases on the left side and increases on the right side.

convergence in all phases even when p is moderately large, i.e. about 0.5. The impact though is larger in the transient phase. This is due to the fact that fast eigenvalues, i.e. contributing in the transient phase, may be moved to the region of eigenvalues contributing in the asymptotic phase, due to edge stochasticity. Thus, the number of fast eigenvalues reduces. Consequently, the convergence rate during the transient phase reduces.

An interesting remark can be made in the case that one considers designing a network that is robust under link failure . In the right-most side of (*fig.*12), the eigenvalues decrease with p, thus the convergence rate of the expected state increases. Therefore, having the eigenvalues in the region $[2/(p+1), 2)$ allow

for robustness under link failures. Nevertheless, this holds only for the system in (*eq.*3.19). The true system (*eq.*3.21) has more complex behaviour. Hence, this should be considered with care and experimentation. We avert from a detailed discussion, since this evades our purpose.

Summarising, in this section, we have shown that the rate of convergence is expected to be smaller in both phases. Though, the impact in the transient phase is greater. Particularly, the contribution to the deviation from consensus and its duration are both affected as p decreases. This is due to the fact that eigenvalues can be transfered, as an effect of link failures, from the central spectral region of the *Laplacian* to the boundary. Therefore, these contribute less in the transient phase. Furthermore, in both phases, the convergence rate has been shown to be more influenced as p reduces.

3.5.1.1 *Approximate stochastic dynamical system*

In the previous section, we have determined the expected state of a system in presence of stochastic link failures given the initial state. Moreover, we have deduced the impact of link failures based on the spectrum of the expected weight matrix. Our direct interest would be to determine the system's expected deviation from consensus, given the initial state $\mathbf{x}(0)$.

$$\mathrm{E}[\|\mathbf{x}(t) - \frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x}(0))\|^2|\mathbf{x}(0)] = \mathbf{x}(0)^T\mathrm{E}[\prod_{k=0}^{t}\mathbb{W}(k)^T\prod_{k=t}^{0}\mathbb{W}(k)]\mathbf{x}(0) - \mathbf{x}(0)^T\frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x}(0) \tag{3.20}$$

Alas, in the right hand side the matrix products are dependent and cannot be separated.

In fact, that discrepancy is introduced as well when we consider the system's expected state (*eq.*3.19). Instead, we make another approximation and consider that the state $\mathbf{x}(0)$ is given by

Averaged dynamical system model

$$\mathbf{x}(t+1) = \mathrm{E}[\mathbb{W}]\mathbf{x}(t) \tag{3.21}$$

where $\mathrm{E}[\mathbb{W}]$ is the expected weight matrix $\mathrm{E}[\mathbb{W}] = (\mathbf{I} - p\mathbf{L})$.

Hence, the expected variance of the averaged dynamical system is simply

$$\mathrm{E}[\|\mathbf{x}(t) - \frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x}(0)\|^2|\mathbf{x}(0)] = \mathbf{x}(0)^T(\mathbf{I} - p\mathbf{L})^{2(t+1)}\mathbf{x}(0) - \mathbf{x}(0)^T\frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x}(0) \tag{3.22}$$

which implies that the products of stochastic matrices in (*eq.*3.20) are independent. This is obviously not true. However, we shall see with a simple numerical

experiment that in random geometric graphs, this approximation does not affect the qualitative behaviour of the system.

Our interest now is in determining the expected variance of the averaged system, given the assumption that $\mathbf{x}(0)$ is sampled from a gaussian distribution $\mathcal{N}(\mu\mathbf{1}, \sigma^2\mathbf{I})$. The expected deviation from consensus of the averaged system can be retrieved with an identical analysis to (*Section* 3.3) where (*eq.*3.10) was obtained.

$$\mathrm{E}[\Phi(t)] = \frac{1}{n-1}\sum_{i=2}^{n}\lambda_i(\mathrm{E}[\mathbb{W}])^{2t}\sigma^2 \tag{3.23}$$

Subsequently, an equation for the expected deviation from consensus under failing links can be retrieved by substituting $\mathrm{E}[\mathbf{W}]$ in (*eq.*3.23).

$$\mathrm{E}[\Phi(t)] = \frac{1}{n-1}\sum_{i=2}^{n}(1-p\lambda_i(\mathbf{L}))^{2t}\sigma^2 \tag{3.24}$$

Therefore, the deviation from consensus of the stochastic dynamical system, during the asymptotic and the transient phases, relates to the edge presence probability and the eigenvalues of the weighted graph *Laplacian*.

figure generation specifics

We provide (*fig.*13) as evidence of our claims. For the purpose of generating (*fig.*13) we selected an example geometric graph where the presence of the transient phase is evident. The latter has a vertex set size of 100 and the average degree was 4.8. Thereafter, we sampled randomly 1000 initial states $\mathbf{x}(0)$ from a normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Subsequently, we have simulated, for each of the initial states, the stochastic linear dynamical system and the stochastic model, (*eq.*2.21) and (*eq.*3.21) respectively. The link presence probability was selected at $p = 0.4$. We utilised two different weight assignments, for each model, the Metropolis-Hastings weights and weights obtained by minimisation of $\|\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{n}\|$. These two weight assignment schemes result in weight matrices with eigenvalues contained in different regions. These, in the first case, are found in a smaller region of (-1,1] in comparison to the second case where the eigenvalues are spread over the entire (-1,1] region. These two weight assignments and the two cases of the actual stochastic simulated system and the model, result in four combinations.

We have simulated the stochastic dynamical system for these two different weight assignments, computed the standard deviation for each sequence, and afterwards obtained the average over all sequences at each iteration. The results are presented in (*fig.*13), denoted as simulation. There, a comparison is made

with the model in (*eq.*3.21) which is the deviation of the expected state, denoted as *theoretical*. There the evolution of the state's standard deviation is obtained by multiplying at each iteration with the true expectation of the weight matrices. A third comparison is made with the standard deviation of a dynamical system where the expected weight matrix is approximated by $\frac{1}{m-1}\sum_{k=1}^{m}\mathbb{W}(t)]$ at each time sequence. This is denoted as experimental on the plot.

figure explanation

The plots on the right illustrate the accordance of our model with the standard deviation obtained during the actual simulation of the system. The ratios of the experimental and theoretical versus the simulation precision are plotted, respectively. Both remain along the diagonal, which signifies that the model introduced in (*eq.*3.21) approximates well the actual system. Carefully inspecting the **Theoretical** curve in the left-most plots in (*fig.*13), allows us to arrive at the conclusion that discrepancies between the average model and the simulations occur during the transient phase. In comparison with the Simulation the transient phase seems to be much smaller. However, comparing the **Theoretical** curve alone between the upper and lower left most plots, we observe that the length of the transient phase is qualitatively in accordance with the one in the Simulation curve.

Inspecting top and bottom rows of (*fig.*13) allows us to arrive at the following conclusion. On the upper two plots the graph has been assigned the *Metropolis-Hastings* weights. The other case is with edge weights obtained by convex optimisation of $\|\mathbf{W} - \mathbf{1}\mathbf{1}^{\intercal}/n\|$. In the latter case, the eigenvalues are well distributed over the entire $[0, 2]$ range of the spectrum. Whereas in the first case, *Metropolis-Hastings*, the eigenvalues are mostly concentrated within the $[0, 2/(p+1))$ range. In (*fig.*13) one observes, by inspecting mainly the Simulation curve in the case of *Metrqopolis-Hastings*, that the rate of convergence is reduced uniformly throughout the execution of the algorithm. The transient phase lasts from 0 to about 200, where the asymptotic phase commences. However, in the case of the optimised weights the transient phase lasts up to about 100. Moreover, the deviation in the transient phase, comparing at 100, is smaller in the case of *Metropolis-Hastings*. This further justifies that the impact of edge stochasticity is more pronounced on the transient phase when the eigenvalues are spread throughout the entire $[0, 2)$ range, as has been described in (*Section* 3.5.1).

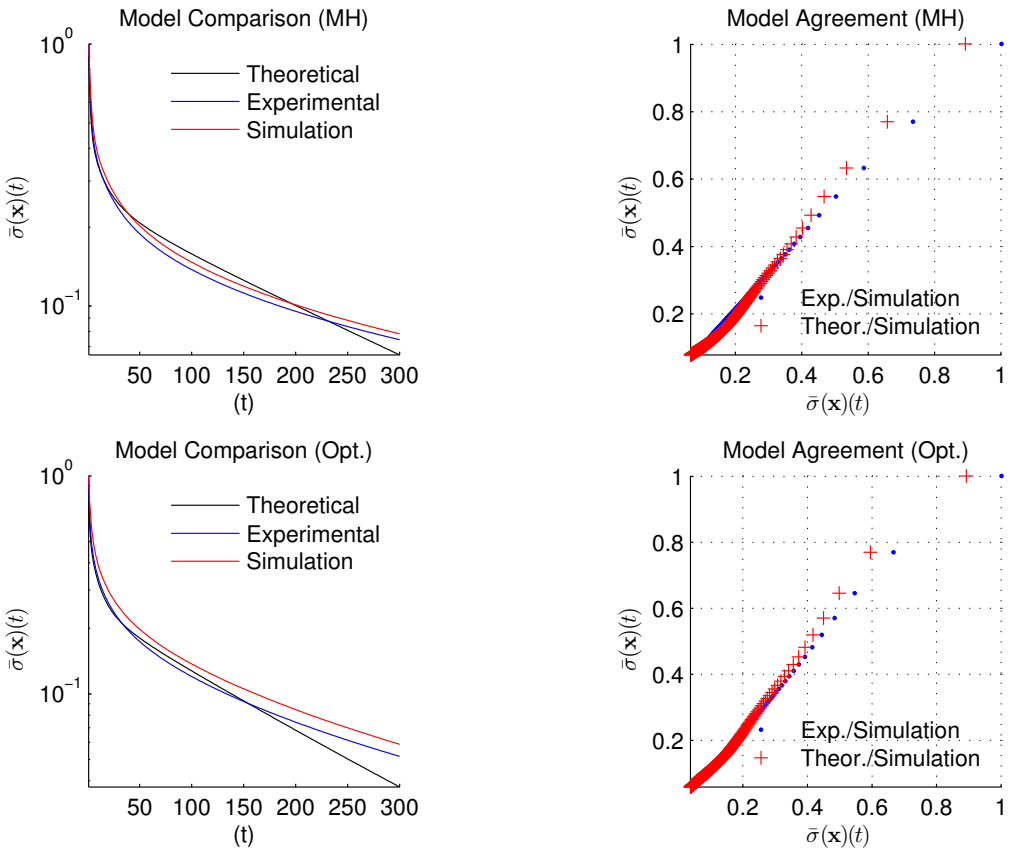


Figure 13: **Averaged System Verification**. MH: Metropolis-Hastings weights, Opt.: Convex Optimized weights. **Left-Top**: Comparison for the model under MH weights. **Theoretical** is the standard deviation of the system in (*eq.*3.21), where $\mathbb{E}[\mathbb{W}]$ has been approximated as the average over all generated matrices. The Experimental curve of average standard deviation is computed by simulating (*eq.*3.21) with $\mathbb{E}[\mathbb{W}]$ determined for each specific sequence of matrices. Thereafter, the average standard deviation is computed over all sequences. The Simulation curve corresponds to the average standard deviation $\bar{\sigma}(\mathbf{x}(t))$ of the simulated stochastic dynamical system. **Right-Top**: Correspondence of the theoretical and the simulation output. **Left-Bottom**:As Left-Top plot but using the optimised weights. **Right-Bottom**: As Right-Top plot but using the optimised weights.

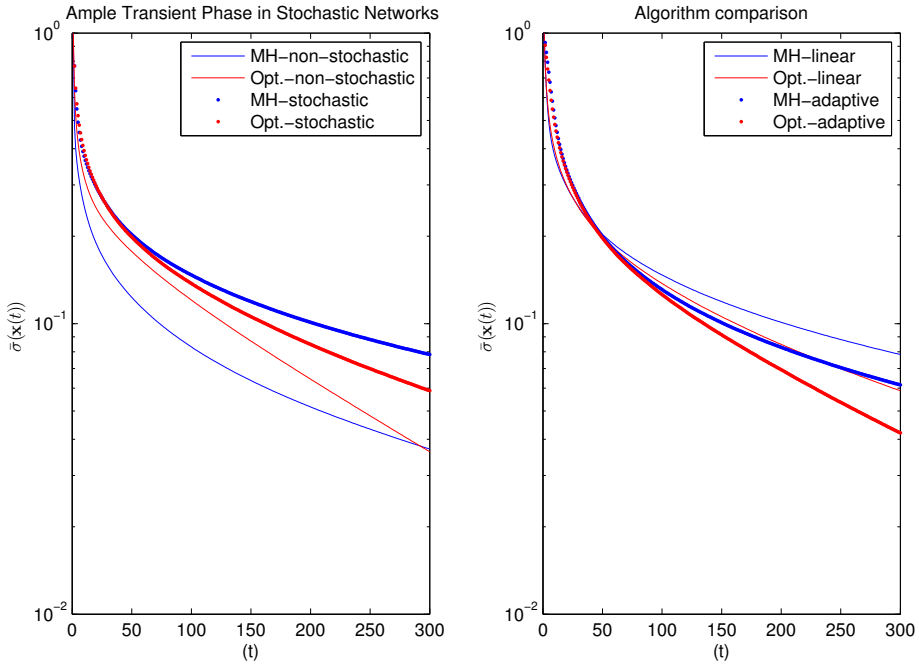


Figure 14: **Impact of Stochasticity**. The mean of the standard deviation $E[\sigma(\mathbf{x}(t))|\mathbf{x}(0)]$ is plotted. MH:Metropolis Hastings weights, Opt.: Convex Optimized weigths wrt asymptotic speed. **Left**: The transient phase is evident in the non-stochastic case (lines). The extended transient phase in the stochastic case (dots) is prominent and asymptotic speed is not achieved. **Right**: A comparison between the adaptive nonlinear algorithm (lines) and the linear (dots). The latter performs worse in comparison to the adaptive for both weight assignments.

3.5.2 *Adaptive Nonlinear Consensus Algorithm*

Drawing from the results presented in section 3.5.1, we claim that it is not preferable to optimise the weights with respect to the asymptotic convergence rate, for the case of the stochastic dynamical network defined in (*eq.*3.21), since this might result in slower convergence rate during the transient phase. The latter seems to be more important in the case of unreliable communications, since the effect of failing links seems to pertubate both the spectrum and the eigenvectors of the graph. Thus, there is a non-constant set of eigenvalues that contributes at each iteration to the rate of convergence. This effect reduces as the state vector approaches slowly $\mathbf{1}$, as in the non-stochastic case. However, now the system remains longer in the transient phase.

In fact, the optimisation of the weights with respect to the asymptotic convergence rate of the non stochastic graph does not aid much, because the edge fail probability has a larger impact and scales the convergence rate approximate proportionally. Therefore, a modification of the consensus algorithm such that it performs better during the transient phase and smoothes the effect of failing links, would be appropriate. Herein, we present such an algorithm which is a modification of the nonlinear consensus algorithm presented in (*Section* 3.4).

We augment the nonlinear dynamical system defined in (*eq.*3.14) and (*eq.*3.16) with a policy to adapt the weights,

$$\theta_i(t+1) = \epsilon_i \left(1 - \frac{1}{2 + e^{-\Delta_2 x_i(t)\Delta x_i(t)}}\right) \tag{3.25}$$

where θ_i is the value on vertex v_i of the parameter θ in (*eq.*3.16), $\Delta x_i(t) = x_i(t) - x_i(t-1)$, and $\Delta_2 x_i(t) = x_i(t) - x_i(t-2)$. Parameter ϵ_i defines the speed of adaptation and finally the maximum of θ_κ. Selecting appropriately these parameters allows to satisfy the conditions in (*eq.*2.12).

Examining the left plot in figure (*fig.*14) one can make four principal observations. Firstly, the two different weight assignments result in different durations of the transient phase in each case. Secondly, the transient phase can be longer in case of stochastic communications. This is evident by examining the curvature in the evolution of $\bar{\sigma}(\mathbf{x}(t))$ for the stochastic system. The delay in achieving asymptotic rate of convergence is also evident upon careful examination.

At the right plot, within figure (*fig.*14), it is observable that the adaptive algorithm results in better convergence rate. This is due to the fact that the weights are adapted to the state. An interesting property which may be further investigated.

Concluding, the adaptive algorithm attempts to retain the transient phase on the stochastic network, and modulates the weights to increase the rate of convergence in both phases, transient and asymptotic. This according to our simulations results in improved precision throughout the operation of the algorithm, in the case of stochastic communications.

3.6 SUMMARY

The consensus algorithm has been known to have slow rate of convergence in large graphs. We have researched towards modifications of the algorithm to increase the precision of the algorithm in early stages. We have introduced the

nonlinear consensus algorithm for this purpose. Specifically, we have shown by simulation that the algorithm enhances the so called transient phase.

Furthermore, we have been concerned with the case of a network with unreliable communication under the model in (*eq.*2.21). A detailed analysis of the effect in this model is not of our direct interest. However, approximating with the averaged dynamical system we have been able to describe qualitatively the impact of the edge presence probability. Based, on our observations, we have introduced an algorithm that adapts the edge weights in respect to the two previous states.

4

DEFINITIVE CONSENSUS

4.1 INTRODUCTION

The principal concern in the previous chapter (*Chapter* 3) has been the speed of convergence of the consensus algorithm which is arguably slow. We have provided evidence for this and illustrated how the speed of convergence can be improved throughout the entire process by modulating the weights on the graph (*Section* 3.4). This furthers our attention in the modulation of the weights throughout the execution of the consensus algorithm. Our main concern in this chapter is to determine a sequence of edge weights such that the network arrives at consensus as fast as possible. In fact we show that given any graph, there is a sequence of weight assignments such that their subsequent application allows the consensus algorithm to be at precise consensus in a finite number of iterations. We call this the definitive consensus algorithm.

Most importantly we provide an efficient method for retrieving these weight assignments for medium sized graphs. The entire set of solutions may be retrieved by computing the Groebner base of the associated system of polynomials. However, the algorithm has exponential time complexity with respect to the number of variables. These are the edge weights, which increase quadratically in the number of vertices and linearly with respect to the diameter of the graph. Instead, we propose to obtain these weight assignments by numerically solving a system of equations, which is surprisingly easy for medium sized graphs.

Specifically, we show that the machines in any given network of connected machines can arrive at agreement in exactly k iterations, where k is larger or equal to the diameter of the graph $d(\mathcal{G})$ and smaller than $2d(\mathcal{G})$. Furthermore, we provide an algorithm that allows to obtain these weights. Moreover, from our experience, we are led to claim that in fact definitive consensus can be attained in exactly $d(\mathcal{G})$ iterations. Subsequently, two methods for numerically obtaining a solution, given a graph, are developed. Finally, we provide some results to cultivate the interest of the reader. Further, results are included in (*Chapter* 6) and (*Appendix* B).

4.1.1 *Motivation*

The speed of convergence of the consensus algorithm is related to the asymptotic rate of convergence. The minimisation of the spectral radius of $\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{n}$ allows to obtain optimal asymptotic rate of convergence. Still, a large number of iterations is required for the algorithm to terminate. The algorithm has practically terminated when the standard deviation of the state vector $\mathbf{x}$ is of order equal to maximum machine precision. The main advantage of this method of finding the weights is that the objecetive function $\|\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{n}\|$ is convex. Therefore, there is a unique global minimum, space and time complexity scale linearly and quadratically with respect to the number of edges on the graph, respectively. However, this only results in obtaining best rate of convergence at the end of the consensus process.

recursive consensus

Alternatively, consider that the weights are switched at subsequent iterations. Hence, the weight matrix is applied $\mathbf{W}_1$ at odd iterations and $\mathbf{W}_2$ at even. The natural question that arises is whether there is a selection of weight matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ such that the consensus algorithm converges faster that having just one matrix applied at each recurrence. Surprisingly, the minimisation of $\|\mathbf{W}_1\mathbf{W}_2 - \frac{\mathbf{1}\mathbf{1}^T}{n}\|$ leads to impressive improvement of the rate of convergence. We ponder whether this can be extended to include a larger number of matrices.

$$\min_{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_r} \|\mathbf{W}_r\mathbf{W}_{r-1}\ldots\mathbf{W}_1 - \frac{\mathbf{1}\mathbf{1}^T}{n}\| \tag{4.1}$$
$$\text{s.t. } \mathbf{W}_r\mathbf{W}_{r-1}\ldots\mathbf{W}_1\mathbf{1} = \mathbf{1},\ \mathbf{W}_s = \mathbf{W}_s^T,\ \forall s \in \{1, 2, \ldots, k\}$$

Then the consensus algorithm (*Algo.*2.1) can be trivially modified to employ the recursion of these matrices.

The weight matrices in (*eq.*4.1) can be obtained by nonlinear optimisation, which is a computationally intense process. Moreover, the value of the minimum is unknown, termination cannot be guaranteed, and the obtained solution may not be a global minimum. Hence, the speed of convergence from the solution of (*eq.*4.1) cannot be guaranteed to be better than minimising for just one weight matrix.

Nevertheless, our interest within the context of this section remains exploratory,thus we need not justify this further. However, as in many nonlinear optimisation problems, one has to be lucky, and usually obtaining a good solution takes more than just one attempt. That is, a solution that performs better

Algorithm 4.1 Recursive Consensus Algorithm, $\mathfrak{C}_r(\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_r, \mathbf{x}, q)$

Execute the while loop for every machine simultaneously
$s \leftarrow 0$
for $t = 1$ to q **do**
  $s \leftarrow s + 1$
  if $s \leqslant r$ **then**
    $x_i \leftarrow \sum_{j=1}^{n} [\mathbf{W}_s]_{ij} x_j$
  else
    $s \leftarrow 0$
  end if
end for

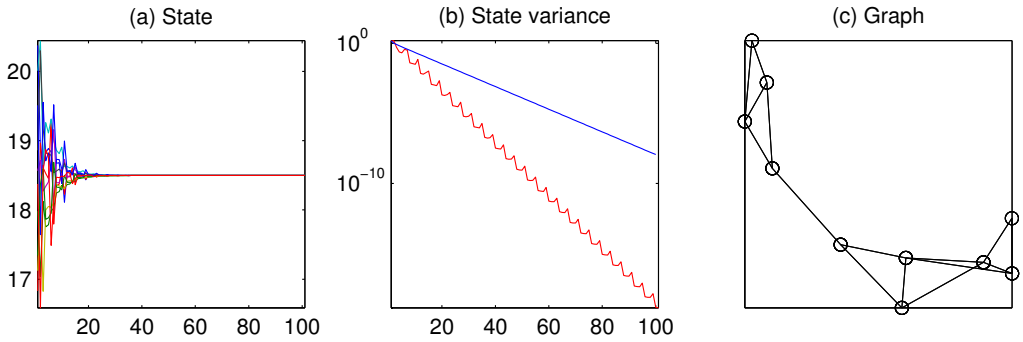


Figure 15: **Definitive Consensus Motivation**. In this figure, evidence is given for the improvement in terms of speed of convergence that can be obtained by employing (*Algo.*4.1). (**a**) The output of the state for 100 iterations of (*Algo.*4.1) is shown. (**b**) The variance of the state is compared for the two algorithms. The blue line designates the execution of the consensus algorithm (*Algo.*2.1). The red line indicates the recursive (*Algo.*4.1). (**c**) The graph is shown, upon which this test has been performed.

than using just one matrix. Results for recursively applying four weight matrices, obtained by solving (*eq.*4.1) are provided in (*fig.*4.1.1).

Inspection of (*fig.*4.1.1) allows to make the following two observations. First, that optimising the weights as in (*eq.*4.1), results in reduced time to convergence throughout the entire process. Second, that the reduction of the standard deviation is not monotonous. The latter is due to the fact that we have not taken specific care to enforce the norm of each of the matrices $\mathbf{W}_r, \mathbf{W}_{r-1}, \dots \mathbf{W}_1$ but only to reduce the norm of their product minus $\frac{\mathbf{1}\mathbf{1}^T}{n}$. Thus, some of the matrices may have $\|\mathbf{W}_r\| > 1$. Therefore, the state variance may increase at the some iterations. However, the improvement is evident overall.

Our experience shows that increasing the number of matrices, i.e. increasing r, drastically improves the rate of convergence. However, the drawback is that the number of variables in (*eq.*4.1) increases as well. The impact of the increase of the size of the network is similar. Having said this, the nonlinearity of the objective function and the constraints further deteriorate the feasibility of finding a good solution. However, the findings of this section draw our attention to the improvements that can be obtained by application of (*Algo.*4.1).

4.2 DEFINITIVE CONSENSUS

The reduction of time to convergence of the consensus algorithm by recursively applying matrices raises the following question.

Problem 4.1 (Definitive Consensus Problem)**.** *Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, does a sequence of weight assignments, represented by matrices $\mathbf{W}_1, \mathbf{W}_2, \dots \mathbf{W}_r$, exist such that by application of (Algo.4.1) the dynamical system arrives at definitive average consensus in r iterations,i.e. $\mathbf{x}(r) = \frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x}(0)$, without having any other sequence $\mathbf{W}'_1, \mathbf{W}'_2, \dots \mathbf{W}'_r$, such that $r > r'$?*

In subsequent sections we are going to show that the necessary number of such matrices is $r = d(\mathcal{G})$, equal to the diameter of the graph, and that the maximum is $2d(\mathcal{G})$. Moreover, we are going to provide evidence that justifies claiming that consensus is always feasible in exactly $d(\mathcal{G})$ iterations. Finally, we are going to provide a method to determine these weight matrices in small and medium sized graphs.

Assume that the answer to (*problem* 4.1) is positive, then algorithm (*Algo.*4.2)would allow the system to be in consensus after r iterations. This is in fact a slight

modification of (*Algo.*2.1). Hence, solving (*problem* 4.1) is of major importance, as well as the problem of finding such matrices.

Algorithm 4.2 Definitive Consensus Algorithm

```
for t = 1 to r do
  for i = 1 to n do
    for j = 1 to n do
      s_i ← Σ_{j=1}^{n} [W_r]_ij x_j
    end for
  end for
  for i = 1 to n do
    x_i ← s_i
  end for
end for
```

It is reminded that the loops for i are executed distributively at each machine separately.

4.2.1 *Formulation*

Assume that for some r the sequence of weight matrices in (*problem* 4.1) exists. Then determining the sequence of these weight matrices is simply the solution of the following matrix equation.

$$\mathbf{W}_r \mathbf{W}_{r-1} \dots \mathbf{W}_1 = \frac{\mathbf{1}\mathbf{1}^T}{n} \tag{4.2}$$

This is not a trivial problem, since the matrices $\mathbf{W}_r, \mathbf{W}_{r-1}, \dots, \mathbf{W}_1$ have a structure imposed by $\mathcal{G}$. Furthermore, it would be of interest to determine such matrices that (*problem* 4.1) is solved.

The above equation can be expanded to a set of n^2 algebraic equations in $r(2m-n)$ variables

$$\sum_{k_{r-1}=1}^{n} \sum_{k_{r-2}=1}^{n} \dots \sum_{k_1=1}^{n} [\mathbf{W}_r]_{i k_{r-1}} [\mathbf{W}_{r-1}]_{k_{r-1} k_{r-2}} \dots [\mathbf{W}_1]_{k_1 j} = 1/n \tag{4.3}$$

where $m = |\mathcal{E}|$ is the edge set size, $n = |\mathcal{V}|$ is the vertex set size, and $[\mathbf{W}_k]_{ij}$ is the ij-th entry of the k^{th} matrix in the sequence of matrices.

4.2.2 *Existence of Solutions*

The solution of (*problem* 4.1) is conjoined with the solution of (*eq.*4.2). In order to solve the latter, two subproblems have to be addressed. The first being the specification of the required number of matrices in the product such that (*eq.*4.2) has a solution. In other words, it is the search for a lower limit for r. Given that the associated graph is not fully connected, then some matrix elements in the weight matrices $\mathbf{W}_t$ must be zero. Therefore, it is possible that in the product some elements are zero. In this case, the system of equations will have no solution since it will include the inconsistent equation $0 = \frac{1}{n}$. This subproblem is addressed in the next section (*Section* 4.2.2.1). The second subproblem is to specify the sufficient number of matrices, i.e. an upper limit for r. We show that a solution can be obtained for r matrices such that $d \leqslant r \leqslant 2d$.

4.2.2.1 *Necessary Iterations*

The evolution of the dynamical system under the operation of the recursive consensus algorithm (*Algo.*4.1) is dictated by the selection of the weight matrices $\mathbf{W}_t$. However, the freedom in the selection of these matrices is still limited from the graph imposed by the communications network. The next theorem states that the minimum number of algorithm iterations for (*eq.*4.2), necessary for a solution, is at least equal to the diameter of the simple graph of communications.

Theorem 4.1. *Assume a connected simple graph $\mathcal{G}$ with diameter d. Then the switching dynamical system defined on $\mathcal{G}$ from the operation of the definitive consensus algorithm in (Algo.4.2) requires at least d state transitions to arrive at definitive consensus.*

Proof. If the product of matrices has less matrices than the diameter of the graph then the matrix $\mathbf{M} = \prod_{t=d}^{1} \mathbf{W}_t - \frac{\mathbf{1}\mathbf{1}^T}{n}$ has at least one element that is equal to $-1/n$.

Let the adjacency matrix of $\mathcal{G}$ be $\mathbf{A}$ such that $[\mathbf{A}]_{ij}$ is zero, unless there is an edge between the pair of vertices $\{v_i, v_j\}$ in which case $[\mathbf{A}]_{ij} = 1$, and consider $\mathbf{I} + \mathbf{A}$. The elements of the p^{th} matrix power of the adjacency matrix $[(\mathbf{I} + \mathbf{A})^p]_{ij}$ enumerate the number of paths from vertex i to vertex j of length smaller or equal to p. Therefore when p is smaller than the diameter d, then there is a pair $\{v_i, v_j\}$ such that $[(\mathbf{I} + \mathbf{A})^p]_{ij} = 0$.

The matrices $\mathbf{W}_t$ must have the same structure as $\mathbf{A}+\mathbf{I}$. Thus, the product of p such matrices cannot have non zero elements where $(\mathbf{A}+\mathbf{I})^p$ has zero elements. Consider that (*eq.*4.2) can be rewritten as

$$
\begin{aligned}
[\mathbf{W}_r\mathbf{W}_{r-1}\dots\mathbf{W}_1]_{ij} = \\
[((\mathbf{I}+\mathbf{A})\circ\mathbf{X}_r)((\mathbf{I}+\mathbf{A})\circ\mathbf{X}_{r-1})\dots((\mathbf{I}+\mathbf{A})\circ\mathbf{X}_1)]_{ij} = \\
\sum_{k_{r-1}=1}^{n}\sum_{k_{r-2}=1}^{n}\dots\sum_{k_1=1}^{n}[\mathbf{I}+\mathbf{A}]_{ik_{r-1}}[\mathbf{I}+\mathbf{A}]_{k_{r-1}k_{r-2}}\dots[\mathbf{I}+\mathbf{A}]_{k_1j} \\
[\mathbf{X}_r]_{ik_{r-1}}[\mathbf{X}_{r-1}]_{k_{r-1}k_{r-2}}\dots[\mathbf{X}_1]_{k_1j}
\end{aligned}
$$

where $\mathbf{X}_r, \mathbf{X}_{r-1}\dots\mathbf{X}_1$ are full matrices in $\mathbb{C}^{n\times n}$, and $\circ$ is the element-wise matrix product. Consequently, the matrix $\mathbf{M}$ will have at least one element equal to $-1/n$. Therefore, we arrive at an inconsistent system which trivially has no solution. □

4.2.2.2 *No Solution for Constant Matrices*

We have to address one matter that naturally arises before we advance further in the discussion. That is to consider one matrix such that $\mathbf{W}^d - \frac{\mathbf{1}\mathbf{1}^T}{n} = 0$. We show that this equation does not have a solution for any graph except for the totally connected graph where $\mathbf{W} = \frac{\mathbf{1}\mathbf{1}^T}{n}$. In these other cases, it follows from

$$\mathbf{W}^d = \frac{\mathbf{1}\mathbf{1}^T}{n}$$

that $\mathbf{W}^d$ has one eigenvalue equal to 1 with eigenvector $\mathbf{1}$ and $n-1$ linearly independent eigenvectors that are orthogonal to 1 with corresponding eigenvalues equal to 0 . Hence, $\mathbf{W}$ has the same eigenvectors with the same eigenvalues, except that instead of the eigenvalue 1, it could also have the eigenvalue -1, for even d. But this would be incompatible with the requirement that $\mathbf{W}$ must have an eigenvalue 1. Thus

$$\mathbf{W} = \frac{\mathbf{1}\mathbf{1}^T}{n}$$

Unless the graph is completely connected, this weight matrix $\mathbf{W}$ is not compatible with the graph.

Remark 4.1. *There is no symmetric weight matrix $W \in \mathbb{R}$ compatible with a graph that satisfies $W^d = \frac{\mathbf{1}\mathbf{1}^T}{n}$ except for the completely connected graph.*

4.2.2.3 *Minimum Diameter Spanning Tree solution*

Having shown that the recursive consensus algorithm requires at least d iterations, where d is the diameter $d(\mathcal{G})$ of the associated graph $\mathcal{G}$, we focus on the second subproblem. We are going to prove that the system of equations in (*eq*.4.2) has at least one solution for any connected graph $\mathcal{G}$ with at most $2d$ matrices. The construction of the solution needs performing two steps. One has to first appropriately prune the edges on the simple graph to construct a so called minimum diameter spanning tree. Then with the appropriate selection of the edges on the associated time-multi-graph of the minimum diameter spanning tree, we show that the system can be at definitive consensus.

This solution is just one of the many that one may be able obtain. However, for the purpose of showing the existence we need not more than one. Apart from a purely theoretical interest in showing the existence of solutions, knowing that the system in (*eq*.4.2) has a solution for at most $2d(\mathcal{G})$ matrices, justifies executing a computationally expensive process, such as a numerical solver, for their retrieval.

construction of the solution

Constructing the solution is straightforward. Every graph $\mathcal{G}$ has a minimum diameter spanning tree. The latter is a spanning subgraph that is a tree and has the minimum diameter among such trees of $\mathcal{G}$. The diameter of this graph is at least d and at most $2d$. A rather more detailed approach to the subject is given in (Hassin and Tamir, 1995) along with a proof based on graph distance.

The retrieval of a minimum diameter spanning tree from $\mathcal{G}$ allows to construct a solution for (*problem* 4.1). Then a simple policy of gathering at the central vertex of the minimum diameter spanning tree (MDST) and then distributing on it, suffices. It is easy to see that if the MDST has an even diameter, then there is a vertex (the centre) with maximum shortest distance to other vertices equal to half the diameter. If the diameter is odd, then there are two adjacent vertices (the centre vertices) such that any other vertex has distance not larger than half the (diameter - 1) to one of these two vertices.

Not having a single vertex as a central vertex introduces some difficulty in constructing the solution with a gather-then-distributed policy. Specifically, we cannot compute the mean at the central vertex. However, this can be overcome by exchanging values at the two adjacent vertices to the central edge and thereafter adjusting the weights on the edges, appropriately.

We proceed by assumption that the (MDST), say $\mathcal{T}$, has been retrieved for a given graph $\mathcal{G}$. The following algorithm when executed on the (MDST) produces

a sequence of weight matrices that solve (*problem* 4.1) in $d(\mathcal{T})$ iterations. We provide (*Algo.*4.3) in the case that $d(\mathcal{T})$ is even and (*Algo.*4.4) if $d(\mathcal{T})$ is odd. In the first case, we denote the central vertex by i^* and in the second case we denote the two central vertices by i^*,j^*.

algorithm operation

Both algorithms follow a similar strategy, and its operation is simple to conceive. In the first case, the algorithm is made from two subprocesses, gather and distribute. The purpose of the gather subprocess is to retrieve the sum of all the values at the centre of the (MDST). This can be achieved by pushing towards the centre the partial sum at each vertex. Proceeding iteratively, this process will cause a transfer of the values at each of the vertices to the central vertex, in $d(\mathcal{T})/2$ steps. In the last step of the gather subprocess, step $d(\mathcal{T})/2$, the average can be computed by dividing by $1/n$, where n is the number of vertices. Alternatively, one could distribute the sum without computing the mean at the central vertex and compute this at the last iteration, at each machine separately.

The second subprocess commences at step $d(\mathcal{T})/2+1$ by pushing outwards from the centre the computed value. Once a machine has received the mean, the value is retained by setting the weight of the self-loop to 1 whereas in the first subprocess it was set to 0. This subprocess, when executed until step $d(\mathcal{T})$, sets all the states to the mean values of the initial states. We provide a detailed explanation of the algorithm to facilitate reading of the formal program in (*Algo.*4.3).

The input to the algorithm is the adjacency matrix $\mathbf{H}$ of the (MDST). The algorithm produces as output a sequence of weight matrices. Each of the aforementioned subprocesses consists of three "for" loops, operating in two directions. One that keeps the time index and the other two running through all the vertices on the tree. At each iteration the distance from the centre, in case of even diameter, and centres in the other case, of an edge connecting vertices v_i and v_j is computed.

The algorithm assigns in the first phase weights for the edges inwards. Specifically, the inwards edges of $\mathcal{T}$ from the vertices found on the layer of the tree having distance from the centre $d(\mathcal{T})/2-t+1$ are assigned value 1 at the t^{th} iteration. The self loops on the layers having distance from the centre smaller than or equal to $d(\mathcal{T})/2-t+1$ are assigned value 1. Therefore, the values on the corresponding vertices are retained until their turn to push the sum towards the centre of the tree. In this manner, the sum of the values found on the vertices at each layer propagates towards the centre. The final outcome of the gather subprocess is that the sum of all the initial values on the vertices are accumulated

at the centre. There, the mean can be computed. The distribute subprocess does not present any important differences from the gather subprocess, except that the outwards edges are selected at each iteration.

Consider the case that the graph has an odd diameter. The previous process has been slightly modified and includes one more part, in order to treat this case. Particularly, the values at the adjacent nodes to the central edge, i^* and j^*, are exchanged. This is executed at the $(d(\mathcal{T})-1)/2+1$ iteration. We present the entire process in in (*Algo.*4.4) as a formal program.

The output of these two algorithms is a sequence of weight assignments that at the last step the average of the initial state assignments is found at each vertex of $\mathcal{G}$. Therefore, these are at average consensus. The weight matrices corresponding to these weight assignments are a solution of the *Definitive Consensus Problem* in $d(\mathcal{T})$ steps. The latter is at most twice the diameter of the original graph. Particularly, this is the case when cycles are present in the graph. In some cases, breaking these cycles, in order to form the tree, does not increase the diameter of the resulting tree. However, consider any circle and form a tree by removing one edge. Removing any of the edges produces a tree of the same diameter, being equal to twice the diameter of the original graph.

The weight assignment obtained through (*Algo.*4.4) and (*Algo.*4.3) provides a solution for (*problem* 4.1) with a factorisation to r matrices , such that $d(\mathcal{G}) \leqslant r \leqslant d(\mathcal{G})$. This is just one of the many solutions that can be obtained. In fact the problem has infinitely many solutions. This is trivial to see. Assume one solution, then one can multiply one matrix by any number and then divide the next matrix by the same number, which produces another solution. Therefore, infinitely many solutions can be produced with this process.

4.2.3 *Numerical Solutions*

Finding an (MDST) of a given graph has $\mathcal{O}(mn + n^2 \log(n))$ time complexity, where m is the number of edges, and n is the number of vertices (Hassin and Tamir, 1995) . One needs to determine the shortest paths which is $\mathcal{O}(m + n\log(n))$ for all vertices. The time complexity of obtaining the weight matrices is $\mathcal{O}(n^2 d(\mathcal{T}))$. Thus this method cannot scale to very large graphs. Alternatively, distributed methods have been proposed (Bui et al., 2004) which have $\mathcal{O}(n)$ time complexity.

