Distributed Average Consensus for Wireless Sensor Networks

THÈSE N^{O} 4509 (2009)

PRÉSENTÉE LE 13 NOVEMBRE 2009

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE COMMUNICATIONS AUDIOVISUELLES 1
PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Florence BÉNÉZIT

acceptée sur proposition du jury:

Prof. M. Hasler, président du jury
Prof. M. Vetterli, Prof. P. Thiran, directeurs de thèse
Prof. V. Blondel, rapporteur
Prof. A. Schiper, rapporteur
Prof. J. N. Tsitsiklis, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2009

Contents

Dedication **v**

Abstract **vii**

Résumé **ix**

Acknowledgments **xi**

Frequently Used Terms, Abbreviations, and Notation **xiii**

1 Introduction **1**
1.1 Motivation 1
1.1.1 Average Consensus 4
1.1.2 Voting 5
1.2 Network and Graph Models 11
1.3 Dissertation Outline 13
1.4 Contributions 13

2 Distributed Average Consensus Algorithms **15**
2.1 Time Models: Synchronicity and Asynchronicity 15
2.2 Description of Average Consensus Algorithms 17
2.2.1 Synchronous average consensus algorithms 18
2.2.2 Gossip or asynchronous average consensus algorithms . . . 20
2.2.3 Broadcast gossip 27
2.2.4 One-way path averaging 28

3 Convergence of Distributed Average Consensus Algorithms **37**
3.1 Convergence of the Estimates 37
3.1.1 Deterministic average consensus algorithms 38
3.1.2 Convergence to consensus with probability 1 40
3.1.3 Convergence in expectation 44
3.1.4 Mean square convergence 45
3.2 Convergence Time 49
3.2.1 (ε, η)-Averaging time T_{ave} 49
3.2.2 Consensus time T_c 54
3.2.3 Consensus cost C_c 72

4 Second Largest Eigenvalue in Magnitude 79
4.1 Symmetry, Primitivity and Stochasticity . . . 79
4.1.1 Definitions and Perron-Frobenius Theorem . . . 79
4.1.2 Non-negativity and symmetry are not necessary . . . 82
4.1.3 Useful results from Markov chain theory . . . 82
4.2 Circles and Lattices . . . 88
4.2.1 Circulant matrices . . . 89
4.2.2 SLEM on cycles and lattices . . . 90
4.2.3 Fastest mixing chain on cycles . . . 95
4.3 Trees . . . 101
4.3.1 Markov Chain on Trees . . . 103
4.3.2 Stationary distribution and induced Markov chain. . . . 105
4.3.3 Upper Bound on the Pseudo-Relaxation Time . . . 106
4.3.4 Applications to Doubly Stochastic Matrices . . . 110
4.3.5 Is the SLE ξ_2 equal to the SLEM λ_2? . . . 118
4.4 Laplacian . . . 123
4.4.1 Definitions . . . 123
4.4.2 Examples . . . 125
4.4.3 Properties of the algebraic connectivity . . . 126

5 Performance of Various Average Consensus Algorithms 131
5.1 Pairwise Gossip . . . 131
5.1.1 Convergence . . . 131
5.1.2 Speed of Convergence . . . 133
5.2 Geographic Gossip and Neighborhood Gossip . . . 144
5.2.1 Speed of Convergence of Geographic Gossip . . . 144
5.2.2 Speed of Convergence of Neighborhood Gossip . . . 149
5.3 Broadcast Gossip . . . 152
5.3.1 Convergence . . . 153
5.3.2 Speed of Convergence . . . 154
5.4 Path Averaging . . . 155
5.4.1 Convergence . . . 155
5.4.2 The Influence of the Routing Protocol. . . . 156
5.4.3 Speed of Convergence on Toroidal Lattices . . . 157
5.4.4 Speed of Convergence on Toroidal Random Geometric Graphs164
5.4.5 Simulations . . . 168
5.5 One-way Path Averaging . . . 171
5.5.1 An Unbiased Estimator . . . 171
5.5.2 Convergence to Consensus . . . 175
5.5.3 Controlling Weights. . . . 184
5.5.4 Robustness, Quantization and Performance. . . . 185

6 Interval Consensus and Voting 195
6.1 From Quantized Average Consensus to Voting . . . 195
6.1.1 The interval consensus problem . . . 196
6.1.2 Interval consensus algorithm . . . 197
6.1.3 Convergence Theorem . . . 200
6.1.4 Back to the voting problem. . . . 202
6.2 Multiple Voting . . . 205
6.2.1 Voting PAGA . . . 206

6.2.2 Simple Examples of Voting PAGA 208
6.2.3 Sufficient Conditions to Construct Voting PAGA 210
6.2.4 Almost Sure Convergence in Finite Time 215
6.3 Evolved Examples of Voting PAGA 217
6.3.1 Binary and ternary voters 217
6.3.2 Quaternary voter 220

7 Conclusion 227
7.1 Future Research 228
7.1.1 Large Deviations 228
7.1.2 Voting Algorithms 228
7.1.3 Mobility and Virtual Mobility 229

A Random geometric graphs 233
A.1 Connectivity of Random Geometric Graphs 233
A.1.1 Connectivity of an RGG on the unit disk 234
A.1.2 Connectivity of an RGG on the unit square 234
A.2 Routing in Random Geometric Graphs 237
A.2.1 The Routing Algorithms 237
A.2.2 Performance 239
A.3 Balanced Random Geometric Graphs 246
A.3.1 Balanced boxes 246
A.3.2 Balanced degrees 247
A.3.3 Balanced Voronoi cells 248

B The quaternary automaton construction 249

Bibliography 257

Curriculum Vitae 263

En mémoire de Patrick Denantes.

Abstract

Wireless sensor networks have emerged a few years ago, enabling large scale sensing at low cost. There are many interesting problems related to this new sensing tool: designing robust and small hardware, defining adapted routing protocols, minimizing the energy consumption of each component, synchronizing the sensors, etc. In this thesis, we focus on the processing of the sensed data within the network itself.

We study a specific network signal processing problem, called distributed average consensus. In this problem, the sensors, which are connected in a wireless network, need to know the average of all the measurements in the network. Instead of gathering the data at a central node, which would compute the average and broadcast it to the network, average consensus algorithms offer a distributed solution to the averaging problem. By local message passing and iterative local computations only, nodes can learn the average of the measurements.

More precisely, in average consensus algorithms, nodes iteratively compute local weighted averages that conserve the global average of the estimates in the network. The estimates at each node contract until they all converge to the average.

Many distributed average consensus algorithms were designed and the literature is vast. This thesis starts by classifying the existing algorithms. Then it describes a small number of useful techniques, which can handle the analysis of all the algorithms. Preexisting algorithms as well as algorithms that we designed are revisited with these unifying and simple techniques.

In addition, the performance of the algorithms depend on the topology of the network, and a variety of networks are explored: simple graphs as circles or trees, lattices, random geometric graphs, complete graphs etc.

Finally, an extension of average consensus to voting consensus is derived. In particular, we show that with two bits of memory at each node, a network can reach consensus in finite time on majority, when the initial measurements are binary. Distributed algorithms to compute majority with finite memory for ternary and quaternary signals are also given.
Keywords: Consensus, Gossip, Distributed Algorithms, Sensor Networks, Voting problem.

Résumé

Les réseaux de capteurs sans fil sont apparus il y a plusieurs années, dans le but d'observer notre environnement à grande échelle et sans grand coût. Ce nouveau système de captation de données est à l'origine d'un grand nombre de problèmes intéressants. Il faut concevoir du matériel informatique qui soit à la fois résistant et de petite taille, inventer des protocoles de routage adaptés, minimiser la consommation d'énergie de chaque composant, trouver le moyen de synchroniser les capteurs à moindre frais, etc. Nous nous intéressons dans cette thèse aux signaux captés, et plus particulièrement à leur traitement au sein même du réseau.

Dans le domaine, assez vaste, du traitement du signal en réseau, nous nous concentrons sur l'étude d'un problème bien spécifique, portant le nom de "problème du consensus distribué sur la moyenne". Il s'agit pour les capteurs, qui sont reliés par un réseau sans fil, d'apprendre la moyenne des valeurs saisies par l'ensemble des différents capteurs du réseau. Au lieu de laisser récupérer toutes les données par un capteur central, qui pourrait alors calculer leur moyenne, des algorithmes de consensus distribués offrent la possibilité de résoudre le problème de manière décentralisée. Il suffit par exemple de communications de faible portée et de calculs locaux, pour permettre aux capteurs d'apprendre, petit à petit, itération après itération, la moyenne des données.

Plus précisément, à chaque itération d'un algorithme de consensus sur la moyenne, les capteurs calculent une moyenne pondérée de données locales qui conserve la moyenne globale des données du réseau. Ainsi, les estimations de la moyenne, détenues par les capteurs, se rapprochent les unes des autres et tendent toutes vers la moyenne des données initiales.

Les algorithmes capables de moyenner de manière décentralisée sont nombreux et la littérature est vaste à ce sujet ; ce manuscrit commence donc par les classifier. Ensuite, nous décrivons un petit nombre de techniques utiles, permettant de les étudier d'un point de vue théorique. Les algorithmes préexistants, tout comme ceux que nous concevons ici, sont analysés grâce à ces techniques simples et générales.

En outre, comme la performance de ces algorithmes dépend de la topologie du réseau dans lequel ils fonctionnent, nous travaillerons sur des topologies nombreuses, en partant de graphes très simples, comme un cercle ou un arbre, avant de s'en rendre, progressivement, à des graphes aléatoires utilisant des schémas booléens de sphères, en passant par l'exemple d'un graphe complet ou bien encore celui de grilles.

En dernier lieu, nous modifierons un algorithme de consensus sur la moyenne pour en faire un algorithme de consensus sur la majorité. Nous montrerons en particulier que si chaque capteur peut contenir deux bits de mémoire, alors le réseau peut parvenir à un consensus sur la majorité en temps fini, dans le cas de

données binaires. Nous étendrons également notre méthode aux cas ternaires et quaternaires.
Mots-clés: Consensus, Commérage, Algorithmes distribués, Réseaux de capteurs, Problème du vote.

Acknowledgments

During my time as a Ph.D. student, I have received help and support from many people.

First of all, I would like to thank my two advisors, Prof. Patrick Thiran and Prof. Martin Vetterli for supporting me (both in the english and in the french sense of the word). Their perseverance in striving for excellence has driven me to always give my best. I especially appreciated their humanity and care in the difficult moments. Finally I would like to thank them for truly loving sciences and for their endless dedication to education.

My research was funded by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation, to which I am grateful for its financial support.

I would also like to thank Prof. John Tsitsiklis, Prof. Vincent Blondel, Prof. André Schiper and Prof. Martin Hasler for being on my thesis committee, for reading this thesis and for sharing their insightful ideas on the topic.

Next, I would like to thank Prof. Pierre Brémaud for all our interesting discussions and for his yellow book, in which I found most of the answers to my thesis problems.

I also thank Dr. Alexandros Dimakis for working with me, for his great creativity and for sharing his enthusiasm for asymptotic upper bounds.

I give special thanks to all the EPFL students for making this place so joyful and rich. It was an honor to meet them and to help them in their studies. I wish them luck and I hope that their scientific dreams come true.

I am very grateful to my LCAV colleagues for their friendship and support. I thank Cédric Paradivin for his interest in my thesis and for translating my abstract into good french. And last but not least, I would like to thank my family and friends in France and at various other places around the world. I thank my parents, my brothers and my sister, for their support, their care, their visits to Switzerland and all their efforts to make my life easier during my convalescence. Finally, I am especially thankful to Vojislav, not only for bringing love and happiness back into my life after my accident, but also for all the promising years of our future life together.

Frequently Used Terms, Abbreviations, and Notation

Terms and abbreviations

RGG: random geometric graph;

Complete graphs, full graphs, circles, cycles, trees, and lattices are defined in Section 1.2.

Balanced trees are defined in Definition 4.3.1.

w.h.p.: with high probability. Event A_n happens with high probability if and only if the probability that A_n does not happen vanishes with n.

w.v.h.p.: with very high probability. Event A_n happens with very high probability if and only if the probability that A_n does not happen vanishes with n faster than $O(1/n)$.

SLE: second largest eigenvalue;

SLEM: second largest eigenvalue in magnitude;

Symmetric Markov chain: it is a Markov chain with a transition matrix which is symmetric. The stationary distribution of such a matrix is uniform.

Self-loop: for any stochastic matrix $\mathbf{P}$, the self-loop in i is $\mathbf{P}_{ii}$.

Notations and variables

Vectors	are denoted by bold lowercase letters ($\mathbf{x}$, $\mathbf{y}$, etc);
Matrices	are denoted by bold uppercase letters ($\mathbf{W}, \mathbf{A}$, etc). By default, matrices are square, and their entries are real;
t	denotes *discrete* time by default;
$\mathbf{A}^T$	is the transposed matrix of $\mathbf{A}$;

$\mathbf{1}$	denotes the all ones vector;
$\mathbb{P}[A]$	denotes the probability of event A;
$\mathbb{E}[X]$	denotes the expectation of the random variable X;
$\otimes$	denotes the Kronecker product;
n	is usually the number of nodes in the network;
$\mathbf{J}$	denotes the exact averaging matrix: $\mathbf{J} = \mathbf{1}\mathbf{1}^T/n$;
$\mathbf{x}(t)$	denotes the vector of estimates at iteration t;
x_{ave}	denotes the average of the initial signal: $\mathbf{J}\mathbf{x}(0) = x_{ave}\mathbf{1}$;
$\mathbf{W}$	is our usual notation for the averaging matrix: $\mathbf{x}(t) = \mathbf{W}(t)\mathbf{x}(t-1)$;
$\mathbf{P}$	denotes a stochastic matrix;
$\boldsymbol{\epsilon}$	denotes the error vector: $\boldsymbol{\epsilon}(t) = \mathbf{x}(t) - x_{ave}\mathbf{1}$;
$\boldsymbol{\eta}$	denotes the error vector with the current average: $\boldsymbol{\eta}(t) = \mathbf{x}(t) - \mathbf{J}\mathbf{x}(t)$;
ρ	denotes the spectral radius: it is the largest eigenvalue in magnitude;
d_i	denotes the degree of node i;
$\mathcal{N}_i$	denotes the neighborhood of node i. By default, $i \notin \mathcal{N}_i$;
$i \sim j$	means that i and j are neighbors;
$f(n) = O(g(n))$	if there is a constant $c > 0$ and an integer n_0 such that $\forall n \geqslant n_0$, $f(n) \leqslant cg(n)$;
$f(n) = \Omega(g(n))$	if there is a constant $c > 0$ and an integer n_0 such that $\forall n \geqslant n_0$, $f(n) \geqslant cg(n)$;
$f(n) = \Theta(g(n))$	if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$;

$\xi_1 \geqslant \xi_2 \geqslant \cdots \geqslant \xi_m$	are the ordered real eigenvalues of an $m \times m$ symmetric matrix. For matrices with complex eigenvalues, $\xi_1, \cdots, \xi_m$ are indexed in any order;
$\lambda_1, \lambda_2, \cdots, \lambda_m$	are the eigenvalues ordered in decreasing *magnitude* of an $m \times m$ matrix: $\lvert\lambda_1\rvert \geqslant \lvert\lambda_2\rvert \geqslant \cdots \geqslant \lvert\lambda_m\rvert$;
λ	can also denote the intensity of an exponential random clock. Since clock intensities arise in different contexts than eigenvalues, there will be no conflict of notations;
$\tau(\mathbf{P})$	denotes the relaxation time (or mixing time) of a stochastic matrix $\mathbf{P}$: $\tau(\mathbf{P}) = 1/(1 - \lvert\lambda_2\rvert)$;
$\tau'(\mathbf{P})$	denotes the pseudo-relaxation time (or pseudo-mixing time) of a stochastic matrix $\mathbf{P}$: $\tau(\mathbf{P}) = 1/(1 - \lvert\xi_2\rvert)$;
$\mathbb{R}_k[X]$	denotes the $\mathbb{R}$-vector space of polynomials of degree less than or equal to k;
$\delta(\mathbf{P})$	denotes the Dobrushin coefficient of matrix $\mathbf{P}$: $\delta(\mathbf{P}) = \frac{1}{2}\max_{ij}\sum_{k=1}^{n}\lvert\mathbf{P}_{ik} - \mathbf{P}_{jk}\rvert$.