However, the main drawback in employing the aforementioned algorithms, (*Algo.*4.3) and (*Algo.*4.4), is that the values are in fact retrieved centrally for

Algorithm 4.3 Gather and Distribute Weight Assignment, even diameter

for $t = 1$ to $d(\mathcal{T})/2$ **do**
  $\mathbf{W}_t \leftarrow \mathbf{0}$
  for $i = 1$ to n **do**
    if $d(i, i^*) \leqslant d(\mathcal{T})/2 - t$ **then**
      $w_{tii} \leftarrow \mathbf{1}$
    end if
    for $j = 1$ to n **do**
      if $d(j, i^*) = d(\mathcal{T})/2 - t + 1$ and $d(j, i^*) > d(i, i^*)$ **then**
        $w_{tij} \leftarrow [\mathbf{H}]_{ij}$
      end if
    end for
  end for
end for
$t \leftarrow d(\mathcal{T})/2 + 1$
$\mathbf{W}_t \leftarrow \mathbf{0}$
$w_{ti^*i^*} \leftarrow 1$
for $i = 1$ to n **do**
  $w_{tii^*} \leftarrow [\mathbf{H}]_{ii^*}/n$
end for
for $t = d(\mathcal{T})/2 + 2$ to $d(\mathcal{T})$ **do**
  $\mathbf{W}_t \leftarrow \mathbf{0}$
  for $i = 1$ to n **do**
    if $d(i, i^*) \leqslant t - d(\mathcal{T})/2 - 1$ **then**
      $w_{tii} \leftarrow 1$
    end if
    for $j = 1$ to n **do**
      if $d(j, i^*) = t - d(\mathcal{T})/2 - 1$ and $d(j, i^*) < d(i, i^*)$ **then**
        $w_{tij} \leftarrow [\mathbf{H}]_{ij}$
      end if
    end for
  end for
end for

Algorithm 4.4 Gather and Distribute Weight Assignment, odd diameter

for $t = 1$ to $(d(\mathcal{T})-1)/2$ **do**
  $\mathbf{W}_t \leftarrow \mathbf{0}$
  for $i = 1$ to n **do**
    if $d(i,i^*) < d(i,j^*)$ **then**
      $k^* \leftarrow i^*$
    else
      $k^* \leftarrow j^*$
    end if
    if $d(i,k^*) \leqslant (d(\mathcal{T})-1)/2 - t$ **then**
      $w_{tij} \leftarrow 1$
    end if
    for $j = 1$ to n **do**
      if $d(j,k^*) = (d(\mathcal{T})-1)/2 - t + 1$ and $d(j,k^*) > d(i,k^*)$ **then**
        $w_{tij} \leftarrow [\mathbf{H}]_{ij}$
      end if
    end for
  end for
end for
$t \leftarrow (d(\mathcal{T})-1)/2 + 1$, $\mathbf{W}_t \leftarrow \mathbf{0}$
$w_{ti^*i^*} \leftarrow 1/n$, $w_{tj^*j^*} \leftarrow 1/n$, $w_{ti^*j^*} \leftarrow 1/n$, $w_{tj^*i^*} \leftarrow 1/n$
for $t = (d(\mathcal{T})-1)/2 + 2$ to $d(\mathcal{T})$ **do**
  $\mathbf{W}_t \leftarrow \mathbf{0}$
  for $i = 1$ to n **do**
    if $d(i,i^*) < d(i,j^*)$ **then**
      $k^* \leftarrow i^*$
    else
      $k^* \leftarrow j^*$
    end if
    if $d(i,k^*) \leqslant t - (d(\mathcal{T})-1)/2 - 1$ **then**
      $w_{tii} \leftarrow 1$
    end if
    for $j = 1$ to n **do**
      if $d(j,k^*) = t - (d(\mathcal{T})-1)/2 - 1$ and $d(j,k^*) < d(i,k^*)$ **then**
        $w_{tij} \leftarrow [\mathbf{H}]_{ij}$
      end if
    end for
  end for
end for

computation. This is somehow contradicting to the main reason for employing the consensus algorithm in the first place. Moreover, the number of iterations required is expected to be larger than $d(\mathcal{G})$ in most cases. Nevertheless, these have been given in order to construct a proof for an existence of solutions. Therefore, these drawbacks are only important in case that we consider the retrieval of such a solution.

Instead, we are interested in determining solutions of the *Definitive Consensus Problem* for exactly $d(\mathcal{G})$ matrices, if they exist. This has not yet been proved. However, our experience does not show otherwise. In order to obtain the weight matrices we employ numerical optimisation. Surprisingly, the solutions are easy to obtain up to medium sized graphs. However, as the size of the network increases, the parameter space becomes enormous. Consequently, the process becomes very slow and the precision attained reduces.

The objective function can be obtained directly from (*eq.*4.3).

$$\mathcal{J}(\mathbf{w}) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij}(\mathbf{w})^2 \tag{4.4}$$

where we have replaced

$$f_{ij}(\mathbf{w}) = \sum_{k_{d-1}=1}^{n} \sum_{k_{d-2}=1}^{n} \dots \sum_{k_1=1}^{n} [\mathbf{W}_d]_{ik_{d-1}} [\mathbf{W}_{d-1}]_{k_{d-1}k_{d-2}} \dots [\mathbf{W}_1]_{k_1 j} - 1/n$$

and $\mathbf{w} \in \mathbb{R}^{2md}$ the weights vector, with m being the number of edges on $\mathcal{G}$, and d its diameter. Then the solution of (*eq.*4.2),

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{J}(\mathbf{w}) \tag{4.5}$$

may be obtained by numerical minimisation of (*eq.*4.4) for those $\mathbf{w}^*$ that $\mathcal{J}(\mathbf{w}^*) = 0$.

4.2.3.1 *Incremental Numerical Solutions*

Unfortunately, directly solving (*eq.*4.2) can become computationally demanding as the size of network increases. However, the computational burden can be reduced based on the following observation.

Consider the product formed by k matrices with $1 < k < d(\mathcal{G})$ and form the system of equations $\{f_{ij}(\mathbf{w}) = 0 | [\mathbf{A}^k]_{ij} = 1\}$. That is the system obtained by ignoring the inconsistent equations $0 = 1$, occurring due to having $k < d(\mathcal{G})$.

Then there is a minimum k such that these equations have a solution for a given graph $\mathcal{G}$. Let this be noted as the nonlinear part of the product.

Interestingly, knowing such a k and the solution for the nonlinear part of the product, for a given graph, allows to obtain the rest of the solution, up to $d(\mathcal{G})$ by incrementally solving linear systems. In fact, the principal difficulty is in obtaining the solution for the nonlinear part.

Formalising what has been discussed in previous paragraphs, we consider the matrix equation

$$\mathbf{W}_k \mathbf{W}_{k-1} \dots \mathbf{W}_1 = \mathcal{R}(\mathbf{A}^k) \tag{4.6}$$

where $\mathcal{R} : \mathbb{N}^{n \times n} \to \{0, 1\}^{n \times n}$ is a matrix function which assigns 0 at ij^{th} element if and only if the corresponding input element is 0.

$$[\mathcal{R}(\mathbf{X})]_{ij} = \begin{cases} 0 & \mathbf{X}_{ij} = 0 \\ 1 & otherwise \end{cases} \tag{4.7}$$

Assume having a solution for (*eq.*4.6) for some $k^* > 1$ such that there is no other k satisfying (*eq.*4.6), with $k^* \leqslant k \leqslant d$. Then the solution for $k^* + 1$ can be obtained by just solving

$$\mathbf{W}_{k^*+1} \mathcal{R}(\mathbf{A}^{k^*}) = \mathcal{R}(\mathbf{A}^{k^*+1}) \tag{4.8}$$

The equations formed by multiplying each row of $\mathbf{W}_{k^*+1}$ with $\mathcal{R}(\mathbf{A}^{k^*})$ do not have common variables. Therefore, these form separate linear systems. Specifically, assume that $\mathbf{u}_i^T$ are the row vectors of $\mathbf{W}_{k^*+1}$ and $\mathbf{b}_i^T$ are the row vectors of $\mathcal{R}(\mathbf{A}^{k^*+1})$. Then the equation above can be separated into n linear systems.

$$\mathbf{u}_i^T \mathcal{R}(\mathbf{A}^{k^*}) = \mathbf{b}_i^T \tag{4.9}$$

Incrementally, the solutions for the matrices from $k^* + 1$ to $d(\mathcal{G})$ can be obtained in this manner.

4.2.4 *Groebner Basis Solutions*

Numerical methods in such complex systems as in (*eq.*4.2), can lead to degenerate solutions. Theoretically, we can only be assured that a system of multivariate polynomial equations has a solution by obtaining its Groebner basis. An introduction to the subject has been provided in (*Section* 2.4). Roughly speaking,

a Groebner base is a set of polynomials which describe all solutions of a set of multivariate polynomial equations. Groebner bases can be retrieved with Buchberger's algorithm (Adams and Loustaunau, 1994), currently implemented in many computer algebra programs, e.g. MATLAB. Having a Groebner base that does not include the unity $\{1\}$ is the necessary and sufficient condition to guarantee that the system of multivariate polynomial equations has at least one solution. Additionally, the Groebner basis gives us the entire set of solutions. In contrast, the solutions obtained by the numerical solver are only specific approximate solutions. Finally, the solutions obtained by Groebner base computation are not dependent on machine precision.

Having obtained a Groebner base, one can retrieve a numerical solution by back-substituting variables. In fact, the polynomials in the Groebner base are in a triangular form. Meaning that no two polynomials have all variables alike. As it has been mentioned already, the system of equations (*eq.*4.2) either has none or an infinite number of solutions. According to Groebner base theory, this implies that, at least one polynomial in the base is multivariate. A simple elimination process produces the desired result. First, select the polynomial with the least number of variables. Then obtain a solution for any of the variables, in terms of the others, and substitute this to the rest. Iteratively, proceeding until all the polynomials have been exhausted, we can obtain one equation in at least one variable which is guaranteed to have a solution in $\mathbb{C}$.

complexity and workarounds

However, the retrieval of a Groebner basis has exponential time complexity with respect to the number of parameters in the worst case. Thus, it is practically impossible to use this method to find the solutions even for some small graphs. However, we can obtain solutions for specific graphs if we carefully select a subset of the variables. That is removing edges or using the same variable for multiple edges. Taking advantage of symmetries is very important for a successful reduction of the equations. In this way, the complexity of the system can be reduced but only a subset of the solutions is preserved. Nevertheless, in such a case the system has been demonstrated to have a set of solutions, which has been our principal purpose. Moreover, we achieve the goal of retrieving an exact subset of the solutions. These can the be employed with maximum precision.

The simplest case are stars. In fact, a star has diameter 2 and (*Algo.*4.3) can be directly applied. However, obtaining the Groebner base includes many other solutions. Subsequently, in the next section, we present a parametrisation that allows to obtain a Groebner base for chord-less cycles of arbitrary size.

4.2.4.1 *Cycles*

A rather more difficult case than the star is that of cycles. In fact, the product of matrices cannot be obtained incrementally as described in (*Section* 4.2.3.1), i.e. $k^* = d(\mathcal{G})$. However, we can parametrise the variables on edges such that solutions are obtainable easily for small cycles, the pentagon, the hexagon, and the septagon. Additionally, this parametrisation reduces the complexity of computing the Groebner base for larger cycles. Though, we have not pursued the solution a cycle larger than $n = 10$.

Particularly, this edge parametrisation is based on a simple observation that one can perform circular permutations of the weights on the edges without in fact affecting the solutions. Hence, it should be possible to retrieve solutions by assigning a variable to the self-loop and the adjacent edges for each iteration. Consider the simple example of the pentagon.

The adjacency matrix is

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

and assigning variables x_1 and y_1 for the self-loop and the adjacent edges to all the nodes gives rise to the first matrix in the product.

$$\mathbf{W}_1 = \begin{pmatrix} x_1 & y_1 & 0 & 0 & y_1 \\ y_1 & x_1 & y_1 & 0 & 0 \\ 0 & y_1 & x_1 & y_1 & 0 \\ 0 & 0 & y_1 & x_1 & y_1 \\ y_1 & 0 & 0 & y_1 & x_1 \end{pmatrix}$$

Similarly, the second matrix can be obtained by replacing x_1 with x_2 and y_1 with y_2. Performing the necessary algebraic operations we obtain just three equations.

$$x_2x_1 + 2y_2y_1 = 1/5 \tag{4.10}$$

$$x_2y_1 + y_2x_1 = 1/5 \tag{4.11}$$

$$2y_2y_1 = 1/5 \tag{4.12}$$

A reduced Groebner base for this system is formed by the following set of polynomials.

$$\begin{aligned}\{-x_1^2 + 2x_1y_1 + 20y_1^4 - 2y_1^2\,,\\ 10x_2y_1^2 - 2y_1 + x_1\,,\\ 10y_1^2 + 5x_1x_2 - 1\,,\\ 5x_2^2 - 10x_2y_2 + 10y_2^2 - 1\,,\\ 5x_1y_2 + 5x_2y_1 - 1\,,\\ 10y_1y_2 - 1\}\end{aligned}$$

It is in fact quite simple to obtain a solution from the original system of equations by substituting $y_2 = \frac{1}{10y_1}$. However, obtaining the Groebner base provides additional information for the original system itself and its solution set. After inspection of the polynomials in the Groebner base, we notice that all the polynomials have at least two variables. This implies, according to theory, that the system has infinitely many solutions. Since the parametrisation introduced is constrained by the original system, we conclude that the original system with the full parametrisation has solutions and their number is infinite, as well.

In larger cycles one may consider a similar parametrisation. Then the k^{th} matrix would be:

$$\mathbf{W}_k = \begin{pmatrix} x_k & y_k & 0 & 0 & 0 & \dots & y_k \\ y_k & x_k & y_k & 0 & 0 & \dots & 0 \\ 0 & y_k & x_k & y_k & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & \dots & y_k & x_k & y_k \\ y_k & 0 & 0 & 0 & \dots & y_k & x_k \end{pmatrix}$$

Therefore, the number of variables is reduced to $2d(\mathcal{G})$ instead of order $3d(\mathcal{G})^2$. Hence, worst case complexity is reduced. Moreover, the resulting equations seem to be much simpler which arguably may further reduce the actual complexity of the task. Specifically, a solution for a circle of 10 vertices can be obtained in about half an hour on a uni-core computing server. In contrast, for the same graph, computing the Groebner base with $3d(\mathcal{G})^2$ variables, did not respond after about 4 months of computation on the same machine.

4.2.5 *Communication Costs*

It is of interest to compare the aforementioned solutions (*Section* 4.2.2.3), (*Section* 4.2.3.1), and (*Section* 4.2.3) with simple approaches such as simple routing to some vertex on $\mathcal{G}$ for processing. Consider having selected a machine to gather all the values distributed over the network. Obviously, one would have to construct a tree out of the original network $\mathcal{G}$. The (MDST) is arguably an appropriate choice. Hence, it has to be retrieved in either case, simple routing or our algorithms (*Algo.*4.4) and (*Algo.*4.3). Thus, the time complexity to assign the routes is computationally identical.

We consider the communication costs for unicast. Its operation on a tree can be abstractly conceived as a push process. A machine retrieves a message from an adjacent machine. The message has a destination , the centre in this case. Each machine has a routing table that in fact states which machine among its adjacent machines is closer to the centre. If the selected machine is not the centre, then the message is forwarded to that closest to the centre. This result of this process is that the central node will obtain finally all the values originally distributed over the network.

In fact, the number of required communications depends on the tree itself. In the best case that $\mathcal{G}$ is a star graph this requires $2(n-1)$ total communications. Assume that the tree is uniform with a branches at each vertex outwards from the centre. If unicast is employed to retrieve the values at the centre and subsequently push them backwards to the nodes of the network, then the cost is given by the equation below.

$$C_{uni} = 2\left(\frac{\ln(an-n+1)}{\ln(a)(a-1)}(an-n+1) - \frac{an}{a-1}\right) \tag{4.13}$$

One could just employ broadcast for the second phase. This in fact does not alter our conclusions, because broadcasting requires $n-1$ communications over the network, which is equal to the communications required by our algorithms. The derivation is included in (*Appendix* A) at (*Section* A.1.2).

Our solution requires exactly $2(n-1)$ total communications over the entire network. That is because each machine waits until all the values have been received, computes locally the summation and pushes the sum to the next vertex closest to the centre. Therefore, each machine only communicates once to send its computed sum to the centre. The same process occurs when receiving the summation. This requires another $n-1$ steps, and therefore the network arrives

at consensus in $2(n-1)$ iterations. This is always smaller than (*eq.*4.13), except for the case of a star graph. Therefore, our protocol is superior to directly employing unicast, even in the case that it is naively being employed directly as a solution to the definitive consensus problem.

4.3 VERIFICATION

Solutions can be obtained with any numerical solver. We have explored the efficacy of such a method in a simple computing server by employing the solver provided in MATLAB. One would probably be able to obtain solutions faster by dedicating time to optimise the solver for this specific problem. However, for our purpose, which is to verify that this method is practically feasible and permits to determine good solutions, it has not been necessary to dedicate effort to such code optimisations.

We have mainly tested the brute force numerical solver and the incremental numerical solver, described in (*Section* 4.2.3). For this purpose we have generated two random geometric graphs of 20 and 30 vertices. The first graph has been used to recover the solutions with direct application of the numerical solver and the second with the incremental solver.

The first graph (*fig.*16) has diameter equal to 8, and the edges are equal to 23. This amounts for 528 variables. The solution, was obtained in about half an hour. In (*fig.*16), two sequences are presented, one for small and the other for large values. In that manner we can naively verify that the solutions are not sensitive to numerical perturbations of the state. Furthermore, one should notice that the standard deviation attains the same value 10^{-9} in both cases. The precision attained is directly dependent on the tolerance of the numerical solver. Therefore, we can in fact select the parameters such that the precision attained by execution of definitive consensus falls within our specific demands for the application.

Furthermore, we can deduce by observing the magnitude of the state that during the execution of the algorithm the state does not converge monotonically. In fact, the standard deviation increases and decreases inconsistently. This is largely dependent on each specific solution, acquired by the numerical solver and the initial state at each execution of the algorithm. In order to address this issue, one can minimise the sum of max norms and impose (*eq.*4.2) as a constraint. However, this remains an even harder optimisation process.

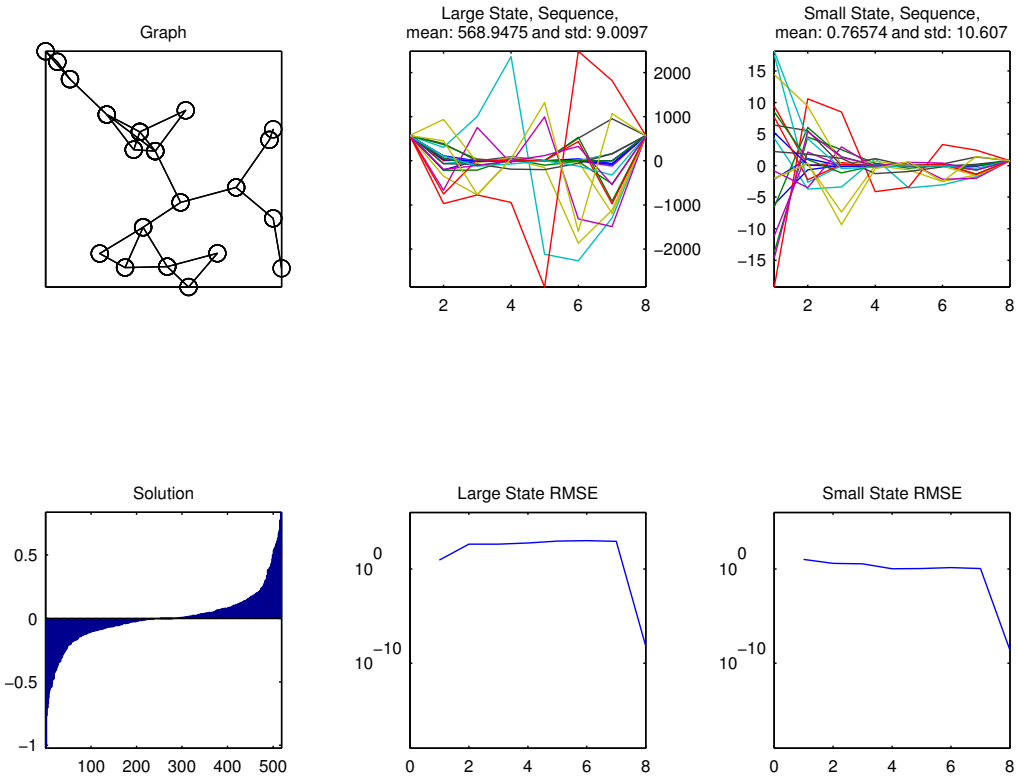


Figure 16: **Convergence of the definitive consensus algorithm using the weights obtained by nonlinear optimisation in a medium sized graph**. From left to right and top to bottom are shown the graph, the states during execution of definitive consensus for an initial state with large values, another with smaller values, the sorted solution (i.e. the values of the parameters), the root mean square error at each step for the first and second execution. Notably, the precision attained in both cases, in $d(\mathcal{G}) = 8$ iterations, is not sensitive to the initial state.

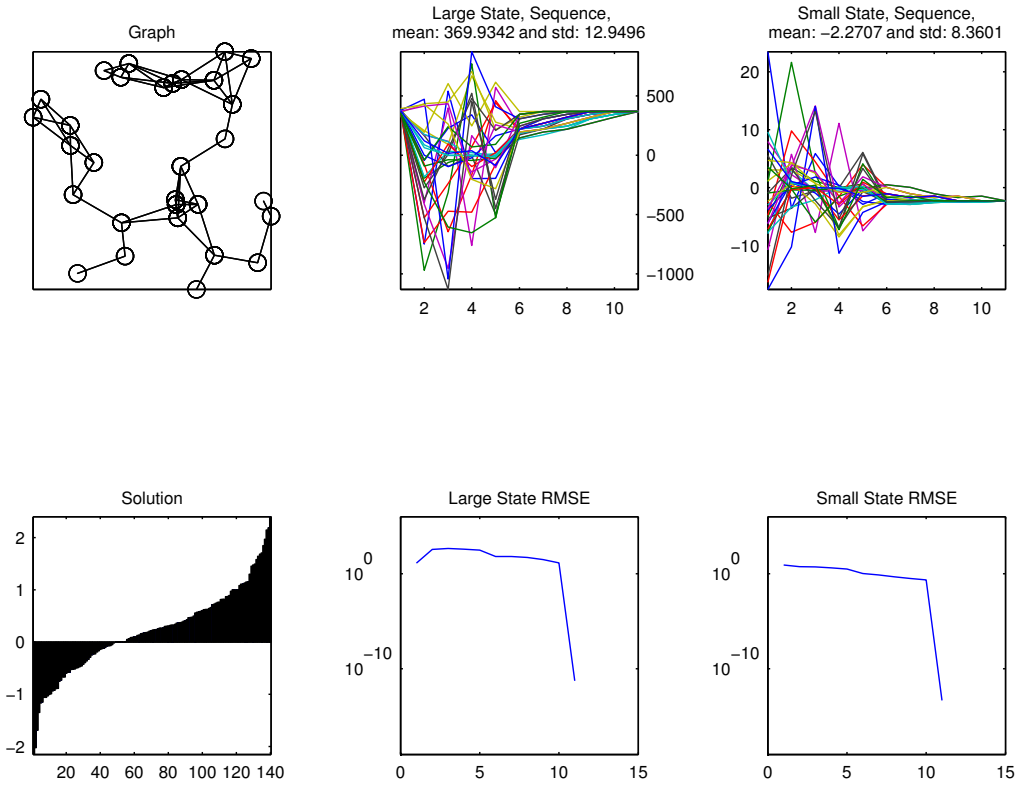


Figure 17: **Convergence of the definitive consensus algorithm using the weights obtained by incremental nonlinear optimisation in a graph of 30 vertices.** From left to right and top to bottom are shown the graph, the states during execution of definitive consensus for an initial state with large values, another with smaller values, the sorted solution of the final matrix (i.e. the values of the edge weights at the last iteration), the root mean square error at each step for the first and second execution. Notably, the precision attained in both cases, in $d(\mathcal{G}) = 12$ iterations, is not sensitive to the initial state.

In larger graphs, it is of interest to reduce the number of variables present in the optimisation. There the incremental numerical solver, described in (*Section* 4.2.3.1), is appropriate. We have tested this method in a graph of 30 vertices and 55 edges with a diameter equal to 10. The nonlinear part of the product of matrices consisted of 5 matrices. This resulted in an optimisation of 700 variables in 584 equations for the nonlinear part. Directly employing the numerical solver on the entire product of these 10 matrices would result in 900 equations and 1400 variables. We have been able to obtain this solution in about 3 hours of computation on the same machine. In contrast, directly solving required 16 hours on the same system, to obtain the same precision.

4.4 CONCLUSIONS

In this chapter we have conceived the *Definitive Consensus Problem*. That is to achieve maximum precision in agreement over a network of connected machines in the minimum number of iterations possible for each given graph. We have determined the necessary conditions, that the graph is connected and that the sequence of matrices consist of $d(\mathcal{G})$ matrices. Furthermore, we have shown that a solution can be obtained for k matrices such that $d(\mathcal{G}) \leqslant k \leqslant 2d(\mathcal{G})$. Moreover, we have conjectured, based on our experience, that a solution exists for any given graph $\mathcal{G}$ in exactly $d(\mathcal{G})$ iterations.

In order to retrieve these solutions, we have suggested three methods, the first being the retrieval of the Groebner base of the system of multi-linear polynomials. We have also provided a parametrisation for cycles that allows to efficiently compute the base up to medium sized chord-less cycles. This we believe to be significant for further theoretical work on affirming the conjecture.

The other two methods are based on numerically solving the system of equations by numerical optimisation. The brute force method has limitations that cannot be easily exceeded. Therefore, an incremental method has been introduced, which allows to obtain solutions for larger graphs faster. The so-called nonlinear part of the matrix product dictates the limitations of this method. This results in arguably smaller parameter spaces. Thus, reducing the complexity of the numerical solver.

The problem of affirming the existence of solutions for the definitive consensus in exactly $d(\mathcal{G})$ iterations remains open. Nevertheless, its theoretical importance is not directly related with the efficiency of retrieving solution. Hence, it is of

greatest interest to conceive algorithms that solve this specific problem for large graphs. Both of these issues indicate areas of auspicious scientific work.

5

DISTRIBUTED DATA INFERENCE

Having shown the efficacy of the consensus algorithms in determining the mean of some quantity over a field of communicating machines, we move onto demonstrating how this can aid in performing machine learning over fields of such machines. Our motivation draws mainly from the case of wireless sensor networks. However, this work may arguably be extended to other cases as well. These include but are not limited to agent computing, distributed computation of large data sets, distributed computation when data is private. However, these have not been extensively studied and the validity of this claim remains to be verified as a future study.

Within this chapter we make the following contributions. Firstly, we propose an abstract framework for supervised machine learning in a distributed manner by employing any of the consensus algorithms presented in this thesis, (*Chapter* 2), (*Chapter* 3) and (*Chapter* 4). This framework is a combination of the consensus algorithm and some iterative machine learning algorithm. The latter shall be referred to, within this text, as the centralised or non-distributed learning algorithm. The framework may also be referred to as the distributed learning algorithm. Secondly, we show that within this framework and given some mild assumptions, the distributed version has three properties. First, that the distributed learning algorithm converges, given that the non-distributed does. Second, that the distributed learning algorithm converges in the same way as the non-distributed counterpart. That is that the learned model of the data will be the same under the same initialisation and given that the learning algorithm is deterministic. Third, that the distributed learning algorithm surpasses the performance of the non-distributed, when the latter is trained only with local data. These are further detailed in subsequent sections.

The rest of the chapter is organised as follows. In (*Section* 5.1) we abstractly introduce the notions discussed, give further motivation, and introduce possible applications where such a framework for distributed inference can be utilised. Afterwards, we discuss our view of a common learning framework (*Section* 5.2.1), thereafter we introduce the modification for distributed learning by consensus. Particularly, we are interested in the application of the definitive

consensus algorithm, which is presented in (*Section* 5.3). The application of this framework in view of the multilayer feedforward neural networks is considered thereafter in (*Section* 5.4.1). This provides a proper baseline comparison for the framework within the scope of an elementary and well known machine learning algorithm. Simulations and conclusions are found in (*Section* 5.5) and (*Section* 5.6), respectively.

5.1 INTRODUCTION

We are concerned with the application of the consensus algorithms in the wider field of distributed computing; particularly the case of distributed machine learning. Specifically, we address the problem of learning from distributed data over a number of connected machines without exchange of this data between the networked machines. Our purpose is to train a neural network at each machine as if the entire dataset was locally available. This is accomplished by employing the consensus algorithms. We introduce an abstract framework for a so called consensus learning, and derive a distributed version for the multilayer feed forward neural network with back-propagation and early stopping. Specifically, the linear consensus algorithm and the definitive consensus algorithm are tested. We discuss the matters that should be given attention in both cases. A careful selection of the parameters allows our method to perform like the non-distributed case. The principal drawback of the framework is that the total computational effort over all machines is larger, but less at each distinct machine. Since, each machine has only to compute the local dataset.

5.1.1 *Problem Layout and Applicability*

The problem at hand can be presented in a few sentences, but its applications are numerous. Suppose a dataset that is distributed over a set of machines with the capabilities of computation and communication. Furthermore, assume that we, for some reason, are not able, or unwilling to communicate the data between machines, or gather it centrally for computation. Instead of just using the local data at each machine, we would like to learn from the entire dataset. In order to illustrate our method and show its feasibility, we have specified our analysis to the case of supervised learning for binary classification of data with feed forward neural networks. However, this study is not limited to this class of

algorithms, since it is founded on a principal operation of many known machine learning algorithms. Evidently, enumerating all these algorithms to demonstrate the applicability of this framework goes way beyond our specific purpose. We present an abstract framework that demonstrates the generality of our approach. (*Section* 5.2.1). The convergence of this method is carefully being considered, which we show, under mild assumptions, in the case that the non-distributed learning algorithm converges.

characteristics of data for consensus learning

There are numerous occasions where the entire dataset may not be available. However, we may consider them as combination the following four basic cases. First, the dataset is too large to be handled by a single machine due to either hardware, software implementation, and or algorithmic limitations. Second, data is intrinsically distributed. The latter can be generated by a set of machines that acquires data by observation or examination (e.g. a set of sensors for environmental monitoring). Third, when data has to remain private but decisions are better performed globally (e.g. when working with clinical patient data). Finally, when data cannot be collected. Hence, it is inaccessible or its access is practically infeasible. This might be due to many reasons, among those just a few are communication costs and failures, energy consumption, and machine downtime. A few of the possible applications where the problem arises are WSN, data mining in large datasets, distributed databases, social networks, and confidential biological data.

For the rest of this chapter we assume a dataset, partitioned and distributed over different machines in an arbitrary manner. Our purpose is to learn from the dataset such that any of the machines when presented with an example from the same generating process can successfully classify it. Moreover, we wish that the classification performed by any machine is identical and the performance equivalent to a centralised case. Also, this has to be achieved without exchanging any datapoints between the machines. Moreover, any machine can communicate with other machines but not necessarily with every other machine. However, we require that the connection graph has no disconnected components. Obviously, the disconnected components cannot be brought to agreement by means of communication algorithms.

a two phase learning process

In our algorithm, the elementary learning process, risk computation and model update, is modified, and becomes a two phase process. The first phase consists of learning with the dataset available locally. This is performed simultaneously at every machine. Therefore, identical learning machines are trained locally but with different data drawn from the distribution of the same generating process.

Actually, this process may even be masked due to the partitioning process. We address this throughout the text. In the second phase, the parameters of the model are communicated to initiate the consensus algorithm and estimate the mean of the learned parameters. We shall demonstrate that this is equivalent under mild conditions to having learned these parameters from the entire dataset. This iterated two phase process, has roughly the same effect as if a single network was trained on the entire dataset.

The process outlined until now is based on the following three assumptions. First, that the iterated two phase process converges, if the non-distributed counterpart does. Second, that consensus on the learned parameters is equivalent to minimisation of the empirical risk of the entire dataset. Third, that the data need not be identically distributed along different machines. We prove the first two based on mild assumptions on how the data is partitioned (*Section* 5.2.1.1). Thereafter, we verify the proof by numerical simulations.

5.1.2 *Related Work*

Related work might be found in the field of distributed optimisation (Rabbat and Nowak, 2004). However, the interest of the researchers there is different. Specifically, the problem of optimisation is not equivalent to a learning problem. In the case of optimisation, the objective function is known a priori. This allows for the computation of the Jacobian, the Hessian, and the retrieval of KKT conditions that designate the proper execution of the algorithm.; especially of the sub-gradient computed with the consensus algorithm, which is equivalent within bounds with the non-distributed case of evaluating the gradient. In contrast, in a machine learning process such facilities are not available since the objective function is unknown.

Other notable attempts are found in the field of distributed support vector machines (Navia-Vazquez et al., 2006), (Lu et al., 2008), (Ang et al., 2008). A notable study has been targeted in modifying the EM-algorithm for distributed computation (Kowalczyk and Vlassis, 2005). An interesting study in the field, which bares specific interest for many applications, is (Kokiopoulou and Frossard, 2010) who has initially brought in our attention this field of research. Nevertheless, most of these studies differentiate from ours since they are specific to each of the learning algorithms in perspective. In contrast, this work introduces an abstract framework that can be applied to a large class of algorithms, since our proof is based on the contraction principle, which is central to many learning algorithms.

Moreover, unlike other studies, we do not exchange any of the data, and do not make explicit assumptions about the manner that the data is partitioned.

5.2 DISTRIBUTED DATA INFERENCE BY CONSENSUS

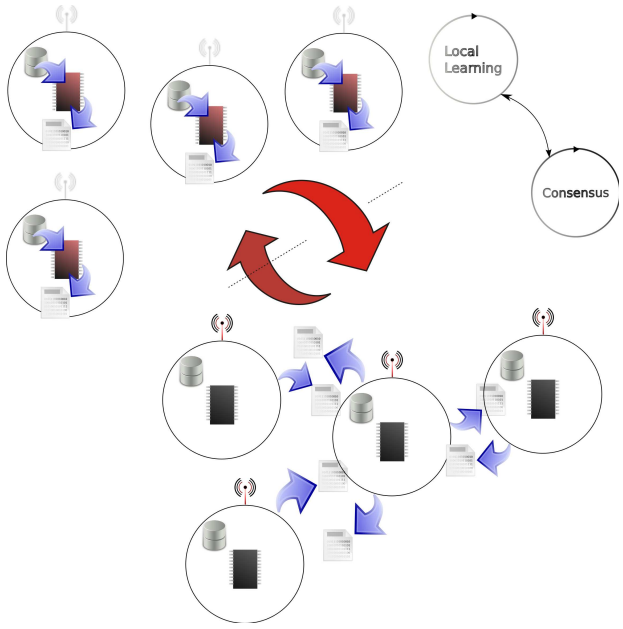


Figure 18: **Abstract operation of a Distributed Inference Framework**. This two step iterative process converges such that all the participating machines achieve the same level of performance. However, the two phases are also iterative. This introduces an interplay between the number of iterations performed at each phase.

Assume a network of machines, connected in an arbitrary manner, such that these are able to communicate with each other but not necessarily directly with every other machine. The communication network can be described by a graph, and each machine by vertex. An edge connects two vertices when the represented machines are able to communicate directly, that is without the intermediation of another. The graph is denoted as $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the set of vertices and edges respectively.

The algorithm for distributed learning has two main phases, as shown in (*fig.*18). Each machine hosts an identical learning algorithm. The first phase consists of the local training, executed at each machine. The second phase is

the execution of the consensus algorithm. There, the learned quantities are exchanged between machines while data is retained locally. Our purpose is to obtain the model as if the entire dataset was available locally at each machine.

5.2.1 *Abstract Framework*

Suppose the dataset $\mathcal{D}$ is segregated in n non-overlapping subsets such that $\mathcal{D} = \cup_{k=1}^{n} {}^{k}\mathcal{D}$. Each subset is ${}^{k}\mathcal{D} = \{{}^{k}\mathcal{X}, {}^{k}\mathcal{Y}\}$, where ${}^{k}\mathcal{X} = \{{}^{k}\mathbf{x}_1, {}^{k}\mathbf{x}_2, \ldots, {}^{k}\mathbf{x}_{m_k}\}$ is the subset of examples, and ${}^{k}\mathbf{x}_i \in \mathbb{R}^s$ is the i^{th} example in the k^{th} subset with $s \in \mathbb{N}$. Similarly for the subset of class labels ${}^{k}\mathcal{Y} = \{{}^{k}y_1, {}^{k}y_2, \ldots, {}^{k}y_{m_k}\}$, ${}^{k}y_i \in \{0, 1\}$ is likewise the label associated with the i^{th} example in the k^{th} subset. Hence the entire dataset is $\mathcal{D} = \{{}^{k}\mathbf{x}_i, {}^{k}y_i\}, k \in \{1, 2, \ldots, n\}, i \in \{1, 2, \ldots, m_k\}$, and m_k is such that $m = \sum_{k=1}^{n} m_k$ is the number of examples in $\mathcal{D}$.

Furthermore, suppose that these datasets are distributed over a network of n machines, described by a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ such that each subset ${}^{k}\mathcal{D}$ is related to one vertex. The problem at hand is presented below (*problem* 5.1).

Problem 5.1 (Distributed Learning)**.** *Given graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and a segregated dataset $\mathcal{D}$ such that ${}^{k}\mathcal{D}$ is related with the k^{th} vertex on the graph, with $k \in \{1, 2, \ldots, n\}$, how can each machine learn a mapping f of examples to class labels $\mathcal{D}$ without communicating any datapoint $\{{}^{k}\mathbf{x}_i, {}^{k}y_i\}$?*

Assume some non-distributed iterated learning algorithm $\mathcal{L}$ that learns on the dataset by minimising the empirical risk (*eq.*5.1). The empirical risk for the entire dataset $\mathcal{D}$ is

$$\mathcal{R}_{emp}(\mathcal{D}, f) = \frac{1}{m} \sum_{k=1}^{n} \sum_{i=1}^{m_k} \mathcal{Q}({}^{k}\mathbf{x}_i, {}^{k}y_i, f) \tag{5.1}$$

where $\mathcal{Q}(\mathbf{x}, y, f)$, $\mathcal{Q} : \mathbb{R}^s \times \{0, 1\} \times \mathcal{F} \to \mathbb{R}$ is a loss function, $f \in \mathcal{F}$ is a classifier $f : \mathcal{X} \to \mathcal{Y}$ that maps examples to class labels, and $\mathcal{F}$ is a set of functions, the set of admissible classifiers.

In general, many non-distributed learning algorithms, let $\mathcal{L}$ denote one, can be conceived as a simple iterative two step process. At the first step the empirical risk is computed. Subsequently, at the second step, an update of the mapping f is determined by some deterministic process $\mathcal{A}$. The latter can be conceived as function $\mathcal{A}(\mathcal{D}, f)$, $\mathcal{A} : \mathcal{D} \times \mathcal{F} \to \mathcal{F}$, such that a new mapping f^* is given. The iterations proceed until a stopping criterion $\mathcal{C}(\mathcal{T}, e, f, f^*)$,$\mathcal{C} : \mathcal{T} \times \mathbb{N} \times \mathcal{F} \times \mathcal{F} \to \{0, 1\}$ becomes true (i.e. 1). Let $\mathcal{T}$ be another set like $\mathcal{D}$, the validation set, and

e is a positive integer, the iteration index. We conceive the learning algorithm $\mathcal{L}(\mathcal{D}, \mathcal{T})$ as a function $\mathcal{L} : \mathcal{D} \times \mathcal{T} \to \mathcal{F}$ which takes as input some dataset from the same generating process and returns the learned model f. This process is summarised in (*Algo.*5.1) where f^* is the model update.

Algorithm 5.1 Non-Distributed Learning Algorithm $\mathcal{L}(\mathcal{D}, \mathcal{T})$

1: Initialise with some f^*, $t \leftarrow 0$
2: **repeat**
3: $\quad t \leftarrow t + 1$
4: $\quad$ Assign $f \leftarrow f^*$
5: $\quad f^* \leftarrow \mathcal{A}(\mathcal{D}, f)$
6: **until** $\mathcal{C}(\mathcal{T}, t, f, f^*)$

However, in the distributed case the dataset is segregated over different machines, and the main problem lies in computing the update in step 5 of (*Algo.*5.1) such that the empirical risk is reduced. The non-distributed learning process with consensus solves the aforementioned problem by determining, at each iteration of $\mathcal{L}$, the update Δf of the classifier f by consensus (*Algo.*2.1). Given a partition ${}^k\mathcal{D}$, the empirical risk is

$$\mathcal{R}({}^k\mathcal{D}, f) = \frac{1}{m_k} \sum_{i=1}^{m_k} \mathcal{Q}({}^k\mathbf{x}_i, {}^k y_i, f) \tag{5.2}$$

and the empirical risk on $\mathcal{D}$ is just the weighted mean over the empirical risk at each machine.

$$\mathcal{R}(\mathcal{D}, f) = \frac{1}{n} \sum_{k=1}^{n} m_k \, \mathcal{R}({}^k\mathcal{D}, f) \tag{5.3}$$

In the process of empirical risk minimisation, the updated classifier should satisfy

$$\mathcal{R}(\mathcal{D}, f^*) - \mathcal{R}(\mathcal{D}, f) \leqslant 0 \tag{5.4}$$

but in the distributed setting this becomes

$$\mathcal{R}({}^k\mathcal{D}, f_k^*) - \mathcal{R}({}^k\mathcal{D}, f_k) \leqslant 0 \tag{5.5}$$

where f_k^* are the updates determined for each subset ${}^k\mathcal{D}$. However, this does not imply (*eq.*5.4), since an update that reduces the empirical risk on one subset

might increase it on another. Therefore, updating such that for some functions f_k^* the local empirical risk is reduced cannot guarantee a reduction of the empirical risk for the entire dataset. Instead the updates f_k^* have to be coordinated such that the empirical risk over the entire dataset is reduced. This is in fact the principal difficulty for distributed learning.

We propose to employ consensus for the coordination of the update functions f_k^*. Therefore, the update step in the distributed setting is

$$f^* = \frac{1}{n} \sum_{k=1}^{n} f_k^* \tag{5.6}$$

We are going to treat its convergence in (*Section* 5.2.1.1); here we concentrate on describing the framework. We define the consensus learning process $\mathcal{CL}(\mathcal{D}, \mathcal{T}, q)$ (*Algo.*5.2) as a modification of the learning process $\mathcal{L}$ (*Algo.*5.1) by executing all the steps locally and adding one more step for the computation of the update by consensus. Thereotically, the latter should be identical for all machines as a result of step 6. Then in step 5 of (*Algo.*5.2), the update of f at each machine is computed as if the entire dataset was locally available. Therefore, step 5 is equivalent to the update step in $\mathcal{L}$ but not identical. The process is presented in (*Algo.*5.2), where f_k^* is the model update of the k^{th} machine, and $\mathbf{f} = (f_1, f_2, \ldots, f_n)$ is the vector of functions $f_1, f_2, \ldots, f_n$ found at each vertex of the graph.

Algorithm 5.2 Consensus Learning Algorithm $\mathcal{CL}(\mathcal{D}, \mathcal{T}, q)$

1: Initialize the local learning machines to f_k^*, $\forall k \in \{1, 2, \ldots, n\}$ and $e \leftarrow 0$
2: **repeat**
3: $\quad e \leftarrow e + 1$
4: $\quad f_k \leftarrow f_k^*, \forall k \in \{1, 2, \ldots, n\}$
5: $\quad f_k^* \leftarrow \mathcal{A}({}^k\mathcal{D}, f_k)$
6: $\quad$ (Consensus algorithm) $\mathbf{f}^* \leftarrow \mathcal{S}(\mathbf{f}^*, q)$.
7: **until** $\mathcal{C}(\mathcal{T}, e, f_k, f_k^*)$, $\forall k \in \{1, 2, \ldots, n\}$

However, there are a few matters that demand careful consideration. First, the algorithm for consensus has only asymptotic convergence to the arithmetic mean value and the agreement among machines. Second, the consensus process is slow in comparison to local computations. Third, the termination function might cause machines to stop asynchronously. These might hinder the success of the framework and are discussed below.

Due to the asymptotic convergence of the consensus algorithm, the estimation of the overall function f_k^* at step 6 may be both inaccurate and different among machines. Subsequently, these are propagated in the update step 5. Therefore, the update f_k^* can diverge between machines. Mainly, this depends on two factors, the update function $\mathcal{A}$ and the precision attained at the consensus step 6. This effect might be even more evident in case that the distribution of the datapoints in the partitions of the dataset is not uniform. Then the empirical risk computed at each machine might be largely diverging among machines. Subsequently, the variance of the empirical risk could be larger and consensus, step 6, may produce larger differences in the determined classifiers f_k. Alas, the algorithm would not necessarily converge as expected.

A workaround in this case is to execute the consensus algorithm for a larger amount of iterations. The convergence up to precision of (*Algo.2.1*) is dependent on a number of parameters. Among them are the algebraic connectivity of the graph $\mathcal{G}$, the cardinality of $\mathcal{V}$, and the variance of the determined quantity (Xiao et al., 2007), in this case the values of the parametrisation of the locally determined function updates f_k^* at each machine. Thus, given a specific graph and the initial variance, we are able only to guess the number of iterations required to achieve some precision and a level of agreement. However, given that the variance of the values cannot be known a priori, the result may still be uncertain.

The second aforementioned problem is that the consensus process is arguably slow. Specifically, it is exponentially fast but dependent on the exponent (*Chapter 3*). The exponent may not be sufficiently large for a number applications. Moreover, due to the fact that a learning process might require a large number of iterations to converge, the entire process could become practically infeasible. However, if we execute the local learning process differentially from the consensus process, then we are able to overcome this. With the term differentially we imply that each phase is executed for a different number of iterations.

However, this modification raises introduces some issues that require consideration. We are going to show in (*Section 5.2.1.1*) that convergence is guaranteed for any combination of learning and consensus iterations. However, the performance of the algorithm is dependent on their combination among other matters, e.g. initialisation, learning step, etc. Though, the performance of any machine learning algorithm is in general sensitive to such parameters. Therefore, this interplay of consensus and learning iterations should be considered in a similar scope as parameters to tune per application. In conclusion, the combination of

local learning steps and iterations for consensus should be carefully designed case by case. However, utilisation of the definitive consensus algorithm allows to overcome this problem. Nevertheless, other matters should be considered in that case. We consider these in (*Section* 5.3).

The final matter having to be examined is the termination function $\mathcal{C}$, which stops the algorithm. Even though in the non-distributed case this would be trivial, here this has some complications. As mentioned before, the updated functions f_k^* may be possibly different. Therefore, the criteria at some machines may be valid for termination but on other machines may have not reached the same decision. Put simply, $\mathcal{C}(\ {}^k\mathcal{T}, e, f_k, f_k^*) = \mathcal{C}(\ {}^l\mathcal{T}, e, f_l, f_l^*)$ cannot be guaranteed for all $l \neq k$. In turn, this implies that not all machines terminate simultaneously.

Algorithm 5.3 Extended Consensus Learning Algorithm $\mathcal{ECL}(\mathcal{D}, \mathcal{T}, l, q)$

1: Initialise $f_k^*,\ \forall k \in \{1, 2, \ldots, n\}$
2: **repeat**
3: $\quad e \leftarrow e + 1$
4: $\quad f_k \leftarrow f_k^*, \forall k \in \{1, 2, \ldots, n\}$
5: $\quad$ **for** $t = 1$ to l **do**
6: $\quad\quad f_k^* \leftarrow \mathcal{A}(\ {}^k\mathcal{D}, f_k)$
7: $\quad$ **end for**
8: $\quad$ Run the consensus algorithm $\mathbf{f}^* \leftarrow \mathcal{S}(\mathbf{f}^*, q)$. $\forall k \in \{1, 2, \ldots, n\}$
9: $\quad c_k \leftarrow \mathcal{C}(\ {}^k\mathcal{T}, e, f_k, f_k^*),\ \forall k \in \{1, 2, \ldots, n\}$
10: $\quad \mathbf{c} \leftarrow \tilde{\mathcal{S}}(\mathbf{c})$
11: **until** $c_k, \forall k \in \{1, 2, \ldots, n\}$

A solution is to run consensus on boolean decisions instead of scalar values. However, (*Algo.*2.1) is not fit for this purpose. Either majority voting or unanimous decision can be employed. The choice of the method depends on many factors but most importantly the application at hand. Recent advancements in the field solve this problem for the case of majority voting when the number of machines is finite, see (Benezit et al., 2009) and (Benezit, 2009). However, the discussion of this algorithm is beyond the purpose of this study, thus the interested reader is directed to the aforementioned literature. Herein, it is assumed that it just works and that it can be employed directly to aid in the termination phase. Let us denote the consensus algorithm on boolean decisions as $\tilde{\mathcal{S}}(\mathbf{x})$. These modifications have been included in (*Algo.*5.3).

We refer to each iteration of (*Algo.*5.3) as an epoch e. At each epoch we distinguish three phases, local learning phase, global learning phase, and the criterion phase. In the first phase the empirical risk is computed and the classifiers are updated locally for l iterations. In global learning phase, the risk over the entire dataset is estimated by running consensus for q iterations. The final phase consists of computing the stopping criteria locally. Then termination can be decided by consensus on the decision value. The last step allows to terminate all machines simultaneously.

5.2.1.1 *Convergence*

Two mild assumptions are made. First that the learning algorithm $\mathcal{L}$ has local convergence in the sense of (*Theorem* 5.1). Second, that the local learning algorithm makes an update step such that some quantity dependent on the update and associated with the data is subsequently reduced. This is an assumption that holds for a large number of machine learning algorithms.

The following well known theorem, see (Agarwal et al., 2001), is useful for the proof of convergence of the distributed learning framework.

Theorem 5.1 (Banach's Contraction Principle)**.** *Let* $(\mathbb{K}, d)$ *be a complete metric space where* $\mathbb{K}$ *is a set and* d *is a metric. Assume* $\mathcal{H} : \mathbb{K} \mapsto \mathbb{K}$ *is a contraction. That is* $d(\mathcal{H}(x), \mathcal{H}(y)) \leqslant \kappa d(x, y)$ *for all* $x, y \in \mathbb{K}$ *with* $0 < \kappa < 1$*. Then* $\mathcal{H}$ *has a unique fixed point* $x^* \in \mathbb{K}$ *that it converges at* $\forall \mathbf{x} \in \mathbb{K}$ *according to*

$$\lim_{n \to \infty} \mathcal{H}^n(x) = x^*$$

where $\mathcal{H}^{n+1} = \mathcal{H}(\mathcal{H}^n)$ *with* $\mathcal{H}^0(x) = x$ *and* $n \in \{1, 2, \ldots\}$.

A learning algorithm optimises some cost function that depends on the training data. Typically, the cost function is the empirical risk or a regularised version of it. It is quite common that this cost function has various local minima. Therefore, an optimisation may get trapped in any of them.

In this paragraph, we want to show that in a very general setting the distributed learning algorithm converges to a classifier that is close to the classifier that would be obtained by the centralised learning algorithm, given that the learning data and the initial classifier are the same. Because of the problem of local minimum, we cannot expect to get such a result for any choice of initial classifier, but we have to restrict it to one that is well within the basin of attraction of a minima (local or global) of every local objective function.

Hypothesis 5.1. *Assume $\mathcal{F}$ the Banach space of all admitted classifiers ("the learning machine"). Let $\mathcal{C} \subset \mathcal{F}$ be a closed bounded convex subset that has the following properties with respect to the learning set $\mathcal{D}$, its partition $\mathcal{D} = \cup_{k=1}^{n} {}^{k}\mathcal{D}$ and the local learning step $\mathcal{A}({}^{k}\mathcal{D}, f)$:*

(a) $\mathcal{A}({}^{k}\mathcal{D}, f)$ leaves $\mathcal{C}$ invariant for all $k = 1, 2, \ldots, n$

(b) For any $k \in \{1, .., n\}$ there exists a constant $0 < \mu_k < 1$ such that for any $f, g \in \mathcal{C}$

$$\|\mathcal{A}({}^{k}\mathcal{D}, f) - \mathcal{A}({}^{k}\mathcal{D}, g)\| \leqslant \mu_k \|f - g\|$$

Remark 5.1. *(Hypothesis 5.1) may seem to be too restrictive to be realistic. However, in a neighbourhood of a minimum of a smooth objective function, it is likely that one can find such a set $\mathcal{C}$. At least when the partition of the data is uniform and there is sufficient data locally.*

To make this case, suppose that the set $\mathcal{F}$ of classifiers is parameterised by a vector $\mathbf{a} = (a_1, a_2, \ldots, a_p)^T \in \mathbb{R}^p$. Suppose that the cost function $\mathcal{R}(\mathcal{D}, \mathbf{a})$ is smooth in $\mathbf{a}$. Let $\mathbf{a}^*$ be a local minimum of $\mathcal{R}$. Then in a ball

$$B(\mathbf{a}^*, r) = \{\mathbf{a} | \|\mathbf{a} - \mathbf{a}^*\| \leqslant r\} \tag{5.7}$$

we have

$$\mathcal{R}(\mathcal{D}, \mathbf{a}) \approx (\mathbf{a} - \mathbf{a}^*)^T M (\mathbf{a} - \mathbf{a}^*) \tag{5.8}$$

where M is a positive definite matrix. Assume that the learning update step of the learning algorithm is gradient descent. In terms of parameter vectors $\mathbf{a}$ a learning step becomes

$$\tilde{\mathbf{a}} = \mathcal{A}(\mathcal{D}, \mathbf{a}) \approx \mathbf{a} - \gamma M (\mathbf{a} - \mathbf{a}^*) \tag{5.9}$$

where γ is a small positive number. Then for sufficiently small γ, $\mathcal{A}$ is a contraction on $B(\mathbf{a}^*, r)$. Indeed for $\mathbf{a}, \mathbf{b} \in B(\mathbf{a}^*, r)$ it holds that

$$\begin{aligned} \|\tilde{\mathbf{a}} - \tilde{\mathbf{b}}\| &\approx \|(\mathbf{a} - \gamma M(\mathbf{a} - \mathbf{a}^*)) - (\mathbf{b} - \gamma M(\mathbf{b} - \mathbf{a}^*))\| \\ &= \|(I - \gamma M)(\mathbf{a} - \mathbf{b})\| \\ &\leqslant \|I - \gamma M\| \|\mathbf{a} - \mathbf{b}\| \end{aligned}$$

Let $0 < \lambda_{min} < \lambda_{max}$ be the minimum and maximum eigenvalues of M. Then choosing $\gamma = \frac{2}{\lambda_{min}+\lambda_{max}}$ we obtain

$$\|I - \gamma M\| = \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} = \mu < 1 \tag{5.10}$$

and therefore

$$\|\tilde{\mathbf{a}} - \tilde{\mathbf{b}}\| \leqslant \mu \|\mathbf{a} - \mathbf{b}\| \tag{5.11}$$

Actually, this inequality holds only approximately. However, by increasing μ somewhat, without surpassing 1, the inequality becomes tight (if the ball $B(\mathbf{a}^*, r)$ is not too large).

Note that since $\mathbf{a}^*$ is a fixed point of $\mathcal{A}$, we deduce also

$$\|\tilde{\mathbf{a}} - \mathbf{a}^*\| \leqslant \mu \|\mathbf{a} - \mathbf{a}^*\| \tag{5.12}$$

which means that $\mathcal{A}$ maps $B(\mathbf{a}^*, r)$ into $B(\mathbf{a}^*, \mu r)$.

Suppose that the data is partitioned. For simplicity we suppose that there are only two parts

$$\mathcal{D} = {}^1\mathcal{D} \cup {}^2\mathcal{D} \tag{5.13}$$

Apply for each part the aforementioned reasoning. There are the local minima ${}^1\mathbf{a}^*$ and ${}^2\mathbf{a}^*$ of the cost functions $\mathcal{R}({}^1\mathcal{D}, \mathbf{a})$ and $\mathcal{R}({}^2\mathcal{D}, \mathbf{a})$, respectively. The functions are then well approximated in the balls $B({}^1\mathbf{a}^*, {}^1r)$ and $B({}^2\mathbf{a}^*, {}^2r)$ by $(\mathbf{a} - {}^1\mathbf{a}^*)^T M_1 (\mathbf{a} - {}^1\mathbf{a}^*)$ and $(\mathbf{a} - {}^2\mathbf{a}^*)^T M_2 (\mathbf{a} - {}^2\mathbf{a}^*)$. The gradient descent learning step functions

$$\tilde{\mathbf{a}} = \mathcal{A}({}^1\mathcal{D}, \mathbf{a}) \tag{5.14}$$

$$\tilde{\tilde{\mathbf{a}}} = \mathcal{A}({}^2\mathcal{D}, \mathbf{a}) \tag{5.15}$$

are contractions on the respective balls

$$\|\tilde{\mathbf{a}} - \tilde{\mathbf{b}}\| \leqslant {}^1\mu \|\mathbf{a} - \mathbf{b}\| \tag{5.16}$$

$$\|\tilde{\tilde{\mathbf{a}}} - \tilde{\tilde{\mathbf{b}}}\| \leqslant {}^2\mu \|\mathbf{a} - \mathbf{b}\| \tag{5.17}$$

Define the intersection of the balls to be

$$\mathcal{C} = B({}^1\mathbf{a}^*, {}^1r) \cap B({}^2\mathbf{a}^*, {}^2r) \tag{5.18}$$

which is indeed a closed bounded convex set. It remains to be determined under what condition it is invariant. This is the case when

$$B({}^1\mathbf{a}^*, {}^1\mu\,{}^1r) \subseteq B({}^2\mathbf{a}^*, {}^2r) \tag{5.19}$$

$$B({}^2\mathbf{a}^*, {}^2\mu\,{}^2r) \subseteq B({}^1\mathbf{a}^*, {}^1r) \tag{5.20}$$

The first inclusion holds if

$$\|{}^1\mathbf{a}^* - {}^2\mathbf{a}^*\| + {}^1\mu\,{}^1r \leqslant {}^2r \tag{5.21}$$

$${}^1\mu \leqslant \frac{{}^2r}{{}^1r} - \frac{\|{}^1\mathbf{a}^* - {}^2\mathbf{a}^*\|}{{}^1r} \tag{5.22}$$

Similarly for the second inclusion it has to hold that

$${}^2\mu \leqslant \frac{{}^1r}{{}^2r} - \frac{\|{}^1\mathbf{a}^* - {}^2\mathbf{a}^*\|}{{}^2r}$$

Given that the learning data is uniformly partitioned and that both ${}^1\mathcal{D}$ and ${}^2\mathcal{D}$ are large enough, then ${}^1\mathbf{a}^*$ and ${}^2\mathbf{a}^*$ are close to each other and ${}^1\mu$, ${}^2\mu$ as well as ${}^1r$ and ${}^2r$ are of almost the same value. In that sense, we can expect the two inequalities to hold and therefore $\mathcal{C}$ to satisfy (*Hypothesis* 5.1).

Theorem 5.2. *Suppose that a distributed learning algorithm makes in one epoch* l *learning steps and* q *consensus iterations. Assume that (Hypothesis 5.1) holds for local learning. Furthermore, suppose that a consensus step is of the form*

$$\tilde{f}_k = \sum_{j=1}^{n} w_{kj} f_j$$

with $w_{kj} \geqslant 0$ *for all* k, j *and* $\sum_{j=1}^{n} w_{kj} = 1$ *for all* k. *Then for any* $l, q \geqslant 1$ *the aforementioned distributed learning algorithm converges, if it is initialised with classifiers* $f_k \in \mathcal{C}$ *and* $k = 1, \ldots, n$.

Proof. We prove that the transformation $(f_1, \ldots, f_n) \mapsto (\tilde{f_1}, \ldots, \tilde{f_n})$ corresponding to one epoch of the distributed learning algorithm is a contraction in $\mathcal{C}^n$. This implies convergence of the distributed learning algorithm to the unique fixed point $(f_1^\infty, \ldots, f_n^\infty) \in \mathcal{C}^n$ of the transformation.

Let us define the norm in $\mathcal{C}^n$ by

$$\|(f_1, \ldots, f_n)\| = \max_k \|f_k\| \tag{5.23}$$

Starting from $f_k \in \mathcal{C}^n$ the classifier $\tilde{f}_k$, obtained by l iterations of the local learning steps $\mathcal{A}({}^k\mathcal{D}, f_k)$, remains in $\mathcal{C}^n$. This is due to the fact that (*Hypothesis*

5.1) is satisfied by f_k. Furthermore, the mapping $f_k \mapsto \tilde{f}_k$ is a contraction in $\mathcal{C}$ with the factor ${}^2\mu_k$.

A consensus step maps $(f_1, .., f_k)$ to $(\tilde{f_1}, \ldots, \tilde{f_n})$ where

$$\tilde{f}_i = \sum_{j=1}^{n} w_{ij} f_j \tag{5.24}$$

Since $\mathcal{C}$ is convex, $f_j \in \mathcal{C}\, \forall j$ implies $\tilde{f}_i \in \mathcal{C}\, \forall i$. Furthermore,

$$\begin{aligned}
\|\tilde{f}_i - \tilde{q}_i\| &= \|\sum_{j=1}^{n} w_{ij} f_j - \sum_{j=1}^{n} w_{ij} g_j\| \\
&= \|\sum_{j=1}^{n} w_{ij} (f_j - g_j)\| \\
&\leqslant \sum_{j=1}^{n} w_{ij} \|f_j - g_j\| \\
&\leqslant \sum_{j=1}^{n} w_{ij} \max_j \|f_j - g_j\| \\
&= \max_j \|f_j - g_j\|
\end{aligned}$$

Thus

$$\|\tilde{f}_i - \tilde{q}_i\| \leqslant \max_j \|f_j - g_j\| \tag{5.25}$$

The combination of local learning and consensus iterations defines a transformation in $\mathcal{C}^n$

$$\mathbf{f} \xrightarrow{l} \tilde{\mathbf{f}} \xrightarrow{q} \tilde{\tilde{\mathbf{f}}} \tag{5.26}$$

where $\mathbf{f} = (f_1, \ldots, f_n)$, $\tilde{\mathbf{f}} = (\tilde{f_1}, \ldots, \tilde{f_n})$ and $\tilde{\tilde{\mathbf{f}}} = (\tilde{\tilde{f}}_1, \ldots, \tilde{\tilde{f}}_n)$, which is a contraction. Observe that

$$\begin{aligned}
\max_i \|\tilde{\tilde{f}}_i - \tilde{\tilde{g}}_i\| &\leqslant \max_i \|\tilde{f}_i - \tilde{g}_i\| && (5.27)\\
&\leqslant \max_i \mu_i^l \|f_i - g_i\| && (5.28)\\
&\leqslant (\max_i \mu_i)^l \max_i \|f_i - g_i\| && (5.29)
\end{aligned}$$

□

The question remains, what classifiers $f_1^\infty, \ldots, f_n^\infty$ does the distributed converges to? First of all, q should be large enough such that the classifiers $f_1^\infty, \ldots, f_n^\infty$ are very close to each other. Subsequently, we consider the idealised case where in each epoch consensus has been reached such that in the limit it holds that $f_j^\infty = f_k^\infty = f^\infty$.

Consider the case that $l = 1$. If the learning step satisfies:

$$\mathcal{A}(\mathcal{D} \cup \tilde{\mathcal{D}}) = \frac{1}{|\mathcal{D}| + |\tilde{\mathcal{D}}|}(|\mathcal{D}|\mathcal{A}(\mathcal{D}, f) + |\tilde{\mathcal{D}}|\mathcal{A}(\tilde{\mathcal{D}}, f)) \tag{5.30}$$

for all $f \in \mathcal{F}$, e.g. in the case of gradient descent on the empirical risk, and if we choose the matrix $\mathbf{W}$ for the consensus algorithm such that the left eigenvector for the eigenvalue 1 is $\left(\frac{|{}^1\mathcal{D}|}{|\mathcal{D}|}, \ldots, \frac{|{}^n\mathcal{D}|}{|\mathcal{D}|}\right)$, then the distributed learning algorithm converges to the same classifier as the non-distributed.

This can be seen as follows. The centralised algorithm transforms a classifier f in one learning step into $\mathbf{f}$ with

$$f_k = \mathcal{A}({}^k\mathcal{D}, f) \tag{5.31}$$

Then the consensus algorithm transforms $(f_1, \ldots, f_n)$ into $(\tilde{f}, \ldots, \tilde{f})$ with

$$\begin{aligned}\tilde{f} &= \frac{|{}^1\mathcal{D}|}{|\mathcal{D}|}f_1 + \ldots + \frac{|{}^n\mathcal{D}|}{|\mathcal{D}|}f_n \\ &= \frac{1}{|\mathcal{D}|}\sum_{k=1}^{n} |{}^k\mathcal{D}|\mathcal{A}({}^k\mathcal{D}, f) \\ &= \mathcal{A}(\mathcal{D}, f)\end{aligned}$$

Therefore, one step of the centralised learning performs exactly the same transformation as one epoch of the distributed learning.

We consider the case $l = \infty$, which is an idealised case as well. Let the weight matrix $\mathbf{W}$ have left eigenvector $\left(\frac{|{}^1\mathcal{D}|}{|\mathcal{D}|}, \ldots, \frac{|{}^n\mathcal{D}|}{|\mathcal{D}|}\right)$ for eigenvalues 1. Then the distributed algorithm converges to

$$f^\infty = \frac{1}{|\mathcal{D}|}\sum_{k=1}^{n} |{}^k\mathcal{D}|f_k^\infty \tag{5.32}$$

where f_k^∞ is the classifier that local learning converges to with input the subset ${}^k\mathcal{D}$ of the entire dataset $\mathcal{D}$.

The definitive consensus algorithm can be applied in a straightforward manner. We need only replace the step for consensus with the definitive consensus algorithm. The result would be equivalent to the first extreme case in the distributed algorithm's proof for convergence, where it has been assumed that the consensus algorithm is executed for infinitely many steps. However, this is only the case when the local learning iterations are equal to one. We are going to see later on by numerical simulations (*Section* 5.5) how the number of learning iterations affect the performance of the learned model. Nonetheless, in the case that local learning iterations are set to be equal to one, the definitive consensus algorithm allows to perform learning in a distributed manner exactly as if an identical learning machine had been trained on the entire dataset.

Therefore, we can control the learning parameters with standard methods. Notably, employing the definitive consensus algorithm allows to perform regularisation directly, just like in the case of the non-distributed counterpart. The latter provides better control over the final outcome of the algorithm.

Algorithm 5.4 Definitive Consensus Learning Algorithm $\mathcal{DCL}(\mathcal{D}, \mathcal{T}, l, \zeta)$

Initialize the local learning machines to some $f_k^*,\ \forall k \in \{1, 2, \ldots, n\}$
repeat
  $e \leftarrow e + 1$
  $f_k \leftarrow f_k^*, \forall k \in \{1, 2, \ldots, n\}$
  for $t = 1$ to m **do**
    $f_k^* \leftarrow \mathcal{A}({}^k\mathcal{D}, f_k, \zeta), \forall k \in \{1, 2, \ldots, n\}$
  end for
  $\mathbf{f}^* \leftarrow \mathcal{DS}(\mathbf{f}^*)$. (Definitive Consensus)
  $c_k \leftarrow \mathcal{C}({}^k\mathcal{T}, e, f_k, f_k^*),\ \forall k \in \{1, 2, \ldots, n\}$
  $\mathbf{c} \leftarrow \tilde{\mathcal{S}}(\mathbf{c})$
until $c_k, \forall k \in \{1, 2, \ldots, n\}$

The definitive consensus algorithm is presented in (*Algo.*5.4). The process is identical to (*Algo.*5.3). However, the update now is computed by definitive consensus, denoted $\mathcal{DS}(\mathbf{f})$, instead of consensus. Each epoch of the algorithm is one iteration of the outer loop; the epoch count is denoted by e. The update function has been modified to include a regularisation term ζ, which reflects the usage of regularisation in the empirical risk function. However, we do not

extend to the treatment of selecting the regularisation function, since this matter is specific to the learning algorithm and the generative function of the dataset. Nevertheless, the effect of regularisation is important, thus we examine it by numerical simulation in (*Section* 5.5).

The internal loop depicts the operation of learning at each machine in the network. This can be executed in a serial or parallel manner. However, the main requirement is that the machines terminate at this phase within some given time-frame, but not necessarily together. Once this phase has been completed, then the updates are communicated by consensus to determine the new update. Subsequently, this is used to evaluate the termination criteria c_k, which may be different when the test sets ${}^k\mathcal{T}$ at each machine are different. Thereafter, termination on the network can be decided with boolean consensus algorithm in the next step, e.g. majority voting, denoted $\mathcal{S}(\tilde{\mathbf{c}})$; already discussed for (*Algo.*5.3). In the case that a common test set is used, then these two steps are obsolete.

We have completed the purely theoretical side of this work with the presentation of the definitive consensus algorithm. Having founded the abstract framework, it is of interest to verify the theory by numerical simulations of distributed learning on abstract network. Therefore, the rest of this chapter is devoted to the applied side of this research.

5.4 APPLICATION OF THE ABSTRACT FRAMEWORK

We are concerned with the practical application of the framework for distributed learning by consensus. Therefore, we have selected one of the elementary but well adopted algorithms in the field to test its practicality. Below, we introduce the necessary modifications and thereafter we verify by simulations the validity of our claims.

5.4.1 *Feed Forward Multilayer Neural Networks with Back-propagation*

A description of this type of neural networks is given in a number of books with the most prevalent being (Bishop, 1996). Though, we remind that the inner product of the parameters $\mathbf{a}_j^l$ and the input vector $\mathbf{u}_j^l$ of the j^{th} neuron at the l^{th} layer is endowed with a non-linear activation function g. Each neuron on the network realizes a function $h(\mathbf{x}_j^l, \mathbf{a}_j^l, b) = g((\mathbf{u}_j^l)^T \mathbf{a}_j^l) + b$. The learning process of the neural network is governed by the following set of equations, referred

as the backpropagation rule. The update of the synaptic coefficients a_{jl} upon presentation of a learning sample (x_i, y_i) is given by

$$\delta_j = g'(\mathbf{u}^T \mathbf{a}_j) \frac{\partial Q(\mathbf{x}_i, y_i, h)}{\partial h} \tag{5.33}$$

$$\delta_j^{l-1} = g'((\mathbf{u}_j^{l-1})^T \mathbf{a}_j^{l-1}) \sum_q a_{kj}^{l-1} \delta_{q_l}^l \tag{5.34}$$

$$\Delta a_{jq}^l = -\eta \delta_j^l u_{jq}^l \tag{5.35}$$

where δ_q^l is the error contribution of the q^{th} neuron at the l^{th} layer, g' is the derivative of the activation function, Δa_{jh}^l is the parameter adjustment, and η is a small constant, the learning rate. The first equation defines the error contribution of the last layer's output, before the output neuron. The second equation defines the error contribution of the j^{th} neuron of the layer indexed $l-1$ by backpropagating the error contributions of neurons at the l^{th} layer. The summations are done over the parameters of the input connections a_{qj}^{l-1} of the j^{th} neuron on the layer indexed $(l-1)$. Then (*eq.*5.35) allows the parameter updates of every connection on the network. The reader is directed to(Bishop, 1996) for a concise presentation of the algorithm.

In the setting of distributed consensus, it is better to consider batch updates.

$$\delta = g'(\mathbf{u}^T \mathbf{a}) \sum_{i=1}^{m} \frac{\partial Q(\mathbf{x}_i, y_i, h)}{\partial h} \tag{5.36}$$

In view of the partitioned dataset and batch updates, the error of the last layer before the output neuron is modified as follows.

$$\delta = g'(\mathbf{u}^T \mathbf{a}) \sum_{k=1}^{n} \sum_{i=1}^{m_k} \frac{\partial Q(\mathbf{x}_i, y_i, h)}{\partial h} \tag{5.37}$$

However, at each k^{th} machine the error that is evidently

$$[\delta]_k = g'(\mathbf{u}^T \mathbf{a}) \sum_{i=1}^{m_k} \frac{\partial Q(\mathbf{x}_i, y_i, h)}{\partial h} \tag{5.38}$$

and $\delta = \sum_{k=1}^{n} [\delta]_k$. Therefore the parameter update is just the sum of the partial weight updates determined at each machine. The computation of the parameter update rule (*eq.*5.39) is straightforward in a distributed fashion by consensus as

in (*Algo.*2.1). The consensus update equation is given in (*eq.*5.40), where, $[\Delta a^l_{jh}]_k$ is the parameter update at the k^{th} machine.

$$\Delta a^l_{jh} = -\eta \sum_{k=1}^{n} [\Delta a^l_{jh}]_k \tag{5.39}$$

$$[\Delta a^l_{jh}]_i \leftarrow \sum_{k=1}^{n} w_{ik} [\Delta a^l_{jh}]_k \tag{5.40}$$

The consensus learning framework has been specified, and consensus is executed on the update parameters. The level of agreement on the parameters is what determines the convergence of the algorithm.

5.4.1.1 *Early stopping*

Early stopping is used to terminate the algorithm. This is mainly due to the difficulty in communicating a decreasing learning rate throughout the network. Additionally, early stopping provides a simple method to avoid over-fitting. In the framework, presented in (*Algo.*5.3), we can directly implement this as the termination criterion.

We can employ early stopping, with the termination criterion in (*eq.*5.41), in a distributed fashion by validating the learned network after each epoch on another dataset (validation set) locally.

$$\mathcal{C}(\mathcal{T}, e, \mathbf{a}_k, \mathbf{a}^*_k) = \begin{cases} 0 & \text{if} \quad \mathcal{R}(\mathcal{T}, f(\mathbf{a}^*_k)) - \mathcal{R}(\mathcal{T}, f(\mathbf{a}_k)) \leqslant 0 \\ 1 & \text{if} \quad \mathcal{R}(\mathcal{T}, f(\mathbf{a}^*_k)) - \mathcal{R}(\mathcal{T}, f(\mathbf{a}_k)) > 0 \end{cases} \tag{5.41}$$

As described in (*Section* 5.2.1) due to discrepancies in the computation of consensus the decisions may be different. To overcome this we run consensus on the local decisions $\{0, 1\}$ taken at each machine, as in (*Algo.*5.3).

In (*eq.*5.41) it has been implied that the validation set is the same for every machine. However, it is likely that the validation set is also partitioned, $\mathcal{T} = \cup_{k=1}^{n} {}^k\mathcal{T}$. Then (*eq.*5.41) can no longer be applied as is. Instead, we compute the differences locally

$$\Delta\, {}^k\mathcal{R} = {}^k\mathcal{R}({}^k\mathcal{T}, f(\mathbf{a}^*_k)) - {}^k\mathcal{R}({}^k\mathcal{T}, f(\mathbf{a}_k)) \tag{5.42}$$

and augment the aforementioned process by running one more consensus step to determine the difference over the entire validation set. Alternatively, consensus should be executed in order to agree on local decisions.Otherwise, the machines might not terminate simultaneously.

in order to validate the distributed data inference framework, we evaluate the multilayer feedforward neural network with the modified equations presented in the previous section (*Section* 5.4.1). Extensive experimental results are presented for the given dataset in case of uniform and non-uniform data partitioning in a relatively small communication network. Additional results can be found in (*Chapter* 6).

5.5.1 *Consensus Learning Results Preamble*

Testing of the algorithm has been performed on a variant of the so called two-moons dataset (Zhou et al., 2004). The generation is performed by uniformly sampling points along a circle of radius 1. The points from the upper half of the circle are displaced vertically and horizontally, then Gaussian noise has been added in the vertical direction. These were 0.9, 0.5 and 0.1 for the vertical, horizontal, and standard deviation, respectively. The upper half has been labelled as class 0 and the lower half 1. The main advantage of this dataset is that it presents a problem where the researcher can adjust the difficulty of the problem with just a few parameters. Particularly, bringing the two semicircles closer and increasing the variance of the noise on the vertical axis, increases the overlap of the classes. The uncertainty on the vertical axis follows a normal distribution, thus in principle the dataset presents a non-separable problem. However, we can move from having empirically separable classes to non-separable by transposing the two semi-circles from further to closer, respectively.

5.5.1.1 *Example of Classification*

We have selected a neural network of 7 neurons placed in two hidden layers [5 2]. The sigmoid function was implemented as activation function of the neurons in these two layers. The input and output layers had linear functions. For the purpose of the experiment, we have generated 50 datasets, with 80 examples per class. We consider both non-uniform and uniform data partitioning. In the later case, each dataset has been divided into 10 parts. The points have been sampled uniformly such that each partition contains examples from both classes. Then these partitions have been allocated on the vertices of the communication graph

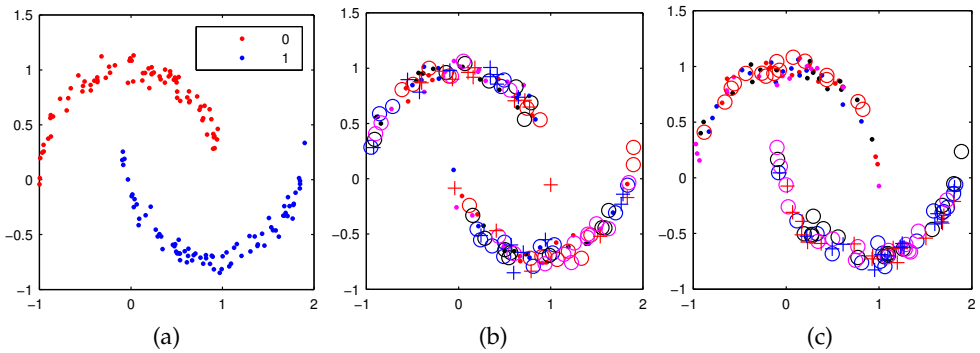


Figure 19: **Example of the *Two Moons Dataset*.** For left to right, in (a) an example of the two moons dataset is shown. This has two classes, top red is class 0 and bottom blue is class 1. In (b), the partitions occurring with uniform sampling are presented. Different, shape, size and colour of the points, indicate different partitions. The given example consists of 10 partitions. Cautious examination of the figure leads to the observation that all partitions have examples from both classes. In contrast, in (c), this does not hold, e.g. blue points occur only on the top circle.

and the algorithm (*Algo.*5.3) has been executed. An example of the results is given in (*fig.*20). The connection graph $\mathcal{G}$ of 10 vertices is also shown.

The dataset has been locally partitioned into training and validation for the purpose of locally training the neural networks.Thereafter, the partitions were allocated to the vertices for each of the 50 datasets. Additionally, one dataset, $\mathcal{T}$ was generated for the purpose of measuring performance, given by (*eq.*5.1). For the purpose of comparing performance, a neural network for each of the 50 datasets has been trained on the entire training set.

We have also considered the case of non-uniform data partitioning. Again 10 parts have been allocated randomly on the vertices. Non-uniform segregation has been performed by uniformly sampling points from only one class for each given partition until datapoints have been exhausted in the given class. In such a case, partitions for the other class are created in the same manner. Any remaining datapoints are placed in one partition. Therefore, there can be only one partition with datapoints from both classes. The rest of the partitions contain examples from only one class. This increases the difficulty of the distributed problem.