Norms and metrics

Vector norms

$\|\mathbf{x}\|_1 \quad = \sum_i |\mathbf{x}_i|$;

$\|\mathbf{x}\|_2 \quad = \sqrt{\sum_i \mathbf{x}_i^2}$;

$\|\mathbf{x}\|_\infty \quad = \max_i |\mathbf{x}_i|$;

Matrix norms and metrics

A matrix norm is by definition submultiplicative: $\|\mathbf{AB}\| \leqslant \|\mathbf{A}\|\,\|\mathbf{B}\|$.

$\|\cdot\|_1$	denotes the ℓ_1 norm: $\|\mathbf{A}\|_1 = \sum_{i,j=1}^{n} \lvert\mathbf{A}_{ij}\rvert$;
$\|\cdot\|_2$ or $\|\cdot\|_F$	denotes the ℓ_2 norm, also called the Frobenius norm: $\|\mathbf{A}\|_2 = \sqrt{\sum_{i,j=1}^{n} \mathbf{A}_{ij}^2}$;
$\|\cdot\|_\infty$	denotes the ℓ_∞ norm: $\|\mathbf{A}\|_\infty = \max_{i,j} \lvert\mathbf{A}_{ij}\rvert$;
$\lvert\lvert\lvert\cdot\rvert\rvert\rvert$	is the associated norm of the vector norm $\|\cdot\|$: $\lvert\lvert\lvert\mathbf{A}\rvert\rvert\rvert = \max_{\|\mathbf{x}\|=1} \|\mathbf{Ax}\|$;
$\lvert\lvert\lvert\cdot\rvert\rvert\rvert_1$	denotes the matrix norm associated with $\|\cdot\|_1$:

$|||\mathbf{A}|||_1 = \max_j \sum_{i=1}^n |\mathbf{A}_{ij}|$;

$|||\cdot|||_2$ denotes the matrix norm associated with $\|\cdot\|_2$.
It is also called the spectral norm:
$|||\mathbf{A}|||_2 = \max\{\sqrt{\lambda} : \lambda \text{ is an eigenvalue of } \mathbf{A}^T\mathbf{A}\}$.
Replace $\mathbf{A}^T$ by $\mathbf{A}^*$ if $\mathbf{A}$ has complex entries;

$|||\cdot|||_\infty$ denotes the matrix norm associated with $\|\cdot\|_\infty$:
$|||\mathbf{A}|||_\infty = \max_i \sum_{j=1}^n |\mathbf{A}_{ij}|$;

$\delta(\cdot)$ denotes the Dobrushin's semi-norm:
$\delta(\mathbf{A}) = \frac{1}{2} \max_{j,k} \sum_i |\mathbf{A}_{ij} - \mathbf{A}_{ik}|$.
Note that we use the same notation for the Dobrushin's semi-norm and for the Dobrushin coefficient. Since they arise in different contexts, there will be no conflict of notations. The Dobrushin's semi-norm of $\mathbf{A}$ is the Dobrushin coefficient of $\mathbf{A}^T$.

Chapter 1

Introduction

1.1 Motivation

Wireless sensor networks are an emerging technology, that aims to revolutionize large-scale sensing in areas such as environmental monitoring, industrial control, health monitoring, target tracking, and robotics. Sensors are devices that interact with the physical world. For example, in Switzerland, the SensorScope project[1] deploys environmental wireless stations with seven sensors each, including a thermometer, a pluviometer and an anemometer. As these weather stations are installed in vast and remote mountainous zones, gathering the data would be inconceivable without wireless communications. It is not feasible to send someone every day to check all the weather stations, and the mountain area is too large to wire the stations together. In the SensorScope project, the data is collected in a base station by means of short-range wireless communications between the stations. Some stations serve as relays for the stations that are out of reach of the base station, using a scheme called multi-hop routing. From time to time, the base station, which is more powerful than the weather stations, sends the collected data by GPRS to a server at EPFL.

The use of a sensor network in the SensorScope project not only allowed for the very existence of the collected data, but it also made data gathering possible at a low price. With the increasing interest in environmental issues in western societies, very high expectations are placed on sensor networks. They should be reliable, efficient, inexpensive, flexible, small, and should consume as little energy as possible in order to have the longest lifetime possible.

Batteries are at the nerve of the sensor network war. There is little point in building a very smart sensor network if the battery of every sensor needs to be frequently replaced. Sensor networks should deliver data over months, at a rate that scales in minutes, over areas that scale in square kilometers, with only a small battery at each sensor and possibly some solar panels. The scope requirement of sensor networks is of such unprecedented order of magnitude that everything needs to be designed from A to Z. This is why sensor networks give rise to an immense number of new problems, all the way from hardware design to energy saving communication protocols, passing by clock synchronization and possibly by the local processing of the data.

[1] `http://sensorscope.epfl.ch/index.php/Main_Page`

We are specifically interested here in the signal processing issues for sensor networks, and more particularly in consensus/agreement problems. Agreement problems do not show up in centralized sensor networks such as SensorScope networks, but rather in decentralized sensor networks, where there are no base stations. In order to add robustness to the network, sensor network designers do not want to rely on one or a few vulnerable base stations. In the ideal vision of decentralized sensor networks, all the sensors nodes are equally equipped and have the same function. If one node dies, due to battery failure for instance, the network should automatically adapt and continue to deliver coherent data. This very ambitious decentralized vision has not been fulfilled yet, and the goal of this dissertation is to help achieve it.

Coherent self-organization does not come without agreement and compromise. Based on the collected data, the whole network needs to agree on a decision in order to carry on coordinated operations. For example, a sensor network whose purpose is to track enemies needs to agree on the presence of such an enemy. A flock of robots wants to move in the same direction. Or sensors should agree on a common time and synchronize their clocks. Our approach to agreement differs from what computer scientists call "distributed consensus". Reaching a "distributed consensus" in computer science implies that the nodes *know* that they have reached a consensus. This problematic knowledge in unreliable networks [52, 37, 36] is not our focus. We are more interested in the value on which the nodes agree, in how to compute in a distributed way a common value of interest, and in how much energy and time is needed for every node to have a good enough approximation of this ideal common value.

As the title of this dissertation indicates, the *ideal value* we will seek chapter after chapter is the *average* of the data measured by the sensors. This might sound disappointingly simple, but the main topic of this dissertation is the computation of averages in a distributed way. If the sensor network has only two nodes, with respective data values a and b, we want both nodes to learn $(a+b)/2$. Averages are indeed simple, but they are very useful above all. Being able to compute averages allows for the computation of linear combinations, which is undeniably useful. The applications of distributed average consensus algorithms are numerous: distributed computing [12, 75, 11], multiagent coordination [71, 13, 67, 46, 33], time synchronization [29, 41], distributed load-balancing [63, 21], wireless sensor network monitoring [80], and decentralized source localization [65, 64]. Distributed consensus algorithms are not confined to wireless sensor networks, and they can be applied to distributed processor computing, distributed database management, or distributed signal processing on the Internet for example.

Distributed signal processing and the average consensus problem are not new. The earliest work on distributed averaging in a network was in the 1984 thesis of Tsitsiklis [75, 11], with applications in distributed computing with several processors. With the advent of wireless sensor networks, and their need for decentralized schemes, the distributed average consensus algorithms designed by Tsitsiklis have received a considerable renewed attention. The specificity of this dissertation is to apply the distributive computing ideas of Tsitsiklis to a wireless sensor network setting. Boyd et al. were the first to show that pairwise gossip, which is a distributed average consensus algorithm, converges slowly on

random geometric graphs [14] (see Definition 1.2.1). Random geometric graphs model well wireless sensor networks, and they will be one of our main points of focus. In particular, we will design averaging algorithms that are particularly efficient on random geometric graphs [8].

In the last six years, many papers were dedicated to distributed average consensus algorithms on random geometric graphs. In this dissertation, we propose an extensive review of the algorithms designed for these graphs. Furthermore, for all the interesting results that we cite, we did our best to offer novel proofs using as few mathematical tools as possible, in order to unify the proofs. This has several advantages. First, it allows us to understand better the differences in performances between the algorithms. Second, it alleviates the reading of the proofs, as the same techniques are repeated over and over again. Finally, it shows that the techniques are general enough to handle all the present cases, and we hope that they will ease the analysis of future algorithms.

In Chapter 6, we introduce a distinction between two averaging problems. In the first averaging problem, all the agents want to agree on a value, which is preferably equal to the average of some initial values. For instance, if a flock of robots wants to move in the same direction, the most important issue is to have the robots move in parallel. The robots need to compute precise directions, and these directions should not differ much from one robot to the other. This problem is the classic "average consensus problem". The second problem, which we introduce in [10], is called the "interval consensus problem". Take now an enemy tracking sensor network. Every sensor measures either 1: "yes, there is an enemy" or 0 :"no, there is not an enemy". The sensors in the network might not be interested in the exact number of positive and negative sensors. They might just want to know if there is a majority of yes or no answers. Or, they might be interested in a more precise piece of information: "Is there at least two thirds of yes or no answers"? These two questions can be formulated as follows. Given a set of disjoint intervals ($\{[0, 0.5], (0.5, 1]\}$ and $\{[0, 1/3], (1/3, 2/3], (2/3, 1]\}$ in our two examples), in which interval does the average measurement lie? Under quantization and memory constraints, the problem of reaching a distributed consensus on the *majority* question, also called the voting problem or the density classification problem, has remained unsolved for two decades. One of the major contributions of this thesis is to solve this problem in a distributed way for any connected network.

We will introduce in further detail the contents of the main two topics of this thesis: average consensus algorithms on random geometric graphs and voting algorithms. In particular, Section 1.1.1 explains in a few words how distributed averaging algorithms work, and it will list the algorithms that we are going to study. Section 1.1.2 explains why the voting problem is famous, and what its implications are. It shows how previous work approached the problem, and then describes the main ideas of our solution. Then, Section 1.2 lists the graphs which are useful in this thesis, Section 1.3 gives a short outline of the dissertation, and Section 1.4 lists the contributions of the thesis.

1.1.1 Average Consensus

Distributed Averaging Algorithms

Distributed averaging consensus algorithms are a class of algorithms that solves the following problem. Every sensor/agent in a network holds a value and wants to learn the average of all the values in the network. However, the agents do not know the shape of the network. They do not even know how many agents there are. All they can learn easily is the identities of their neighbors in the network. In some evolved cases, we can assume that agents have access to their locations, thanks to GPS or to a distributed localization algorithm.

All distributed averaging algorithms use the same basic mechanism, which is to iteratively perform local averages until each agent holds an accurate enough estimate of the global average. At every iteration, all or part of the agents send their estimates to some other agents using the edges of the network. Then the agents update their estimates using linear weighted averages of the received data. To design a distributed averaging algorithm, one should specify if agents send their estimates synchronously or asynchronously. Then one should design mechanisms to decide at each iteration which agents send data, which agents update their states, and which are the averaging weights to be used. Distributed averaging algorithms are thus a large class of algorithms, and we are interested in finding those that are efficient on graphs that model wireless sensor networks, namely random geometric graphs (see Definition 1.2.1).

Previous Work

The first algorithm that has been studied on a random geometric graph is *pairwise gossip* [15]. This asynchronous average consensus algorithm activates a pair of neighbors at every iteration and updates their estimates to their pairwise average. Boyd et al. show that pairwise gossip converges as slowly as flooding all the values through the network and locally computing the average based on all the stored values. More precisely, pairwise gossip demands order $O(n^2)$ messages on a random geometric graph of n nodes, whereas it demands only $O(n \log n)$ messages on the complete graph (see Section 1.2 for a definition). Note that pairwise gossip cannot converge in less than $O(n)$ messages, because every node should send its value at least once.

After this disappointing result, Dimakis et al. decided to accelerate pairwise gossip, by routing estimates in the network [27]. Based on the observation that pairwise gossip is very efficient on complete graphs, the idea is to enable any pair of nodes to pairwise average their estimates at the expense of routing, as if the network were a complete graph. By routing estimates in the network and by allowing any pair of nodes to exchange their estimates, we say that we create a complete overlay graph on the network. The resulting algorithm is called *geographic gossip*, because the chosen routing demands some geographic knowledge of the network. At every iteration, a node wakes up and generates a random route. Both the node that started the route and the destination node update their estimates to their pairwise average. In order to do this, the route must be utilized first forward and then in reverse. As the gain in iterations is larger than the expense of routing the estimates back and forth, geographic gossip necessitates less messages ($O(n^{1.5}\sqrt{\log n})$) and thus less energy than the classic pairwise gossip algorithm.

We also review algorithms that take advantage of the broadcast nature of wireless communications, namely *neighborhood gossip* introduced in [57, 58] by Nazer et al., and *broadcast gossip* introduced in [7] by Aysal et al.

Our Algorithms

Given the encouraging results of geographic gossip, we decided, jointly with Dimakis, to explore a simple question. What happens if all the estimates of the random routes are averaged together, instead of only their extremities? The idea is to aggregate the estimates on one way of the route, the destination node computes the path average and sends it back through the same route so that all the nodes in the route update their estimates. The resulting algorithm was named *path averaging*, and we showed that it necessitates as few messages as pairwise gossip on the complete graph, under some simplifying conditions [9].