An example is given in (*fig.*21) for the case of non-uniform data partitioning. There the output classification on the test set is presented for three cases, the output of a network having been trained with the consensus framework, an

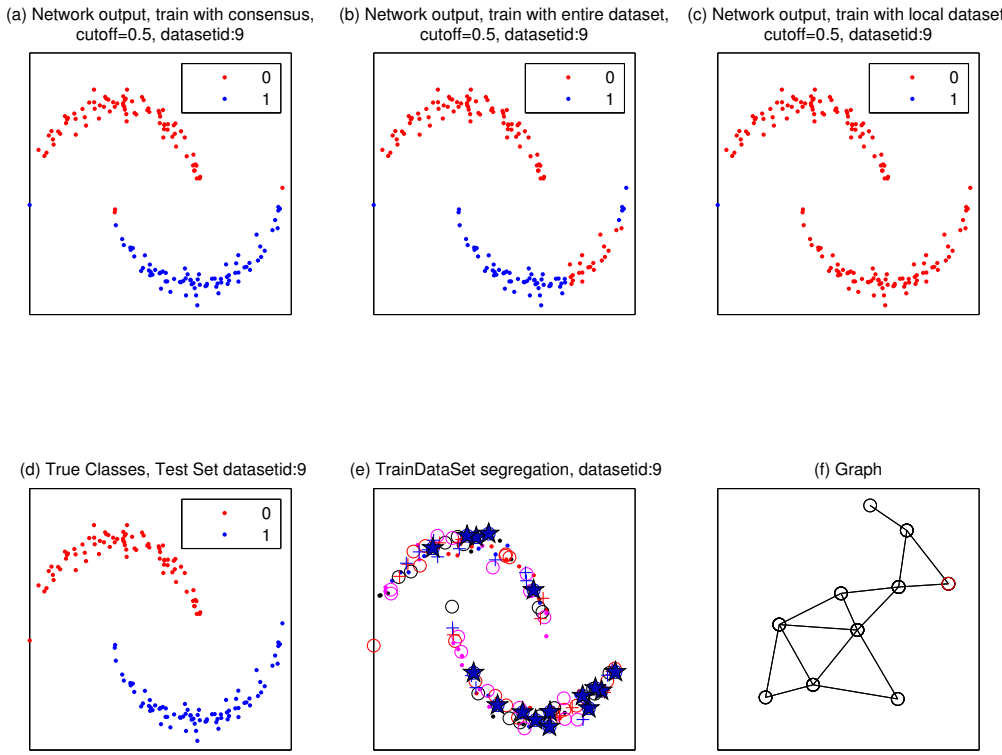


Figure 20: **Dataset and Classification output - Uniform case** (a) The output of the distributively trained network on the Test set. (b) The output of the non-distributively trained network. (c) The output by training with just the local dataset. (d) The true Test set. (e) The segregation of the training dataset. Each combination of colour and figure designates a different partition. The datapoints for the selected dataset are specified with a blue star. (f) The communication graph. The red circle designates the selected node for which the results are shown

identical network trained against the entire dataset, and an identical neural network trained only with the corresponding dataset. The latter totally fails because there are examples only from one class. Surprisingly, in this example, the distributed version demonstrates better classification in comparison with the centralised, which was trained on the entire set. This is a complex effect of the differential execution of the local learning algorithm and the consensus algorithm.

Overall results for the case of non-uniform data partitioning are summarised in (*fig.*22). There the mean performance at each epoch over all 50 datasets, measured on $\mathcal{T}$, is presented for different configurations of the algorithm. All

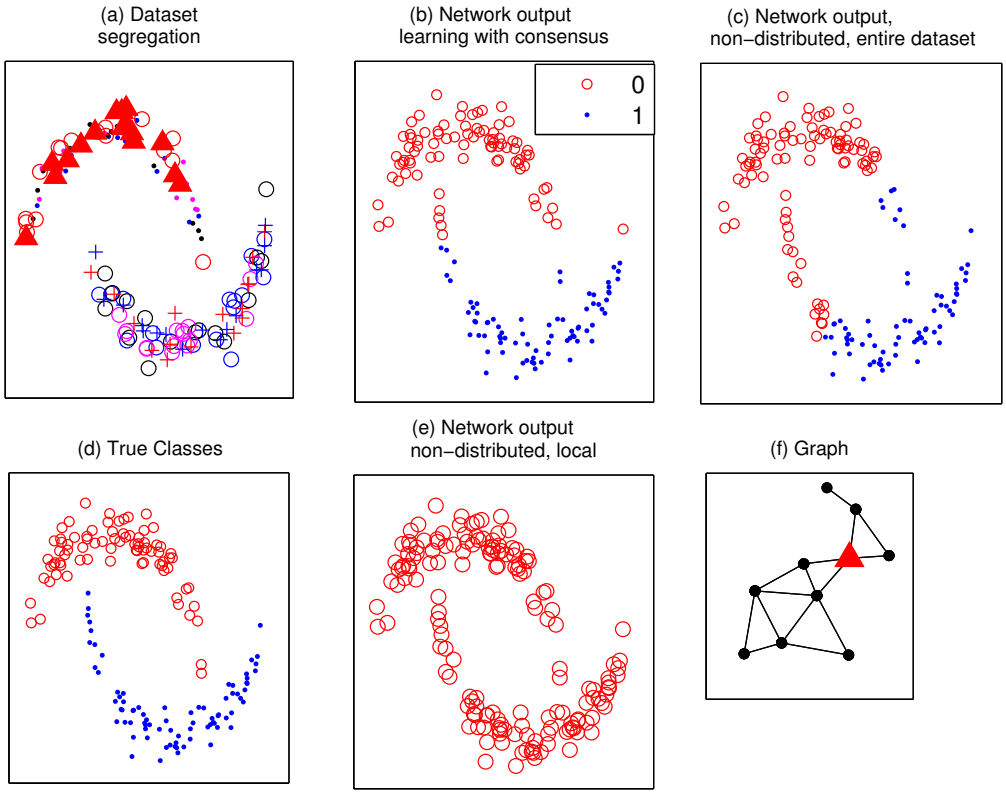


Figure 21: **Dataset and Classification output - Non-Uniform case.** **(a)**Training set segregation. Different colours and shapes designate points belonging to different machines. The red triangles designate the datapoints available at the selected machine. **(b)** The output of the neural network (NN) for a selected machine, trained with (*Algo.*5.3). **(c)**The output of the NN trained non-distributively on the entire dataset. **(d)**The true output. **(e)**The output of the NN using only the data available locally. **(f)**The connection graph. The red triangle designates the selected machine for which the network output is shown in (b).

the neural networks, local, global, and distributed had been initialised identically for each distinct dataset.

In this preliminary presentation of results, we observe that the distributed consensus learning algorithm with careful selection of the parameters can perform just as well as in the case that the entire dataset has been locally available. In specific cases, it might even outperform the centralised on the test set. Consider a neural network trained with the consensus learning framework on a uniformly partitioned dataset and an identical neural network trained over

the entire dataset. The second is expected to learn the dataset better. That is, to retrieve solutions such that performance on a test set is better. This in fact depends on the number of samples at each partition and the selection of the parameters of consensus learning. However, with an appropriate selection of learning parameters, the distributed case can surpass local learning, when the number of local samples is small.

We have selected an exemplary case, under uniform data partitioning, where consensus learning surpasses the classification on the test set of both local learning and centralised learning (*fig.*20). The different machines on the displayed communication graph had been allocated partitions of the dataset with examples from both classes. However, in the absence of regularisation, centralised learning does not manage to predict the classes of the test better than consensus learning. We point out that for the sake of comparison training has been performed with the same initialisation and training parameters in all networks and in all three cases of, consensus, non-distributed and local learning.

Under the non-uniform partitioning the results further designate the power of the proposed framework. An example is given in (*fig.*21) of the classification output on the test from the aforementioned neural network. In the case of non-uniform segregation, the partitions are not guaranteed to include examples from both classes. Instead, we have enforced that there are more than half of the partitions that contain examples from only one class. The latter further hardens the problem for both local learning and consensus learning.

However, from the given example we observe that local learning utterly fails. Something that is well expected. In contrast, consensus learning not only achieves to classify the examples of the test, but also attains better generalisation than the non-distributed case. However, we have to admit this is not always case. Nevertheless, this result demonstrates the potential of this method. As we have already illustrated theoretically, the algorithm's performance, and consequently its generalisation performance, depends on both the learning l and the consensus q iterations. We are going to further examine this interplay in (*Section* 5.5). There overall results are provided in order to give a better view of the average overall performance of the distributed algorithm.

A preamble of the results is presented subsequently, (*fig.*22). There the overall performance throughout the epochs of the consensus learning algorithm is displayed. Specifically, the average of the mean square error over all 50 datasets with random initialisation is computed at each epoch on the test set. The minimum empirical error is also presented along with the average base performance.

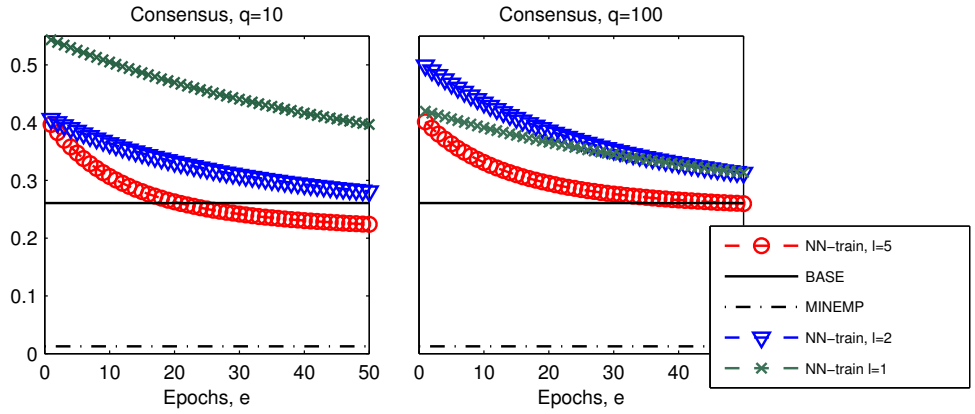


Figure 22: **Overall Classification Performance with Non-uniform Partitioning**. The average over all the neural networks mean squared error (MSE) on the test set, at each epoch, trained over all 50 datasets, is compared with the non-qdistributed counterpart. **Left**: Iterations for consensus was $q = 10$, **Right**: Iterations for consensus was $q = 100$. (MSE) was measured over the same unique test set. Different colour curves, red, blue, green, correspond to different number of local learning iterations, $l = 5$, $l = 2$, $l = 1$, respectively. **BASE** denotes the performance obtained by training an identical neural network with the entire dataset. MINEMP is the minimum empirical risk.

That is the average of the mean square error attained by the same neural network, which had been trained with the entire dataset and same initialisation as in the distributed version. We provide two plots, each for a different number of consensus iterations per epoch. At each of the two plots, there are three curves designating the overall performance for three different choices of the number of local learning iterations. One can observe the complicated nature of selecting the parameters. Observing the green lines on the left plot one may conclude that more local iterations are better. However, that is not the case when the iterations for consensus are increased, demonstrated in the right plot in (*fig.*22). These are further investigated in (*Section* 5.5).

Summarising, we draw the two following conclusions based on (*fig.*22). First, that neural networks trained with the proposed algorithm can perform distributively just as well as an identical neural network trained over the entire dataset. Second, that there is a tradeoff between number of iterations for learning locally and the iterations for consensus. However, on average it is favourable to have a large number of local learning iterations.

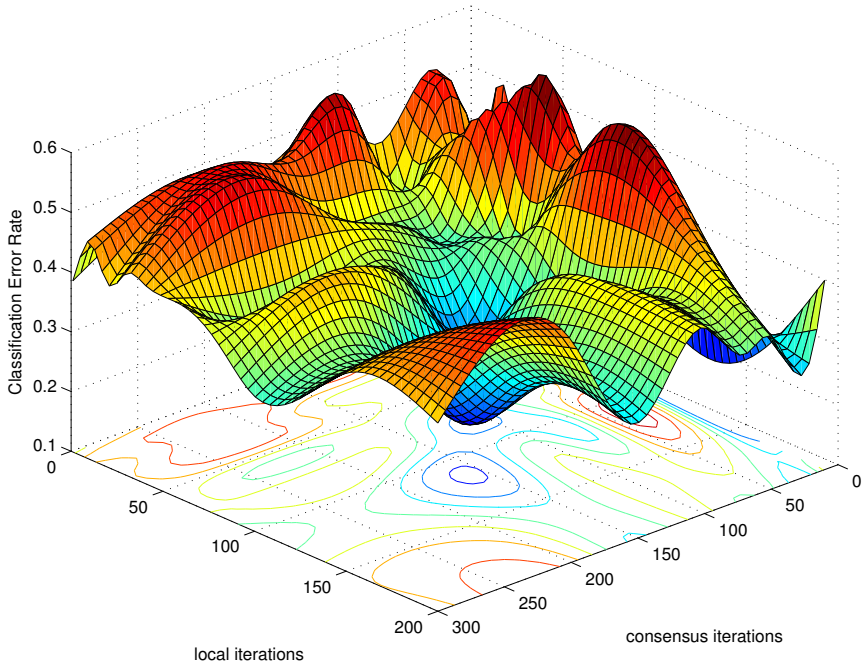


Figure 23: **Classification Error Rate - Uniform data partitioning**. The surface shown is a two dimensional cubic smoothing spline of the weighted averages of the datapoints at each configuration site. There is a region of local iterations between 100 and 150 and consensus iterations 120 to 200 where the algorithm performs best.

We have further performed experiments to determine the interplay of consensus iterations with learning iterations. The experiments have been performed in the following manner. We have fixed a communication graph, and have realized at each vertex the given neural network with 2 hidden layers, having 5 and 2 neurons at the first and second layer respectively, as in (*Section* 5.5.1.1). The reason for selecting this type of network was that it separates well the classes in the given dataset. We have selected 112 configurations {maximum local learning iterations, consensus iterations} in the range {1 -200,1-300}. Slightly different parameters have been employed to reduce computational effort. The parameters were 0.9 for the vertical displacement of the two semicircles, 0.3 for the horizontal, and white noise added on the y-axis was 0.07 .

Subsequently, for each configuration 50 different datasets had been generated, upon which the aforementioned neural network has been trained, both

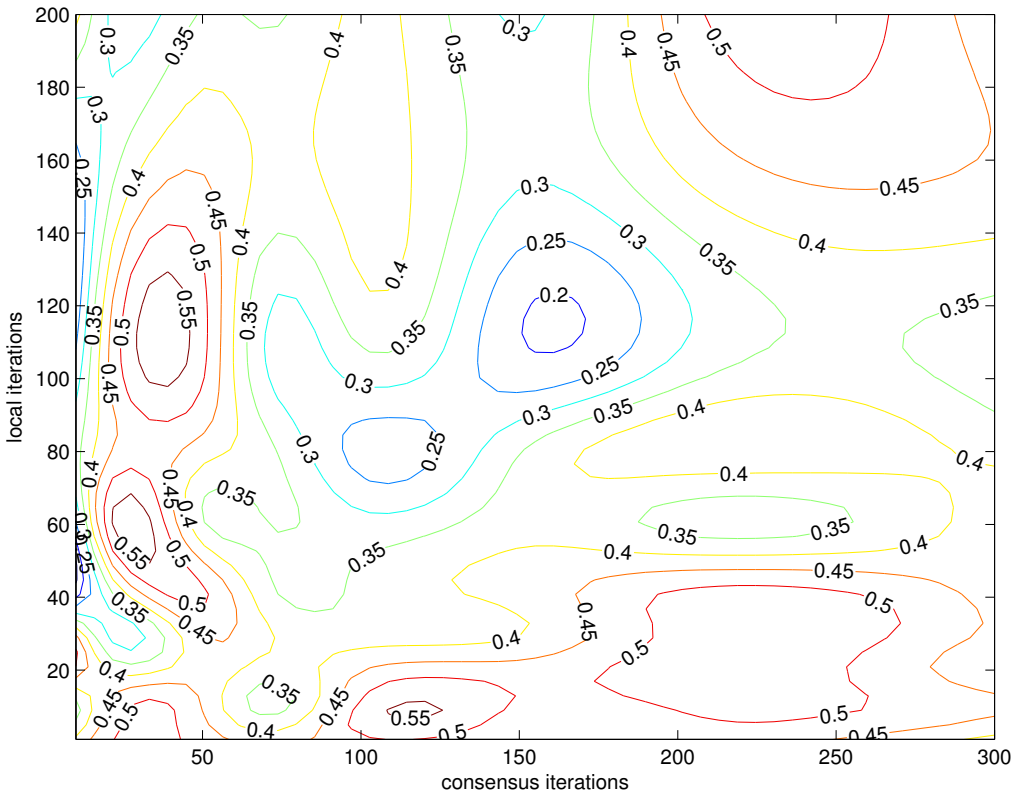


Figure 24: **Classification Error Rate - Uniform data partitioning - Contour plot**. As in (*fig.*23) one can observe in the contour plot the a region where the algorithm performs best.

distributively and non-distributively. Each of the 50 resulting networks has been tested on a unique predefined test set, generated in the same manner as the training set. Thus, for each configuration we have obtained 50 datapoints, each corresponding to the final classification error rate, totalling a dataset of 112 by 50.

For each configuration we have selected the datapoints where the non-distributed version of the algorithm had a classification error rate below 0.2. This is justified since initialisation has an important effect on the learned model, thus we cannot know the proper initialisation a priori. Hence, it is appropriate to select those models that fit the data to compare the performance of the two cases. Inclusion of the cases that even the non-distributed algorithm has not performed is pointless. On the resulting datapoints, a weighted average cubic smoothing spline was fitted appropriately for all the tested configurations, in order facilitate visual

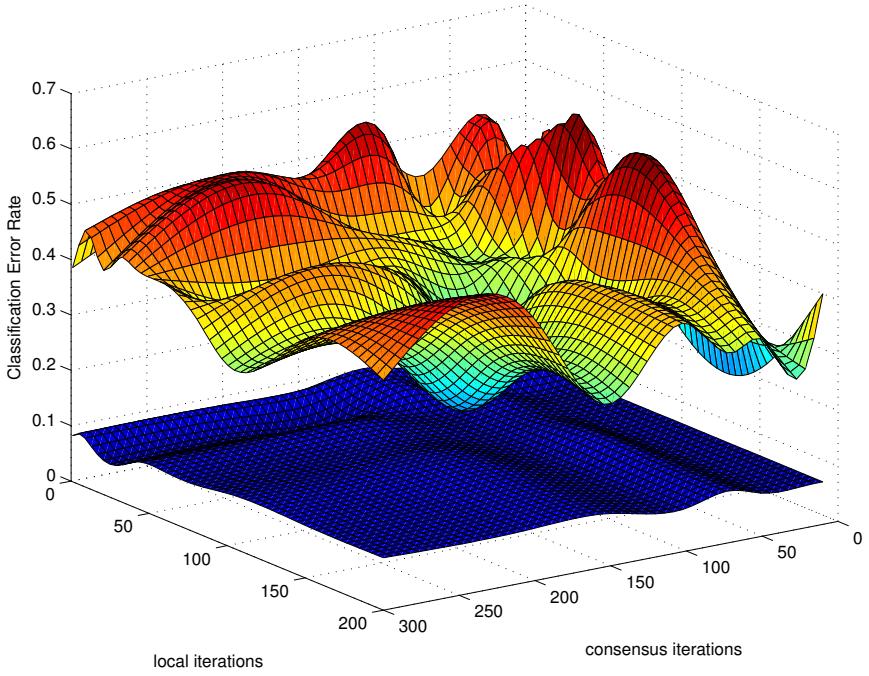


Figure 25: **Classification Error Rate - Comparison with non-distributed - Uniform data partitioning**. As in (*fig.*23), the same surface is plotted versus a two dimensional cubic smoothing spline fitted on the weighted average classification error rate when training non-distributively (bottom surface). The minimum difference is about 0.05 .

representation of the results. Detailed results are presented in (*Section* 5.5.2.2) and (*Section* 5.5.2.1).

One should notice in all figures that there is a region on the plane, corresponding to a group of configurations where the distributed version attains on average results near the non-distributed counterpart. Let us emphasise that these results are on average. This implies that there are cases where the distributed algorithm is better than the non-distributed and vice versa. Therefore, no safe conclusions can be made about the performance of the distributed algorithm being better or worse in comparison to having the entire dataset on just one site. However, what is illustrated from these results is that the distributed learning algorithm by consensus performs nearly as good on average with the non-distributed counterpart. Nevertheless, the main comparison, is not to be made with the non-distributed having the entire dataset, but with the local having only one partition of the dataset. In that case the distributed algorithm performs much

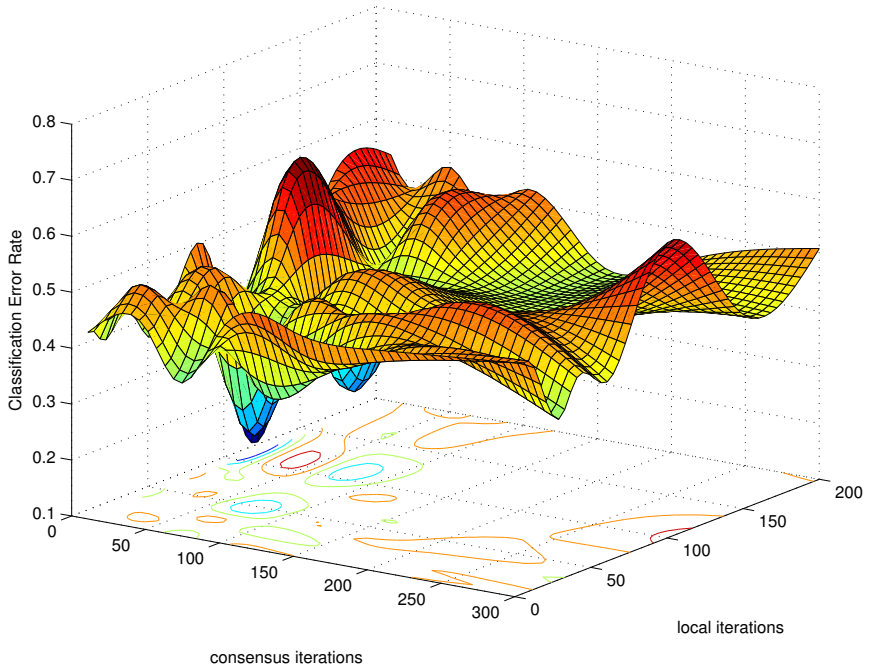


Figure 26: **Classification Error Rate - Non-uniform data partitioning**. The surface shown is a two dimensional cubic smoothing spline of the weighted averages of the datapoints at each configuration site. There are two a regions where the algorithm performs best. These are {80-100,50-100} and {80-100, 40-80}, local and consensus iterations respectively.

better. Nonetheless, the distributed performs sufficiently close even when compared with the non-distributed counterpart, which bears significance for many applications.

5.5.2.1 *Classification Error Rate - Uniform*

In the case of uniform data partitioning, we observe that the classification is equivalent to random assignment for the greater region of the plane. However, there are two minima where the error rate is about 0.2 . That is when consensus iterations are in the neighbourhood of 130 and the local iterations are in the region of 120. We further observe the formation of two valleys on the classification error rate surface. These occur when either of the local or consensus iterations are held constant while the other quantity is increased. There, the classification error rate is in the region of 0.3 .

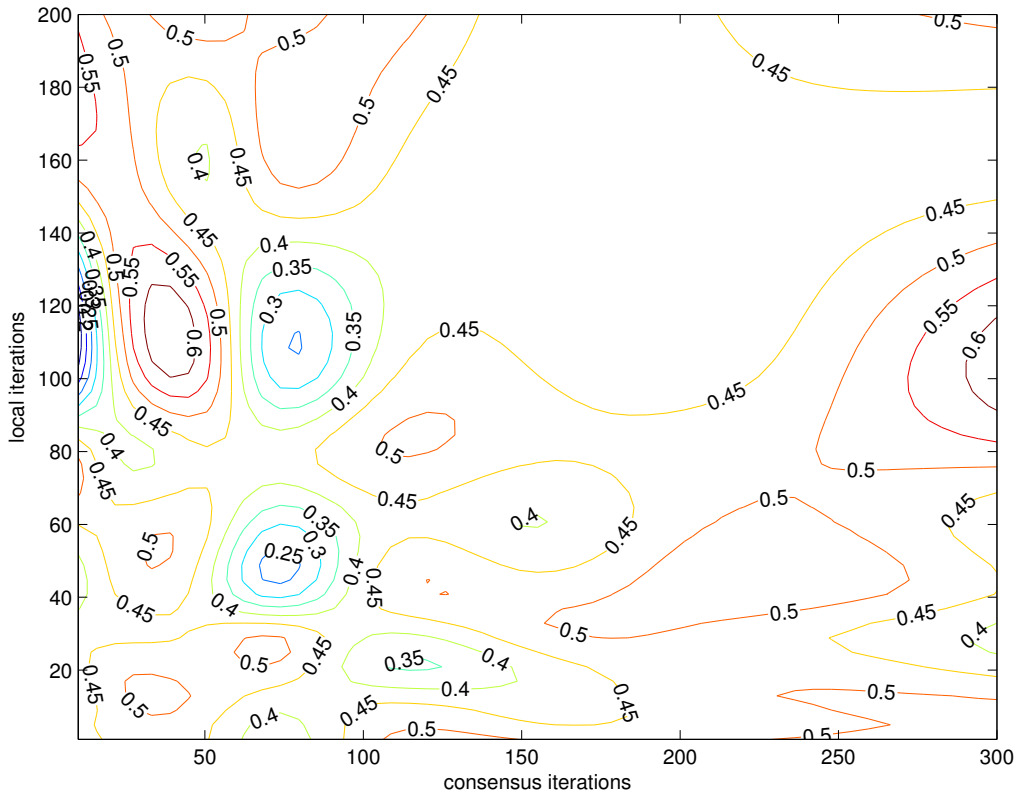


Figure 27: **Classification Error Rate - Non-uniform data partitioning - Contour plot**. As in (*fig.*26) the regions where the algorithm performs best, {80-100,50-100} and {80-100, 40-80} can be adduced.

That further verifies our claim about the interplay of the two parameters. Originally, we had found theoretically that selecting $q > Q$ is not sufficient to model the data successfully. The latter is only sufficient for the local convergence of the algorithm. This is verified here as well, due to the presence of the valley which forms when increasing the iterations for consensus. There, the classification error rate is much better than random selection, between 0.2 and 0.3, but still large. These are also observable in more detail in the contour map (*fig.*24).

The comparison with the classification error rate of the local learning algorithm is given in (*fig.*25). There only learning iterations and the initialisation is important. These have been identical with the case of the non-distributed counterpart . The latter's performance is depicted in the bottom surface. Particularly, different initialisation has been used at each experiment. Hence, each pair of

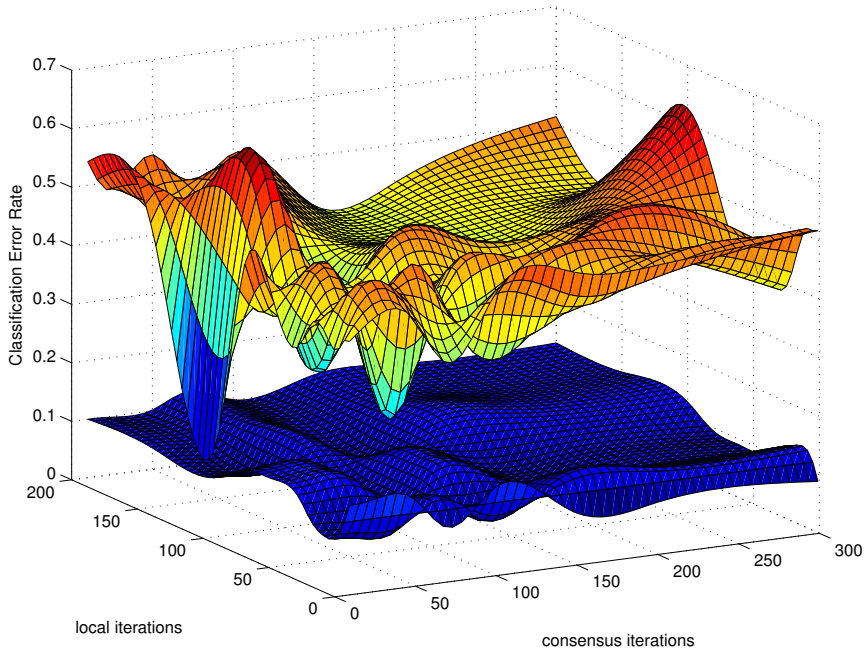


Figure 28: **Classification Error Rate - Comparison with non-distributed - Non-uniform data partitioning**. As in (*fig.*23), the same surface is plotted versus the classification error rate achieved on average when training non-distributively bottom surface. The minimum difference is about 0.1 .

distributed and non-distributed has been using the same initialisation at each experiment but different between experiments. This causes a slight fluctuation on the surface at the bottom, corresponding to the non-distributed, when the consensus iterations vary.

That is somehow counter intuitive but does not alter the validity of these numerical experiments. This was treated by running experiments at each configuration on multiple datasets. Retrieving the average of the performance, (RMSE) or (CER), provides a concise view at each configuration point. Hence, it allows to draw safe conclusions about the interplay of the parameters and the comparison with the centralised counterpart.

The centralised algorithm performs better overall, given that the entire dataset is available locally. In contrast when the latter cannot be, the distributed learning algorithm, presents an interesting alternative. Appropriate selection of the parameters allows to obtain results that are adequately close to having the entire dataset available on a single machine.

5.5.2.2 *Classification Error Rate - Non-Uniform*

We have built the figures as presented in (*Section* 5.5.2.1) for the case that the data is not uniformly allocated over the machines. The classification error rate surface is positioned near higher values in comparison to the uniform case. Furthermore, the valleys, that we had observed in the case of uniform partitioning, have vanished. Instead, there are only two steep regions where the value is about 0.25. The most prominent of the two is found in the region around 60 local and 60 consensus iterations. There the distributed algorithm has its best classification error rate on average. However, in the larger region, the convergence of the algorithm is such that the learned model of the data completely fails to classify properly the examples of the test set. Furthermore, the classification error is verified to be more sensitive with respect to the selection of the parameters than in the case of uniform segregation. Details are better depicted on the contour plot (*fig.*27).

Additionally, a comparison with the non-distributed data learning algorithm is given in (*fig.*28). There, one should notice that there is a region that the classification error rate which is near the one associated to the non-distributed counterpart. Obviously, partitioning of the dataset in any manner does not affect the training that is performed with the entire dataset. Hence, the blue surface below is in fact an idealised comparison. The fluctuation when varying the consensus iterations is both, an effect of the different initialisation at each configuration site, and an artifact of fitting with a two dimensional cubic smoothing spline. The conclusion from (*fig.*28) is that even if data partitioning has not been performed in a favourable manner, then the consensus learning framework may still attain good performance near the non-distributed. However, this case is more difficult than the case of uniform partitioning.

5.5.3 *Definitive Consensus Machine Learning with Regularisation*

The study in the case of the definitive consensus algorithm is simpler. We employ regularisation on the risk function of the neural network directly. We use the same dataset and setup as has already been described in (*Section* 5.5.1.1). However, we have selected to change the communication graph. The graph that we use has a somehow irregular form. This would pose some difficulties for the original consensus algorithm. The less connected vertices should present differences from the value of agreement. However, employing the definitive

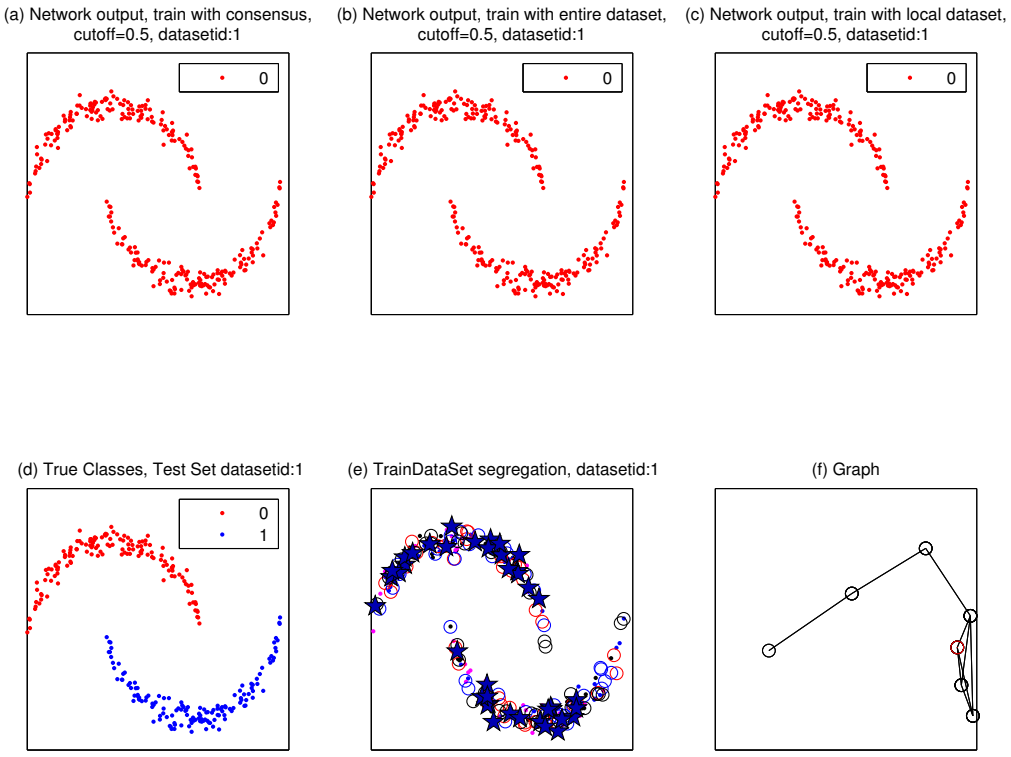


Figure 29: **Classification with little regularisation $\zeta = 0.9$ - Uniform partitioning**. Classification on the test set is shown, for the distributed and non-distributed case, trained on the 9^{th} of the 50 datasets. Little regularisation may result in misclassification in all cases. Details: (a) trained with $\mathcal{DCL}$, (b) trained on the entire dataset $\mathcal{L}$, (c) trained with the subset, (d) the true classes, (e) the partitioning of the training set. Blue stars indicate the subset. (g) The graph, a circle designates the specific machine.

consensus algorithm should alleviate such problems. Furthermore, we present the results on different random graphs to demonstrate that the outcome is independent of the network topology.

As has already been mentioned, the definitive consensus algorithm enables the utilisation of a regularisation function on the empirical risk function. We have used the same regularisation function for both distributed and the non-distributed for the training of the same neural network. We are going to present the results for both uniform and non-uniform partitioning for the given dataset. Surprisingly, we shall observe that in contrast to what we would expect the distributed and the non-distributed do not always perform similarly. This is due

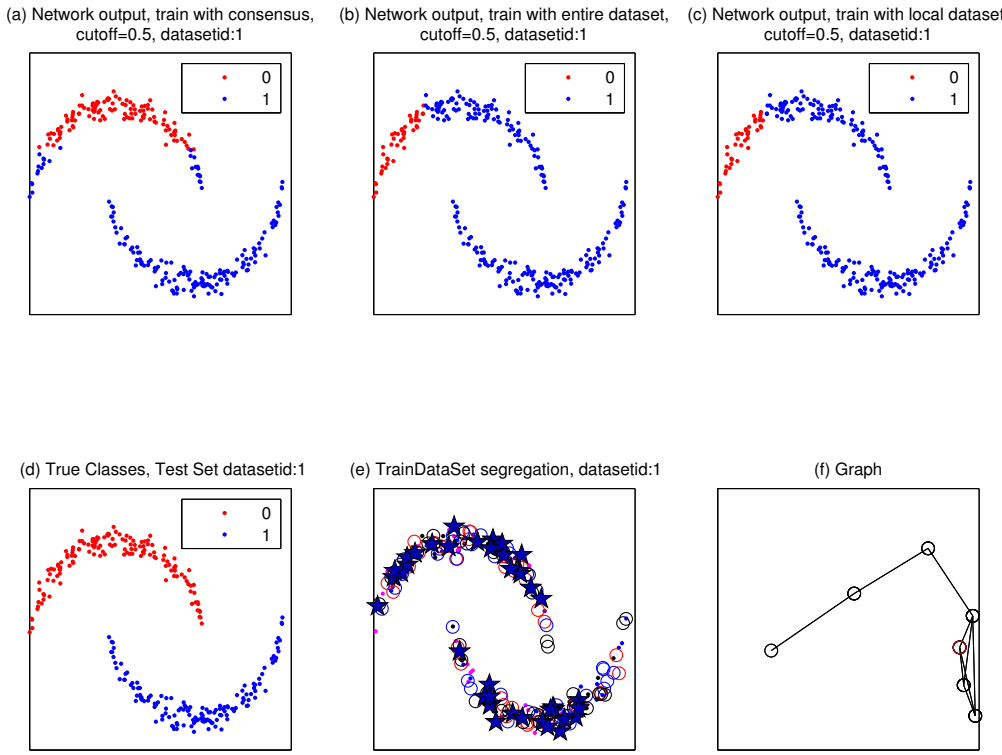


Figure 30: **Classification with regularisation $\zeta = 0.5$ - Uniform partitioning**. Classification on the test set is shown, for the distributed and non-distributed case, trained on the 9^{th} of the 50 datasets. Surprisingly the distributed has better generalisation than the rest. Details: (a) trained with $\mathcal{DCL}$, (b) trained on the entire dataset $\mathcal{L}$, (c) trained with the subset, (d) the true classes, (e) the partitioning of the training set. Blue stars indicate the subset. (g) The graph, a circle designates the specific machine.

to the effect of the local learning algorithm being executed for more than one iteration.

As a regularisation function in this sequence of experiments, we have used the mean of the square of the learning parameters, i.e. the neural network weights. The empirical risk is modified to include this as follows

$$r_k = \mathcal{R}({}^k\mathcal{D}, f_k) + (1-\zeta)\frac{1}{m_a}\sum_{j=1}^{m_a}[a_j]_k^2 \tag{5.43}$$

where m_a is the number weights on the entire neural network, and j is some indexing of the weights throughout the entire neural network. This representation is simpler than having to include an index for layers and neuron to neuron just

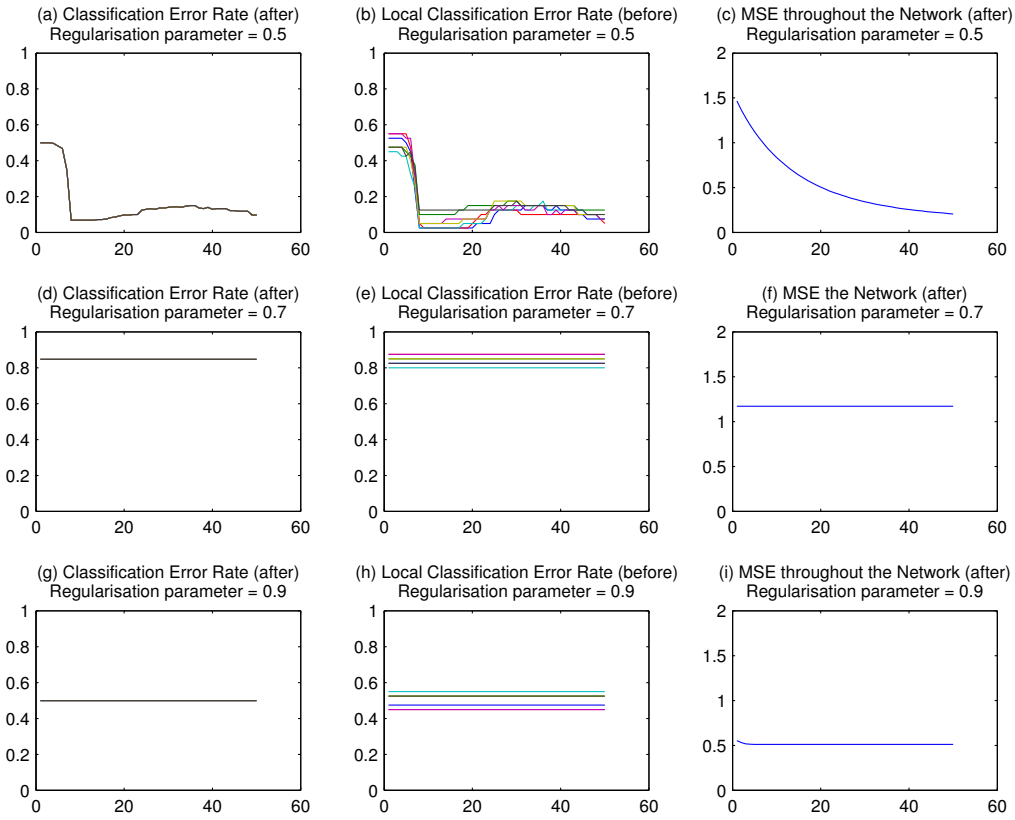


Figure 31: **Acceptable performance on the Test set during training - Uniform partitioning**. The effect of regularisation diminishes from top $\zeta = 0.5$ to bottom $\zeta = 0.9$. On the given dataset and initialisation, only the first case is behaving in a satisfactory manner. Details: (a), (d) and (g) display (CER) after consensus has been executed at each iteration. (b), (e) and (h) exhibit (CER) before consensus has been executed. (c), (f) and (i) display the (MSE) after consensus.

to describe this simple summation over the squares of all learning parameters. The outer index k indicates the machine.