Both geographic gossip and path averaging look good on paper, but in practice they require that sensor networks are reliable enough to route a message on a single route both forward and back. In reality, this can be difficult to achieve because wireless links are time-varying and thus unreliable; worse, the network might be mobile. Hence, applying path averaging could require more messages than expected in an unreliable network, because nodes would need to send their messages many times before they reach their destination, especially when routing the average back.

However, path averaging is not an idea to be abandoned. It shows that adding directionality to the averaging pattern can help to accelerate distributed averaging in random geometric graphs. To finish with an optimistic touch, we propose an algorithm called *one-way path averaging*, which keeps this random route feature, but does not need to route back any information. The very nature of one-way path averaging is different from previous algorithms, although it is still based on a weighted average mechanism. The algorithm is inspired from the Push-Sum protocol [51]. We do not yet have formal proofs of its convergence or of its performance. However, we provide some clues to understand its behavior, and we show on simulations that one-way path averaging is very promising.

1.1.2 Voting

The voting problem is famous mostly because it is linked to the following important question:

How powerful are local interactions?

Although local interactions rule our physical world, they are believed to have strong limitations, because a networked set of agents holding binary opinions does not seem to be able to compute its majority opinion by means of local interactions only. This is intriguing as majority is the simplest global piece of information one can retrieve from a binary system: Are there more positive opinions than negative opinions? This question requires a simple yes or no answer and yet, so far, no distributed scheme has managed to drive all the agents to reach a consensus on majority.

In this dissertation, we rehabilitate local interactions, not only because all our averaging algorithms are based on local interactions, but also because we

point out some simple local updating rules that lead a system to a consensus on the majority. Interestingly, these rules are directly inspired by a well-known physical phenomenon based on local interactions: heat diffusion. Indeed, as we will explain later, heat diffusion is linked to distributed averaging algorithms, which are the basis of the voting algorithms described here.

Local interactions in large systems can lead to surprisingly complex coordinated macroscopic behaviors and to global computations. For example, fireflies manage to flash synchronously, flocks of birds and schools of fish can move in harmony, spins align in ferromagnetism. The interest in these phenomena is threefold:

1. based on simple local interactions, explain the global coordination we can observe in some physical and biological systems,

2. imitate such phenomena to design distributed computational algorithms or coordinate large distributed systems,

3. draw the limit between what can be computed by local means and what requires centralized organization.

In this thesis, we address the voting problem, which is more commonly called the density classification task in the literature. As for the averaging problem, we consider a set of agents that can communicate through a connected network. Every agent initially produces a binary piece of information (bit): 0 or 1. The density-classification task consists in letting all the agents know whether 0 or 1 was the initial majority bit (the bit with higher density), by means of local interactions only.

The applications of the voting problem are vast: distributed computing, neural networks, system-level diagnosis, distributed database management, fault-local mending and more [43, 26, 62]. In sensor networks, it could be useful to recover binary sensing errors for instance, if it can be assumed that the correct measurement is the majority measurement.

Previous Work

A considerable amount of work has been done in the past two decades to design local rules that would successfully solve the density-classification task. The most active early work concentrated on *cellular automata* (CA) [70]. CA are local rules often designed for circular lattices where agents can communicate within a radius r. Each agent stores one memory bit (its state), and at each iteration, sends it to all its neighbors. An iteration ends when all the agents have synchronously updated their state according to the states they have received. Ideally, all states in the network should converge to the initial majority bit.

Although it was proven in [53] that no successful CA exists, some interesting CAs were previously designed with the hope of solving the problem. The most intuitive CA is the local majority rule, in which agents update their state to the majority state of their neighborhood [56, 61]. The inability of this rule to lead all the states to a consensus on 0 or 1 constitutes its major flaw (see Fig. 1.1). Additionally, even when a consensus is reached, the bit (0 or 1) on which agents agree is not necessarily the initial majority bit. As a consequence

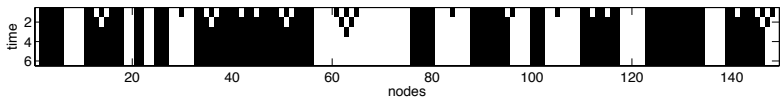
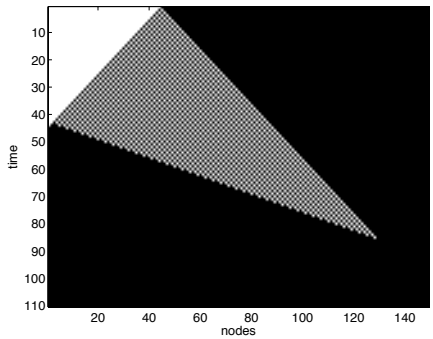


Figure 1.1: The synchronous majority rule does not always lead the states to a consensus. Black squares represent state 0, white squares represent state 1 and the successive state configurations are shown with time increasing down the page.

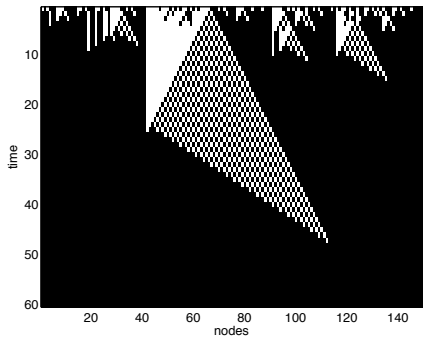
of this failure, CAs were developed with a crucial new feature: *information flow*. To reach consensus, the updating rule should make the information travel in the system and reveal disagreements. The Gacs-Kurdyumov-Levin (GKL) automaton is the most efficient CA found so far. It operates on the circle with radius $r = 3$ [22] and it is empirically successful on approximatively 82% of initial configurations sampled at random (each bit is drawn independently at random with probability 1/2). The GKL rule proceeds as follows: An agent in state 0 changes to the majority state among its own state and the states of the two neighboring agents to its left removed a distance 1 and 3, respectively. An agent in state 1 applies the same rule but uses the neighbors to its right. With these rules, blocks of bits 0 and blocks of bits 1 interlace, bits 0 drifting to the left, and bits 1 to the right. Eventually, larger blocks absorb smaller ones, which explains the efficiency of the GKL rule (see Fig. 1.2).

By slightly modifying the problem setting, two exact solutions were found on the circle [39, 18]. Both are based on the traffic flow rule, also called the 184 rule, in which the number of 0's and 1's stays unchanged, while the 0's and 1's interlace with time, until the minority bits are systematically surrounded by 2 majority bits. Inevitably, some neighboring agents have common majority bit states. Both solutions are based on simple observations. In the first solution, at each iteration, this interlaced configuration shifts by one position in the network so that all the agents eventually discover two consecutive identical bits, which are equal to the majority bit. In the second solution, the network switches to the local majority rule, which is successful on any interlaced configuration reached by rule 184.

Both solutions confirm the power of information flow in the density classification task. Furthermore, they highlight the importance of *conserving the global information* through iterations. In [17], Capcarrere and Sipper proved that density conservation is a necessary condition for a CA to be successful. Despite their success on circles, neither solution however is extendable to arbitrary graph topologies, so the research effort was devoted to designing rules adapted to any network, focusing back on bringing states to a consensus. To avoid the impossibility result [53], asynchronous and probabilistic automata were proposed [44]. An automaton is used asynchronously when nodes wake up at random times (usually exponentially distributed), communicate with their neighbors and update their state according to the automaton. The order in which nodes update their state is thus random, which could appear as an additional difficulty [74]. Most probabilistic automata are run synchronously. Examples of probabilistic automata converging to a consensus are numerous; in probabilistic polling, nodes update their state to the state of a randomly chosen node. If the probabilities are well tuned, this algorithm can reach pro-



(a) 44 white states out of 149.



(b) 67 white states out of 149.

Figure 1.2: Evolution of the GKL rule. Black squares represent state 0, white squares represent state 1 and the successive state configurations are shown with time increasing down the page. In (a), the initial configuration has two blocks: 44 white states and 105 black states. In (b), 67 white states and 82 black states are positioned at random. In both example the GKL automaton is able to solve the voting problem.

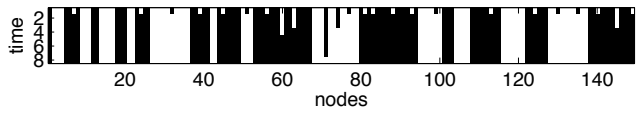
portionate agreement: the probability of reaching a consensus on 1 is equal to $\rho(1)$ (the initial density of 1's), and a consensus on 0 has probability $\rho(0)$. In [38], Fuks designs a probabilistic automaton on the circle that conserves density in expectation and thus achieves proportionate agreement. However, none of these algorithms converges to the correct consensus with probability 1.

A Novel Approach: 2 bits per node instead of 1.

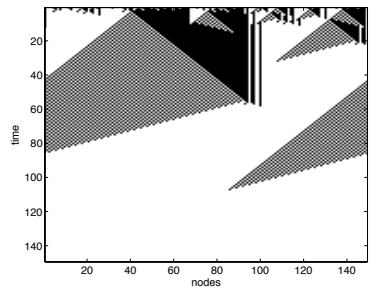
Chapter 6 is devoted to an algorithm that classifies density, almost surely (a.s.). Loyal to the philosophy of CA, the solution is inspired by a physical phenomenon: heat diffusion. Without heat loss, a material with heterogeneous temperature stabilizes to a single temperature equal to the average initial temperature. In other words, this locally interacting system is able to compute averages. Discretizing the heat diffusion equation reveals that the local interactions are local averaging computations. Note that computing the average of initial 0's and 1's and comparing it to 0.5 solves the density task. All the averaging algorithms in this thesis are based on this observation [14]. To run gossip, agents have an infinite memory to store a real number state. When an agent wakes up in pairwise gossip, it calls a neighbor at random to exchange their states. Then they *both* update their states to their average; if agents i and j with states x_i and x_j communicate, they update their states to $(x_i + x_j)/2$. Whereas previous work on the density classification task focused only on states of size 1 bit, we describe in Chapter 6 a quantized version of gossip, which we call "voting automaton"; it solves the problem with states of 2 bits.

In other words, at any iteration, every node is in one out of four states. Two states decide that 1 is the majority bit (black state and dark gray state), the two other states decide 0 (white state and light gray state). To initialize the algorithm, nodes that voted for 1 start with the black state, and nodes that voted for 0 start with the white state. Fig. 1.3 (e) shows an example of the evolution of the states in a circular network, where there were more black states at the beginning. Fig. 1.3 (d) colors in black the black and the dark gray states, it colors in white the white and the light gray states. Eventually all the states become black, which is the majority color in this example. The remaining of the figure shows how previous algorithms behave on the same initial configuration of votes. In this example, only our algorithm successfully converges to black.

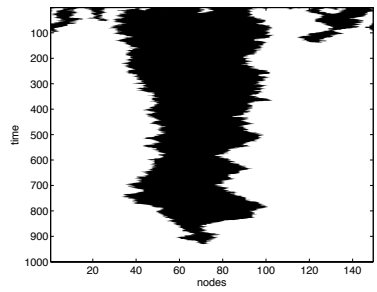
Chapter 6 starts from the distributed averaging algorithms described in the thesis and shows how to get to the voting problem by considering the interval consensus problem: "In which interval is the average of the signal?". The voting problem is a particular instantiation of the interval consensus problem as it is equivalent to ask "Is the average of the 0's and 1's in $[0, 0.5]$ or in $[0.5, 1]$?". Chapter 6 solves the interval consensus problem with a finite number of bits in a quite general case. Applying the general solution to the voting problem leads to the 2 bit automaton. Then Chapter 6 focuses more specifically on the voting problem and generalizes the solution to the multiple voting problem, where nodes choose between more than 2 candidates. It shows that the ternary voting problem can be solved with 15 states and that the quaternary voting problem can be solved with 100 states. Whether these problems can be solved with less states than, respectively, 15 and 100, remains an open problem. To prove that our multiple voter solutions converge, we develop a framework in



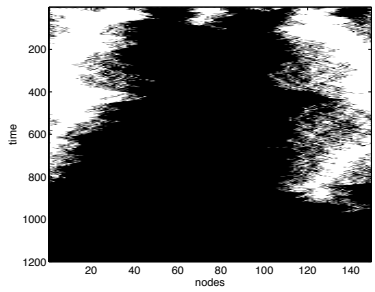
(a) Asynchronous local majority rule.



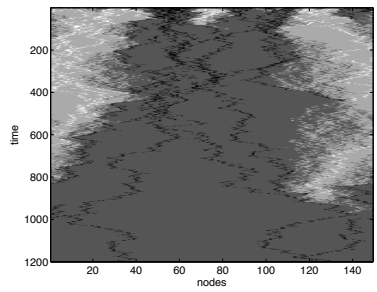
(b) GKL rule.



(c) Async. probabilistic polling [44].



(d) Our algorithm, 1 bit displayed.



(e) Our algorithm, 2 bits displayed.

Figure 1.3: Evolution of several automatons with identical initial condition: 73 whites and 76 blacks in a circular topology. In (a), (c), (d) and (e), the automata were used asynchronously (149 iterations per time level display). In (a), the active node updates its state according to the majority state among his state, its left neighbor state and its right neighbor state. In (b) the GKL rule is used synchronously. In (c), the active node chooses uniformly at random its left or right neighbor and adopts its state. The figures show that the local majority rule, GKL rule and probabilistic polling can fail. Our algorithm always converges to the right state. Note that there were initially 3 black states more than white states, and that in the end of our algorithm, there are three black states left, all the other states being dark gray (e).

which automata solve the multiple voting problem. It suffices to notice that the proposed automata fall in the correct framework to conclude that they are successful with probability 1 in finite time in any connected network.

1.2 Network and Graph Models

In the literature, wireless sensor networks are often modeled as random geometric graphs, and we adopt this model as well.

Definition 1.2.1 (Random geometric graph $G(n, r)$). A random geometric graph $G(n, r)$ is a random graph, where n nodes are drawn independently and uniformly in a unit area. Two nodes are connected by an edge if and only if their distance is smaller than radius r.

The unit area is often the unit square, but it is sometimes the unit disk or the unit torus. The random geometric graph model assumes that the wireless electromagnetic reach of an antenna forms a circular zone, which range is independent of the agent, of its location, and of the location of the other agents. In order to make this ideal model a little more realistic, we often assume that edges can fail independently in time, and we denote by p the probability that the link is active (does not fail).

In order to study the algorithms on random geometric graphs, we need a number of technical results concerning them. These results have been gathered in Appendix A. This appendix is worth reading because it clarifies a number of hasty assertions on random geometric graphs, which we encountered in the literature. Furthermore, we will need to refer to simpler graphs, listed and defined here. Graphs are denoted by the letter $\mathcal{G}$, their set of vertices by V, and their set of edges by E.

- **Complete graph**, also called the **full graph** or the **fully connected graph**. A complete graph is a graph $\mathcal{G} = (V, E)$ with vertices V and edge E such that $E = V \times V$. In other words, any pair of vertices is connected by an edge.

- **Connected graph**. A graph $\mathcal{G} = (V, E)$ is connected if, for any pair of vertices i and j, there is a path of edges in E which connects node i and j.

- **Undirected graph**. A graph $\mathcal{G} = (V, E)$ is undirected if and only if for any pair of node i and j,

  $$(i, j) \in E \iff (j, i) \in E.$$

  A graph, which is not an undirected graph, is a **directed graph**.

- **The circle**, also called the **cycle**. A circle with n nodes indexed by $[0, n-1]$ has $3n$ edges: For any $i \in [0, n-1]$, $(i, i-1 \mod n) \in E$, $(i, i) \in E$ and $(i, i+1 \mod n) \in E$.

  Note that, if for a given graph $\mathcal{G} = (V, E)$, one can find a subgraph $\mathcal{G}' = (V' \subseteq V, E' \subseteq E)$ that follows the above characteristics, we say that $\mathcal{G}'$ is a cycle. The term of circle is not used in that case.

- **Tree** A graph is a tree if it does not have cycles.

- **The circle of radius** k, also called the **cycle of radius** k. A circle of radius k with n nodes indexed by $[0, n-1]$ has $(2k+1)n$ edges: For any $i \in [0, n-1]$, and any $j \in [0, k]$, $(i, i-j \mod n) \in E$ and $(i, i+j \mod n) \in E$.

  Note that the circle of radius 1 is called circle in short.

- **The toroidal lattice**, also called the **lattice embedded on the torus**. A toroidal lattice with n^2 nodes indexed by $[0, n-1] \times [0, n-1]$ has $5n^2$ edges: for any node of index (i, j), with $0 \leqslant i, j \leqslant n-1$,

$$\begin{aligned}
\big((i,j),(i,j)\big) &\in E,\\
\big((i,j),(i,j-1 \mod n)\big) &\in E,\\
\big((i,j),(i,j+1 \mod n)\big) &\in E,\\
\big((i,j),(i-1 \mod n,j)\big) &\in E,\\
\big((i,j),(i+1 \mod n,j)\big) &\in E.
\end{aligned}$$

- **The lattice**. A lattice with n^2 nodes indexed by $[0, n-1] \times [0, n-1]$ has fewer edges than the toroidal lattice: for any node of index (i, j), the possibly redundant following edges are in E:

$$\begin{aligned}
\big((i,j),(i,j)\big) &\in E,\\
\big((i,j),(i,\max(j-1,0))\big) &\in E,\\
\big((i,j),(i,\min(j+1,n-1)\big) &\in E,\\
\big((i,j),(\max(i-1,0),j)\big) &\in E,\\
\big((i,j),(\min(i+1,n-1),j)\big) &\in E.
\end{aligned}$$

- **The toroidal lattice of radius** r, also called the **lattice of radius** r **embedded on the torus**. A toroidal lattice of radius r with n^2 nodes indexed by $[0, n-1] \times [0, n-1]$ has $(2r+1)^2 n^2$ edges: for any node of index (i, j), and any $(k_i, k_j) \in [-r, r] \times [-r, r]$,

$$\big((i,j),(i+k_i \mod n, j+k_j \mod n)\big) \in E.$$

  Note the neighborhoods in toroidal lattices are square shaped and that the toroidal lattice of radius 1 is not equal to the toroidal lattice.

- **The lattice of radius** r**.** A lattice of radius r with n^2 nodes indexed by $[0, n-1] \times [0, n-1]$ has less edges than the toroidal lattice: for any node of index (i, j), and any $(k_i, k_j) \in [-r, r] \times [-r, r]$ the possibly redundant following edge is in E:

$$\Big((i,j),\Big(\max\big(\min(i+k_i,n-1),0\big),\max\big(\min(j+k_j,n-1),0\big)\Big)\Big) \in E,$$

  The lattice of radius 1 is not equal to the lattice.

1.3 Dissertation Outline

We briefly describe here the architecture of this dissertation.

- Chapter 2 describes all the distributed averaging algorithms that we study on random geometric graphs or lattices.
- Chapter 3 gives the theoretical tools needed to study the convergence and the performance of a distributed averaging algorithm. In particular, it points out the importance of the second largest eigenvalue of the expected matrix of averaging weights.
- Chapter 4 is dedicated to the second largest eigenvalues in magnitude. First, it lists useful theorems on the topic. Then it computes the second largest eigenvalues on basic topologies as cycles of radius k, lattices of radius k, and some special trees. The goal is to understand the limits of distributed consensus on lattices of radius k, and on trees, which symbolize the presence of bottlenecks. Finally, it briefly lists some useful formulas on Laplacians.
- Chapter 5 exhaustively computes the performance of the algorithms of Chapter 2. It often refers to theorems in Chapters 3 and 4.
- Chapter 6 is two-fold. First it solves the interval consensus problem, which gives as a by-product a solution to the voting problem. Second, it gives a solution for multiple voting problem, where agents vote among 3 or 4 candidates.

Finally, as the chapters are rather long, we placed framed summaries of the main results at the end of each chapter, and at the end of Section 3.1. We make an exception for Chapter 5, where the summary is replaced by a table that gathers the performances of all the gossip algorithms (Table 5.5.4).

1.4 Contributions

Any theorem, definition or lemma, that is a contribution of this thesis, will be followed by a star. For example,

Theorem* 1.4.1. is a contribution of this thesis.

We distinguish three kinds of contributions: first we list the novel algorithms produced in this thesis, then we describe our theoretical contributions, and finally, we explain why we believe that this thesis contributes on a pedagogical level to the field.

The detailed list of the contributions is as follows:

- Algorithmic Contributions:
  - We propose path averaging.
  - We propose both the naive and evolved versions of one-way path averaging.
  - We improve geographic gossip, by showing that rejection sampling is not necessary.

  - We design interval consensus gossip, as well as binary, ternary and quaternary voting automata.
  - We design a simple and efficient algorithm to compute an approximation of the fastest mixing chain on circles of radius k, in the case where k is significantly smaller than the number of nodes in the circle.

- Theoretical Contributions:
  - We introduce the notion of consensus time T_c and consensus cost C_c, as well as their bounds[2]. We also introduce the empirical consensus times and costs as tools to measure performance on simulations.
  - We derive a novel and accurate bound on the mixing time of Markov chains on balanced trees. We also list some applications of the bound, in particular in the computation of the fastest mixing chain on balanced trees.
  - We prove that path averaging converges in $O(n \log n)$ messages with high probability under some conditions.
  - We prove that one-way path averaging converges correctly with probability 1.
  - We find sufficient conditions under which greedy routing provably reaches destination in random geometric graphs with high probability.
  - We find sufficient conditions under which all the nodes of a random geometric graph have degree $\Theta(\log n)$ with high probability.
  - We state and solve the interval consensus problem. We solve the voting problem, and we state and solve the multiple voting problem.

- Pedagogical and Technical Contributions:
  - We perform an extensive review of gossip algorithms on random geometric graphs, and we clear some inaccuracies left in published in papers.
  - We gather in a chapter the tools needed to study gossip algorithms on random geometric graphs.
  - We point out the utility of Poincaré's inequality to study distributed averaging algorithms. Using this theorem, we produce alternate proofs for the performance of all the algorithms studied in this thesis. This unifies the analysis.

[2] Joint work with Patrick Denantes

Chapter 2

Distributed Average Consensus Algorithms

A set of agents can communicate through a connected network. Each agent initially holds a scalar measurement and wants to learn the average of all the measurements in the network in a distributed way. The goal of gossip algorithms is to solve this distributed average consensus problem. Agents do not necessarily know much about the network: they may not know the number of nodes, the network topology, etc. In addition, in some frameworks, the network topology can vary with time because of link instability or node mobility. However, the network wants to reach average consensus in a reliable and robust manner.

2.1 Time Models: Synchronicity and Asynchronicity

We distinguish two categories of average consensus algorithms: the synchronous algorithms and the asynchronous algorithms. In both cases, the algorithms are iterative and every node holds an estimate of the measurements' average. The algorithms are designed such that all the estimates in the network converge to the sought average up to any desired level of precision. An iteration is threefold.

- First one or several nodes wake up.
- Then the woken nodes send their estimates to one or several neighbors in the network.
- Finally the nodes update their estimates to a value which depends on their estimates and on the estimates they have potentially received.

On the one hand, in a synchronous algorithm, all the nodes in the network wake up at each iteration and broadcast their estimates. All the nodes update their estimates before the next iteration can start. On the other hand, in an asynchronous algorithm, only one node wakes up at each iteration. The woken node calls some chosen neighbors. Only a subset of nodes in the network update their estimates at the end of each iteration.

Synchronous algorithms require the synchronization of the network, which might be a difficult constraint in some distributed settings. Also the nodes should agree on the time the algorithm starts running. For these reasons, synchronous algorithms seem unadapted to solve the distributed average consensus problem, yet it is important to study and understand them. Indeed synchronous algorithms are often much simpler to analyze than asynchronous algorithms, and they form a powerful theoretical benchmark for the average consensus problem.

Asynchronous algorithms do not require network synchronization, which is a great asset when solving distributed computing problems. The asynchronous algorithms described in this thesis run with the asynchronous time model developed in [15] and [14]. In this model, each agent has an independent clock whose "ticks" are distributed as a rate λ Poisson process. An agent wakes up and starts an iteration whenever its clock ticks. Nodes therefore wake up at random times in a random order. In the analysis, time is measured in terms of the number of ticks of an equivalent single virtual global clock ticking according to a rate $n\lambda$ Poisson process, where n is the number of nodes in the network. At each tick of this equivalent global clock, one node in the network wakes up uniformly at random.

In this model, the inter-time between each iteration is inversely proportional to $n\lambda$. If λ is too large, then there is a high chance that a node wakes up while some ongoing iteration is being run. If several iterations coexist in the network, they might interfere and nodes might fail to update their estimates. In sensor networks, energy limitations are such that it is preferable not to waste energy in useless communications. If λ is small enough then there is only one iteration at a time with high probability, and each communication has greater chances to succeed. There is therefore a tradeoff between the number of communication used to have the algorithms converge, and the delay of the algorithms. All the asynchronous algorithms in this thesis can handle coexisting iterations in the network, but the associated congestion control issues are beyond the scope of this work. As a consequence, in the analysis, we assume that iterations do not interfere, which is equivalent to consider that there is always only one iteration at a time in the network. Furthermore, we count time in terms of iteration number. We will refer to the time between two consecutive clock ticks as one timeslot and at each timeslot, one iteration occurs.

It is possible to relate number of iterations and absolute time thanks to Cramer's theorem from large deviations theory. If Z_k denotes the time of the kth tick of the global clock, then $Z_{k+1} - Z_k$ are IID exponentials of rate $n\lambda$. As shown in [14], $\mathbb{E}[Z_k] = \sum_{j=1}^{k} \mathbb{E}[Z_j - Z_{j-1}] = k/(n\lambda)$, and, for any $\delta > 0$, Cramer's theorem [23] implies that:

$$\mathbb{P}\left[\left|Z_k - \frac{k}{n\lambda}\right| \geqslant \frac{\delta k}{n\lambda}\right] \leqslant 2\exp\left(-\frac{\delta^2 k}{2}\right).$$

As a consequence, for $k \geqslant n$,

$$Z_k = \frac{k}{n\lambda}\left(1 \pm 2\sqrt{\frac{\log n}{n}}\right),$$

with probability at least $1 - 2/n^2$.

2.2 Description of Average Consensus Algorithms

This section lists a number of synchronous and asynchronous average consensus algorithms. In order to eliminate quantization effects, it is assumed in this thesis that nodes can store scalars with infinite precision and that the communication channels transmit with infinite rate. Many quantized averaging algorithms have been derived, but they are not the main concern of this thesis. Nevertheless, Section 6.1 incorporates quantization constraints to the picture, and briefly reviews literature on this matter. A novel quantized algorithm developed in this thesis is the object of Chapter 6.

Mathematical set up Each of the n agents forming the network $\mathcal{G}$ initially measures some scalar value, and the measurement of agent i is denoted by $x_i(0)$. The goal of an average consensus algorithm is to let all the agents know the average of measurements

$$x_{ave} = \frac{1}{n}\sum_{i=0}^{n} x_i(0).$$

In order to achieve this goal, each agent i holds at any iteration t an estimate $x_i(t)$ of the average x_{ave}. All the estimates are gathered in a vector of size n denoted by $\mathbf{x}(t)$. The goal of an average consensus algorithm can thus be restated as:

$$\begin{aligned}\lim_{t\to\infty} \mathbf{x}(t) &= \frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x}(0)\\ &= x_{ave}\mathbf{1},\end{aligned}$$

where $\mathbf{1}$ denotes the vector of all ones.

At each iteration t one or several nodes receive the estimates of one or several other nodes. Based on the received estimates, nodes update their own estimate according to certain update rules. The common feature between all the algorithms described in this chapter is the linearity of these estimate update rules. For any algorithm and any of its iteration t, there is a matrix $\mathbf{W}(t)$ such that:

$$\mathbf{x}(t) = \mathbf{W}(t)\mathbf{x}(t-1). \tag{2.1}$$

The design of the matrix $\mathbf{W}(t)$ differs from one algorithm to another, and the rest of this section is devoted to their precise description. The next section shows that all these algorithms converge correctly, but a foretaste of the mathematical constraints on $\mathbf{W}(t)$ is needed to properly appreciate the reading of the algorithms.

In a nutshell, there are three main constraints on the matrices $\mathbf{W}(t)$. First, consensus has to be stable. If, at time t, all nodes agree on one estimate, then they should keep agreeing on this estimate for times greater than t. Because of Eq. (2.1), this mathematically translates to:

$$\mathbf{W}(t)\mathbf{1} = \mathbf{1}. \tag{2.2}$$

Second, in order to guarantee that the only possible consensus is *average* consensus, the average of the estimates at any time should be equal to x_{ave}. In

other words, the sum of the estimates should remain constant:

$$\mathbf{1}^T\mathbf{W}(t) = \mathbf{1}^T. \tag{2.3}$$

Finally, the estimates should contract to the true average. This notion will be clarified in Section 3.1 thanks to spectral tools.

2.2.1 Synchronous average consensus algorithms

In synchronous average consensus algorithms (see Alg. 2.2.1), all the nodes activate at each time slot t, communicate with their neighbors and update their current estimate. The $(i,j)^{th}$ entry of the matrix $\mathbf{W}(t)$ is denoted by $\mathbf{W}_{ij}(t)$, and the set of neighbors of node i at time t is denoted by $\mathcal{N}_i(t)$. Then, for any time t, and any node i, the local update rule can be written as

$$x_i(t) = \mathbf{W}_{ii}(t)x_i(t-1) + \sum_{j\in\mathcal{N}_i(t)} \mathbf{W}_{ij}(t)x_j(t-1). \tag{2.4}$$

Algorithm 2.1: Synchronous average consensus with averaging matrices $\{\mathbf{W}(t)\}_{t\geqslant 1}$.