5.5.3.1 *Uniform*

In our sequence of experiments, we range ζ from 0.5 to 0.9 in order to determine the significance of regularisation in the distributed learning under the definitive consensus algorithm. Results for the same dataset are given in (*fig*.30) and (*fig*.29) for $\zeta = 0.5$ and $\zeta = 0.9$, respectively.

There the classification output on the test set is given for three cases. The first sub-figure (a), in (*fig*.30) and (*fig*.29), exhibits the output for a specific machine

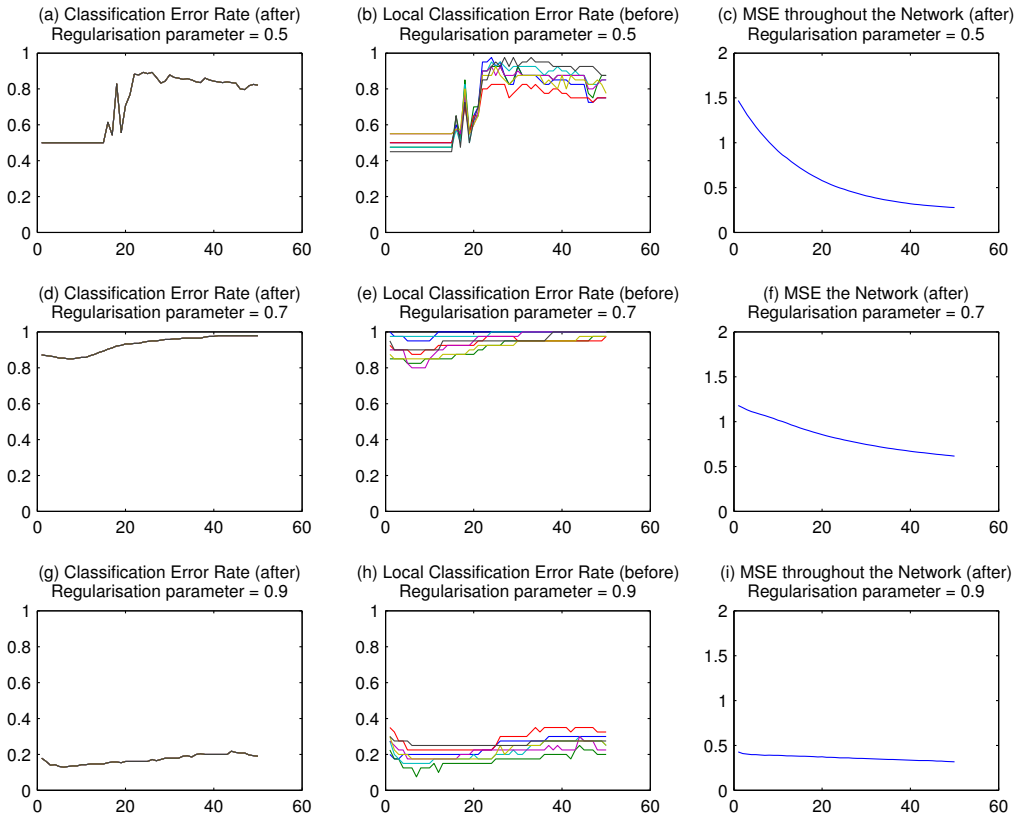


Figure 32: **Not acceptable performance on the Test set during training - Uniform partitioning**. The effect of regularisation diminishes from top row $\zeta = 0.5$ to bottom $\zeta = 0.9$. Here, the classification error rate is not acceptable even though the (MSE) diminishes throughout training. Details: (a), (d) and (g) display (CER) after consensus has been executed at each iteration. (b), (e) and (h) exhibit (CER) before consensus has been executed. (c), (f) and (i) display the (MSE) after consensus.

in the communication graph after having trained with the consensus machine learning algorithm. The specific machine is designated on the communication graph (f) with a red circle. Second, the output of an identical neural network that has been trained with the entire dataset is displayed in (b). Third, the output of the same network which was trained only on the local subset of the designated machine is displayed in (c). A similar layout is followed in (*fig*.36) and (*fig*.37), as well.

Specifically, in (*fig*.30) and (*fig*.29) is demonstrated that without regularisation, then generalisation, i.e. classification on the test set, may fail in all three learning settings. This cannot be overcome with early stopping, because we

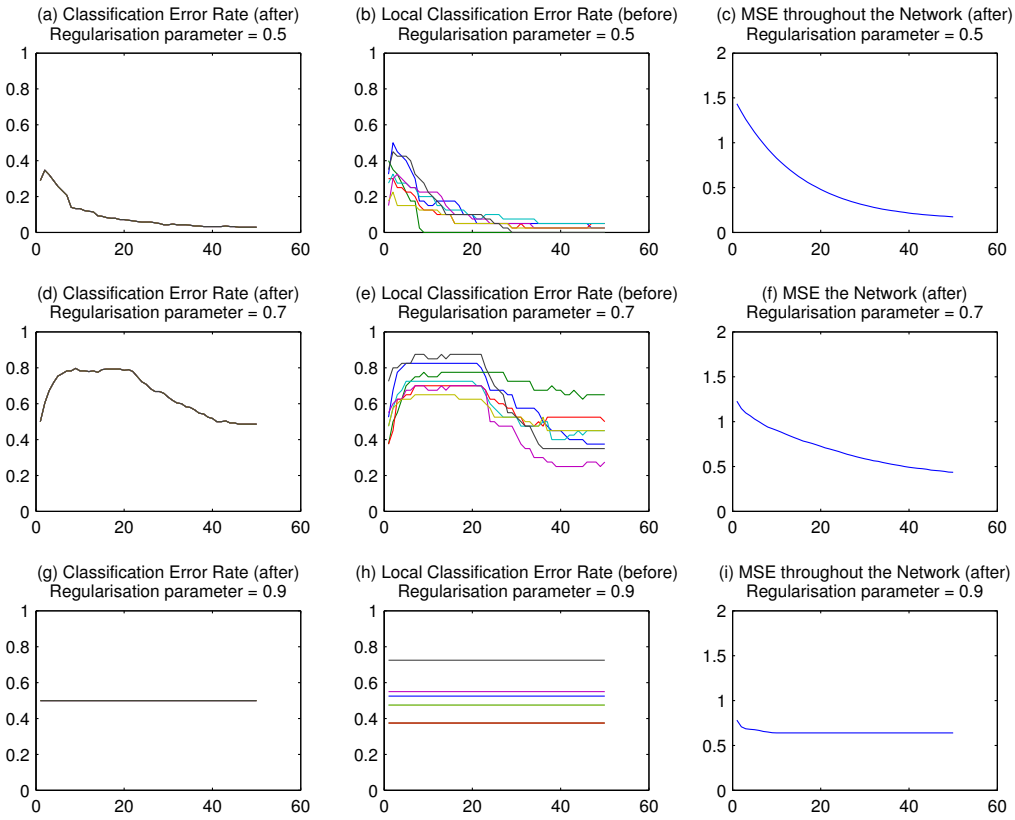


Figure 33: **Classification Error Rate of a network with 2 hidden layers - Non-uniform data segregation - Contour plot**. The effect of regularisation diminishes from top row $\zeta = 0.5$ to bottom $\zeta = 0.9$. Here, very good classification error rate on the test set is attained only for the first case $\zeta = 0.5$. Details: (a), (d) and (g) display (CER) after consensus has been executed at each iteration. (b), (e) and (h) exhibit (CER) before consensus has been executed. (c), (f) and (i) display the (MSE) after consensus.

track the mean squared error instead of the classification error rate on the test set. Unfortunately, the classification error rate is not a smooth function, rendering it unusable throughout the training phase of the neural network. This can be better examined in (*fig.*32).

In (*fig.*30) with $\zeta = 0.5$, the definitive consensus appears to generalise better than the rest. However, this is not always the case. In most cases, definitive consensus learning performs just as the centralised learning and always better than learning with the local partition. The results of (*fig.*30) are controversial. One would not expect to observe better generalisation under the operation of the definitive consensus learning algorithm than in the case of the non-distributed

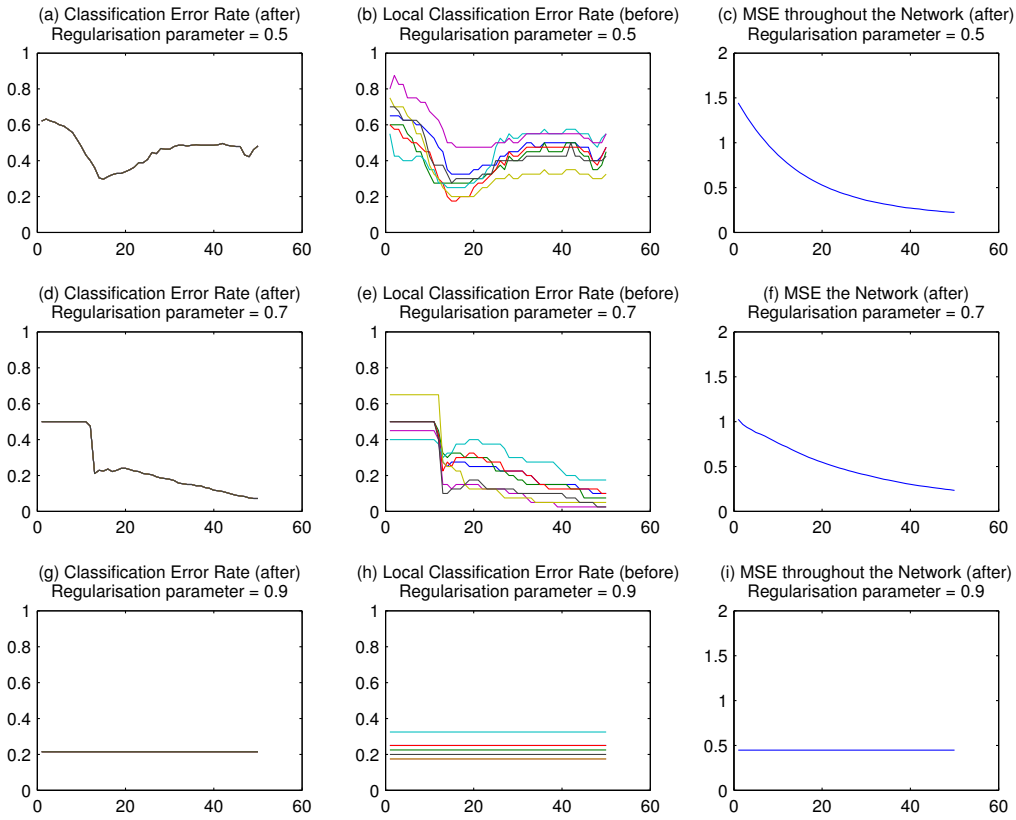


Figure 34: **Good performance on the Test set during training - Uniform partitioning**. The effect of regularisation diminishes from top row $\zeta = 0.5$ to bottom $\zeta = 0.9$. Here, very good classification error rate on the test set is attained for the first two cases. Details: (a), (d) and (g) display (CER) after consensus has been executed at each iteration. (b), (e) and (h) exhibit (CER) before consensus has been executed. (c), (f) and (i) display the (MSE) after consensus.

counterpart. Examination of the update equations for the neural network shows that the learning parameter updates are identical to those obtained by performing the updates centrally. However, that holds in the case that the local learning phase is not executed differentially from consensus; hence in one learning iteration is executed locally. Therefore, the path followed on the parameter space is different for each of the two cases of definitive consensus learning; that is with one or multiple local learning iterations before consensus. The first case, according to the proof of (*Theorem* 5.2) is identical with learning centrally with the entire dataset. However, the other case, for which we have

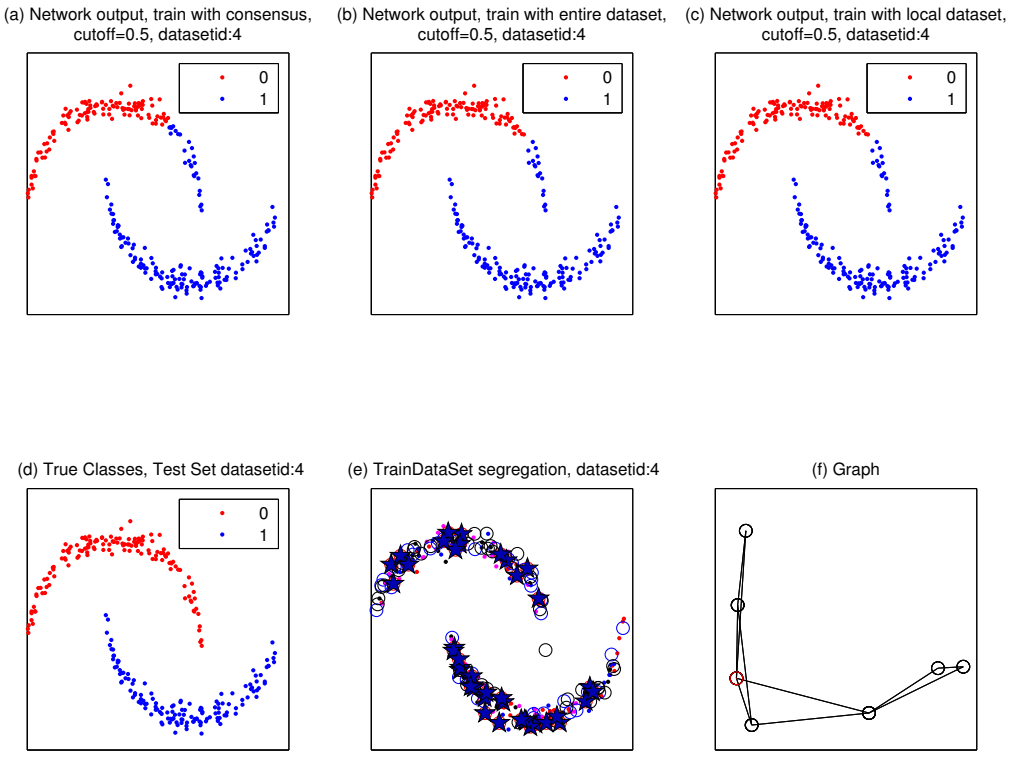


Figure 35: **Classification without differential learning and regularisation $\zeta = 0.8$ - Uniform partitioning**. Exactly the same results are obtained for both the centralised and distributed, when we the local learning phase consists of only one iteration of the learning algorithm. The number of epochs has been increased to 200. Details: (a) trained with $\mathcal{DCL}$, (b) trained on the entire dataset $\mathcal{L}$, (c) trained with the subset, (d) the true classes, (e) the partitioning of the training set. Blue stars indicate the subset. (g) The graph, a circle designates the specific machine.

noticed the controversy, is not identical, which in turn justifies the difference in the generalisation ability.

Specifically, in these experiments the local neural networks have been allowed to execute up to 100 iterations per epoch of the distributed algorithm. Thus, it becomes evident that the local parameter updates are not identical in practice with those of the centralised. The local learning algorithm being executed differentially permits the local neural networks to perform some kind of local over-fitting of the data. Thus the global learning process does not follow the same path on the parameter space as if it had been executed centrally with the entire dataset. Moreover, executing the consensus algorithm for a finite number

of iterations can have a complex effect, which has already been examined in (*Section* 5.5.2). These conclusions can be verified by examining (*fig.*35), where the local learning phase was executed for only one iteration.

We also present here some results that relate to training the neural networks. In (*fig.*31), (*fig.*32), (*fig.*33), and (*fig.*34), examples of acceptable, bad, good, and best cases of training are demonstrated, respectively. Let us remind that these results are always evaluations on the test set at each epoch of the algorithm.

In the first case (*fig.*31) the classification error rate is given for three different configurations of ζ. In the second column, the local classification error rate is presented throughout the epochs of the algorithm. In the last column, the mean squared error on the validation set has been included. The mean squared error is not always in accordance with the classification error rate. Moreover, we see that the local updates after the learning and before the consensus phase provide different classification error rates on the test set. This can be observed in the central column of (*fig.*31) but also in the rest of the figures as well. This is another effect of the differential execution of the local learning phase. There, the local neural networks overfit the local partition of the data but are brought into agreement afterwards by consensus.

The latter can be verified by observing (*fig.*35). There we have imposed only one iteration for the local learning phase. Then the update of the algorithm is exactly as described in (*eq.*5.39). It is evident that the results are near identical. Minor discrepancies can be justified from the fact that the precision of updates determined with the definitive consensus algorithm are obtained with less precision than those obtained through direct updates. The precision for the updates of the definitive consensus and the direct updates is 10^{-12} and 10^{-16} respectively, in the given sequence of experiments.

5.5.3.2 *Non-Uniform*

We include two exemplary results for $\mathcal{DCL}$ in the case of non-uniform data partitioning. These illustrate the potential of the distributed learning algorithm with definitive consensus. Regularisation, is still a matter of concern. Too little regularisation may cause the system to over-fit the data (*fig.*36), whereas if selected appropriately the result can be as good as having learned from the entire dataset (*fig.*37). In contrast to consensus learning $\mathcal{ECL}$, the definitive consensus $\mathcal{DCL}$, allows direct control over the regularisation of the data.

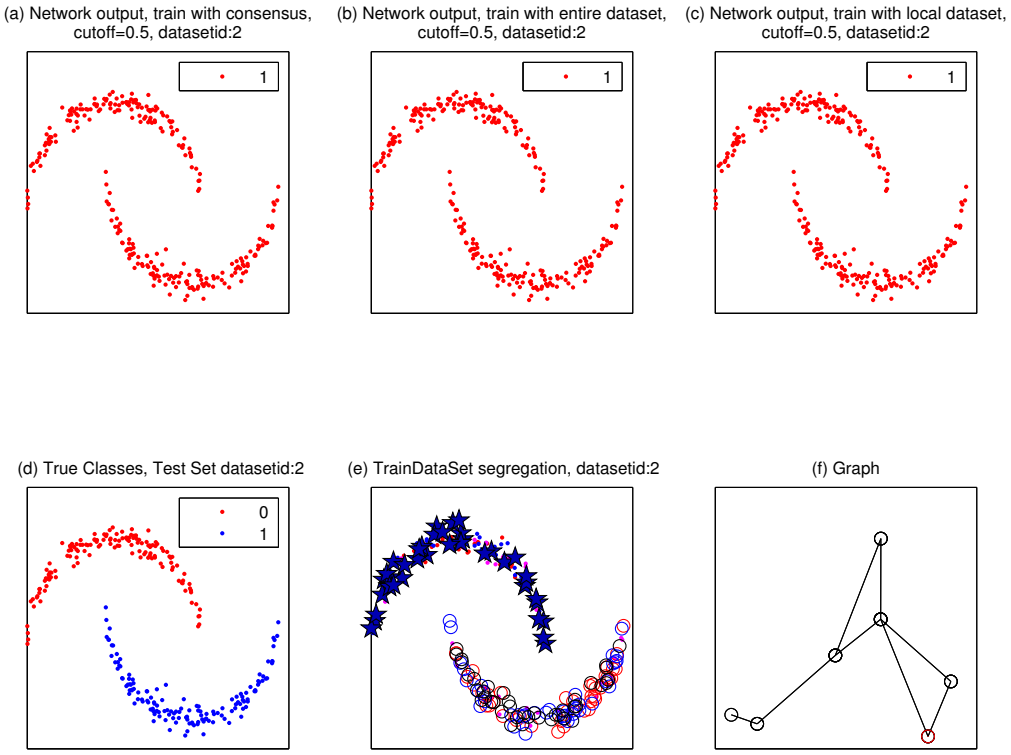


Figure 36: **Classification with little regularisation $\zeta = 0.9$ - Non-uniform partitioning**. Classification on the test set is shown, for the distributed and non-distributed case, trained on the 2^{nd} out of 50 datasets. Too much regularisation obstructs learning. Details: (a) trained with $\mathcal{DCL}$, (b) trained on the entire dataset $\mathcal{L}$, (c) trained with the subset, (d) the true classes, (e) the partitioning of the training set. Blue stars indicate the subset. (g) The graph, a circle designates the specific machine.

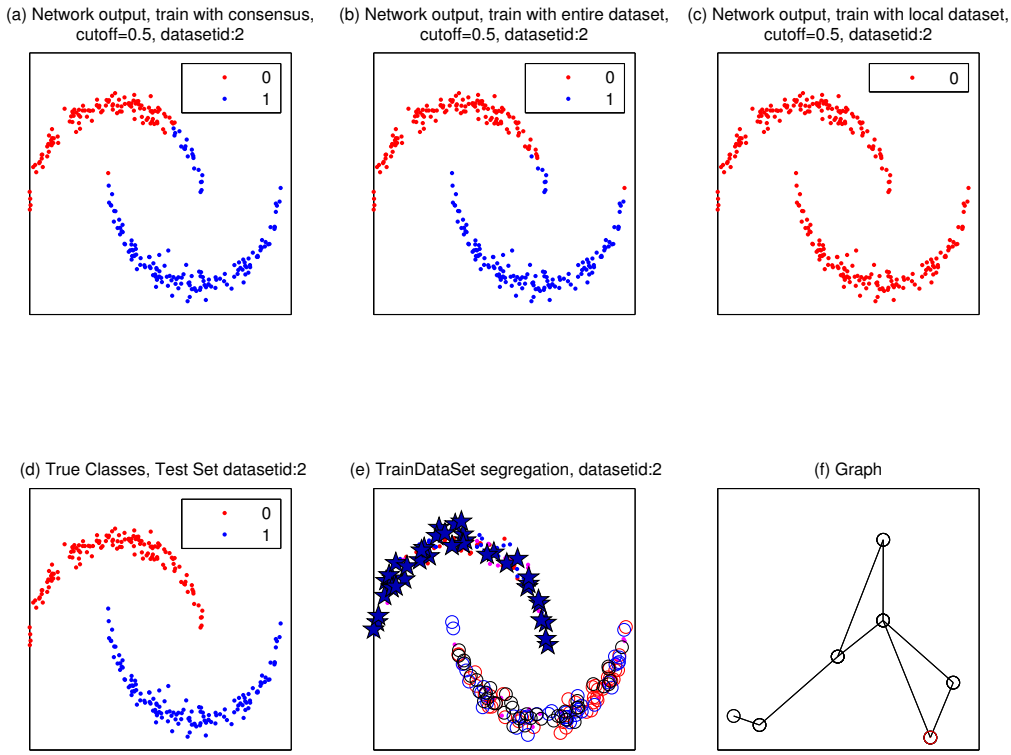


Figure 37: **Classification with regularisation $\zeta = 0.5$ - Non-uniform partitioning**. Classification on the test set is shown, for the distributed and non-distributed case, trained on the 2^{nd} of the 50 datasets. Local learning (c) recognises only class 0 which does not generalise well. The distributed (a) performs as well as the centralised (b). Details: (a) trained with $\mathcal{DCL}$, (b) trained on the entire dataset $\mathcal{L}$, (c) trained with the subset, (d) the true classes, (e) the partitioning of the training set. Blue stars indicate the subset. (g) The graph, a circle designates the specific machine.

We have presented a theoretical framework for training a learning machine distributively by incorporation of the consensus algorithms. We have proven its convergence under mild assumptions; these are realistic in many applications. Furthermore, this framework has been specified for a well understood learning algorithm, the multi-layer feed-forward neural network with back-propagation. Hence, the framework for distributed inference has been well founded and initially verified.

Furthermore, we have performed extensive numerical tests; additional results are included in (*Chapter* 6). Herein, the algorithm was tested with a well-known, simple, but not separable dataset under uniform and non-uniform data partitioning.

Importantly, during the distributed process there is no exchange of data, only the learned quantities are shared between neighbouring machines. Moreover, the effort required at each machine is reduced in comparison to having to process the entire dataset. These are favourable in many applications due to communication, energy, other costs, communication, and privacy considerations. Consequently, this designates that this method may be appropriate for a large number of applications.

Part III

DISCUSSION

6

NUMERICAL VALIDATION

Our purpose in this chapter is to provide further evidence of the work presented in (*Chapter* 3), (*Chapter* 4), and (*Chapter* 5). In subsequent sections, the methodology of the numerical experiments is detailed for each of these three cases. Moreover, exemplary results are provided. Further figures of the results are found in (*Appendix* B).

6.1 DYNAMIC CONSENSUS PERFORMANCE

Herein we attempt further verification of the analysis performed on the algorithms in (*Chapter* 3) by additional numerical simulations. In contrast to the results already presented, we attempt to compare the overall performance for the consensus algorithm, the nonlinear and the adaptive under the schemes of non-failing and failing links. Various link fail probabilities have also been used. However, it is understandable that this work can only be indicative. Since, we can perform experiments up to some scale whereas the field of application is much larger. In fact it is infinitely countable, since one has to consider all graphs. Therefore, the outcome of this comparison may be challenged with experiments in a larger domain.

We have performed a series of experiments on the following combinations of consensus algorithms and weighting schemes. These are

1. Convex: The consensus algorithm (*Algo.*2.1) with weights assigned by convex optimisation, as in (*Section* 2.3.5.4).
2. MaxDegree: The consensus algorithm with the Maximum Degree weights, (*Section* 2.3.5.1).
3. Metropolis: The consensus algorithm with the Metropolis-Hastings weights, (*Section* 2.3.5.2).
4. The nonlinear consensus algorithm (*Section* 3.4) with weights assigned by convex optimisation and parameters $k = \theta = 1$

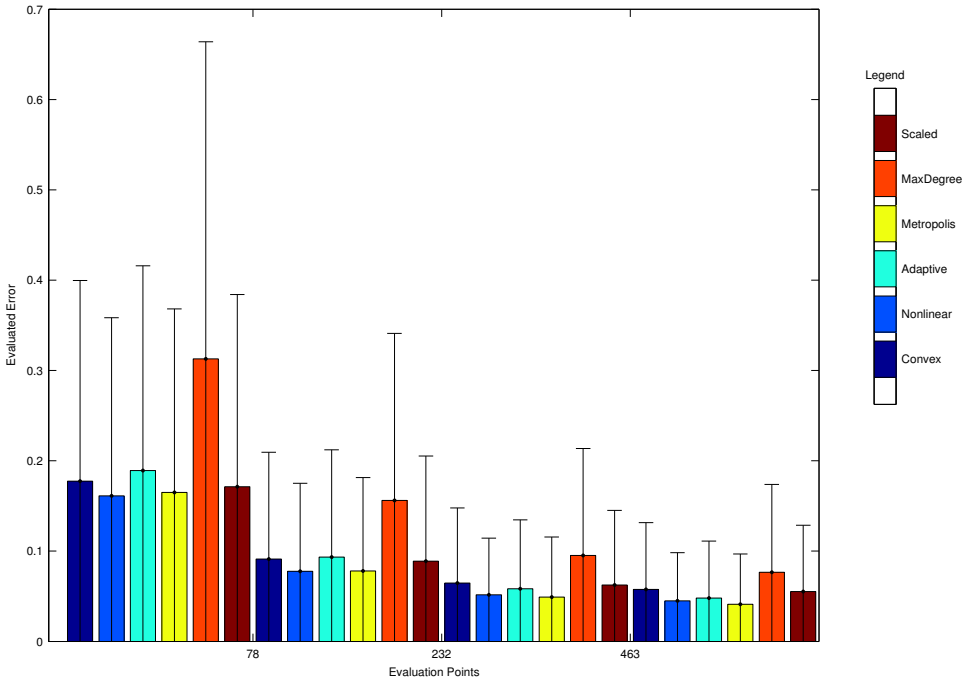


Figure 38: **Small graphs overall performance**. The performance measure E_3 is given in the intervals [0 78], [0 232], [0 463], [0 700]. Each colour designates one of the algorithm-weight scheme configurations. The labels Convex, metropolis, maxdegree, and scaled designate the performance of the consensus algorithm under the corresponding weighting schemes. The labels adaptive and nonlinear designate the corresponding consensus algorithms with convex weighting schemes.

5. The adaptive nonlinear consensus algorithm (*Section* 3.5) on an un-weighted graph with parameters $k = \theta = 1$.

6. The scaled weight matrix is $\mathbf{W} = \mathbf{I} - \varepsilon\mathbf{L}$ and ε can be retrieved by one parameter minimisation of $\varepsilon = \arg\min_{\varepsilon} \|\mathbf{W} - \mathbf{1}\mathbf{1}^T/n\|$ subject to $\mathbf{W}\mathbf{1} = \mathbf{1}$ and $\mathbf{W} = \mathbf{W}^T$ as in the convex optimisation scheme.

For each experiment an identical process has been followed. A random graph is created by randomly placing points on a plane, sized $[-1, 1]$ on both axes, until we obtain a connected graph, for a radius of interaction determined by the number of vertices and edges according to a predefined threshold; in our case we have used $\frac{|\mathcal{E}|}{8|\mathcal{V}|^{1/2}(|\mathcal{V}|+2)}$. The vertices are considered connected when they are found on the grid to be closer than that. Subsequently, a number of initial

states is sampled from a normal distribution with predefined standard deviation and mean. For each of the initial states each of the algorithms has been executed and their performance measured at specific intervals and at specific iterations.

The same type of experiments is executed for stochastic link failures. The model for communication has been described in (*Section* 2.3.4). In that case we have generated sequences of link failures and each algorithm is tested on the same multiple of sequences. Thus, each algorithm is executed multiple times for each of the generated initial states.

generated graph details

We have generated 20 different random graphs for each of two different vertex sizes. The first had 20 vertices and a number of edges ranges from 40 to 50. The second had 100 vertices and the number of edges ranged from 250 to 400. Then 10 initial states for each vertex were sampled from a normal distribution. Particularly, the sampling distribution had standard deviation 0.2 , 0.5, 1, and a mean of 10. For the stochastic link failures experiments, we generated 10 sequences of link failures with edge alive probabilities 0.2, 0.7 and 0.95 . The algorithms have been allowed to execute for a limited number of iterations which was set at 700.

The reason for selecting the aforementioned combinations of algorithms and weighting schemes is that the latter affects the performance of the algorithm. Notably, the adaptive algorithm has been executed without edge weighting. This due to according to our experience weighting has little effect on the performance of the aforementioned algorithm.

the error measures

We introduce three different error measures to compare against. Each error measure allows to designate the utility of the algorithms, in view of different possible applications. These are the average initial arithmetic mean deviation (IAMD ϵ_1)

$$\epsilon_1 = \frac{1}{n_k} \sum_{k=1}^{n_k} \left(\frac{1}{n} \|\mathbf{x}_k(t)\|^2 - \mu_k(0)^2 \right)^{1/2} \tag{6.1}$$

the true mean average deviation (TMAD or ϵ_2)

$$\epsilon_2 = \frac{1}{n_k} \sum_{k=1}^{n_k} \left(\frac{1}{n} \|\mathbf{x}_k(t)\|^2 - \|\mathbb{E}[\mathbf{x}(0)]\|^2 \right)^{1/2} \tag{6.2}$$

and the average standard deviation (ASTD or ϵ_3)

$$\epsilon_3 = \frac{1}{n_k} \sum_{k=1}^{n_k} \left(\frac{1}{n} \|\mathbf{x}_k(t)\|^2 - \mu_k(t)^2 \right)^{1/2} \tag{6.3}$$

where $\mathbf{x}(t) \in \mathbb{R}^n$ is the state, $\mu_k(t) = \frac{1}{n}\mathbf{1}^T\mathbf{x}(t)$ is the arithmetic mean of the network at the t^{th} iteration, and k is the index of the realisation in the given batch of experiments with n_k being the number of experiments.

These error measures accommodate the comprehension of each of the algorithms performance in different circumstances, which may be related to possible applications. Specifically, the error measure ϵ_1 determines the accuracy of the algorithm with respect to determining the arithmetic mean of the initial values found on the network. This is important for the distributed machine learning (*Chapter* 5). The second error measure ϵ_2 is more appropriate in cases where we address the problem of the determination of the data's expectation, e.g. in the case of environmental monitoring. In contrast, error measure ϵ_3 provides information about the agreement over the network at each iteration. This would be of importance in coordination tasks. The network may be in agreement but not at the desired value, which herein we assume to be the arithmetic mean.

measures for interval evaluation

In order to evaluate the expected performance in various intervals of execution, we have introduced interval versions for each of these measures. These modifications are simply obtained by dividing the measure with the interval's duration, which are simply

$$E_1 = \frac{\epsilon_1}{\Delta t}, \quad E_2 = \frac{\epsilon_2}{\Delta t}, \quad E_3 = \frac{\epsilon_3}{\Delta t} \tag{6.4}$$

where $\Delta t = t - t_0$ is the interval duration, with t_0 marking the start of the interval and t its end.

figures' explanation

Overall results for the case of the small graphs are exhibited in (*fig.*38). Hence, the error average of the measure in each sequence, inclusive of all the executions of the algorithms for both non-failing and failing links, is shown. The layout of (*fig.*38) is as follows. The vertical axis is the error measure E_3. The horizontal axis is separated in 4 segments. Each segment corresponds to a time interval from 0 to the designated time point. In each segment the bar plots designate the performance of a distinct configuration up to the separating point, e.g. the overall performance for the entire execution is given in the last segment. The rest of the figures follow a similar layout. Extensive examination of (*fig.*38) is not needed to conclude that the nonlinear algorithm outperforms on average the rest in these relatively small random graphs. This is further justified by detailed results presented in (*Appendix* B) in (*fig.*51) to (*fig.*61).

Summarising the results of these experiments, the main conclusion is that the nonlinear algorithm using weights obtained by convex optimisation, performs better on average. Specifically, in small sized graphs the nonlinear algorithm

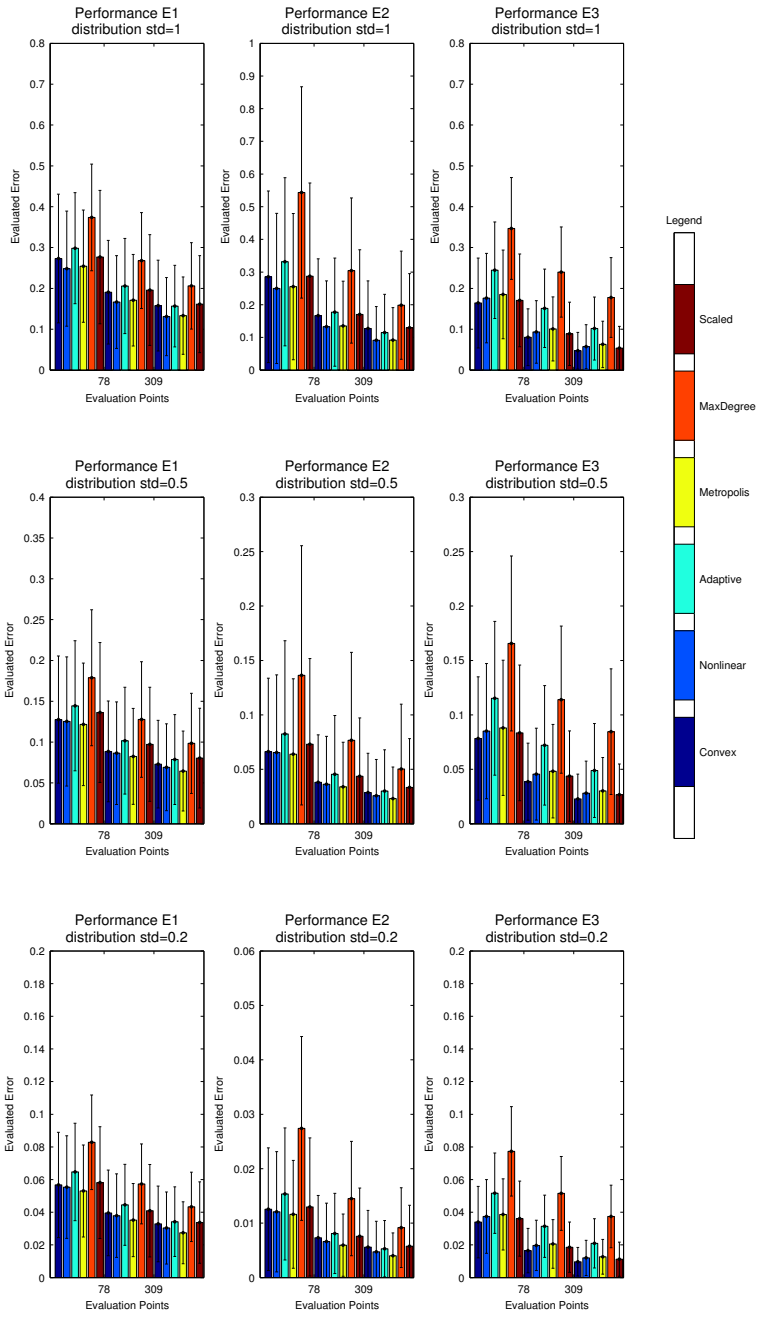


Figure 39: **Overall errors wrt different initial standard deviation in small graph**. The errors E_1, E_2, E_3 computed over 10 random graphs of 20 vertices, are given. The initial standard deviation was 1, 0.5 and 0.2 with mean equal to 10.

outperforms the rest with respect E_1 and E_2 in almost all segments. This implies that it is better fitted for the determination of the arithmetic mean and the expectation, given normality of the data. In most of the cases the linear consensus algorithm exhibits better performance with respect to E_3. This implies that the nonlinear algorithm facilitates agreement between the participating machines. The adaptive algorithm performs better than the linear in some cases, but its performance is superseded in most of the cases by the nonlinear. Further examination of the figures with respect to edge and vertex failure designate that the nonlinear algorithm exhibits better performance than the rest overall.

The main conclusion from this sequence of experiments is that the nonlinear consensus algorithm exhibits superior performance in comparison to the linear. The adaptive algorithm even though in our experience has exhibited superior performance to the nonlinear in specific graphs, see results in (*Section* 3.5.2), exhibits inferior performance on average. However, its performance is comparable with the other algorithms. Even though the adaptive consensus algorithm has been executed on un-weighted graphs, it has performed near the best in each case. Specifically, these conclusions indicate the application domain for each algorithm. The nonlinear algorithm is an "all-around" algorithm that has superior performance than the original consensus algorithm in weighted graphs. This is due to the enhancement of the convergence rate during the transient phase. The adaptive algorithm is appropriate for cases of uncertain communication links and in graphs that weighting is difficult, e.g. huge graphs.