For any node i, let $x_i(0)$ be its measurement, let t_{end} be a number of iterations and let $\{\mathbf{W}(t)\}_{t\geqslant 1}$ be the sequence of averaging matrices. The matrices $\{\mathbf{W}(t)\}_{t\geqslant 1}$ can be defined in various ways. For example, $\{\mathbf{W}(t)\}_{t\geqslant 1}$ can be constructed with uniform weights, metropolis weights or adaptive weights, the three examples given in this thesis.

1: **for** $t = 1 : t_{end}$ **do**
2: Every node i broadcasts its estimate $x_i(t-1)$
3: Every node i updates its estimate:

$$x_i(t) \leftarrow \mathbf{W}_{ii}(t)x_i(t-1) + \sum_{j\in\mathcal{N}_i(t)} \mathbf{W}_{ij}(t)x_j(t-1),$$

where $\mathcal{N}_i(t)$ is the set of nodes from which i has received estimates.
4: **end for**

The main challenge for any node i in the design of $\mathbf{W}$ is to be able to locally learn its weights $\mathbf{W}_{i\cdot}$. Based on the knowledge of its neighborhood $\mathcal{N}_i(t)$ and maybe on some extra information sent by its neighbors, node i should choose weights that are coordinated with the weights of the other nodes, so that in particular the global constraint $\mathbf{1}^T\mathbf{W}(t) = \mathbf{1}^T$ and the local constraint $\mathbf{W}(t)\mathbf{1} = \mathbf{1}$ are verified.

If the communication graph $\mathcal{G}$ does not change with time, and if it is known, it is interesting to centrally compute a efficient matrix $\mathbf{W}$ and to implement its weights at each node. But if the communication graph is time-varying, which is most often the case in wireless settings, then nodes should have agreed beforehand on a strategy to distributively adapt their weights to the topology variations.

There are three main average consensus strategies for the choice of the weights when $\mathcal{G}$ is time-varying ([78, 79, 24]).

- **Uniform weights** Uniform weights is the simplest strategy. For any neighboring nodes i and j, weight w_{ij} is chosen to be equal to a constant α, which is common to the whole network. Nodes locally adapt the value of their self loop weight $w_{ii}(t)$ by counting how many neighbors they have at time t:

$$\mathbf{W}_{ij}(t) = \begin{cases} \alpha & \text{if } j \in \mathcal{N}_i(t) \\ 1 - \alpha\,|\mathcal{N}_i(t)| & \text{if } i = j \\ 0 & \text{otherwise,} \end{cases} \tag{2.5}$$

  where $|\cdot|$ denotes cardinality. For the estimates to converge, the constant α should be designed small enough. Section 3.1 shows that any $\alpha < 1/\max_i d_i$ is a proper constant, where d_i denotes the degree of node i. Note that $\alpha = 1/n$ works for any graph topology.

- **Metropolis weights** In the Metropolis weights strategy, nodes are required to know their neighbors' degrees. Assuming that the communication graph has no unidirectional links, a possible way to implement Metropolis weights is the following: nodes first broadcast their estimates, then they broadcast the number of estimates they have received. Finally, the following weights can be distributively computed:

$$\mathbf{W}_{ij}(t) = \begin{cases} \frac{1}{1+\max\{d_i(t),d_j(t)\}} & \text{if } j \in \mathcal{N}_i(t) \\ 1 - \sum_{k\in\mathcal{N}_i(t)} w_{ik}(t) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

- **Adaptive weights** In this scheme, the network is modeled as a given graph $\mathcal{G}$ whose links may fail over time. Define a weight matrix $\mathbf{W}^{\mathcal{G}}$ which realizes a successful average consensus on the fixed graph $\mathcal{G}$. At each iteration t, weights $w_{ij}(t)$ are designed based on current failures and on $\mathbf{W}^{\mathcal{G}}$. Node $j \in \mathcal{N}_i(t)$ if and only if the link between nodes i and j does not fails at time t. If link (i,j) does not fail, then $\mathbf{W}_{ij}(t) = \mathbf{W}^{\mathcal{G}}_{ij}$. If link (i,j) fails, then necessarily $\mathbf{W}_{ij}(t) = 0$, and nodes i and j add weight $\mathbf{W}^{\mathcal{G}}_{ij}$ to their self loop coefficients $\mathbf{W}_{ii}(t)$ and $\mathbf{W}_{jj}(t)$:

$$\mathbf{W}_{ij}(t) = \begin{cases} \mathbf{W}^{\mathcal{G}}_{ij} & \text{if } j \in \mathcal{N}_i(t) \\ 1 - \sum_{k\in\mathcal{N}_i(t)} \mathbf{W}_{ik}(t) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

  This strategy is called *optimal adaptive weights* when $\mathbf{W}^{\mathcal{G}}$ is the fastest mixing averaging matrix on $\mathcal{G}$ when there is no link failure. The fastest mixing averaging matrix on a graph is the doubly stochastic matrix whose positive entries correspond to edges of the graph, and which minimizes the second largest eigenvalue in magnitude. Fastest mixing averaging matrices will be studied in Section 4.1.

  Note that links are assumed here to fail in both directions: If link (i,j) fails, then node i cannot send a message to node j *and* node j cannot send a message to node i either. Unidirectional link failures have been studied in [30], with some interesting first results despite the difficulty of the problem. The authors in [30] consider two synchronous algorithms. One

is the adaptive weights algorithm, which they call "the biased compensation model". The other algorithm is called "the balanced compensation model" in which the weights are designed in the following way:

$$w_{ij}(t) = \begin{cases} \mathbf{W}^{\mathcal{G}}_{ij}/K & \text{if } j \in \mathcal{N}_i(t) \\ \mathbf{W}^{\mathcal{G}}_{ii}/K & \text{if } i = j \\ 0 & \text{otherwise,} \end{cases} \tag{2.8}$$

where

$$K = \mathbf{W}^{\mathcal{G}}_{ii} + \sum_{j \in \mathcal{N}_i(t)} \mathbf{W}^{\mathcal{G}}_{ij}. \tag{2.9}$$

They show on simulations (with random independent unidirectional link failures) that the balanced compensation model seems to converge faster to a consensus than the biased model, but that the latter seems to reach a better estimation of x_{ave} than the former. Some analytical results were obtained only when $\mathbf{W}^{\mathcal{G}}$ is a Cayley matrix (refer to [30] for further details). For most algorithms (except for broadcast gossip), this thesis will avoid unidirectional link failures and directional networks, because their analysis is hard and little is known today in this area.

2.2.2 Gossip or asynchronous average consensus algorithms

In asynchronous average consensus algorithms, only one node wakes up at each iteration. This node initiates an averaging process, which can be more or less complex, depending on the asynchronous algorithm.

Gossip is a vast sub-class of asynchronous average consensus algorithms that is extensively studied in this thesis. There is an asynchronous algorithm called broadcast gossip which is not strictly speaking a gossip algorithm, because it does not fit the mathematical definition of gossip which we give in this section. Broadcast gossip being important in practice, Section 2.2.3 is especially dedicated to this algorithm.

Definition* 2.2.1 (Gossip algorithm). A gossip algorithm is a distributed iterative algorithm, where at each iteration t, a random subset $S(t)$ of nodes update their estimates to the average of the estimates of $S(t)$: for all $j \in S(t)$,

$$x_j(t+1) = \frac{1}{|S(t)|} \sum_{i \in S(t)} x_i(t). \tag{2.10}$$

There are four well-known gossip algorithms: standard gossip (also called pairwise gossip or nearest-neighbor gossip), neighborhood gossip, geographic gossip and path averaging. In both standard gossip and geographic gossip, $S(t)$ always has two nodes, whereas in neighborhood gossip and in path averaging, where whole neighborhoods and paths are being averaged at each iteration, $S(t)$ has a random number of nodes. Note that Definition 2.2.1 is specific to this thesis and that gossip may have a different meaning in other contexts.

Gossip algorithms are convenient, because they verify Property 2.2.1, which later greatly simplifies their theoretical analysis:

Property 2.2.1 (Gossip matrices). For any gossip algorithm (see Definition 2.2.1), at any iteration t, the matrix $\mathbf{W}(t)$ is a projection matrix:

$$\mathbf{W}(t)\mathbf{W}(t) = \mathbf{W}(t).$$

Furthermore, $\mathbf{W}(t)$ is a symmetric matrix:

$$\mathbf{W}(t)^T = \mathbf{W}(t).$$

Proof: $\mathbf{W}(t)\mathbf{W}(t) = \mathbf{W}(t)$: If a neighborhood is averaged twice in a row, the second averaging does not bring any change to the vector of estimates.
$\mathbf{W}(t)$ is a symmetric matrix: obvious. ■

Because gossip algorithms are easier to study than other asynchronous algorithms, they are a good reference. It is now time to describe the four gossip algorithms that were mentioned earlier. Note that they are listed in increasing performance order.

Standard gossip (or pairwise gossip)

Standard gossip is the most famous averaging consensus algorithm ([15, 14, 24, 31]).

Assumption: The network is jointly connected in time.

Algorithm 2.2: Standard gossip or Pairwise gossip. [15, 14, 24, 31]

For any node i, let $x_i(0)$ be its measurement, and let t_{end} be a number of iterations.

1: **for** $t = 1 : t_{end}$ **do**
2: Node i's clock ticks, so that node i wakes up.
3: Node i broadcasts its estimate $x_i(t-1)$.
4: Any node that has received $x_i(t-1)$ draws a random back off time.
5: Let j be the neighbor with the smallest back off time. Node j sends its estimate $x_j(t-1)$ to node i.
6: Node i acknowledges the reception.
7: Neighbors different than j hear the acknowledgement and do not send their estimates.
8: Both nodes i and j update their estimates to

$$\begin{aligned} x_i(t) &\leftarrow \frac{x_i(t-1) + x_j(t-1)}{2}, \\ x_j(t) &\leftarrow \frac{x_i(t-1) + x_j(t-1)}{2}. \end{aligned}$$

9: **end for**

Comments: In a more classical version of standard gossip, when node i wakes up, it sends its estimate to a neighbor j that it chooses at random with probability p_{ij}. The two versions are mathematically equivalent, since the distribution of back off times can be tuned so that the first neighbor to answer is indeed node j with probability p_{ij}. For example, if neighbors are chosen with uniform probability, it is equivalent to set identical back off distributions in each node.

The first version of the algorithm might be preferable since nodes might not be able to maintain an accurate table of neighbors. Indeed, note that in the first version, node i does not need to know its neighborhood, whereas in the second version, it does. In the second version, nodes shall always have their radio turned on to listen if they are called. Otherwise neighborhoods are too time-varying. On the contrary, in the first version, nodes might turn their radio off from time to time without disturbing the algorithm. Remember that the only constraint is that the sequence of graphs should be jointly connected in time. As a consequence, the first version can potentially save energy compared to the second version.

If for some reason, node j does not receive the acknowledgement of node i, then node j does not update its estimate. Indeed, it assumes that if it has just received the first packet, it should be able to receive the acknowledgement as well. However, it is possible that node i updates its estimate, but that node j does not receive the acknowledgement. In that case, the final consensus will have some error, which is hopefully not very large.

Neighborhood gossip

Neighborhood gossip (see Alg. 2.3) is very similar to standard gossip. It was introduced by Nazer et al. in [57, 58]. As its name indicates, the estimates of a whole neighborhood are averaged at each iteration of neighborhood gossip.

Assumptions: The network is jointly connected in time.

Algorithm 2.3: Neighborhood gossip [57, 58].

For any node i, let $x_i(0)$ be its measurement, and let t_{end} be a number of iterations.

1: **for** $t = 1 : t_{end}$ **do**
2: Node i's clock ticks, so that node i wakes up.
3: Node i broadcasts a wake up message.
4: Any node that has received the wake up message sends its estimate to node i.
5: Let $\mathcal{N}_i(t)$ be the set of nodes from which node i receives estimates. Node i computes

$$y \leftarrow \frac{1}{|\mathcal{N}_i(t)|} \sum_{k \in \mathcal{N}_i(t)} x_k(t-1),$$

6: Node i broadcasts y and the identities of $\mathcal{N}_i(t)$.
7: Node i and all the nodes in $\mathcal{N}_i(t)$ (which received y) update their state to y.
8: **end for**

Comments: As explained in [57], step 4 can be modified to save both time and energy. Instead of sending their estimates to node i one by one, or on orthogonal channels, the neighbors of node i can use computation coding and send their estimates all at the same time. In this scheme, node i receives the sum of the signals sent by its neighbors. A computation code being linear, node i can decode the sum of its neighbors estimates directly. To achieve this, each node j needs to know the channel characteristics $(r_{ij}, \theta_{ij}(t))$ from itself to node i, where r_{ij} is the distance between nodes i and j, and $\theta_{ij}(t)$ is a phase

coefficient. This scheme saves time because nodes can send their estimate at the same time, and not in sequence. It saves energy because a sum of signals has a better signal over noise ratio (SNR) than one signal on its own. For further details, refer to [57, 58].

Geographic gossip

As the next chapter shows, on grids and random geometric graphs, both standard gossip and neighborhood gossip require an enormous amount of communications to converge. On complete graphs however, these algorithms are very efficient. From this observation came the idea of creating virtual complete graphs. Instead of averaging estimates with their close neighbors only, nodes can route estimates in the network so that any pair of nodes can average their values. Although routing is expensive in terms of communication cost, the gain in the number of averaging rounds is such that routing is worth the effort. Since the resulting gossip algorithm assumes some spatial knowledge of the network, Dimakis et al. called it geographic gossip [27].

Assumptions: Geographic gossip (Alg. 2.5) works on networks where nodes know their locations as well as the locations of their neighbors, so that greedy routing (Alg. 2.4) can be performed. The network is assumed to be dense enough so that greedy routing is able to generate long routes. More precisely, the network is assumed to be a grid or a random geometric graph with radius $r(n) \geqslant \sqrt{52 \log n/9n}$, embedded in a unit square. This choice of connection radius is explained in Appendix A.2.2, where Lemma A.2.2 shows that if $r(n) \geqslant \sqrt{52 \log n/9n}$, then greedy routing reaches the closest node to the target with high probability.

Algorithm 2.4: Greedy Routing

Assume that a node s has a message m to transmit, and a target location T.