6.2 DEFINITIVE CONSENSUS

We provide exemplary results of the weights obtained by nonlinear optimisation for definitive consensus. We have generated for this purpose a couple of random geometric graphs, with vertex sizes 10 and 25 and edge set size 13 and 42, respectively. We have executed an optimisation to obtain the numerical solution with an error tolerance of 10^{-8}. For the first graph the results could be obtained in a regular 8-core server in less than a minute. The second graph required less than half an hour on the same machine.

figures' explanation

In these figures, the graph, the solution, and two executions of the definitive consensus algorithm are given, along with their root mean square error (RMSE). The two executions of the definitive consensus algorithm are for small and large initial state. With the terms small and large we imply that the initial state values have been small or large numbers. Specifically, in the large state sequence the

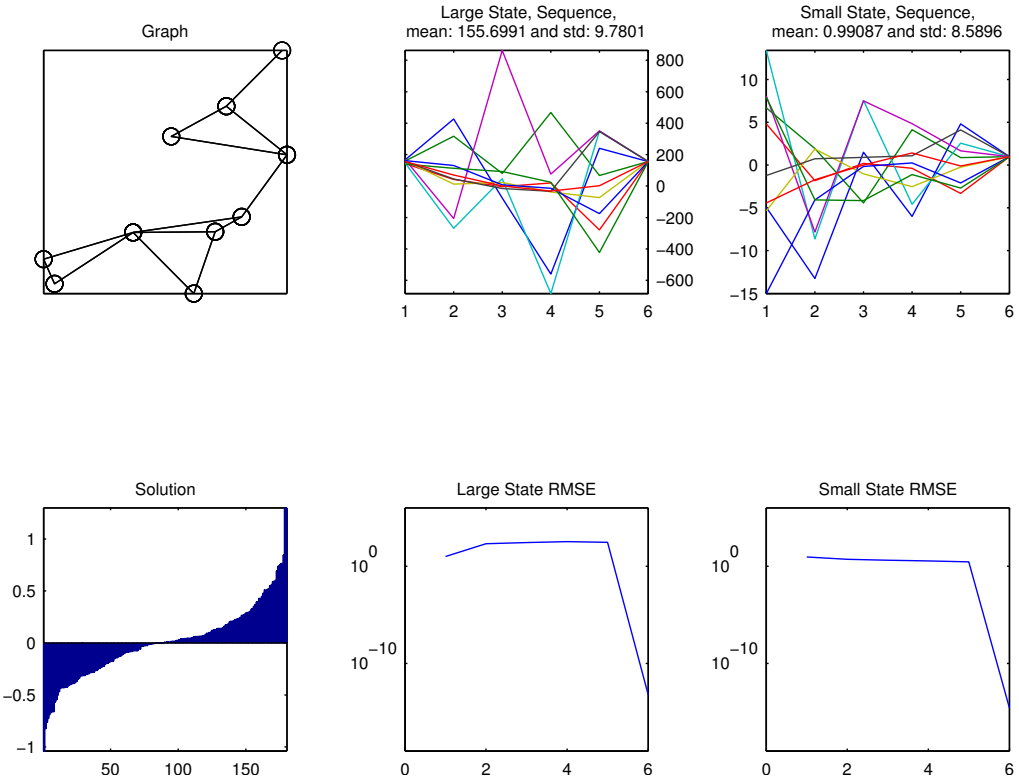


Figure 40: **Definitive consensus example in a graph of 10 vertices with asymmetric weight matrices**. From left to right and top to bottom are shown the graph, the states during execution of definitive consensus for an initial state with 's, another with smaller values, the sorted solution, the root mean square error at each step for the first and second execution.

mean and variance has been about 100 times larger that in the small case. These had mean and variance sampled from normal distribution centred at 0 and variance 1. The purpose of this comparison is to demonstrate that the accuracy obtained is not sensitive to the initial state. This can be verified in all figures by observing the small and large state root mean square error (RMSE). The latter, in all four figures, is large throughout the process. However, after the last iteration it attains near machine precision, 10^{-14} specifically.

Both cases of symmetric and non symmetric weight matrices are presented. These are in (*fig*.41) and (*fig*.40) for a graph of vertex size 10, and (*fig*.43) and (*fig*.42) display results for a graph of vertex size equal to 25. In most cases,

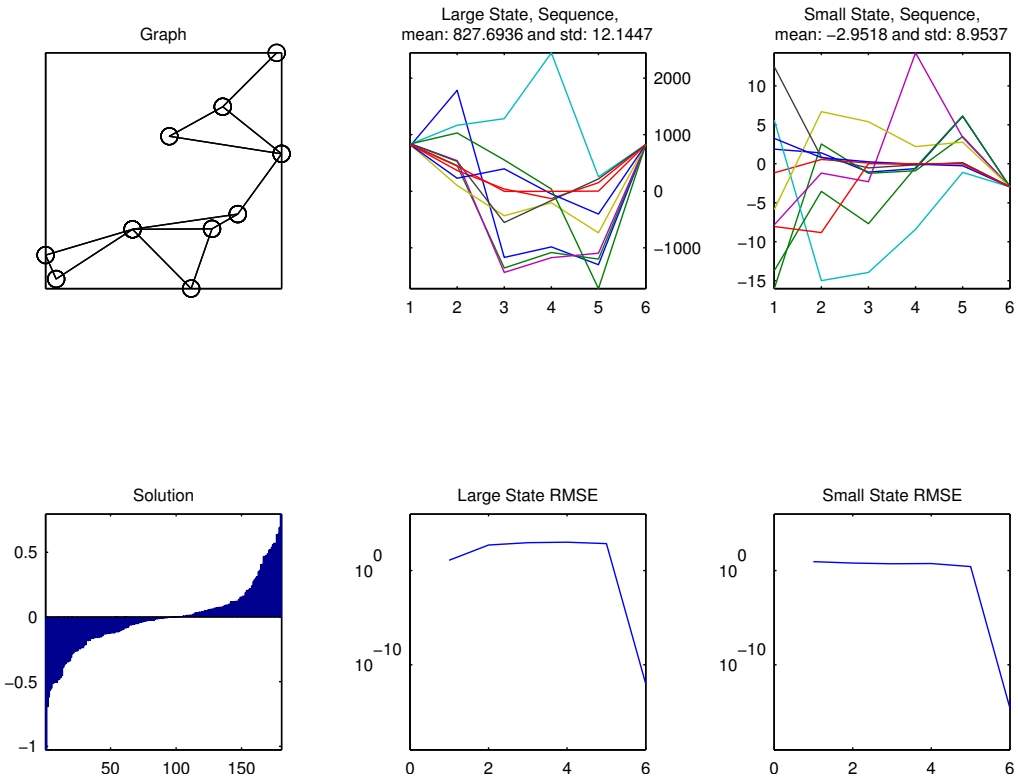


Figure 41: **Definitive consensus example in a graph of 10 vertices with symmetric weight matrices**. From left to right and top to bottom are shown the graph, the states during execution of definitive consensus for an initial state with large values, another with smaller values, the sorted solution, the root mean square error at each step for the first and second execution.

solutions obtained for symmetric matrices are better behaved. That is in the sense that the states show less variation, which can be better noticed by inspecting the case of the small graph, i.e. (*fig*.40) and (*fig*.41). In the first case, values are spread over $[-2, 2]$ even up to before the last iteration. However, in the second case (*fig*.41), the values are constrained from early in the region $[-1, 1]$. Similar behaviour is observed for the large state sequence, as well.

The sorted values of the edge weights for each given solution are also shown in these figures. It is interesting to observe how the number of parameters increases with the vertex size even for these small graphs. The first graph has vertex size 10 and about 160 edge weights to be determined. The second has vertex

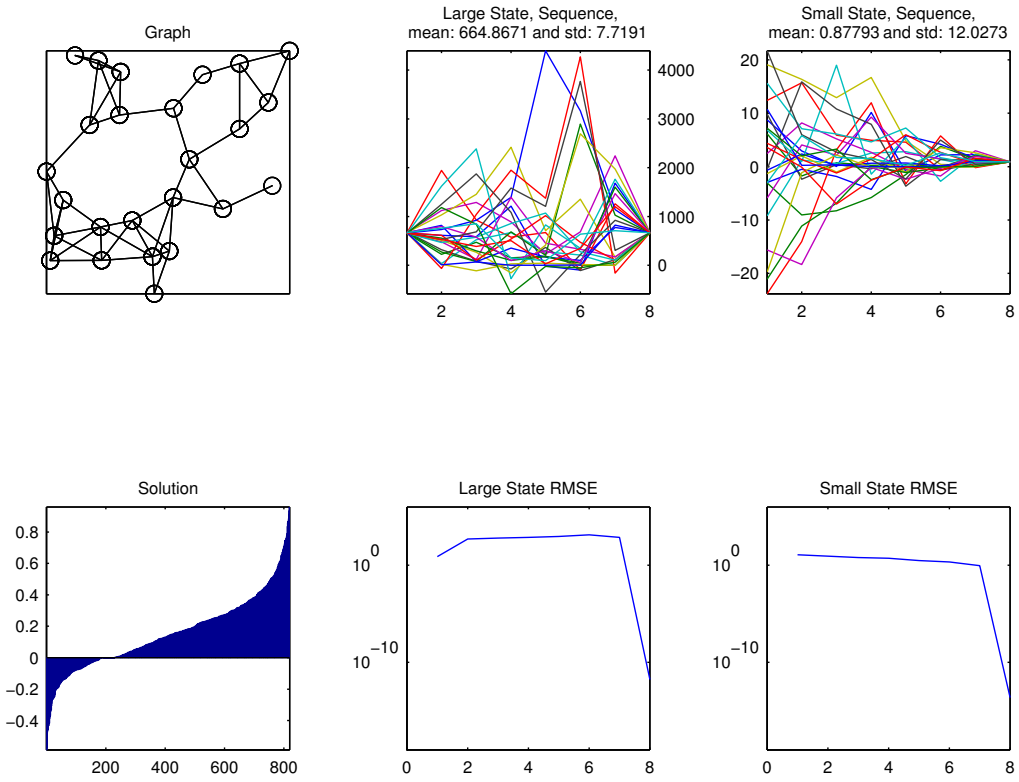


Figure 42: **Definitive consensus example in a graph of 25 vertices with asymmetric weight matrices**. From left to right and top to bottom are shown the graph, the states during execution of definitive consensus for an initial state with large values, another with smaller values, the sorted solution, the root mean square error at each step for the first and second execution.

size 25 and 800 edges to solve for. This hardens the problem of determining the solutions for larger graphs, since the number of parameters does not scale linearly with respect to the diameter. Specifically for the asymmetric case, the parameters scale with $(2|\mathcal{E}(\mathcal{G})|)^{d}$, where d is the graph diameter. The latter, scales at most linearly with the vertex size $|\mathcal{V}(\mathcal{G})|$.

The difficulty in finding the edge weights is becoming evident on normal computing servers from vertex size 40 and upwards for random geometric graphs. Specifically, we have solved for a random geometric graph of vertex size 40. The process required about 4 hours to complete. Obtaining the weights for a random geometric graph of 45 vertices required about a day on the same

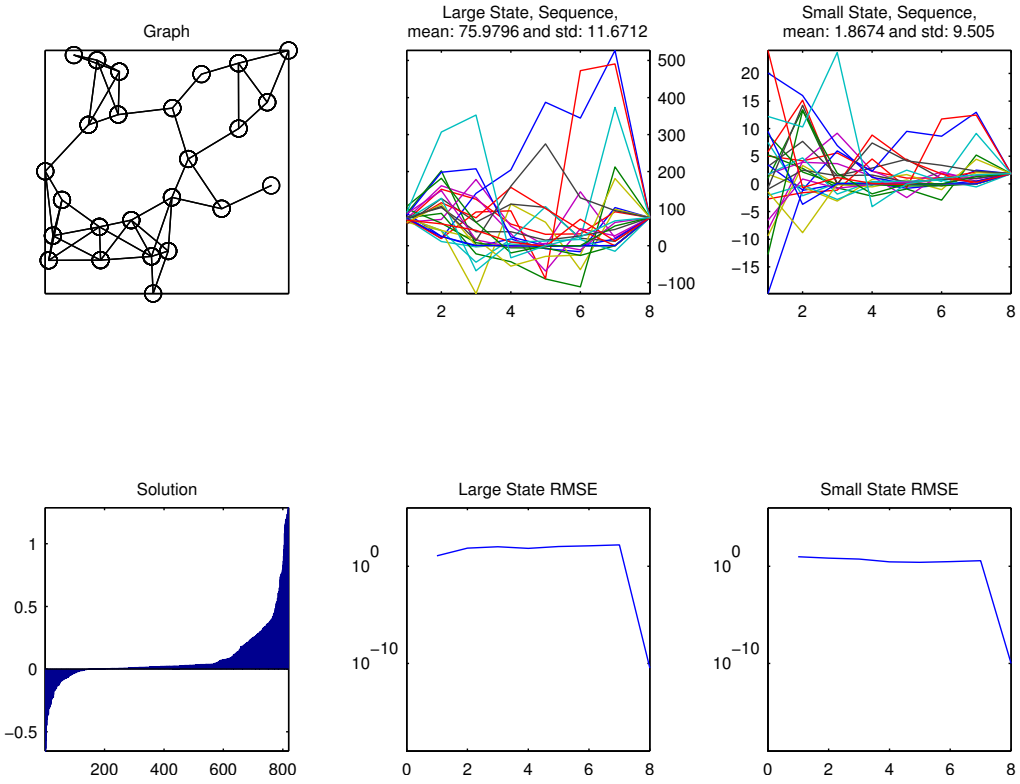


Figure 43: **Definitive consensus example in a graph of 25 vertices with symmetric weight matrices**. From left to right and top to bottom are shown the graph, the states during execution of definitive consensus for an initial state with large values, another with smaller values, the sorted solution, the root mean square error at each step for the first and second execution.

machine. The results for the second graph are found in (*Appendix* B), in (*fig.*49) and (*fig.*50), for the symmetric and non symmetric case. As the graph size increases, apart from the effort needed to determine a solution, the difficulty in recovering the parameters that attain maximum precision is also larger. This can be verified in (*fig.*49) where maximum precision is at 10^{-7}. The difficulty in attaining better precision in larger graphs can be overcome by using floating point arithmetic at the expense of increased computational effort.

The computational effort needed to obtain the solutions may be discouraging. However, this is something that is needed only once for any given network. Once the coefficients have been retrieved, then there is no need for additional

computation. Particularly, once the network has been setup with these coefficients, then the reduction in operating costs shall be enormous in comparison to executing simple consensus (*Section* 4.1). Therefore, the effort for the computation of the coefficients should not be regarded a drawback; once the operating, communication, energy, and other costs on the network have been taken into consideration. However, the computational effort is of concern, since it imposes a limit to size of the networks where this method can be applied to.

6.3 DISTRIBUTED MACHINE LEARNING

In (*Chapter* 5) we have mainly used as an example a toy-dataset to evaluate the distributed data inference framework. However, real world problems usually lead to the production of large datasets. Our main purpose is to evaluate the convergence of the consensus machine learning framework in large real world data. For that purpose we have tested a few neural networks on the 2007 TREC Public Spam Corpus (Cormack and Lynam, 2005), also known as the NIST spam/ham dataset. The performance attained in these tests is not intended for comparison with any of the algorithms having been tested on this dataset, because our interest has been to verify the convergence property of the distributed machine learning framework.

6.3.1 *The 2007 TREC Public Spam Corpus*

The TREC spam dataset consists of 75419 emails and an index file that designates if the email is spam or ham. A description of the process to retrieve the spam corpus is given in (Cormack and Lynam, 2005). The dataset can be obtained directly from http://plg.uwaterloo.ca/ gvcormac/treccorpus07/ .

dataset pre-processing

Significant pre-processing is needed before the dataset can be used and a machine learning algorithm can be applied. We have chosen to treat each email with a bag-of-words approach; the process is intuitive. One first determines the tokens that are found at each e-mail; a token is a string between two delimiters. Then the number of occurrences of each token is counted for each different email. Thus, each email can be represented as a very big vector that has as elements the number of occurrences of all the tokens in all e-mails in the specific email. In fact, frequencies have to be computed from the number of occurrences as a subsequent step. However, the number of different tokens in the entire

dataset is huge. We have determined 155579 different tokens. However, this is dependent on the tokeniser itself, i.e. the program the extracts the tokens from the e-mails. Other implementations might produce a different number of tokens. Finally, after this pre-processing step the dataset would require 87GB memory space; this needs to be reduced in order to be usable.

Instead of using the entire dataset, we chose to perform a reduction of the dimension of the token vector by performing random sampling in three parts of the vector. We have sorted the tokens depending on their number of occurrences on the corpus. Then three regions have been determined for this sort, the head, the middle, and the tail. The head and the tail have been specified to be the first and last 10000 tokens, respectively. The middle section has been specified as the region 5000 tokens before and after the token in the middle of the sorting. Subsequently, we have randomly sampled 1000 tokens from each section, totalling 3000 tokens. Finally, these tokens formed the vector of tokens that represents each email by accounting the frequencies of tokens at each email.

However, this would still produce a dataset of about 2GB. Our purpose had been to use this dataset for consensus learning by simulating a network of 10 connected machines. In order to further reduce the computational effort and overcome some memory issues with Matlab, we used only half of the dataset. Specifically, 30000 e-mails out of the total 75419 had been used, resulting in a dataset of 1GB. Nevertheless, these pre-processing steps do not alter its appropriateness for our purpose. Particularly, such pre-processing steps are quite common, and depending on the machine learning algorithm these may even be advantageous.

6.3.2 *Definitive Consensus Data Inference*

The generated dataset of 30000 vectors has been separated into 2/3 training set and 1/3 test set. The training set has been partitioned into 10 subsets. Then each of these 10 subsets has been partitioned into train and validation, (2/3 training set). These 10 subsets have been used for the simulation of the consensus machine learning process in Matlab.

Arguably, it might have been more appropriate to implement this in a completely distributed setting. That would require to implement a separate process on different computing servers and distribute the data on these servers. The process could run in multi-threaded mode to exploit the full potential of each server. Also, communications would be of concern, but simple implementations

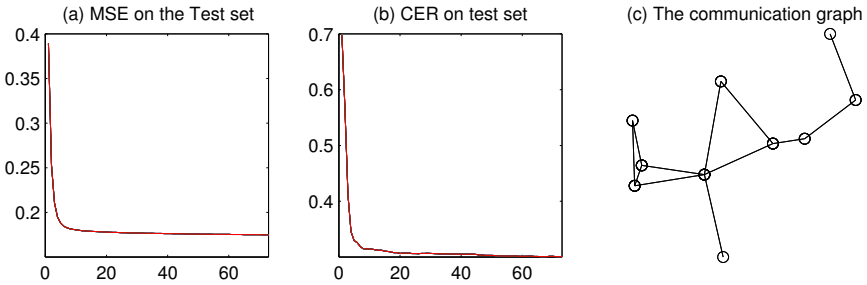


Figure 44: **Convergence on the reduced TREC Spam Corpus on a simple neural network**. The convergence of the consensus machine learning framework on the reduced TREC Spam Corpus can be evaluated by examination of the loss function during training. Results are shown on the test set for a neural network of [10 50 10] neurons. Early stopping with 6 validation checks has been used. The regularisation parameter was set at 0.80. The (MSE) and (CER) of the non-distributed was 0.18 and 0.31, respectively. The sub-figures demonstrate the convergence of the distributed algorithm. Particularly, (a) The mean squared error (MSE). (b) The classification error rate (CER). (c) The communication graph

such as TCP/IP or IPC or even simple file-sharing should be sufficient. This would require a realisation of the aforementioned framework which had not been in our intentions. Then speed, memory and communication utilisation benchmarks would allow us to draw safe conclusions. However, it has not been our main concern to verify the supremacy of the solution, but its equivalence in convergence with respect to the non-distributed.

Nevertheless, a few obvious conclusions can be drawn from inspection of these simulations with respect to the required computation time. Consider the fact that the distributed has been in fact executed in sequence for each neural network at each iteration, and it did not fully utilise all 8-cores of the system. In contrast, the centralised had been running with fully optimised code base, i.e. the Matlab implementation of feed-forward neural network, and had been utilising the entire processing power of the system. In our experience, the time consumed from a distributed process was comparable to that of training just one neural network on the entire dataset when divided by the number of machines on the simulated communication network.

We have employed the definitive consensus algorithm during the execution. Therefore, the only options having to be selected for these experiments are the number of local learning iterations for the neural network, the regularisation

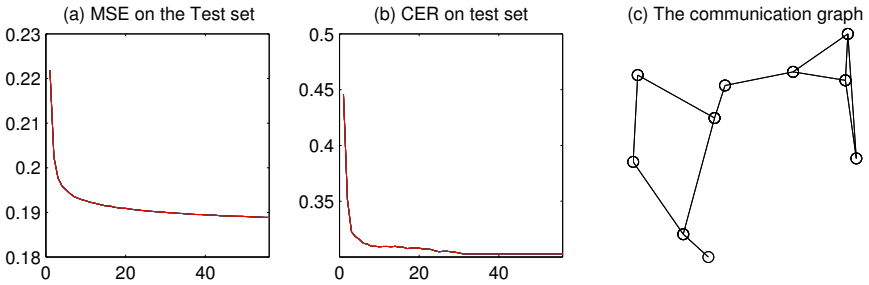


Figure 45: **Convergence on the reduced TREC Spam Corpus on a larger neural network**. The convergence of the consensus machine learning framework on the reduced TREC Spam Corpus can be evaluated by examination of the loss function during training. Results are shown on the test set for a neural network of [12 20 30 10] neurons. Early stopping with 6 validation checks has been used. The regularisation parameter was set at 0.90. The (MSE) and (CER) of the non-distributed was 0.19 and 0.30, respectively. The sub-figures demonstrate the convergence of the distributed algorithm. Particularly, (a) The mean squared error (MSE). (b) The classification error rate (CER). (c) The communication graph

amount, the number of validation checks for termination. The experiments are also affected by initialisation. We have slightly modified the default Matlab initialisation method. The latter calculates the weight and bias values for each layer using the Nguyen-Widrow initialisation method (Nguyen and Widrow, 1990). This was slightly modified. At each initialisation we have sampled randomly from the entire training set a random number of datapoints. Thus using only a part of the entire training set for the initialisation of the neural networks with the aforementioned method. Then, these parameters were communicated by consensus for the initialisation of all 10 neural networks.

Furthermore, we have used a variety of communication networks. However, this did not seem to affect the convergence or the results. Specifically, during the process of training, issues common to training feedforward neural networks with back-propagation have been of concern. Nonetheless, the distributed algorithm converged in each case.

We exhibit three cases of interest on differently sized neural network. The first case in (*fig.*44) is a simple three layer neural network with one hidden layers, having 10 neurons on the first, 50 on the second and 10 in the third. We denote such an architecture as [10 50 10]. Subsequently, we present a neural network

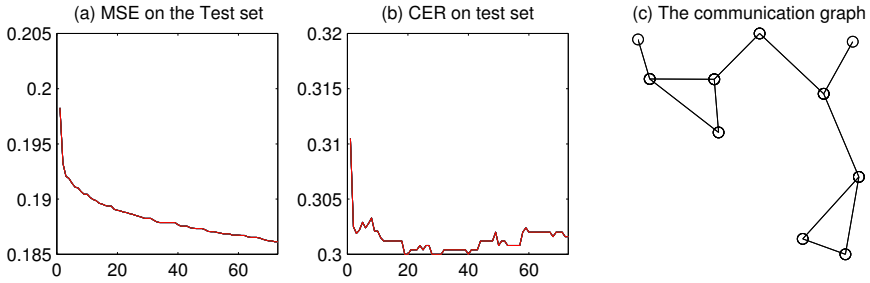


Figure 46: **Convergence on the reduced TREC Spam Corpus on an even larger neural network**. The convergence of the consensus machine learning framework on the reduced TREC Spam Corpus can be evaluated by examination of the loss function during training. Results are shown on the test set for a neural network of [12 20 30 40 10] neurons. Early stopping with 20 validation checks has been used. The regularisation parameter was set at 0.91. The (MSE) and (CER) of the non-distributed was 0.19 and 0.30, respectively. The sub-figures demonstrate the convergence of the distributed algorithm. Particularly, (a) The mean squared error (MSE). (b) The classification error rate (CER). (c) The communication graph

with architecture [12 20 30 10] and another one with [12 20 30 40 10] in (*fig.*45) and (*fig.*46), respectively.

In all three examples the distributed data inference converges just as the non-distributed. The classification error rate is not sufficiently low, but comparison with the non-distributed reveals that the non-distributed performs similarly. Nevertheless, in all figures the reduction of the classification error rate and the mean square error is evident. However, it can be better verified in (*fig.*44). There the initial error rate was at 0.7 and with subsequent iterations it had been reduced to about 0.3.

6.3.3 *Summary*

In this chapter our main purpose has been to exhibit additional results for the notions presented in (*Chapter* 3), (*Chapter* 4) and (*Chapter* 5). These additional results verify our claims throughout this thesis.

Specifically, the improvement provided by incorporating the nonlinear consensus and adaptive algorithm has been validated in (*Section* 6.1). There we compared the three consensus algorithms, linear, nonlinear, and adaptive, and various cases of reliable and unreliable communications have been tested. More-

over, their performance has been compared under different weight assignments schemes. Therefore, a large number of the possible cases has been simulated, thus taken into account. The result has been that the nonlinear consensus algorithm presents a better solution overall. The adaptive algorithm should be employed in un-weighted communication networks with unreliable communication links.

Thereafter, in (*Section* 6.2) additional results have been presented for the definitive consensus algorithm. Our purpose has been to present solutions for random geometric graphs that verify convergence up to machine precision. Moreover, we have shown that employing symmetric matrices may allow for a somewhat smoother convergence. Specifically, the weight matrices are not restrained with respect to their norm. This can lead to large values of the states during the execution of the algorithm, which might be unwanted in some applications. Let us report here that in our experience, it is possible to restrain the norm of the matrices in the sequence. This enables a better behaved state sequence during execution of the algorithm. However, this would require imposing $\mathtt{md}(\mathcal{G})$ additional nonlinear constraints on the numerical solver, and therefore increase the computational burden.

We have exhibited solutions with near machine precision 10^{-15} for graphs with vertex set size equal to 25. Additional results for larger graphs and lower precision have been included in (*Appendix* B).

Finally, we have included results that demonstrate the convergence property of the definitive consensus machine learning algorithm in (*Section* 6.3.2). The algorithm has been tested on the TREC Spam Corpus, which is a large dataset for binary classification. Our results verify that the distributed neural networks, presented in (*Section* 5.4.1) converge just as an identical non-distributed neural network on the global dataset. Arguably, performance in both cases of distributed and non-distributed is not adequate for comparison with current approaches. However, this is mainly due to the trivial architecture employed on the neural network, and not due the distributed learning process. This is evident due to the fact that comparison of the two cases, distributed and non-distributed, does not indicate significant differences in the performance. Significantly more complex solutions have been proposed for this given dataset. Therefore, a comparison with our simplistic approach would not be appropriate. However, our purpose has been to verify convergence of the distributed learning algorithm in real world datasets, which has been achieved.

Concluding, in this chapter, additional numerical results have been provided for the algorithms presented in previous chapters. The results demonstrate the truthfulness of our claims and the theoretical derivations. These have been the following three. First, that the nonlinear and adaptive consensus algorithms reduce the number of iterations for convergence to machine precision. Second, that solutions can be obtained for the definitive consensus algorithm relatively easy for medium sized graphs. Third, that the distributed data inference framework converges just as the non-distributed counterpart. All three have been shown to be valid.

7

DISCUSSION

In previous chapters a number of problems, located in the intersection of communication and computer sciences, have been presented and dealt with. Within this chapter we approach the issue from an applied perspective. Our main concerns are to specify the application domain of the algorithms, the advantages and disadvantages of the algorithms, and finally provide directions for extending and improving this work.

7.1 SUMMARY

The first part of this document (*Part* i) has mainly been concerned with introducing the reader to the notions and tools related to this research. Most importantly, the main issues found in the thesis have been introduced.

In (*Part* ii), the main theme has been the distributed inference problem. The latter, in its simplest form, can been conceived as a consensus problem. In that case, we consider having one dimensional data and making the assumption of normality, then our interest is in determining the expectation of the data. Thus, inferring in a very simple manner the expectation for the data. This has been solved in the past (Tsitsiklis, 1984) with the so called average consensus algorithm. However, the latter has some important drawbacks with the most important being the large number of operations required on the network until convergence. Moreover, the number of iterations needed to attain some precision depends on the initial state vector norm. Hence, we argue that in many applications of interest the algorithm is not appropriate. The task of inference in a distributed manner is such a case.

In order to apply such a solution to more complex and larger datasets, it is of great importance to ameliorate the aforementioned predicaments. This has been addressed with the definitive consensus algorithm. The latter is a switching dynamical system that converges to consensus in number of iterations equal to the diameter of the communication graph. Comparing with the asymptotic convergence of the consensus algorithm our solution provides a major improvement.

However, there is a tradeoff to consider. In order to determine the weights on the edges one has to resort to numerically solving a large set of multi-linear polynomial equations with a large number of unknowns. We have shown in (*Section* 4.2.2.3) that this system of equations has a solution when the number of matrices is between $d(\mathcal{G})$ and $2d(\mathcal{G})$. Moreover, we have specified one of the many solutions that may be obtained in quadratic time complexity with respect to the number of edges with centralised algorithms and in linear time when considering distributed algorithms. Moreover, based on our experience it has been conjectured that this system of equations has infinitely many solutions given any graph for $d(\mathcal{G})$ matrices. Therefore, definitive consensus can be reached in a number of iterations equal to the diameter of the graph that the algorithm is executed at.

These theoretical considerations are important because their affirmation encourages the extensive numerical calculations needed for the retrieval of a solution. However, it is of even greater importance to know if these solutions can be obtained. Surprisingly, the solutions are easy to retrieve, up to medium sized graphs. Particularly, we have obtained solutions for graphs having up to 60 vertices without significant programming effort. Admittedly, one can obtain better results with specific code optimisation and floating point arithmetic. However, this has not been our main research interest.

Though, we recognise that the definitive consensus algorithm cannot be applied in every possible situation. Therefore, we have been concerned with improving the speed of convergence in cases of partially known topology, stochastic communications, and for larger graphs. The nonlinear consensus algorithm and the adaptive consensus algorithm, presented respectively in (*Section* 3.4) and (*Section* 3.5), are improvements of the linear consensus algorithm (*Section* 2.3.2) for the aforementioned cases. Moreover, the nonlinear consensus algorithm improves the overall speed of convergence when the sampling distribution of the initial state is presumably Gaussian. In conclusion, these algorithms improve finite time behaviour of the network.

In (*Chapter* 5), we have presented an abstract framework that enables distributed data inference in supervised machine learning problems. The convergence of consensus machine learning relates to that of the specific machine learning algorithm. Hence, with appropriate selection of the parameters, the number of local learning iterations and consensus iterations, the distributed learning process converges if the non-distributed learning algorithm does for the specified initialisation and learning parameters. The introduction of the

definitive consensus algorithm in this network alleviates partially the problem of specifying both the consensus and learning iterations, since is is equivalent to running consensus until convergence, up to machine precision. This allows us to have better control over the process. In this case, the number of local learning iterations affects the learning step of the entire network.

Furthermore, the learning equations have been specified in the case of multi-layer feedforward neural network with back-propagation. The update equations for batch training have been duly modified and included in (*Section* 5.4.1). This has enabled us to perform numerical experiments to verify the theory in (*Section* 5.2.1). We have presented numerous numerical results for the distributed inference framework for a toy example, the two moons dataset; these are presented in (*Section* 5.5) and (*Appendix* B). In that manner, the convergence property of the framework has been verified.

Furthermore, the applicability of the framework to large datasets has been verified by training a simulated distributed learning framework on the 2007 TREC Spam Corpus. The results for those experiments permit two conclusions. First, that the convergence property is verified. Second, that the results are as good as the non-distributed counterpart. Specifically, given the fact that we have tested simple network architectures, there was no expectation of training a neural network that will have notable performance on such a difficult task. Nevertheless, our purpose, to verify the convergence of the distributed learning algorithm in a large dataset has been duly verified.

Another significant observation deduced from these experiments is that the time to convergence is comparable between the distributed and the non-distributed. Something that supports the applicability of this method in real world cases.

7.2 APPLICATION DOMAINS

The main contributions of this thesis are the definitive consensus algorithm and the consensus machine learning framework. We consider the different application domains for each.

7.2.1 *Definitive Consensus*

We have arrived at the following conclusions in numerous parts of this document when considering the definitive consensus algorithm. First, the definitive consensus algorithm allows consensus in a small finite number of iterations. It has been possible to obtain the network weights up to medium sized random graphs. Second, the algorithm switches weights at each iteration. This makes it prone to errors in case of unreliable or asynchronous communications. These designate the domain of application to be that of the consensus algorithm when communications are guaranteed and the network is synchronous. Specifically, in small to medium ad-hoc communication networks where machines are arbitrarily connected. As usual we assume that the entire network is connected.

The application to larger networks may be enabled by building a hierarchical network. There, a consensus algorithm runs at each different level of the hierarchy. Particularly, the machines at one layer wait for those at the subsequent layer to converge before commencing their execution of the definitive consensus algorithm. Such architectures are in fact being employed in real world ad-hoc networks instead of a flat-out architecture.

network applications

Other domains of application associated to this work are related to the algebraic equation (*eq.*4.2). Such issues arise in distribution, transportation, power and computer networks. The general theme of the problem is how can one evenly distribute a quantity over a network by timely regulating the connection parameters. For example in a transportation network, one could consider as connection parameters the rate of vehicles between road segments, which are regulated by the wait states at the traffic lights at the edge of the segments. The purpose would then be to regulate these such that the same amount of vehicles is found at each traffic light.

Particularly, consider the following abstraction. Assume the graph emerging from the road network where each segment $\{i, j\}$ between traffic lights i and j is an edge and each traffic light is a vertex. Assume that the automobiles spent negligible time between traffic lights, i.e. at the road segments, but spent most of their time waiting for the traffic light to turn on. Thus, the amount of time that a traffic light is on, in each direction, determines the rate w_{ij} of vehicles passing though each segment $\{i, j\}$. The amount of vehicles found at a traffic light at any given iteration is given by the state. The rate of vehicles following a path from k_1 to k_d, i.e. $p\{k_1, k_2, \dots, k_d\}$, through such segments, in the considered time frame, is just the product of the rates on the segments $w_{k_1k_2}w_{k_2k_1}\dots w_{k_{d-1}k_d}$.

Then the rate of the vehicles from any traffic light k to any other k_d is the sum of the rates through all the paths connecting k to k_d on the graph. These are the products found in (*eq*.4.2). Therefore, regulating the road network such that the rate of vehicles is the same between any two points can be obtained by solving (*eq*.4.2).

Similarly, in a computer network, one would want to regulate the bandwidth between points , e.g. routers, such that the same amount of packets arrives from any one machine on the network to any other in a given time-frame. It is quite important to notice here that (*eq*.4.2) does not necessarily have a solution for any target matrix, which in the case of consensus is $\frac{\mathbf{1}\mathbf{1}^T}{n}$. This implies that network topology limits the options in regulating the network.

7.2.2 *Consensus Machine Learning*

We distinguish the following about the consensus machine learning framework. The algorithm converges as the non-distributed does. It allows the distributed processing of large datasets by partitioning these into smaller sets and distributing these to networked machines. Thus, both the memory demands and the computational demands at each machine are significantly reduced in comparison to processing the entire dataset centrally. However, the total computational effort is increased.

Presumably, a fully distributed implementation would demonstrate that computation time is less than having the entire dataset on a single machine. This is justified because then the computation of the update step at each machine is performed in parallel. In contrast, central computation would require that all the datapoints in the dataset are evaluated sequentially.

Our experience, mainly from training on the 2007 TREC Spam Corpus, shows that the computational effort is of the same order of magnitude as for the centralised. In fact, the distributed simulation and the executions have been executed on the same machine. Specifically, the distributed algorithm neural networks have been executed sequentially, since we had made a parallelised implementation. Therefore, the execution time in a parallelised implementation should be divided by the number of simulated neural networks in order to compare with the centralised. The latter, in fact had been executed with multi-threaded computation enabled. Therefore, all the 8-cores had been utilised which resulted in increasingly better execution times.

Considering the communication side of the problem, we have to note that the network can be arbitrarily connected without a central computation node. Finally, the exchanged quantities between the nodes in the network are the classifiers' parametrisation.

The properties of the data designate the domain of application. Specifically, it is advantageous in numerous applications that data is not exchanged due to privacy considerations. Such data is usually medical data, banking data and personal data. In many such cases, one would like to be able to learn from the entire dataset. Therefore, one can imagine that a computer program is created and distributed to the participants that have the data. This would learn on the local data and then perform exchanges on the communication network, as described. Thus, allowing to all the participants to infer as if the entire dataset was available locally.

Another case would be when the data is too large. Our purpose would be to distribute this data into a number of machines when we cannot compute it on a single one. Employing the consensus learning framework frees us from the need of specifying the communication network and a computing or a coordination centre. Furthermore, the distributed computation can be inherently performed in a peer to peer manner. Many such applications have been widespread in the past with SETI@HOME (Anderson et al., 2002) being the pioneer. Other notable efforts are BOINC (Anderson, 2004), EGEE (Gagliardi et al., 2005), GLOBuS, IBM-WCMG, OurGrid, and OpenScienceGrid. In cases of such large and dynamic networks, the size of the communication network would be of concern. However, one can overcome this by forming a hierarchy of network and solving the definitive consensus problem for each network in the hierarchy.

In the case of SETI@HOME, someone would connect to a group of central servers to upload and retrieve information. Moreover, the program would not perform any learning locally, but search for predefined signal patterns in the local dataset. In most of the other cases, GRID computing is applied, which is a conceptually different approach to the problem. In fact, it is more general, in the sense that in a GRID processes are distributed on a number of participating computers. Thus, it is a computer systems approach, whereas we have attempted to abstract the problem of machine learning such that the process can be executed distributively. Therefore, such approaches are different in principle with the work developed in this thesis.

Another domain of applications can be defined when considering that the data is in fact generated locally at each machine. We assume that this data is labelled

somehow. Such a case, includes wireless sensor networks for fire detection and or environmental monitoring, swarm robotics and other collaboration schemes.

The WSN for forest fire detection and localisation case is of great interest to us. There, one would assume that each sensor gathers the data locally. Among the many sensors that can be employed the most prevalent are humidity, temperature, infra-red imaging and CO_2 concentration. Based on these quantities and the current models for fire outburst, one would have hard coded the alarms in each of these wireless sensor nodes. When a sensed quantity would be within the alarm zone, then the system would broadcast some kind of an alarm.

However, this approach cannot be easily adapted to many different situations. Specifically, forests tend to differentiate largely from one region to another. Thus, what may cause an ignition in one case, might not in another.

Assume that there is a local process found at each wireless sensor node that assigns automatically a label to each measurement. This has to be carefully designed. Then one would have to define a decision module. The latter would broadcast an alarm by inferring on current and past measurements along with any predefined fire detection models. However, we can augment the information obtained form all sensor nodes by employing the proposed learning framework.

In the contrast, sending all the data to all nodes would be expensive in terms of communications. Moreover, processing all the information at every node to obtain the same result would result in larger computational cost overall the network. One would have to consider the computation effort to determine the update at each machine on the entire dataset. This is much more expensive than doing it on an n times smaller subset. However, the distributed data inference framework allows to reduces the computational effort at each machine. Moreover, it may also decrease the communications. That is when comparing with the case that the data is transferred for centralised computation.

7.3 FUTURE WORK

Given the number of applications being acknowledged in the previous section, it is of interest to mention possible research extensions. There are such related to the main contributions of this thesis, in both the theoretical and the applied side.

Considering the definitive consensus algorithm, the conjecture that it can be attained in a number of iterations equal to the graph diameter, has yet to be proved. However, this is principally of pure theoretical interest. In fact it has no

impact to the assignment of the edge weights. Admittedly, this assignment is a computationally intensive process. An efficient algorithm for this task would be of much greater interest, both theoretically and applied, thus making possible the retrieval of the edge weights for large graphs.