1: Initialize $i \leftarrow s$
2: Node i searches its neighbor j that is the closest to location T.
3: **if** $j = i$ **then**
4: Node i is the final receiver of message m and greedy routing terminates.
5: **else**
6: Node i sends message m and target location T to node j.
7: $i \leftarrow j$ and go to step 2.
8: **end if**

Comments: Note that, if greedy routing is guaranteed to reach the closest node to the target, then, in step 6 of Alg. 2.5, node j can route back the new estimate y using greedy routing as well. This solution is more robust to link failures, but it is risky to assume that greedy routing always succeeds in the first place. To solve this robustness issue in step 6 of Alg. 2.5, there are some evolved routing protocols which can robustly route a message to a node whose location is known [32, 1, 49].

In geographic gossip, each iteration costs L messages, where L is the route length (in hops) generated by greedy routing. In a lattice, $L \leqslant 2\sqrt{n}$, and in a random geometric graph $L \lesssim D/r(n)$, where D is the diameter of the domain where nodes are embedded, and where $r(n)$ is the communication radius. In

the classical case where nodes are embedded in the unit square, $D = \sqrt{2}$.

Algorithm 2.5: Geographic gossip.

For any node i, let $x_i(0)$ be its measurement, and let t_{end} be a number of iterations.

1: **for** $t = 1 : t_{end}$ **do**
2: Node i's clock ticks, so that node i wakes up.
3: Node i draws a target location T uniformly at random in the unit square.
4: Node i starts greedy routing with target T and message $m = (x_i(t-1), \ell_i)$, where ℓ_i is the location of node i.
5: Let j be the destination node of the previous greedy route. Node j computes
$$y = \frac{x_i(t-1) + x_j(t-1)}{2},$$
and updates its estimate to $y : x_j(t) \leftarrow y$.
6: Node j sends the update y back to node i, using the same route backwards.
7: Node i updates its estimate to y: $x_i(t) \leftarrow y$.
8: **end for**

At each iteration, the matrix $\mathbf{W}(t)$ is a standard gossip matrix. Let p_{ij} be the probability that nodes i and j average their estimates at a given iteration. Then, on a random geometric graph, if nodes wake up independently and uniformly at random, and if greedy routing always succeeds to reach the closest node to the target, we have

$$p_{ij} = \frac{V_i + V_j}{n}, \tag{2.11}$$

where V_k denotes the area of the Voronoi cell of node i. As Equation (2.11) shows, nodes are queried with a rate proportional to the area of their Volonoi cell, which introduces an unbalanced situation. In [27], the authors suggest to implement a rejection sampling scheme for large Voronoi cells, so that nodes with such Voronoi cells are not over queried. Rejection sampling advantageously simplifies the analysis of the algorithm, by ensuring the global balance of the algorithm. However, routing first information and next rejecting it, is not a very convincing idea. Once a node receives an estimate, it might as well use it and finish the averaging round. Therefore, this thesis studies geographic without rejection sampling.

Path averaging

In geographic gossip, any pair of nodes can average their estimates to the expense of routing the needed information. The idea of path averaging ([9, 8]) is that the whole route might as well participate to the averaging effort, because all the estimates of the nodes in the route can be averaged together with no extra communication cost. Interestingly, under some circumstances described in Chapter 5, path averaging performs optimally in the number of communications it requires.

In order to increase the performance of path averaging, it is better to use a slightly different routing algorithm, called *random* greedy routing. Random greedy routing is very similar to greedy routing. Assume that a node s has a message m to transmit, and a target location T. Then, in random greedy routing, node s sends message m to a random neighbor closer to the target, whereas in greedy routing, s sends m to its neighbor that is the closest to the target. An important advantage of random greedy routing is that a node does not need to keep track of its neighborhood.

Random greedy routing is described in Alg. 2.6. It is included again in the description of path averaging (Alg. 2.7) because path averaging processes the message along the route, and because we need to make sure that no pair of routes cross each other during their processing. Therefore, random greedy routing in Alg. 2.7 appears in a slightly different and more complicated way than in Alg. 2.6.

Assumptions: The assumptions to run path averaging are the same as in geographic gossip. Nodes are assumed to know their locations. The network should be dense enough so that *random* greedy routing is able to generate long routes. More precisely, the network is assumed to be a grid or a random geometric graph with radius $r(n) \geqslant \sqrt{52 \log n / 9n}$, embedded in a unit square.

Algorithm 2.6: Random greedy Routing

Assume that a node s has a message m to transmit, and a target location T.

1: Initialize $i \leftarrow s$
2: Node i broadcasts m and T.
3: Let $\mathcal{N}_{\text{closer}}$ the set of nodes which have received message m and which are closer to T than node i. Any node in $\mathcal{N}_{\text{closer}}$ draws a random back off time.
4: **if** $\mathcal{N}_{\text{closer}} \neq \emptyset$ **then**
5: Let j be the node with the smallest back off time. Node j acknowledges to node i.
6: Node i sends message m and target location T to node j.
7: $i \leftarrow j$ and go to step 2.
8: **else**
9: Node i is the final receiver of message m and greedy routing terminates.
10: **end if**

Comments: In step 11 of Alg. 2.7, if node i does not overhear the broadcast message m of node j, then node i might send another acknowledgement.

Note that thanks to the activity state, no node is involved in two different routes at the same time.

Ignoring the short acknowledgment communications, path averaging and geographic uses the same order of communications per rounds, although random greedy routes are longer than classical greedy routes. However, path averaging requires less averaging rounds to converge, and it therefore saves energy compared to geographic gossip. The advantage of random greedy routing is that nodes do not need to know their neighborhood, and nodes can thus turn off their radios from time to time. Furthermore, routes are more diverse in random greedy routing, which, as explained later, accelerates the convergence

Algorithm 2.7: Path averaging.

For any node i, let $x_i(0)$ be its measurement. All the nodes are set to an *inactive* state. An *active* node temporarily freezes its clock until it turns back to the *inactive* state. The goal of this activity state is to avoid route collisions. Let t_{end} be a number of iterations.

1: **for** $t = 1 : t_{end}$ **do**
2: At time t, node s's clock ticks, so that node s wakes up and sets its state to *active*.
3: Node s draws a target location T uniformly at random in the unit square.
4: Node s broadcasts message $m = (y, \ell_s, T, C)$, where ℓ_s is the location of node s, C is a counter variable initiated at $C = 1$, and y is the route estimates aggregation initialized at $y = x_s(t)$.
5: $i \leftarrow s$.
6: Let $\mathcal{N}_{\text{closer}}$ the set of inactive nodes which have received message m and which are closer to T than node i. Any node in $\mathcal{N}_{\text{closer}}$ draws a random back off time.
7: **if** $\mathcal{N}_{\text{closer}} \neq \emptyset$ **then**
8: Let j be the neighbor with smallest back off time. When its back off time expires, node j sends a notification message to node i and it sets its state to *active*.
9: Neighbors that overhear the notification message, withdraw from the iteration.
10: Node i acknowledges to node j. (If node i receive other notification messages, it acknowledges them negatively so that they turn back to *inactive*.)
11: When node j receives the acknowlegment, it broadcasts message $m = (y + x_j(t), \ell_j, T, C + 1)$, and it memorizes the location ℓ_i of its predecessor i.
12: $i \leftarrow j$ and go to step 6.
13: **else**
14: After a silent time (no notification message), node i considers itself as the final receiver of message m and random greedy routing terminates.
15: Node i computes $z = y/C$, updates its estimate to z, sets is state to *inactive*, and sends z back to node s by the same route, which is possible since all the nodes in the route memorized their predecessors. On the way, all the nodes update their estimate to z and set their states to *inactive*.
16: **end if**
17: **end for**

of the algorithm.

2.2.3 Broadcast gossip

All the gossip algorithms described in previous section are guaranteed to work if pairwise agreement can always be reached. Unfortunately, it is impossible to have two nodes consciously agree on a common action in a deterministic way, if the communication channel is unreliable [37, 36]. Randomized pairwise agreement algorithms exist, but they require many communications between the nodes, which low-energy sensors cannot afford. In order to intuitively understand this impossibility statement, take two nodes i and j, where node i wakes up and calls node j in the hope that they both pairwise average their estimates. In the description of standard gossip in Paragraph 2.2.2, node i sends its estimate x_i to a node j and node j sends its estimate x_j in return. Node i acknowledges the reception of x_j and updates its estimate. When node j receives the acknowledgement, it also updates its state. Node j knows that node i has updated its state, but node i does not know if node j did. Indeed, the acknowledgement might have failed to reach node j. One might want to solve this issue by adding an acknowledgment from node j to node i at the end of the averaging round, but this just swaps the problem: Node i receives the acknowledgment and knows that node j updated, but node j does not know if node i received the last acknowledgment.

When two kids want to exchange their toys, they have no guarantee that the other kid will actually give up its own toy. As a consequence, none of them wants to give its toy first. As kids in the playground, nodes have a "you go first!" problem. However, in gossip algorithms, nodes accept the risk and continue the algorithm without knowing if previous rounds went well or not. Gossip algorithms are therefore more robust in networks that are stable on an iteration time scale. Also, if an error occurs, then it does not impact heavily the final output, because only a subset of nodes participate to a gossip round.

Since errors are not avoidable, de facto, the sum of estimates is not necessarily conserved through iterations. Broadcast gossip is an algorithm based on this observation. It was introduced by Fagnani et al. in [31], and further developed by Aysal et al. in [7]. In this algorithm, communications are always *one-way*, no pair of nodes need to agree on some common action at any moment, which is an algorithmic simplification of great importance. The trade-off is that broadcast algorithms inherently cannot conserve the sum at each iteration. However, note that it is worth dropping pairwise agreements if the final consensus estimate is close enough to x_{ave}.

The goal of the work in [7], was to design such a broadcast algorithm, and the algorithm they proposed is described in Alg. 2.8, for some mixing parameter $0 < \gamma < 1$.

This algorithm is interesting because it is very simple to implement in practice. Also it takes advantage of the broadcast nature of wireless communication, and only one communication effort is necessary to update several nodes. The main challenge is to optimize the mixing paramenter γ, so that nodes reach a consensus on a value close enough to x_{ave} without making the algorithm drastically slow down. As mentioned in p. 19, averaging algorithms are difficult to study in directed graphs. So paradoxically, broadcast algorithms were designed and optimized in undirected graphs with no link failures. In that case, although

Algorithm 2.8: Broadcast gossip [7]

For any node i, let $x_i(0)$ be its measurement, and let t_{end} be a number of iterations.

1: **for** $t = 1 : t_{end}$ **do**
2: Node i's clock ticks, so that node i wakes up.
3: Node i broadcasts its estimate $x_i(t-1)$.
4: Any node j that has received $x_i(t-1)$ updates its estimate to:

$$x_j(t) \leftarrow \gamma x_j(t-1) + (1-\gamma)x_i(t-1). \tag{2.12}$$

5: **end for**

the sum of estimates is not conserved through iterations ($\mathbf{1}^T\mathbf{W}(t) \neq \mathbf{1}^T$), the expected averaging matrix verifies $\mathbf{1}^T\mathbb{E}[\mathbf{W}(t)] = \mathbf{1}^T$. In other words, the sum of estimates is conserved *in expectation*. This will guarantee that the consensus estimate reached by the nodes is equal to the average x_{ave} in expectation.

Although the work in [7] was carried out in undirected graphs with no link failures, one can easily generalize it to the case of directed i.i.d. link failures. If a directed link fails independently of other links and of time with probability p, then the consensus estimate the nodes reach is also equal to the average x_{ave} in expectation. Indeed, although the network may be directed at any time, the resulting expected graph is undirected. In this thesis, an additional parameter p is thus introduced in the study of broadcast gossip. Finally, note that in [76], synchronous average consensus with constant weight is studied is a similar time-varying directed graph setting.

2.2.4 One-way path averaging

In geographic gossip and path averaging, not only nodes that are direct neighbors need to reach agreement at every single iteration, but faraway nodes need to coordinate and reach agreement in a single iteration as well. In both algorithms, the information therefore needs to travel back and forth in the network in order to coordinate these faraway nodes. Therefore, for robustness concerns, these algorithms are unlikely to be used in practice, although it is proven in this thesis that they converge much faster than the other algorithms. Their interest is thus mainly theoretical: they show that long distance averaging is a rich idea. This section precisely designs distributed averaging consensus algorithms, that do long distance averaging thanks to routing, but that do not route any information back. However, in this approach, neighbors still need to reach local agreements to guarantee the correctness of the algorithms.

Both algorithms designed here use routes that are used one-way only. We prove in Chapter 5 that they converge to the average with probability 1, and we show that they converge fast on simulations. Studying these algorithms in theory is not easy because, unlike geographic gossip and regular path averaging, they do not follow the nice properties of gossip algorithms (see Property 2.2.1). Both algorithms are inspired by the Push-Sum Protocol developed in [51] by Kempe et al. The Push-Sum protocol is synchronous, i.e. all the nodes wake up at each iteration. However, it is important to notice that this algorithm can be made asynchronous by having only one node or a subset of nodes wake

up at the same time. Here the synchronicity of the Push-Sum protocol is a great advantage because it implies that several averaging rounds can *coexist* in the network. The one-way averaging algorithms developed in this section will inherit this nice property, and several routes will be allowed to coexist in the network.

Algorithm 2.9: Protocol Push-Sum [51]

For any node i, let $x_i(0)$ be its measurement, and let t_{end} be a number of iterations. Every node i maintains two variables: a sum $s_i(t)$ and a weight $\omega_i(t)$, which are initialized to $s_i(0) = x_i(0)$ and $\omega_i(0) = 1$.

1: **for** $t = 1 : t_{end}$ **do**
2: **for** node $i = 1 : n$ **do**
3: Let $\{(s_r(t-1), \omega_r(t-1))\}$ be all pairs sent to i in round $t-1$.
4: Node i updates to:

$$\begin{aligned} s_i(t) &= \sum_r s_r(t-1) \\ \omega_i(t) &= \sum_r \omega_r(t-1) \end{aligned}$$

5: Node i picks a random node j uniformly at random in the network.
6: Node i sends the pair $(s_i(t)/2, \omega_i(t)/2)$ to node j and node i (itself).
7: The estimate of node i at step t is

$$x_i(t) = \frac{s_i(t)}{\omega_i(t)}.$$

8: **end for**
9: **end for**

The key property of Algorithm 2.9 is that the sum of all sums $s_i(t)$ and the sum of all weights $\omega_i(t)$ are conserved at any step t. Although the weights vary with time, the algorithm converges to the true average because the contribution of each node i in each weight $\omega_j(t)$ tends to be $\omega_j(t)/n$ (see proof in [51]).

Naive one-way path averaging Naive one-way path averaging is an algorithm inspired by the Push-sum protocol. While the Push-sum protocol is synchronous, naive one-way path averaging is asynchronous, and while in the Push-sum protocol, nodes divide their weights by 2 to distribute them to random nodes in the network, in naive one-way path averaging, nodes divide their weights by a larger number H to distribute them to nodes along random paths.