In respect to the distributed data inference framework there are numerous extensions of this work. Specifically, there are numerous machine learning algorithms that might be more appropriate to be executed in a distributed manner. The un-supervised and semi-supervised machine learning algorithms can be of great utility in many of the aforementioned application domains. The modification of the learning equations would be of interest in such cases. Furthermore, the case of stochastic gradient descent would be an interesting research directions.

The implementation of such methods would be of interest as well. This is something that we have not performed since our principal interest has been in providing a sound foundation for the framework. Admittedly, efficient implementation for distributed and parallel processing for an appropriate machine learning algorithm would be an important contribution, as well.

7.4 CONCLUSION

Herein, the established thesis has been that distributed inference from data can be made possible in a collaborative manner by utilisation of the consensus algorithm. In order to achieve this in an efficient manner the rate of convergence of the consensus algorithm had to be greatly improved. The definitive consensus algorithm converges in a finite number of iterations by timely switching the appropriate communication parameters. The number of iterations for convergence is equal to the diameter of the graph, which enables the execution of the consensus machine learning framework in medium sized arbitrarily connected ad-hoc networks.

Part IV

APPENDICES

A

DERIVATIONS

A.1 RELATED TO (*chapter 3*)

A.1.1 *Derivation of expected state*

We derive (*eq.*3.19), included below.

$$\mathbb{E}[\mathbf{x}(t+1)] = (\mathbf{I} - p\mathbf{L})^{t+1}\mathbf{x}(0)$$

Proof. Notice that:

$$\begin{aligned}
&\text{for } i \neq j \ \mathbb{E}[[\mathbb{L}]_{ij}] = p[\mathbf{L}]_{ij} \\
&\text{for } i = j \ \mathbb{E}[[\mathbb{L}]_{ii}] = \mathbb{E}[\sum_{j=1}^{n} [\mathbb{L}]_{ij}] \\
&\qquad\qquad = \sum_{j=1}^{n} \mathbb{E}[[\mathbb{L}]_{ij}] \\
&\qquad\qquad = \sum_{j=1}^{n} p[\mathbf{L}]_{ij} \\
&\qquad\qquad = p[\mathbf{L}]_{ii}
\end{aligned}$$

Thus $\mathbb{E}[\mathbb{L}] = p\mathbf{L}$. Then it follows for the state that:

$$\begin{aligned}
\mathbb{E}[\mathbf{x}(t+1)] &= \mathbb{E}[\prod_{k=0}^{t} (\mathbf{I} - \mathbb{L}(t))\mathbf{x}(0)] & &= \mathbb{E}[\prod_{k=0}^{t} (\mathbf{I} - \mathbb{L}(t))]\mathbf{x}(0) \\
&= \prod_{k=0}^{t} \mathbb{E}[(\mathbf{I} - \mathbb{L}(t))]\mathbf{x}(0) & &= \prod_{k=0}^{t} (\mathbf{I} - \mathbb{E}[\mathbb{L}(t)])\mathbf{x}(0) \\
&= \prod_{k=0}^{t} (\mathbf{I} - p\mathbf{L})\mathbf{x}(0) & &= (\mathbf{I} - p\mathbf{L})^{t+1}\mathbf{x}(0)
\end{aligned}$$

□

A.1.2 *Communication Cost of Unicast on a Uniform Tree*

Consider a tree with n_k vertices at each level and radius equal to the half of the diameter $r = d/2$. From each vertex to the root the number of communications required is $(r-1)-k$. Hence the total communications towards the root, on a tree, for unicast are

$$u_{in} = \sum_{k=0}^{r} l_k(r-k) - 1$$

where at we assume l_k nodes at each level of the tree. Let these be uniformly distributed according to $l_k = a^k$.

$$\begin{aligned} u_{in} &= \sum_{k=0}^{r} ka^k \\ &= \frac{(ar-r-1)a^{r+1}+a}{(1-a)^2} \end{aligned}$$

We can obtain the radius is closed form since

$$\begin{aligned} n &= \sum_{k=0}^{r} a^k \\ n &= \frac{a^{r+1}-1}{a-1} \\ r &= \frac{\ln(1+n(a-1))}{\ln(a)} - 1 \end{aligned}$$

Therefore we the communication cost for unicast is:

$$u_{in} = \frac{\ln(an-n+1)}{\ln(a)(a-1)}(an-n+1) - \frac{an}{a-1}$$

B

ADDITIONAL NUMERICAL RESULTS

In order to avoid cluttering the text, we have chosen to place some the results in this separate appendix.

B.1 DEFINITIVE CONSENSUS EXAMPLE SOLUTIONS

Exemplary solutions for two random geometric graphs of 20 and 45 vertices are included here.

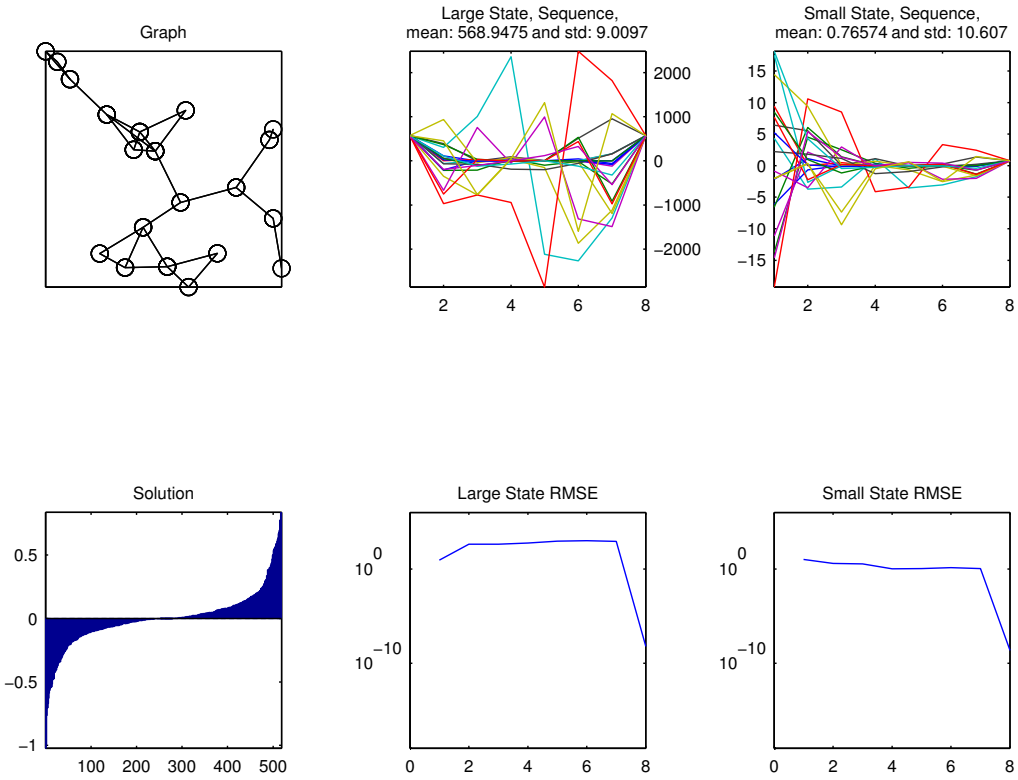


Figure 47: **Definitive consensus example in a graph of 20 vertices with asymmetric weight matrices**. From left to right and top to bottom are shown the graph, the states during execution of definitive consensus for an initial state with large values, another with smaller values, the sorted solution, the root mean square error at each step for the first and second execution.

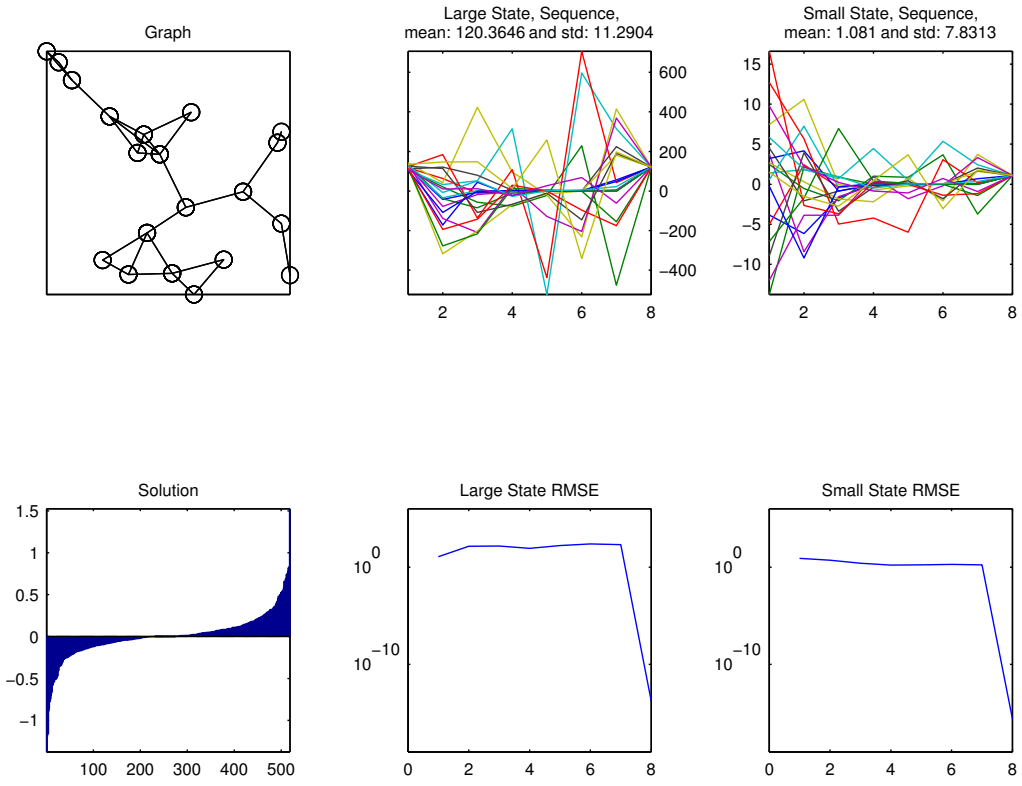


Figure 48: **Definitive consensus example in a graph of 20 vertices with symmetric weight matrices**. From left to right and top to bottom are shown the graph, the states during execution of definitive consensus for an initial state with large values, another with smaller values, the sorted solution, the root mean square error at each step for the first and second execution.

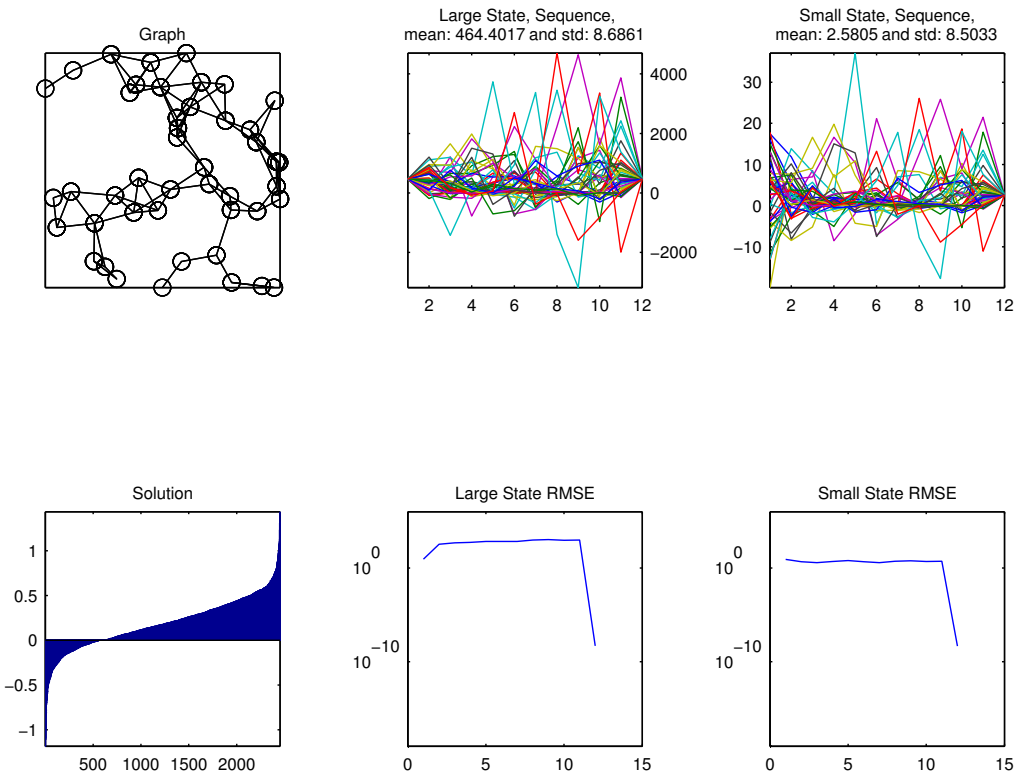


Figure 49: **Definitive consensus example in a graph of 45 vertices with asymmetric weight matrices**. From left to right and top to bottom are shown the graph, the states during execution of definitive consensus for an initial state with large values, another with smaller values, the sorted solution, the root mean square error at each step for the first and second execution.

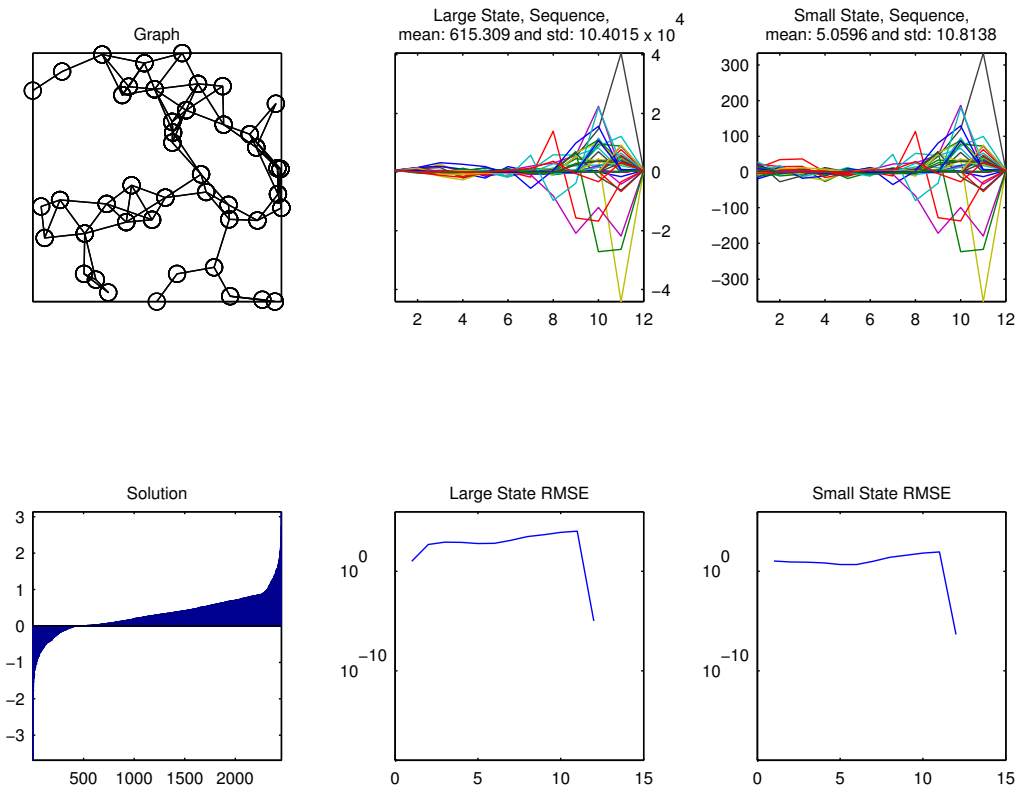


Figure 50: **Definitive consensus example in a graph of 45 vertices with symmetric weight matrices**. From left to right and top to bottom are shown the graph, the states during execution of definitive consensus for an initial state with large values, another with smaller values, the sorted solution, the root mean square error at each step for the first and second execution.

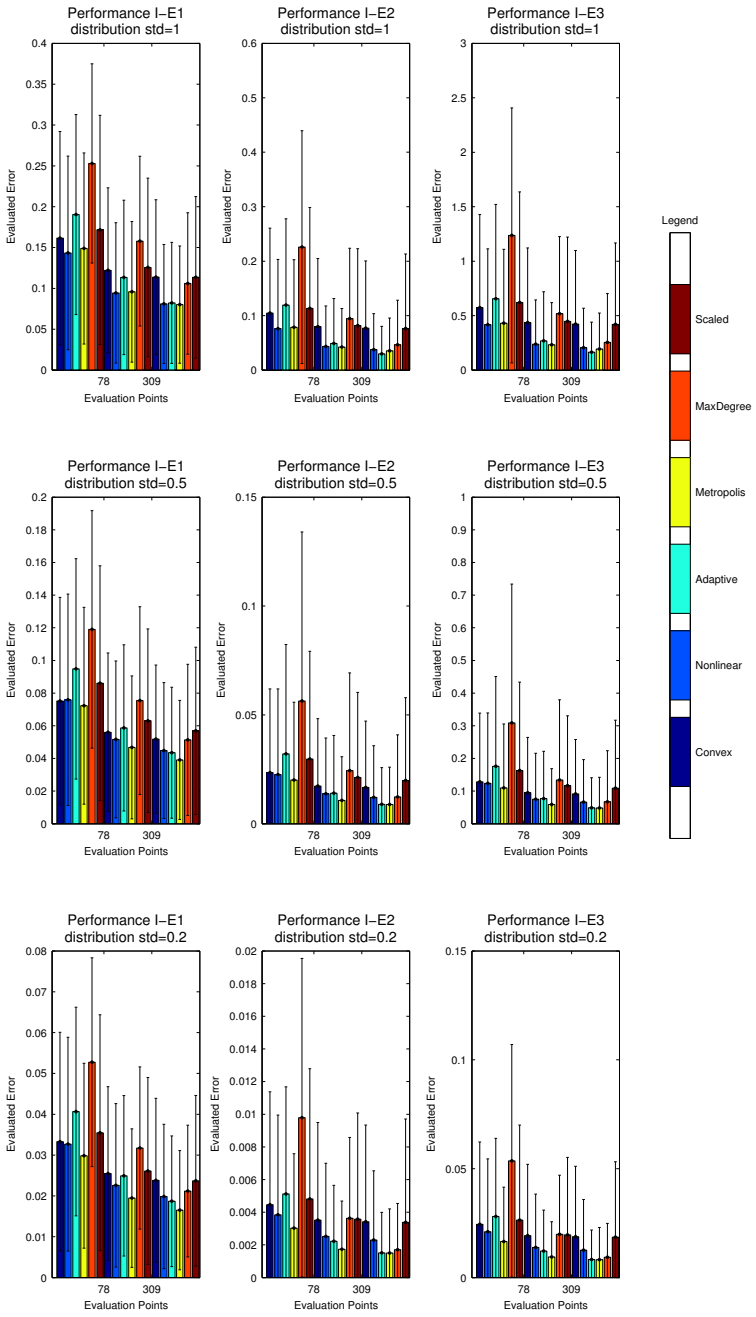


Figure 51: **Instantaneous errors wrt different initial standard deviation in small graphs**.The errors ϵ_1, ϵ_2, ϵ_3 computed over 10 random graphs of 20 vertices, are given. The initial standard deviation was 1, 0.5 and 0.2 with mean equal to 10.

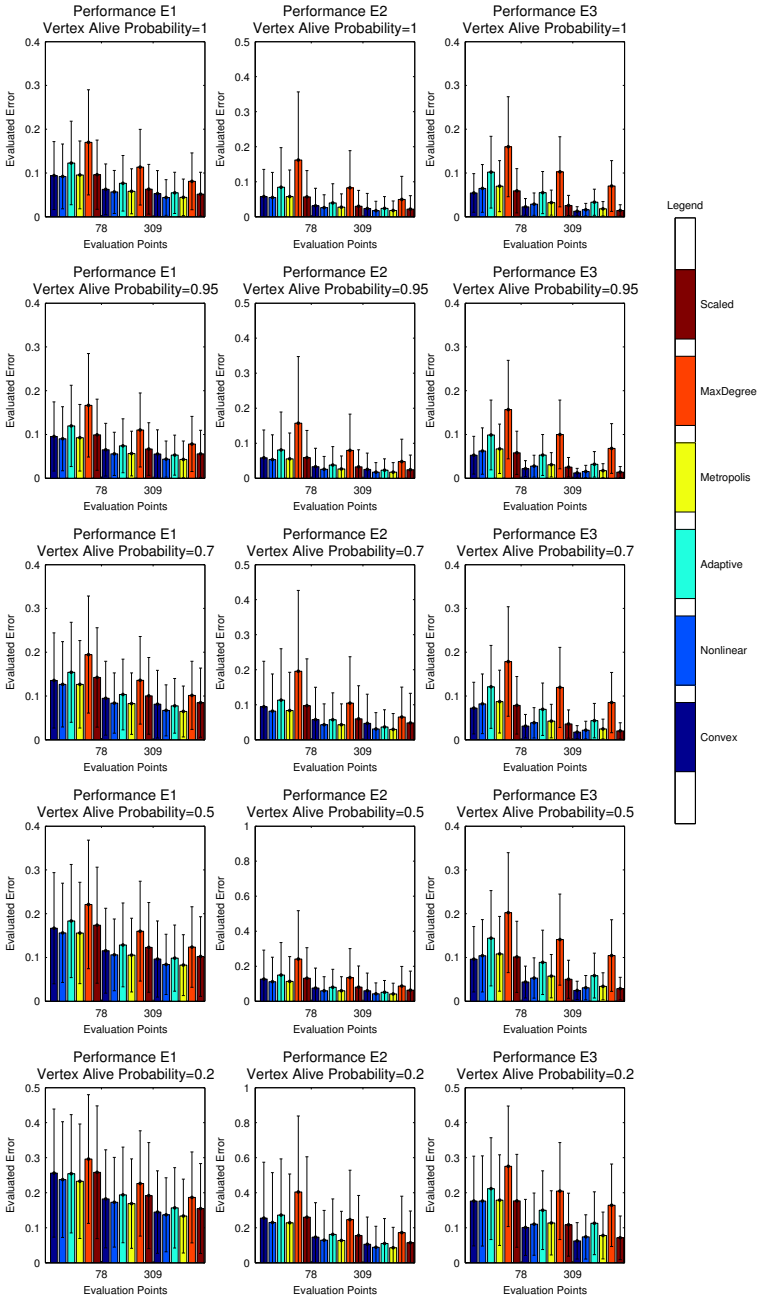


Figure 52: **Overall errors wrt different alive probabilities in small graphs**. The errors E_1, E_2, E_3 computed over 10 random graphs of 20 vertices, are given in three segments, [0 78], [0 309], [0 700]. We have tested against different vertex alive probabilities, 1, 0.95, 0.7, 0.5, 0.2 .

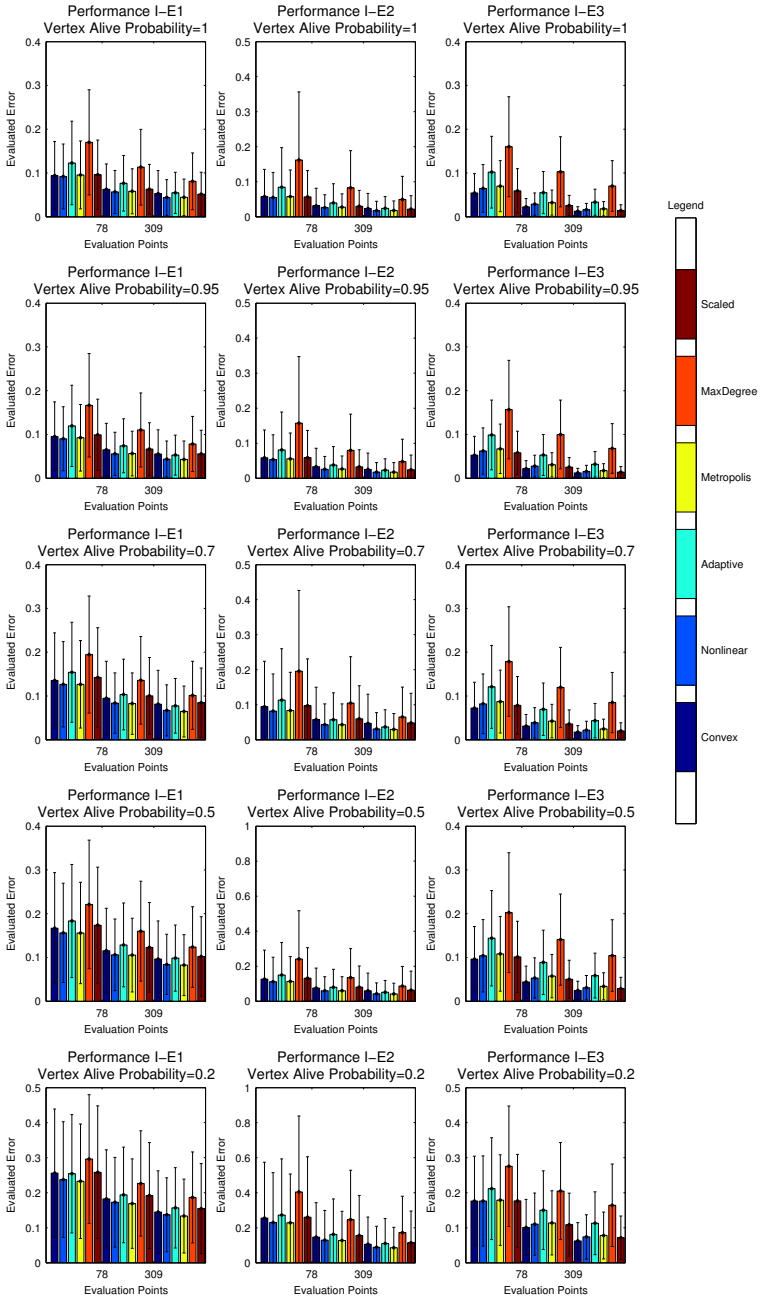


Figure 53: **Instantaneous errors wrt vertex alive probability in small graphs**. The errors ϵ_1, ϵ_2, ϵ_3 computed over 10 random graphs of 20 vertices, are given in three segments, [0 78], [0 309], [0 700]. We have tested against different vertex alive probabilities, 1, 0.95, 0.7, 0.5, 0.2 .

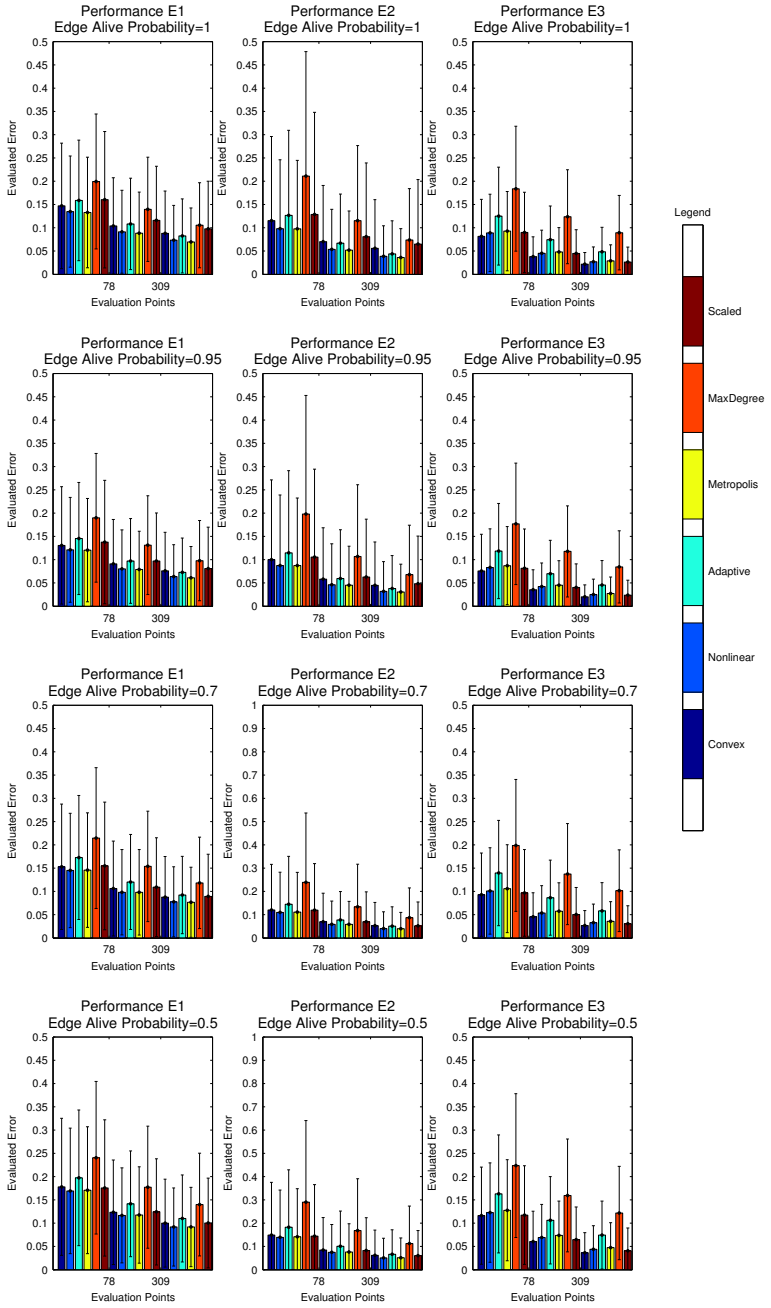


Figure 54: **Overall errors wrt to edge alive probability in small graphs**. The errors E_1, E_2, E_3 computed over 10 random graphs of 20 vertices, are given in three segments, [0 78], [0 309], [0 700]. We have tested against different vertex alive probabilities, 1, 0.95, 0.7, 0.5, 0.2 .

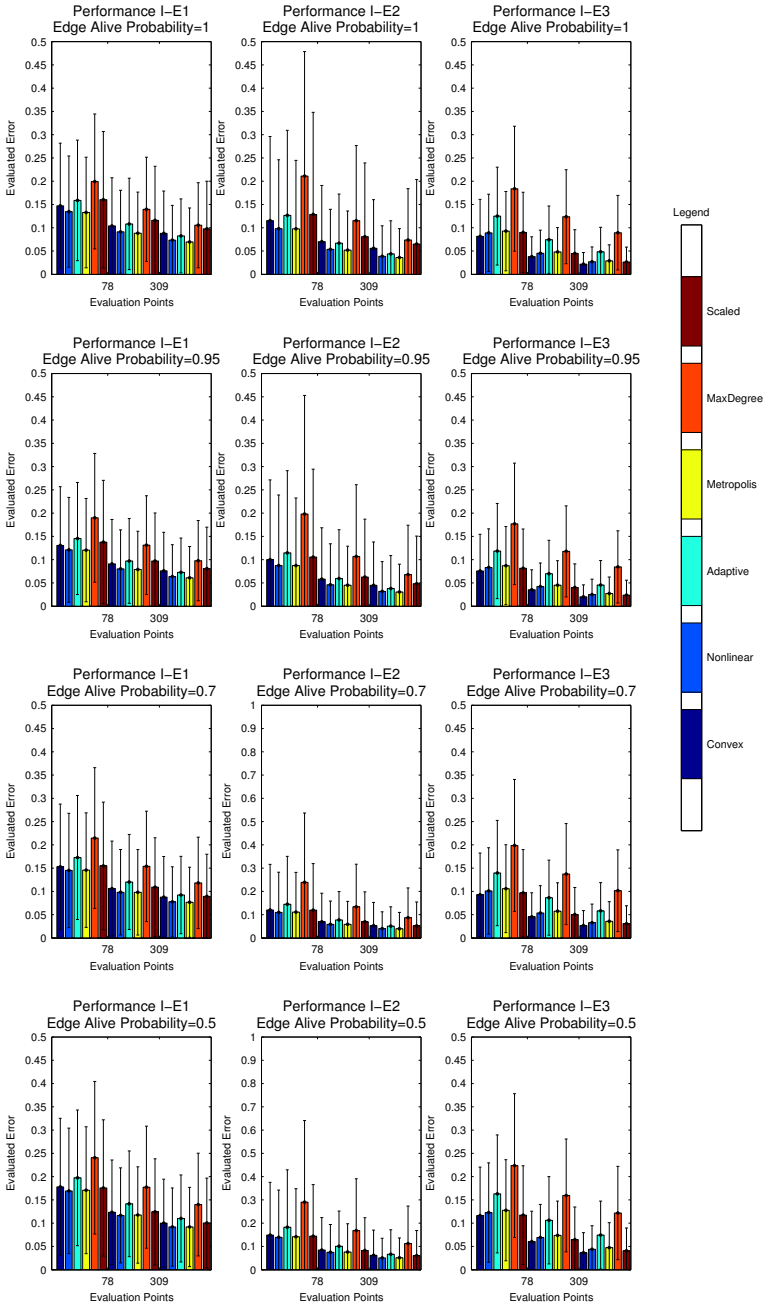


Figure 55: **Instantaneous errors wrt to edge alive probability in small graphs**. The errors ϵ_1, ϵ_2, ϵ_3 computed over 10 random graphs of 20 vertices, are given in three segments, [0 78], [0 309], [0 700]. We have tested against different vertex alive probabilities, 1, 0.95, 0.7, 0.5, 0.2 .

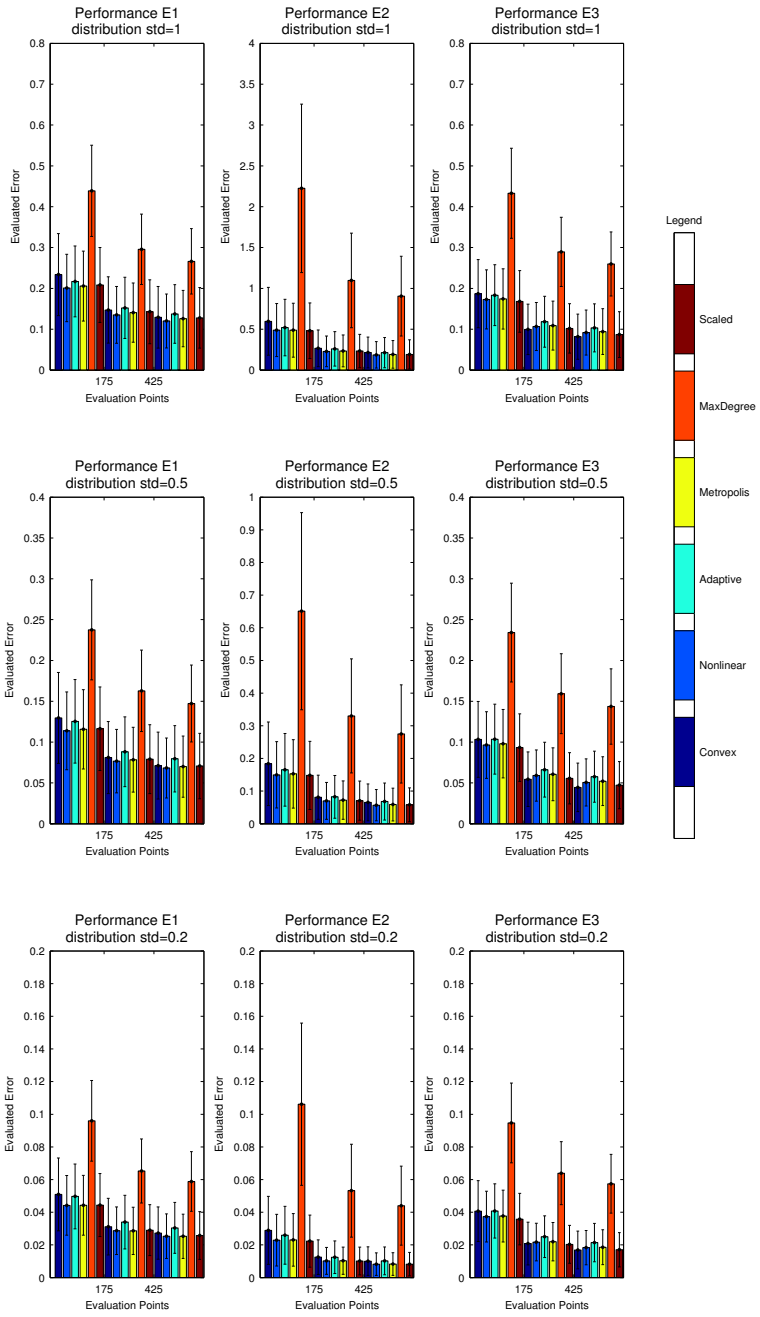


Figure 56: **Overall errors wrt different initial standard deviation in medium graph**. The errors E_1, E_2, E_3 computed over 10 random graphs of 20 vertices, are given. The initial standard deviation was 1, 0.5 and 0.2 with mean equal to 10.

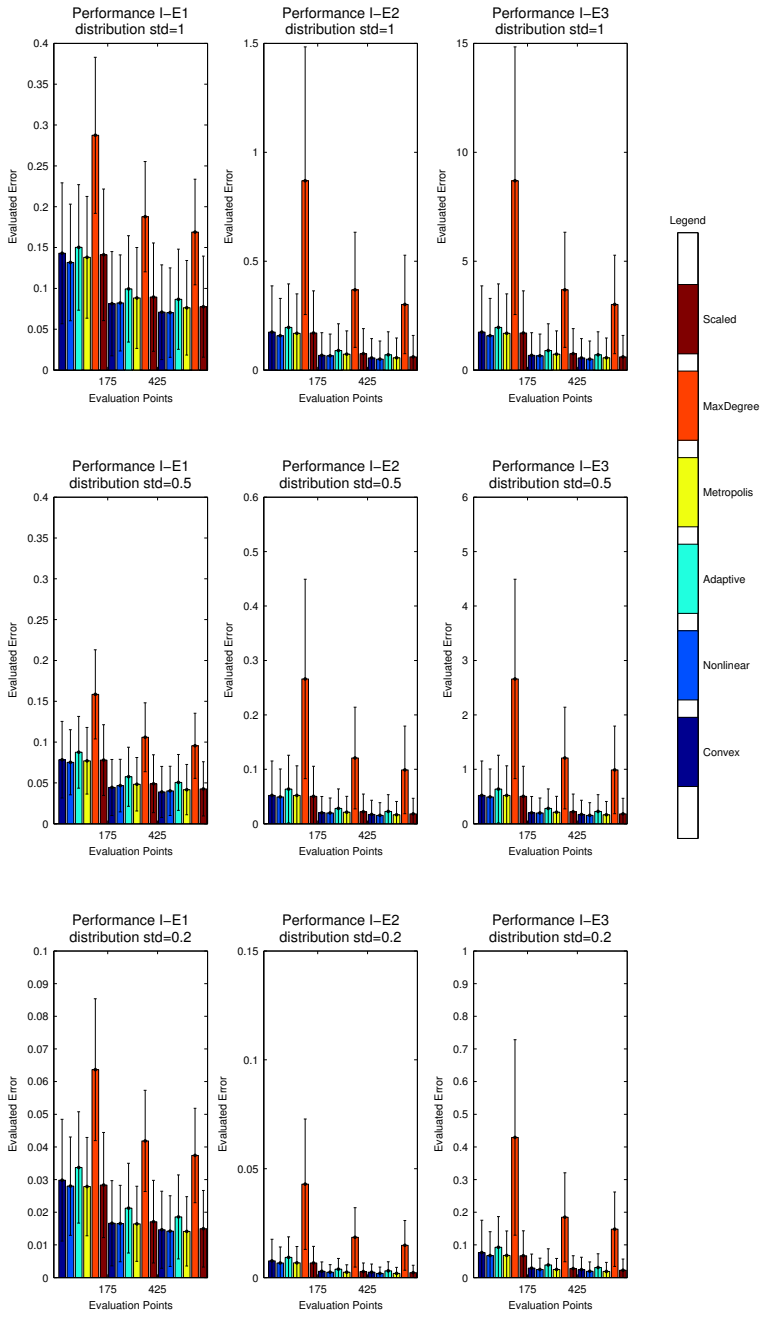


Figure 57: **Instantaneous errors wrt different initial standard deviation in medium graphs**.The errors ϵ_1, ϵ_2, ϵ_3 computed over 10 random graphs of 20 vertices, are given. The initial standard deviation was 1, 0.5 and 0.2 with mean equal to 10.

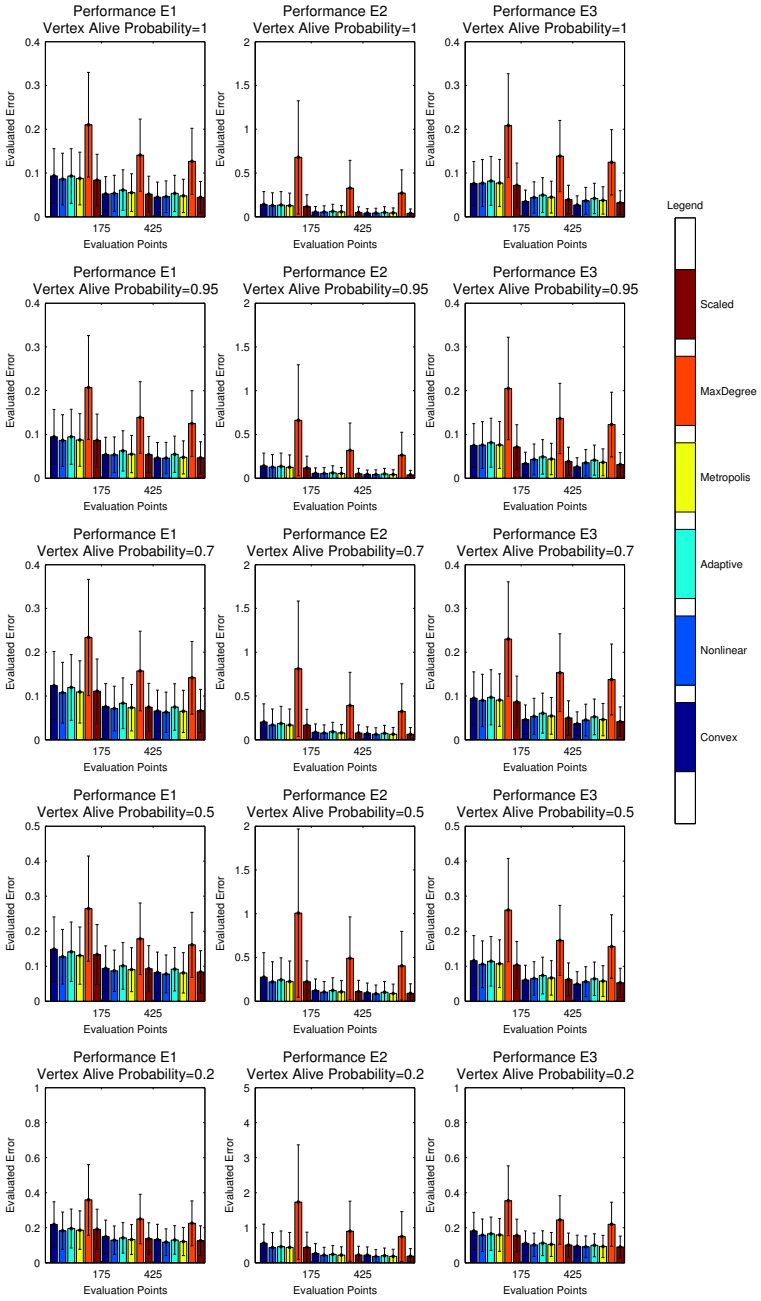


Figure 58: **Overall errors wrt different alive probabilities in medium graphs**. The errors E_1, E_2, E_3 computed over 10 random graphs of 20 vertices, are given in three segments, [0 78], [0 309], [0 700]. We have tested against different vertex alive probabilities, 1, 0.95, 0.7, 0.5, 0.2 .

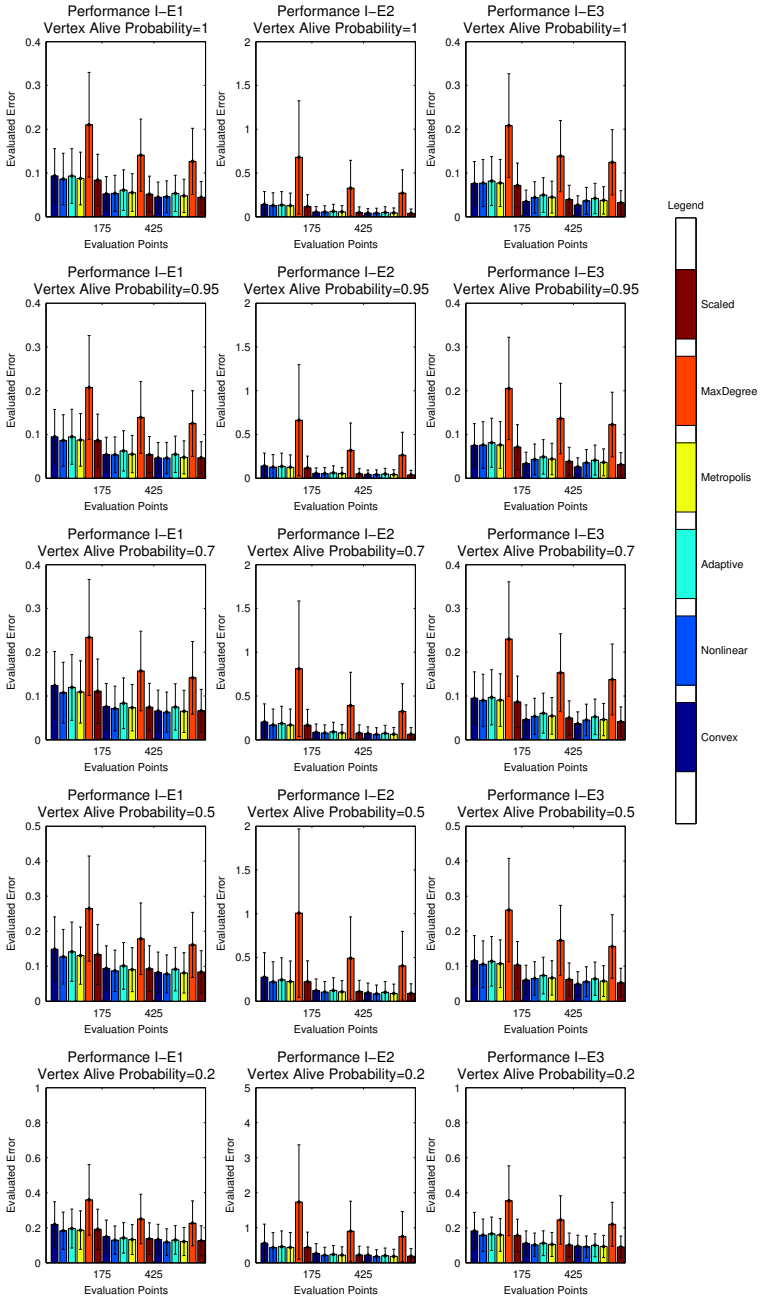


Figure 59: **Instantaneous errors wrt vertex alive probability in medium graphs**. The errors ϵ_1, ϵ_2, ϵ_3 computed over 10 random graphs of 20 vertices, are given in three segments, [0 78], [0 309], [0 700]. We have tested against different vertex alive probabilities, 1, 0.95, 0.7, 0.5, 0.2 .

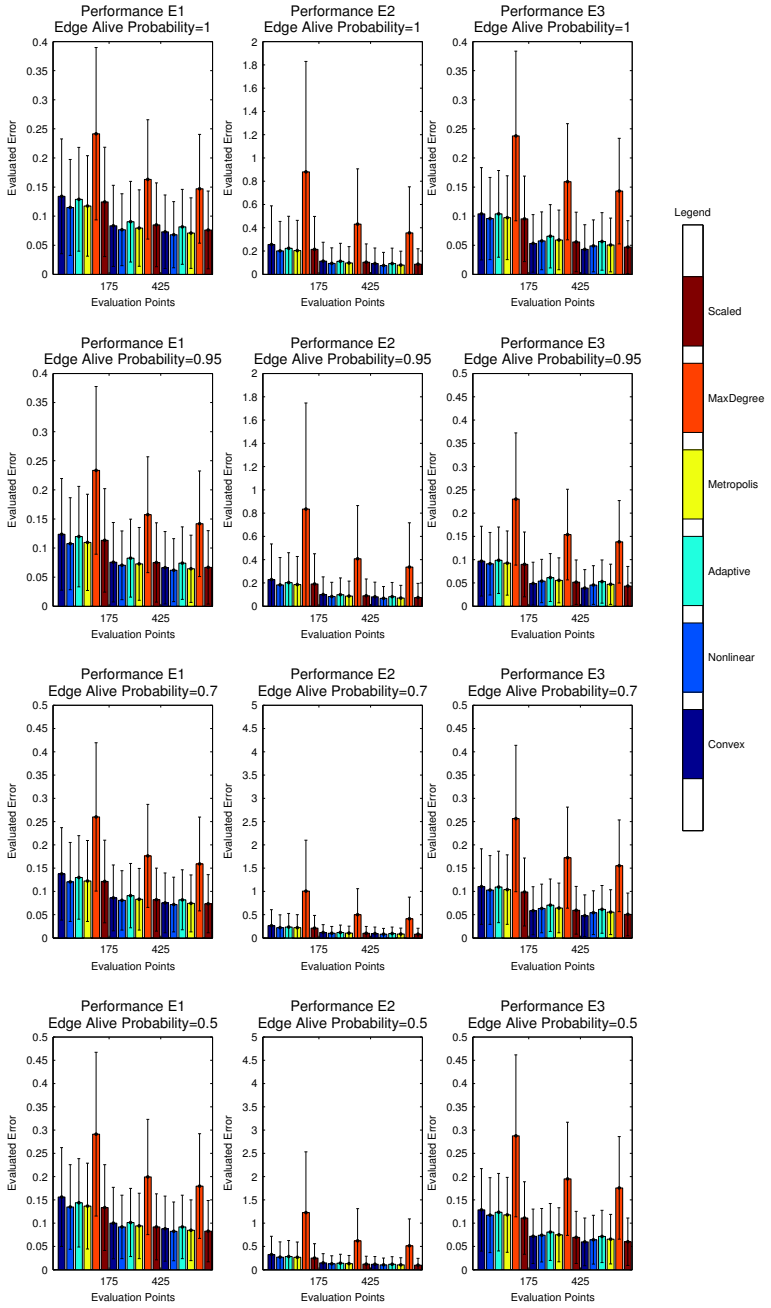


Figure 60: **Overall errors wrt to edge alive probability in medium graphs**. The errors E_1, E_2, E_3 computed over 10 random graphs of 20 vertices, are given in three segments, [0 78], [0 309], [0 700]. We have tested against different vertex alive probabilities, 1, 0.95, 0.7, 0.5, 0.2 .

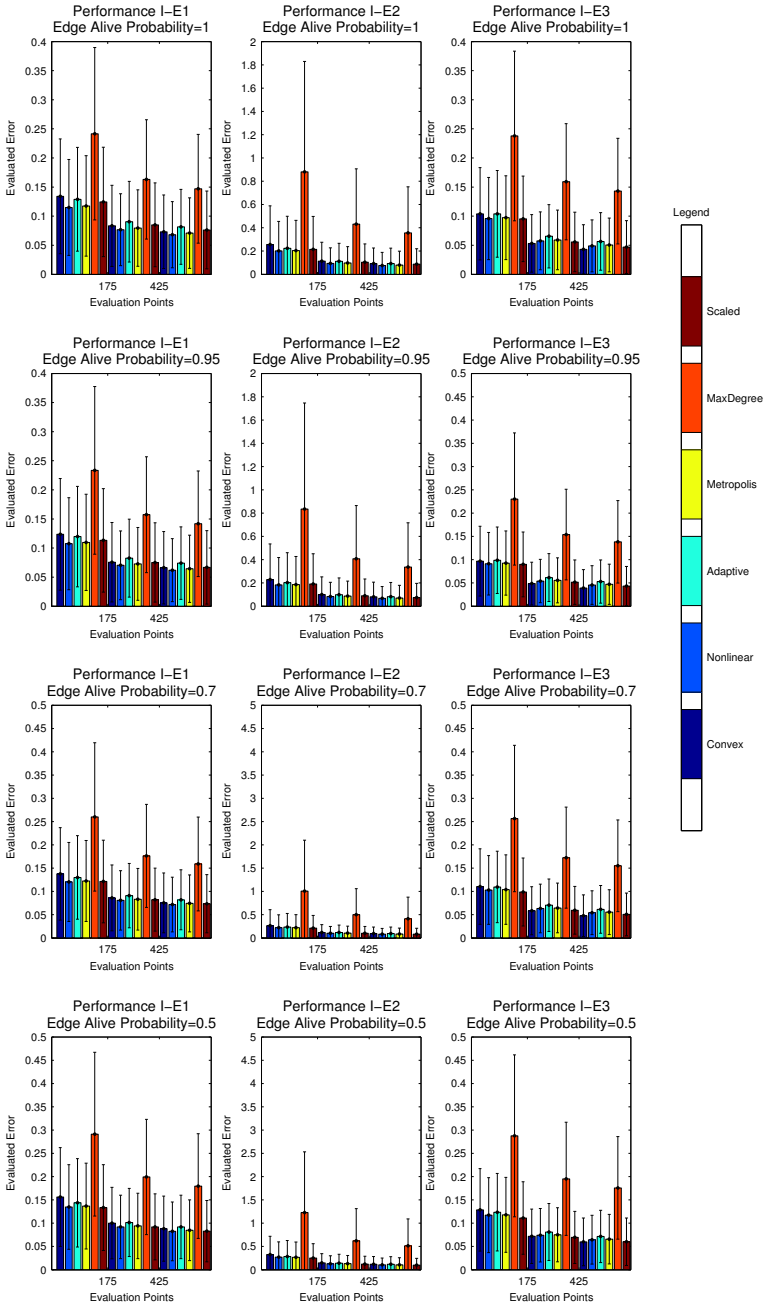


Figure 61: **Instantaneous errors wrt to edge alive probability in medium graphs**. The errors ϵ_1, ϵ_2, ϵ_3 computed over 10 random graphs of 20 vertices, are given in three segments, [0 78], [0 309], [0 700]. We have tested against different vertex alive probabilities, 1, 0.95, 0.7, 0.5, 0.2 .

C

DEFINITIVE CONSENSUS SOLUTIONS

The solutions herein are for equation $\prod_{k=d}^{n} \mathbf{W}_k = \mathbf{1}\mathbf{1}^T$ instead of $\frac{\mathbf{1}\mathbf{1}^T}{n}$, in order to simplify the matrices. The solutions $\frac{\mathbf{1}\mathbf{1}^T}{n}$ can be obtained by dividing any of the matrices with $\frac{1}{n}$.

C.1 STAR GRAPHS

Any star graph can be at absolute consensus by application of the following two matrices.

$$\mathbf{W}_1 = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{W}_2 = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

C.2 CYCLES

C.2.1 *The Pentagon*

An exact solution can be obtained for the pentagon as described in (*Section* 4.2.4.1)

$$\mathbf{W}_1 = \begin{pmatrix} \frac{1}{2}(1-\sqrt{5}) & 1 & 0 & 0 & 1 \\ 1 & \frac{1}{2}(1-\sqrt{5}) & 1 & 0 & 0 \\ 0 & 1 & \frac{1}{2}(1-\sqrt{5}) & 1 & 0 \\ 0 & 0 & 1 & \frac{1}{2}(1-\sqrt{5}) & 1 \\ 1 & 0 & 0 & 1 & \frac{1}{2}(1-\sqrt{5}) \end{pmatrix}$$

$$\mathbf{W}_2 = \begin{pmatrix} \frac{1}{2}(1+\sqrt{5}) & 1 & 0 & 0 & 1 \\ 1 & \frac{1}{2}(1+\sqrt{5}) & 1 & 0 & 0 \\ 0 & 1 & \frac{1}{2}(1+\sqrt{5}) & 1 & 0 \\ 0 & 0 & 1 & \frac{1}{2}(1+\sqrt{5}) & 1 \\ 1 & 0 & 0 & 1 & \frac{1}{2}(1+\sqrt{5}) \end{pmatrix}$$

C.2.2 *The Hexagon*

$$\mathbf{W}_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

$$\mathbf{W}_2 = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 1 \\ 1 & -1 & 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 1 \\ 1 & 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

$$\mathbf{W}_3 = \begin{pmatrix} 1 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 1 \end{pmatrix}$$

NOMENCLATURE

$[x]$	integer part of x
$\pi\{v_i, v_j\}$	denotes the shortest path $p\{v_i, \ldots, v_j\}$
π_{ij}	denotes the shortest path $p\{i, \ldots, j\}$
$\mathbf{A}$	adjacency matrix
iff	if and only if
$\leftarrow$	left side of $\leftarrow$ is assigned the value on the right side
$c\{v_i, \ldots, v_j\}$	a cycle from v_i to v_j
$\cdot^*$	A designated object or a complex conjugate number
m	the cardinality of the edge set, if not otherwise noted
$\mathcal{E}$	denotes an edge set
$\circ$	element-wise matrix product, know also as *Hadamard product*
γ	denotes a graph incidence function
$\mathcal{G}(\mathcal{E}^*)$	edge induced subgraph of $\mathcal{G}$
$\mathcal{G}(\mathcal{V}^*)$	vertex induced subgraph of $\mathcal{G}$
$\mathcal{G}^*$	induced subgraph of $\mathcal{G}$
$\mathcal{G}$	denotes a graph
$\mathbf{Q}$	the incidence matrix
$lc(f)$	leading coefficient of a polynomial f
$lp(f)$	leading power product of a polynomial f
$\rightarrow$	left side of $\leftarrow$ is mapped to the value on the right side
$[\mathbf{x}]_{ij}$	denotes the i^{th} row and j^{th} column of some matrix $\mathbf{x}$

$\mathbf{x}^{\mathbf{a}}$ a monomial term of degree $\mathbf{a}$

$\mathbf{1}$ the all one vector $(1, 1, \ldots, 1)^T$

$p\{v_i, \ldots, v_j\}$ a path from v_i to v_j

$\rho(\cdot)$ spectral norm

$t\{v_i, \ldots, v_j\}$ a trail from v_i to v_j

$\mathbf{x} > \mathbf{y}$ element-wise vector comparison, $x_i > y_i$

n the cardinality of the vertex set, if not otherwise noted

$\mathcal{V}$ denotes a vertex set

$w\{v_i, \ldots, v_j\}$ a walk from v_i to v_j

$\mathbf{0}$ all zeros vector $(0, 0, \ldots, 0)^T$

GLOSSARY

ad-hoc (network)	a network without routing of packets, 6
ASTD	average standard deviation, 161
cardinality	the number of elements in a group, 16
CER	classification error rate, 144
CGS	Connection Graph Stability, 47
connected digraph	the associated undirected graph is connected, 25
connected graph	a graph where for any two vertices exists a path, 20
cycle	a closed path, 20
degree	vertex degree, 33
digraph	directed graph, 24
edge coefficients	edge weights, 38
edge weight	a coefficients associated with an edge, 32
hop	is the distance connecting two machines able to communicate in an ad-hoc network, 6
IAMD	initial arithmetic mean deviation, 161
incidence function	a function that maps an edge to the vertices it relates, 17
information consensus	a distributed process that leads to the participating nodes in agreement on a given scalar value, 3
Laplacian	the Laplacian matrix, 32
loop	the edge $\{v_i, v_i\}$, 24

MDST	minimum diameter spanning tree, 96
MH-weights	Metropolis-Hastings weights, 46
modulus	the absolute value of a real number, or the square root of a complex's product with its conjugate, 39
neighbour	of a machine is a machine that is one-hop away, 6
path	a walk without duplicate vertices, 20
process consensus	a set of independent distributed processes that terminates simultaneously, 3
RMSE	root mean square error, 144
self-edge	the edge $\{v_i, v_i\}$, 17
spanning tree	a spanning subgraph that is a tree, 27
spectral gap	the modulus of the smallest non-zero eigenvalue of the laplacian, 64
strongly connected	any two vertices are connected with a directed path, 25
TMAD	true mean average deviation, 161
trail	a walk without duplicate edges, 20
tree	a graph without cycles, 27
UIDP	Class of problems where data is: Unobtainable, Incomputable, Distributed, Private, 4
unicyclic graph	a graph with exactly one cycle, 27
vertex degree	number of adjacent edges, 33

walk	an interlacing sequence adjacent vertices and edges, 20
weight	a coefficient associated with an edge, 31
WSN	Wireless Sensor Network, 3

BIBLIOGRAPHY

William W. Adams and Philippe Loustaunau. *An Introduction to Gröbner Bases*. Number 3 in Graduate Studies in Math. Amer. Math. Soc., New York, 1994.

R.P. Agarwal, M. Meehan, and D. O'Regan. *Fixed point theory and applications*. Cambridge Univ Pr, 2001. ISBN 0521802504.

David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home: an experiment in public-resource computing. *Commun. ACM*, 45 (11):56–61, November 2002. ISSN 0001-0782. doi: 10.1145/581571.581573. URL http://dx.doi.org/10.1145/581571.581573.

D.P. Anderson. BOINC: A system for public-resource computing and storage. In *proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10. IEEE Computer Society, 2004. ISBN 0769522564.

Hock Hee Ang, Vivekanand Gopalkrishnan, Steven C. H. Hoi, and Wee Keong Ng. Cascade RSVM in peer-to-peer networks. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part I*, volume 5211 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 2008. ISBN 978-3-540-87478-2. URL http://dx.doi.org/10.1007/978-3-540-87479-9_22.

J. M. Anthonisse. The Rush In A Directed Graph. Technical report, NARCIS, 1971. URL http://oai.cwi.nl/oai/asset/9791/9791A.pdf. bibliographical data to be processed – Stichting Mathematisch Centrum. Mathematische Besliskunde ; BN 9/71. ISSN: – Stichting Mathematisch Centrum – 10 p.

Vladimir N. Belykh, Igor V. Belykh, and Martin Hasler. Connection graph stability method for synchronized coupled chaotic systems. *Physica D: Nonlinear Phenomena*, 195(1-2):159 – 187, 2004. ISSN 0167-2789. doi: DOI:10.1016/j.physd.2004.03.012. URL http://www.sciencedirect.com/science/article/B6TVK-4CNJD3M-3/2/13620fb652443d396fbfbb0a5fd16823.

Florence Benezit. *Distributed average consensus for wireless sensor networks*. PhD thesis, Information and Communications Sciences, Lausanne, 2009. URL http://library.epfl.ch/theses/?nr=4509.

Florence Benezit, Patrick Thiran, and Martin Vetterli. Interval consensus: from quantized gossip to voting. In *IEEE, ICASSP 2009*, pages 3661 – 3664, 2009. URL http://www.icassp09.com/default.asp.

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

Dimitri P. Bertsekas. A new class of incremental gradient methods for least squares problems. *SIAM J. Optim*, 7:913–926, 1996.

N. L. Biggs. *Algebraic Graph Theory*. Cambridge University Press, Cambridge, 1974. ISBN 0-521-20335-X.

Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1 edition, January 1996. ISBN 0198538642. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0198538642.

V D Blondel, J M Hendrickx, A Olshevsky, and J N. Tsitsiklis. convergence in multiagent coordination, consensus, and flocking. In *In Joint 44th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC '05*, 2005.

BOINC. Boinc website. http://boinc.berkeley.edu/.

V Borkar and P Varaiya. Asymptotic agreement in distributed estimation. *IEEE Trans. Auto. Control*, 27:16, 1982.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521833787.

Bruno Buchberger. An algorithmic criterion for the solvability of a system of algebraic equations. In Bruno Buchberger and Franz Winkler, editors, *Gröbner Bases and Applications*, volume 251 of *London Mathematical Society Lecture Note Series*, pages 535–545. Cambridge University Press, Cambridge, 1998.

Bui, Butelle, and Lavault. A distributed algorithm for constructing a minimum diameter spanning tree. *JPDC: Journal of Parallel and Distributed Computing*, 64, 2004.

R.L.G. Cavalcante, I. Yamada, and B. Mulgrew. An adaptive projected subgradient approach to learning in diffusion networks. *Signal Processing, IEEE Transactions on*, 57(7):2762 –2774, july 2009. ISSN 1053-587X. doi: 10.1109/TSP.2009.2018648.

Gordon Cormack and Thomas Lynam. Spam corpus creation for trec. In *Stanford University*, 2005.

O.E. Dictionary. Oxford English dictionary online. *Mount Royal College Lib., Calgary*, 14, 2004.

D.V. Dimarogonas and K.J. Kyriakopoulos. Connectedness preserving distributed swarm aggregation for multiple kinematic robots. *Robotics, IEEE Transactions on*, 24(5):1213 –1223, oct. 2008. ISSN 1552-3098. doi: 10.1109/TRO.2008.2002313.

Rick Durrett. *Random Graph Dynamics (Cambridge Series in Statistical and Probabilistic Mathematics)*. Cambridge University Press, New York, NY, USA, 2006. ISBN 0521866561.

EGEE. Egee website. http://www.eu-egee.org/.

P. Erdos and A. Renyi. On random graphs. *Publ. Math. Debrecen*, 6:290–297, 1959.

P. Erdos and A. Renyi. On the strength of connectedness of a random graph. *Acta Mathematica Hungarica*, 12:261–267, 1961.

P. Erdos and A. Rényi. On random graphs. *Publ. Math. Debrecen*, 6:290–291, 1959.

Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases F_4. *Journal of Pure and Applied Algebra*, 139(1-3):61–88, 1999. URL http://www.sciencedirect.com/.

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985. ISSN 0004-5411. doi: 10.1145/3149.214121. URL http://dx.doi.org/10.1145/3149.214121.

L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40:35–41, 1977.

F. Gagliardi, B. Jones, F. Grey, M.E. Bégin, and M. Heikkurinen. Building an infrastructure for scientific Grid computing: status and goals of the EGEE project. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 363(1833):1729, 2005. ISSN 1364-503X.

John Geanakopolos. We can't disagree forever. *Journal of Economic Theory*, 28(1): 192–200, 1982.

L. Georgopoulos and M. Hasler. Nonlinear average consensus. *In Proceedings of the 2009 International Symposium on Nonlinear Theory and its Applications*, pages 12–13, 2009a.

Leonidas Georgopoulos and Martin Hasler. Early consensus in complex networks under variable graph topology. In *Proceedings of the European Conference on Circuit Theory and Design 2009*, pages 575–578, Piscataway NJ, 2009b. IEEE. URL http://ecctd09.dogus.edu.tr/.

GLOBuS. Globus website. http://www.globus.org/.

Chris Godsil and Gordon Royle. *Algebraic Graph Theory*, volume 207 of *Graduate Texts in Mathematics*. Springer, April 2001. ISBN 0387952209. URL http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/0387952209.

V. Gupta, B. Hassibi, and R.M. Murray. On sensor fusion in the presence of packet-dropping communication channels. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, pages 3547 – 3552, 12-15 2005.

Hassin and Tamir. On the minimum diameter spanning tree problem. *IPL: Information Processing Letters*, 53, 1995.

A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2001.

B Hendrickson and T G Kolda. Graph partitioning models for parallel computing. *Parallel Comput*, 26:1519–1534, 2000.

IBM-WCMG. World community grid website. http://www.worldcommunitygrid.org/.

A. Jadbabaie, A. Ozdaglar, and M. Zargham. A distributed newton method for network optimization. In *Decision and Control, 2009 held jointly with the 2009*

28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on, pages 2736 –2741, 15-18 2009. doi: 10.1109/CDC.2009.5400289.

B. Johansson, T. Keviczky, M. Johansson, and K.H. Johansson. Subgradient methods and consensus algorithms for solving convex optimization problems. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 4185 –4190, 9-11 2008. doi: 10.1109/CDC.2008.4739339.

Mark Kac. Can one hear the shape of a drum? *The American Mathematical Monthly*, 73:1–23, April 1966.

Alireza Khadivi and Martin Hasler. Fire Detection and Localization Using Wireless Sensor Networks. *Fire Detection and Localization Using Wireless Sensor Networks*, 2010. doi: 10.1007/978-3-642-11870-8.

Effrosini Kokiopoulou and Pascal Frossard. Graph-based classification of multiple observation sets. *Pattern Recognition*, 43(12):3988–3997, 2010. URL http://dx.doi.org/10.1016/j.patcog.2010.07.016.

Wojtek Kowalczyk and Nikos Vlassis. Newscast em. In *In NIPS 17*, pages 713–720. MIT Press, 2005.

Lamport. The weak byzantine generals problem. *JACM: Journal of the ACM*, 30, 1983.

Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4:382–401, 1982.

Yumao Lu, V. Roychowdhury, and L. Vandenberghe. Distributed parallel support vector machines in strongly connected networks. *Neural Networks, IEEE Transactions on*, 19(7):1167 –1178, july 2008. ISSN 1045-9227. doi: 10.1109/TNN.2007.2000061.

Nancy A. Lynch. *Distributed Algorithms (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 1st edition, 1996. ISBN 1558603484. URL http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/1558603484.

L. Moreau. Stability of multiagent systems with time-dependent communication links. *Automatic Control, IEEE Transactions on*, 50(2):169 – 182, feb. 2005. ISSN 0018-9286. doi: 10.1109/TAC.2004.841888.

A. Navia-Vazquez, D. Gutierrez-Gonzalez, E. Parrado-Hernandez, and J.J. Navarro-Abellan. Distributed support vector machines. *Neural Networks, IEEE Transactions on*, 17(4):1091 – 1097, july 2006. ISSN 1045-9227. doi: 10.1109/TNN.2006.875968.

A. Nedic, A. Ozdaglar, and P.A. Parrilo. Constrained consensus and optimization in multi-agent networks. *Automatic Control, IEEE Transactions on*, 55(4):922 – 938, april 2010. ISSN 0018-9286. doi: 10.1109/TAC.2010.2041686.

M. E. J. Newman and D. J. Watts. Renormalization group analysis of the small-world network model. *Physics Letters A*, 263:341–346, 1999.

D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *1990 IJCNN International Joint Conference on Neural Networks, 1990.*, pages 21–26, 1990.

R. Olfati-Saber. Flocking for multi-agent dynamic systems: algorithms and theory. *Automatic Control, IEEE Transactions on*, 51(3):401 – 420, march 2006. ISSN 0018-9286. doi: 10.1109/TAC.2005.864190.

R. Olfati-Saber and R.M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *Automatic Control, IEEE Transactions on*, 49(9):1520–1533, Sept. 2004. ISSN 0018-9286. doi: 10.1109/TAC.2004.834113.

Reza Olfati-saber, J. Alex Fax, and Richard M. Murray. Consensus and cooperation in networked multi-agent systems. In *Proceedings of the IEEE*, page 2007, 2007.

A. Olshevsky and J. N. Tsitsiklis. Convergence Speed in Distributed Consensus and Control. *ArXiv Mathematics e-prints*, December 2006.

OpenScienceGrid. Open science grid website. http://www.opensciencegrid.org/.

OurGrid. Our grid website. http://www.ourgrid.org/.

M. Penrose. *Random Geometric Graphs*. Oxford University Press, Oxford, 2003.

Michael Rabbat and Robert Nowak. Distributed optimization in sensor networks. In *In 3rd Int. Symp. on Information Processing in Sensor Networks (IPSN 04*, pages 20–27, 2004.

Maneesha V. Ramesh, Sangeeth Kumar, and P. Venkat Rangan. Wireless sensor network for landslide detection. In Hamid R. Arabnia and Victor A. Clincy, editors, *Proceedings of the 2009 International Conference on Wireless Networks, ICWN 2009, July 13-16, 2009, Las Vegas Nevada, USA, 2 Volumes*, pages 89–95. CSREA Press, 2009. ISBN 1-60132-113-9.

Wei Ren, Haiyang Chao, W. Bourgeous, N. Sorensen, and YangQuan Chen. Experimental validation of consensus algorithms for multivehicle cooperative control. *Control Systems Technology, IEEE Transactions on*, 16(4):745 –752, july 2008. ISSN 1063-6536. doi: 10.1109/TCST.2007.912239.

R. O. Saber and R. M. Murray. Consensus protocols for networks of dynamic agents. In *American Control Conference, 2003. Proceedings of the 2003*, volume 2, pages 951–956, 2003. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1239709.

SETI@HOME. Seti at home website. http://setiathome.berkeley.edu/.

W. Strunk. *The elements of style*. Douglas Editions, 1930.

Sturmfels. WHAT IS...a grobner basis? *NOTICES: Notices of the American Mathematical Society*, 52, 2005.

H. G. Tanner, A. Jadbabaie, and G. J. Pappas. Flocking in fixed and switching networks. *Automatic Control, IEEE Transactions on*, 52(5):863 –868, may 2007. ISSN 0018-9286. doi: 10.1109/TAC.2007.895948.

J. Tsitsiklis, D. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *Automatic Control, IEEE Transactions on*, 31(9):803–812, 1986. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1104412.

John N Tsitsiklis. *Problems in decentralized decision making and computation*. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1984. URL http://hdl.handle.net/1721.1/15254.

Xiao Fan Wang and Guanrong Chen. Complex networks: small-world, scale-free and beyond. *Circuits and Systems Magazine, IEEE*, 3(1):6 – 20, 2003. ISSN 1531-636X. doi: 10.1109/MCAS.2003.1228503.

D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

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*, pages 63 – 70, 15 2005. doi: 10.1109/IPSN.2005.1440896.

Lin Xiao and S. Boyd. Fast linear iterations for distributed averaging. *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, 5:4997–5002 Vol.5, Dec. 2003. ISSN 0191-2216. doi: 10.1109/CDC.2003.1272421.

Lin Xiao, Stephen Boyd, and Seung-Jean Kim. Distributed average consensus with least-mean-square deviation. *J. Parallel Distrib. Comput.*, 67(1):33–46, 2007. ISSN 0743-7315. doi: http://dx.doi.org/10.1016/j.jpdc.2006.08.010.

J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, pages 139 – 148, 11 2003. doi: 10.1109/SNPA.2003.1203364.

Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard SchG'Alkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems 16*, pages 321–328. MIT Press, 2004.

Leonidas Georgopoulos

Av. de l' Eglise Anglaise 12
Lausanne 1006 +41786272711
Switzerland georgopl@gmail.com

Education

Ecole Polytechnique Federale de Lausanne Lausanne, Suisse
(PhD, Computer and Communications Sciences) *Aug. 2007- June 2011*
- Subject: Distributed Information Processing

IDIAP Research Institute Martigny, Suisse
(Internship under Scholarship) *Jan. 2007 - July 2007*
- Award: Bourse de la Confederation Suisse pour etudiants etrangers
- Subject of Research: Machine Learning, Visual Reconstruction

The Edinburgh University Scotland, UK
M.Sc., Artificial Intelligence *Sep. 2004 - Sep. 2005*
- Award: Outstanding MSc Project Prize in Informatics class of 2005
- Subject: Robotics, Machine Learning, Neural Computation

University of Athens Athens, Greece
B.Sc., Physics (Astrophysics Option *Oct. 1997- Nov. 2004*

Distinctions

Student Paper Award in International Symposium on Nonlinear Theory and its Application	2009
Machine Learning Summer School	2007
Bourse de la Confederation Suisse pour etudiants etrangers	2007
Outstanding MSc Project Prize	2005

Work Experience

Greek Armed Forces — Greece

SAM/RADAR operator — *Feb. 2006 - Feb. 2007*

- Mandatory army service (1 year)

The University of Edinburgh — Scotland, UK

Teaching Assistant — *Nov. 2004 - Apr. 2005*

- Subject: Physics

Athens 2004 — Athens, Greece

Unix Systems Engineer — *June 2004 - Aug. 2004*

- Venue: Athens Olympic Rowing Center

ALTEC S.A. (Athens 2004 Contractor) — Athens, Greece

Unix Systems Engineer — *Oct. 2003 - Sep. 2004*

- Department: Operations
- Subject: Pre-sales Systems Design, Systems Installation, 2nd level Support, Unix Systems

K.A. S.A. — Athens, Greece

IT Systems Advisor — *2001 - 2004*

Agricultural House of Spyrou — Athens, Greece

Senior Systems Administrator — *Sep. 2000 - Mar. 2002*

- Department: IT
- Subject: Systems Administrator, DBA, Network Administrator

Multimedia S.A (part of Lambrakis Press Group of companies) — Athens, Greece

Junior Systems Administrator — *Arp. 1998 - Sep. 2000*

- Department: Operations
- Subject: Installation, 1st and 2nd level support, Unix, Windows Servers

Language Skills

BULATS	68%	2011
DELF	60%	2006
TOEFL	284/300	2004
Cambridge Lower Certificate	A*	1998

Activities and Memberships

MEMBERSHIPS: Organising committee of Forum EPFL 2010 (Enterprise Relations) , Alumni of The University of Edinburgh

ACTIVITIES: Venture Challenge 2009 at EPFL, Junior Systems Administrator at EPFL-LANOS, Teaching Assistant of Network Security, Teaching Assistant of Pattern Classification and Machine Learning

Academic Interests

Distributed Information Processing, Algorithm Development for Large Scale Networks, Machine Learning on Graphs, Consensus Protocols

Interests

SPORTS: Swimming, Snowboard, Cycling, Basketball

GAMES: Chess, Backgammon

CULTURE: Theatre, Movies, Literature, Philosophy

OTHER: Astronomy, Photography

Publications

- A Forward Model of Optic Flow for Detecting External Forces, L. Georgopoulos, G.M. Hayes and G.D. Konidaris, Proceedings of IEEE/RSJ 2007 International Conference on Intelligent Robots and Systems, October 2007, San Diego

- H. Rovithis-Livaniou; L. Georgopoulos; P. Rovithis:"Interplay of Periodic, Cyclic and Stochastic Variability in Selected Areas of the H-R Diagram". Edited by C. Sterken, ASP Conf. Ser. 292. San Francisco: Astronomical Society of the Pacific, 2003, p. 137.
- L. Georgopoulos and M. Hasler. Early consensus in complex networks under variable graph topology. In Proceedings of the European Conference on Circuit Theory and Design 2009, pages 575-578, Piscataway NJ, 2009. IEEE.
- A. Khadivi, L. Georgopoulos, and M. Hasler. Forest fire localization using distributed algorithms in wireless sensor networks. In Proceedings of PIERS 2009, Moscow, pages 452-455, Moscow, 2009.
- L. Georgopoulos and M. Hasler. Nonlinear Average Consensus. In Proceedings of the 2009 International Symposium on Nonlinear Theory and its Applications, pages 10-13, Sapporo, Hokaido, 2009. IEICE.
- L. Georgopoulos and M. Hasler. Training Distributed Neural Networks by Consensus. In Distributed machine learning and sparse representation with massive data sets, 2011.

COLOPHON

This thesis was type-set with LaTeX 2ε using the *Bookman* font family. In various parts, Hermann Zapf's *Palatino* and *Euler* type faces have been used (Type 1 PostScript fonts *URW Palladio L* and *FPL* were used). The listings are typeset in *Bera Mono*, originally developed by Bitstream, Inc. as "Bitstream Vera". (Type 1 PostScript fonts were made available by Malte Rosenau and Ulrich Dirr.) The figures are typeset in "Helvetica". The typographic style is available for LaTeX via CTAN as "`classicthesis`".
The writing style indicated in (Strunk, 1930) has been followed within this thesis, to the extent of the writer's ability. The *Oxford British English Dictionary* (Dictionary, 2004) has been consulted as needed to verify the usage of terms.

Final Version as of May 21, 2011 at 12:27.