Although the evolved version of one-way path averaging is more efficient, we present this naive version because it is simpler to understand. One-way path averaging requires the same communication scheme as regular path averaging: it uses random greedy routing (Alg. 2.6). The only difference is that nodes update their estimates along the way and that no information needs to be routed back at the end of each round. Naive one-way path averaging is asynchronous, but several rounds can coexist. The algorithm is described in Alg. 2.11.

Algorithm 2.10: Draw target

This algorithm is used to draw targets outside of the network unit square area. Let s be the node of coordinates (x_s, y_s), which needs to draw a target

1: Node s draws a location (X, Y) uniformly at random in the unit square.
2: $x_T \leftarrow x_s + 3(X - x_s)/\sqrt{(X - x_s)^2 + (Y - y_s)^2}$.
3: $y_T \leftarrow y_s + 3(Y - y_s)/\sqrt{(X - x_s)^2 + (Y - y_s)^2}$.
4: **return** target T of coordinates (x_T, y_T).

Remark: For the algorithm to work, the node which wakes up (denoted by a in Alg. 2.11) should not initiate another route between steps 4 and 8. Otherwise its weight would become negative. Therefore, in step 8, node a updates to $s_a(t)/H + s'$ and $\omega_a(t)/H + \omega'$, where (s', ω') is the sum of sums and weights node a has received between steps 4 and 8.

Brief description of naive one-way path averaging: In a few words, in this algorithm, every time a node a wakes up, it divides its sum $s_a(t)$ and its weight $\omega_a(t)$ into H equal "chunks", and distributes them to nodes on a route of length H. If there is no route crossing the route of length H during the averaging round, then node a updates its values to $(s_1/H, \omega_1/H)$. Any other node j on the route updates to $(s_j + s_1/H, \omega_j + \omega_1/H)$. If the route hits the border of the network area before H hops, then the border node b takes all the C remaining chunks for itself: it updates to $(s_b + s_1 * C/H, \omega_b + \omega_1 * C/H)$. As a result, border nodes compensate for the lack of routes passing through them.

Evolved one-way path averaging In the naive one-way path averaging algorithm, only the first node in the route distributes its contribution. On the contrary, in the evolved version, any intermediate node aggregates its own contribution to previous contributions. This allows to accelerate the diffusion of information with no extra cost compared to the naive scheme.

To simplify the description of the algorithm in Alg. 2.12, we will utilize a sub-routing called `find neighbor`, where a node broadcasts its own location and a target location as a request for a neighbor. Any node, which hears the broadcast message, and which is closer to the target, draws a random back off time. When its back off time expires, one of these nodes sends an acknowledgement. `find neighbor` returns the first node to answer the request for a neighbor. The node returned by `find neighbor` will become the next node in a route in Alg. 2.12. In the remainder of this manuscript, evolved one-way path averaging will be simply referred as "one-way path averaging".

Parameters: Since we do not analyze one-way path averaging in theory, a number of parameters deliberately remain unspecified. Further work is needed to optimize them. In particular:

1. The probability density of the target location T suggested in Alg. 2.10 is probably not the optimal one. An alternative is to choose a direction uniformly at random in $[0, 2\pi)$.

2. The probability distribution f of the random variable H (number of hops) should be made specific. In the simulations of Chapter 5, we will chose the uniform distribution on $\mathbb{N} \cap [\sqrt{2}/(2r(n)), \sqrt{2}/r(n)]$.

Algorithm 2.11: Naive one-way path averaging.

For any node i, let $x_i(0)$ be its measurement. Let t_{end} be a duration time for the algorithm, and let f be a discrete probability distribution on $[2, D]$, where D is the diameter of the network. Similarly to Algorithm 2.9, every node i maintains two variables: a sum $s_i(t)$ and a weight $\omega_i(t)$, which are initialized to $s_i(0) = x_i(0)$ and $\omega_i(0) = 1$. We are going to distinguish here the discrete time slot by k, and continuous time by t.

1: **while** $t \leqslant t_{end}$ **do**
2: Node a's clock ticks at time t, so that node a wakes up.
3: Node a draws a target location T at random *outside* the unit square using Alg. 2.10, as well as a random number H of hops using the discrete probability distribution f.
4: Node a broadcasts message $m = (s_i(t)/H, \omega_i(t)/H, \ell_a, T, C)$ at time t, where ℓ_a is the location of node a, C is a counter variable initiated at $C = H - 1$. Let $i \leftarrow a$.
5: Let $\mathcal{N}_{\text{closer}}$ the set of nodes which have received message m and which are closer to T than node i. Any node in $\mathcal{N}_{\text{closer}}$ draws a random back off time.
6: **if** $\mathcal{N}_{\text{closer}} \neq \emptyset$ **then**
7: Let j be the neighbor with the smallest back off time. When its back off timer expires, node j sends a notification message to node i.
8: Node i acknowledges to node j and updates its variables as follows:
9: **if** $i = a$ **then**
10: $s_a(t') \leftarrow s_a(t') - s_a(t) * (H - 1)/H$, where t' is the current time.
11: $\omega_a(t') \leftarrow \omega_a(t') - \omega_i(t) * (H - 1)/H$, where t' is the current time.
12: **else**
13: $s_a(t') \leftarrow s_a(t') + s_a(t)/H$, where t' is the current time.
14: $\omega_a(t') \leftarrow \omega_a(t') + \omega_i(t)/H$, where t' is the current time.
15: **end if**
16: When node j receives the acknowlegment, it updates the counter $C \leftarrow C - 1$.
17: **if** $C > 0$ **then**
18: Node j broadcasts message $m = (s_i(t)/H, \omega_i(t)/H, \ell_j, T, C)$.
19: $i \leftarrow j$, and go to step 5.
20: **else**
21: The averaging round is terminated.
22: **end if**
23: **else**
24: After a silent time (no notification message), node i considers itself as the final receiver of message m and random greedy routing terminates.

25: $s_i(t') \leftarrow s_i(t') + s_a(t) * C/H$, where t' is the current time.
26: $\omega_i(t') \leftarrow \omega_i(t') + \omega_a(t) * C/H$, where t' is the current time.
27: **end if**
28: At any time t, the estimate of node i is

$$x_i(t) = \frac{s_i(t)}{\omega_i(t)}. \tag{2.13}$$

29: **end while**

3. Unlike the other algorithms described previously, here it may be preferable not to have the clock intensity equal across nodes and time. In particular, one can choose a parameter α, and an intensity λ_0 such that the clock intensity at any node i at time t is:

$$\lambda_i(t) = \lambda_0 \omega_i^{\alpha}(t), \tag{2.14}$$

where $\omega_i(t)$ is the weight of node i at time t. Note that if $\alpha = 0$, then the clock intensity is a constant equal to λ_0 across nodes. We will discuss cases $\alpha = 0, 1$ and 2 using simulations in Chapter 5.

Example of one-way path averaging round: If $H = 4$ and if a route with 4 elements is generated, let (s_1, s_2, s_3, s_4) and $(\omega_1, \omega_2, \omega_3, \omega_4)$ be, respectively, the sums and the weights of the 4 nodes in the route. Then, at the end of the rounds, the sums have been updated to

$$\begin{aligned}
s_1 &\leftarrow \frac{1}{4}s_1 \\
s_2 &\leftarrow \frac{1}{4}s_1 + \frac{1}{3}s_2 \\
s_3 &\leftarrow \frac{1}{4}s_1 + \frac{1}{3}s_2 + \frac{1}{2}s_3 \\
s_4 &\leftarrow \frac{1}{4}s_1 + \frac{1}{3}s_2 + \frac{1}{2}s_3 + s_4,
\end{aligned}$$

and the weights have been updated to

$$\begin{aligned}
\omega_1 &\leftarrow \frac{1}{4}\omega_1 \\
\omega_2 &\leftarrow \frac{1}{4}\omega_1 + \frac{1}{3}\omega_2 \\
\omega_3 &\leftarrow \frac{1}{4}\omega_1 + \frac{1}{3}\omega_2 + \frac{1}{2}\omega_3 \\
\omega_4 &\leftarrow \frac{1}{4}\omega_1 + \frac{1}{3}\omega_2 + \frac{1}{2}\omega_3 + \omega_4.
\end{aligned}$$

As a consequence, $\mathbf{x}(k-1) = (\mathbf{x}_1(k-1), \mathbf{x}_2(k-1), \mathbf{x}_3(k-1), \mathbf{x}_4(k-1))$ is updated as $\mathbf{x}(k) = \mathbf{W}(k)\mathbf{x}(k-1)$, where

$$\mathbf{W}(k) = \begin{pmatrix}
1 & 0 & 0 & 0 \\
\frac{\frac{\omega_1}{4}}{\frac{\omega_1}{4}+\frac{\omega_2}{3}} & \frac{\frac{\omega_2}{3}}{\frac{\omega_1}{4}+\frac{\omega_2}{3}} & 0 & 0 \\
\frac{\frac{\omega_1}{4}}{\frac{\omega_1}{4}+\frac{\omega_2}{3}+\frac{\omega_3}{2}} & \frac{\frac{\omega_2}{3}}{\frac{\omega_1}{4}+\frac{\omega_2}{3}+\frac{\omega_3}{2}} & \frac{\frac{\omega_3}{2}}{\frac{\omega_1}{4}+\frac{\omega_2}{3}+\frac{\omega_3}{2}} & 0 \\
\frac{\frac{\omega_1}{4}}{\frac{\omega_1}{4}+\frac{\omega_2}{3}+\frac{\omega_3}{2}+\omega_4} & \frac{\frac{\omega_2}{3}}{\frac{\omega_1}{4}+\frac{\omega_2}{3}+\frac{\omega_3}{2}+\omega_4} & \frac{\frac{\omega_3}{2}}{\frac{\omega_1}{4}+\frac{\omega_2}{3}+\frac{\omega_3}{2}+\omega_4} & \frac{\omega_4}{\frac{\omega_1}{4}+\frac{\omega_2}{3}+\frac{\omega_3}{2}+\omega_4}
\end{pmatrix}. \tag{2.15}$$

If the route has only three nodes because node 3 is a border node, then the

Algorithm 2.12: One-way path averaging

For any node i, let $x_i(0)$ be its measurement. Let t_{end} be a duration time for the algorithm, and let f be a discrete probability distribution on $[2, D]$, where D is the diameter of the network. Similarly to Algorithm 2.9, every node i maintains two variables: a sum $s_i(t)$ and a weight $\omega_i(t)$, which are initialized to $s_i(0) = x_i(0)$ and $\omega_i(0) = 1$. We are going to distinguish here the discrete time slot by k, and continuous time by t. We denote by t' the current time.

1: **while** $t \leqslant t_{end}$ **do**
2: Node a's clock ticks at time t, so that node a wakes up.
3: Node a draws a target location T at random *outside* the unit square using Alg. 2.10, as well as a random number H of hops using the discrete probability distribution f.
4: Node a initializes the counter to $C \leftarrow H - 1$. Let $i \leftarrow a$.
5: Node i uses `find neighbor` with target T, and
6: **if** `find neighbor` outputs a node j **then**
7: Node i sends message $m = (s_i(t')*C/(C+1), \omega_i(t')*C/(C+1), \ell_i, T, C)$, where ℓ_i is the location of node i. Let t be the time at which the message is sent ($t \leftarrow t'$).
8: $s_i(t) \leftarrow s_i(t) - s_i(t) * C/(C+1) = s_i(t)/(C+1)$.
9: $\omega_i(t) \leftarrow \omega_i(t) - \omega_i(t) * C/(C+1) = \omega_i(t)/(C+1)$.
10: Node j receives message m.
11: $s_j(t') \leftarrow s_j(t') + s_i(t) * C/(C+1)$.
12: $\omega_j(t') \leftarrow \omega_j(t') + \omega_i(t) * C/(C+1)$.
13: Node j updates the counter to $C \leftarrow C - 1$.
14: **if** $C > 0$ **then**
15: Repeat step 5 with $i \leftarrow j$.
16: **else**
17: The averaging round is terminated.
18: **end if**
19: **else**
20: Node i considers itself as the last node of the route and the kth averaging round is terminated.
21: **end if**
22: At any time t, the estimate of node i is

$$x_i(t) = \frac{s_i(t)}{\omega_i(t)}.$$

23: **end while**

sums have been updated to

$$\begin{aligned} s_1 &\leftarrow \frac{1}{4}s_1 \\ s_2 &\leftarrow \frac{1}{4}s_1 + \frac{1}{3}s_2 \\ s_3 &\leftarrow \frac{1}{2}s_1 + \frac{2}{3}s_2 + s_3, \end{aligned}$$

and the weights have been updated to

$$\begin{aligned} \omega_1 &\leftarrow \frac{1}{4}\omega_1 \\ \omega_2 &\leftarrow \frac{1}{4}\omega_1 + \frac{1}{3}\omega_2 \\ \omega_3 &\leftarrow \frac{1}{2}\omega_1 + \frac{2}{3}\omega_2 + \omega_3. \end{aligned}$$

In this case, the update matrix $\mathbf{W}(k)$ is

$$\mathbf{W}(k) = \begin{pmatrix} 1 & 0 & 0 \\ \frac{\frac{\omega_1}{4}}{\frac{\omega_1}{4}+\frac{\omega_2}{3}} & \frac{\frac{\omega_2}{3}}{\frac{\omega_1}{4}+\frac{\omega_2}{3}} & 0 \\ \frac{\frac{\omega_1}{4}}{\frac{\omega_1}{4}+\frac{\omega_2}{3}+\omega_3} & \frac{\frac{\omega_2}{3}}{\frac{\omega_1}{4}+\frac{\omega_2}{3}+\omega_3} & \frac{\omega_3}{\frac{\omega_1}{4}+\frac{\omega_2}{3}+\omega_3} \end{pmatrix}. \tag{2.16}$$

Comments:

- We expect that the evolved one-way path averaging algorithm converges faster than the naive one.
- Suppose that in every round a number of routes are initiated and terminated, then at the end of these rounds the estimates have been linearly updated. In other words, for each round k, there is an averaging matrix $\mathbf{W}(k)$ such that $\mathbf{x}(k) = \mathbf{W}(k)\mathbf{x}(k-1)$. It is easy to see that $\mathbf{W}(k)$ verifies $\mathbf{W}(k)\mathbf{1} = \mathbf{1}$, but that $\mathbf{1}^T\mathbf{W}(k) \neq \mathbf{1}^T$. However, since nodes keep the weights ω in memory, if the algorithm converges to a consensus value c, then c is equal to x_{ave}, as we will show in Section 5.5.

Conclusion

In this chapter we listed a number of algorithms in order of increasing performance, but decreasing practicality. Specifically, geographic gossip and path averaging may lack robustness in time-varying networks. We will see in Chapter 5 that one-way path averaging, the final algorithm we present, outperforms all previous algorithms on simulations, while at the same time being robust in time-varying networks. We do not study its performance analytically, but we believe that one-way path averaging is a very promising algorithm.

We end this chapter with an important remark. Each of the algorithms we described in this chapter uses a parameter t_{end}, the time at which the algorithm terminates. This parameter needs to be either fixed in advance, or

imposed in real time by a simultaneous parallel algorithm that detects the level of consensus of the network. In the latter case, there is no issue of algorithm convergence, but such an algorithm is not always available. If t_{end} is to be chosen before running the algorithm, then guarantees are needed that the algorithm will reach a desired level of convergence before it terminates. In particular, it is crucial to have accurate estimates of the convergence speed of the algorithms. In the following chapter we state mathematical results on the convergence of distributed averaging algorithms and on eigenvalues, before we begin to study the performance of individual algorithms in Chapter 5.

SUMMARY

A distributed averaging algorithm is linear: for every iteration t, there is an averaging matrix $\mathbf{W}(t)$ such that:

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

Classification of the averaging algorithms studied in this thesis:

- Synchronous algorithms
  - *Uniform weights*: $w_{ij} = \alpha$.
  - *Metropolis weights*: $w_{ij} = 1/(1 + \max\{d_i, d_j\})$
  - *Adaptative weights*: Let $\mathcal{G}$ be a fixed graph, and let $\mathbf{W}^{\mathcal{G}}$ be an averaging matrix on $\mathcal{G}$. Then $w_{ij}(t) = \mathbf{W}^{\mathcal{G}}_{ij}$ if $i \sim j$ at time t.
- Asynchronous algorithms
  - *Pairwise gossip*: Two neighboring nodes i and j communicate at time t, and update their estimates to their pairwise average.
  - *Neighborhood gossip*: A neighborhood $\{i\} \cup \mathcal{N}_i$ communicate at time t, and update their estimates to the neighborhood average.
  - *Geographic gossip*: Any pair of nodes i and j communicate at time t by routing their estimates to each other. They update their estimates to their pairwise average.
  - *Path averaging*: A random path of nodes is generated at time t. All the nodes on the route update their estimates to the average of the route estimates.
  - *Broadcast gossip*: At time t, one node i broadcasts its estimate. Any node $j \in \mathcal{N}_i$ updates its estimate to $\gamma x_j(t) + (1-\gamma)x_i(t)$.
  - *One-way path averaging*: Variant of the Push-Sum protocol that allows weighted averages on random paths without any routing back.

Gossip algorithms average the estimates of a random group of nodes at each iteration. The following properties hold for any time t:

$$\begin{aligned} \mathbf{W}(t)^2 &= \mathbf{W}(t), \\ \mathbf{W}(t)^T\mathbf{W}(t) &= \mathbf{W}(t). \end{aligned}$$

Chapter 3

Convergence of Distributed Average Consensus Algorithms

This chapter describes the tools needed to show the convergence of a distributed average consensus algorithm. It also states theorems to evaluate their speed of convergence. After each theorem statement, there is a list of algorithms it can be applied to. However, the complete analysis of the algorithms described in previous chapter is deferred to Chapter 5.

3.1 Convergence of the Estimates

Let $\mathbf{x}(0)$ be the vector of measurements, x_{ave} be the average of $\mathbf{x}(0)$, and $\mathbf{x}(t)$ be the vector of estimates at time t. The vector of all ones is denoted by $\mathbf{1}$. The convergence question in distributed averaging algorithms is twofold:

- Question 1: Do the estimates converge to a *consensus*? In other words, is there a scalar c such that

$$\lim_{t\to\infty} \mathbf{x}(t) = c\mathbf{1} \ ? \tag{3.1}$$

- Question 2: If there is such a consensus scalar c, is it equal to the average x_{ave}?

  Note that, if c exists, and if, for any time t, $\mathbf{1}^T\mathbf{W}(t) = \mathbf{1}^T$, then necessarily $c = x_{ave}$. If $\mathbf{1}^T\mathbf{W}(t)$ is not automatically equal to $\mathbf{1}^T$, then Question 2 should be restated as "how is c distributed?". In particular, "is $\mathbb{E}[c] = x_{ave}$?", and "how large is the variance of c?".

Let $\mathbf{J}$ be the matrix $\mathbf{J} = \mathbf{1}\mathbf{1}^T/n$. Then, the distance to the *current* average, which can also be interpreted as the distance to consensus, is measured with

$$\boldsymbol{\eta}(t) = \mathbf{x}(t) - \mathbf{J}\mathbf{x}(t) = (\mathbf{I} - \mathbf{J})\mathbf{x}(t). \tag{3.2}$$

The error $\boldsymbol{\epsilon}$ is defined as

$$\boldsymbol{\epsilon}(t) = \mathbf{x}(t) - x_{ave}\mathbf{1} = \mathbf{x}(t) - \mathbf{J}\mathbf{x}(0). \tag{3.3}$$

$\boldsymbol{\epsilon}(t)$ is useful to measure the distance to the *initial* average.

The topic of this section is to study the convergence of $\boldsymbol{\eta}(t)$ and $\boldsymbol{\epsilon}(t)$, when time t goes to infinity. Since the network can be randomly time-varying and since most algorithms described in Section 2.2 are randomized, $\boldsymbol{\eta}(t)$ and $\boldsymbol{\epsilon}(t)$ are in most cases random variables. Three types of convergence are addressed here: convergence with probability 1, convergence in expectation, and mean-square convergence.

3.1.1 Deterministic average consensus algorithms

It is convenient to start with the deterministic case, where the network is fixed and does not suffer from link failures, and where the algorithm is linear but not randomized. In this context, there is a matrix $\mathbf{W}$ such that:

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{W}\mathbf{x}(t-1) \\ &= \mathbf{W}^t\mathbf{x}(0). \end{aligned}$$

Since $\mathbf{x}_{ave} = \mathbf{1}^T\mathbf{x}(0)/n$,

$$\boldsymbol{\epsilon}(t) = \mathbf{x}(t) - \mathbf{1}\mathbf{1}^T/n \cdot \mathbf{x}(0). \tag{3.4}$$

Then, in the deterministic case,

$$\boldsymbol{\epsilon}(t) = (\mathbf{W}^t - \mathbf{J})\mathbf{x}(0). \tag{3.5}$$

Therefore, $\lim_{t\to\infty} \boldsymbol{\epsilon}(t) = 0$ for any $\mathbf{x}(0) \in \mathbb{R}^n$ if and only if:

$$\lim_{t\to\infty} \mathbf{W}^t = \mathbf{J}. \tag{3.6}$$

In [78], Xiao et al. show the following theorem:

Theorem 3.1.1. $\lim_{t\to\infty} \mathbf{W}^t = \mathbf{J}$ if and only if

$$\mathbf{W}\mathbf{1} = \mathbf{1}, \tag{3.7}$$

$$\mathbf{1}^T\mathbf{W} = \mathbf{1}^T, \tag{3.8}$$

$$\rho\left(\mathbf{W} - \mathbf{J}\right) < 1, \tag{3.9}$$

where ρ denotes the spectral radius of a matrix, the largest eigenvalue in magnitude.

Comments: This theorem can be applied to synchronous averaging algorithms when the network does not vary in time. Conditions 3.7, 3.8 and 3.9 have simple and intuitive interpretations:

1. Condition 3.7: $\mathbf{1}$ is a fixed point of the algorithm, i.e. *consensus* is stable.

2. Condition 3.8: The average of the estimates is conserved at each step.

3. Condition 3.9: Components of the estimates that are not parallel to $\mathbf{1}$ vanish.

Proof: The following proof is taken from [78].

($\Leftarrow$): If $\mathbf{W}$ satisfies Conditions (3.7) and (3.8), then $\mathbf{W}$ satifies a number of convenient formulas, which will be useful in this thesis:

$$\begin{aligned} \mathbf{WJ} &= \mathbf{J}, \\ \mathbf{JW} &= \mathbf{J}, \end{aligned}$$

which implies that $\Rightarrow$ $\mathbf{WJ} = \mathbf{JW}$, i.e. that $\mathbf{W}$ and $\mathbf{J}$ commute. It is also convenient to notice that

$$\begin{aligned} \mathbf{J}^2 &= \mathbf{J} \\ (\mathbf{I}-\mathbf{J})^2 &= (\mathbf{I}-\mathbf{J}), \end{aligned}$$

and that $\mathbf{W}$ also commutes with $(\mathbf{I}-\mathbf{J})$. Gathering this information together, leads to the following calculation:

$$\begin{aligned} \mathbf{W}^t - \mathbf{J} &= \mathbf{W}^t(\mathbf{I}-\mathbf{J}) \\ &= \mathbf{W}^t(\mathbf{I}-\mathbf{J})^t \\ &= (\mathbf{W}(\mathbf{I}-\mathbf{J}))^t \\ &= (\mathbf{W}-\mathbf{J})^t. \end{aligned}$$

But it is well-known that for any real or complex square matrix $\mathbf{A}$, $\lim_{t\to\infty} \mathbf{A^t} = 0$ if and only if $\rho(\mathbf{A}) < 1$ (see [45], page 298). Therefore Condition (3.9) implies that $\lim_{t\to\infty}(\mathbf{W}^t - \mathbf{J}) = 0$, which finishes the sufficiency proof.

($\Rightarrow$): To prove necessity, it is useful to know [54] that $\lim_{t\to\infty} \mathbf{W}^t$ exists if and only if there is a nonsingular matrix T such that

$$\mathbf{W} = T \begin{bmatrix} \mathbf{I}_\kappa & 0 \\ 0 & Z \end{bmatrix} T^{-1}, \tag{3.10}$$

where $\mathbf{I}_\kappa$ is the κ-dimensional identity matrix ($0 \leqslant \kappa \leqslant n$) and Z is a matrix such that $\rho(Z) < 1$. Let $u_1, \ldots, u_n$ be the columns of T and $v_1^T, \ldots, v_n^T$ be the rows of T^{-1}. Then,

$$\begin{aligned} \lim_{t\to\infty} \mathbf{W}^t &= \lim_{t\to\infty} T \begin{bmatrix} \mathbf{I}_\kappa & 0 \\ 0 & Z^t \end{bmatrix} T^{-1} && (3.11) \\ &= T \begin{bmatrix} \mathbf{I}_\kappa & 0 \\ 0 & 0 \end{bmatrix} T^{-1} && (3.12) \\ &= \sum_{i=1}^{\kappa} u_i v_i^T. && (3.13) \end{aligned}$$

Because each $u_i v_i^T$ is a rank-one matrix and their sum $\sum_{i=1}^n u_i v_i^T = TT^{-1} = \mathbf{I}$ has rank n, the matrix $\sum_{i=1}^\kappa u_i v_i^T$ must have rank κ. But $\sum_{i=1}^\kappa u_i v_i^T = \mathbf{J}$ has rank 1, thus $\kappa = 1$ and $u_1 v_1^T = \mathbf{J}$, which implies that both u_i and v_i are multiples of $\mathbf{1}$. In other words, 1 is a non-multiple eigenvalue of $\mathbf{W}$ and $\mathbf{1}^T$ and $\mathbf{1}$ are, respectively, its associated left and right eigenvectors. Hence

Conditions (3.7) and (3.8) hold. Moreover, since $\mathbf{J} = T \begin{bmatrix} \mathbf{I}_1 & 0 \\ 0 & 0 \end{bmatrix} T^{-1}$,

$$\begin{aligned} \rho\left(\mathbf{W} - \mathbf{J}\right) &= \rho\left(T \begin{bmatrix} 0 & 0 \\ 0 & Z \end{bmatrix} T^{-1}\right) && (3.14) \\ &= \rho(Z) < 1, && (3.15) \end{aligned}$$

which is precisely Condition 3.9. ■

Note that this theorem is easily generalizable to:

Theorem 3.1.2. $\lim_{t\to\infty} \mathbf{W}^t = \mathbf{1}\mathbf{v}^T$ if and only if

$$\begin{aligned} \mathbf{W}\mathbf{1} &= \mathbf{1}, && (3.16) \\ \mathbf{v}^T\mathbf{W} &= \mathbf{v}^T, && (3.17) \\ \mathbf{v}^T\mathbf{1} &= 1, && (3.18) \\ \rho\left(\mathbf{W} - \mathbf{1}\mathbf{v}^T\right) &< 1, && (3.19) \end{aligned}$$

where ρ denotes the spectral radius of a matrix.

If $\mathbf{W}$ is non-negative, then $\mathbf{v}$ can be chosen positive by the Perron-Frobenius theorem [16]. As a consequence, the only linear consensus that are reachable in the deterministic case with non-negative $\mathbf{W}$ are barycentric: the consensus scalar c is equal to $\mathbf{v}^T\mathbf{x}(0)$, where the sum of the coefficients in $\mathbf{v}$ is equal to 1.

3.1.2 Convergence to consensus with probability 1

The attention is back to networks that can be time-varying and to algorithms that can be asynchronous. Assume that at each step t, there is a matrix $\mathbf{W}(t)$ such that $\mathbf{x}(t) = \mathbf{W}(t)\mathbf{x}(t-1)$ and let T_ε be the following stopping time with respect to the random process $\{\mathbf{W}(t)\}_{t\geqslant 1}$:

$$T_\epsilon := \inf_t \left\{ t \geqslant 1 : \prod_{p=1}^{t} \mathbf{W}(t-p) \geqslant \varepsilon > 0 \right\}, \tag{3.20}$$

where ε is an arbitrarily small positive number. The notation $\mathbf{A} \geqslant \varepsilon$ means that all the elements of the matrix $\mathbf{A}$ are larger than or equal to ε. Sufficient conditions for the agents to converge to a consensus c are stated in the following theorem[1]:

Theorem 3.1.3 (Sufficient conditions for almost sure consensus). With probability 1 there is a scalar c such that the sequence

$$\mathbf{x}(t) = \prod_{p=0}^{t-1} \mathbf{W}(t-p)\mathbf{x}(0), \quad t \geq 1$$

converges to $c\mathbf{1}$ for large t, if the following conditions hold:

1. The sequence $\{\mathbf{W}(t)\}_{t\geqslant 1}$ is stationary.
2. $\mathbf{W}(t)\mathbf{1} = \mathbf{1}$ for all times t. (fixed point)

[1] This theorem was proven by Patrick Denantes in his Master thesis.

3. $|||\mathbf{W}(t)|||_2 \leq 1$ for all times t. (contracting property)
4. $\forall \varepsilon > 0$, $\mathbb{E}[T_\varepsilon] < \infty$.

Comments:

- This theorem applies to all the algorithms described in Chapter 2, except for one-way path averaging.
- If furthermore $\mathbf{1}^T\mathbf{W}(t) = \mathbf{1}^T$ for all times t, then necessarily $c = \mathbf{1}^T\mathbf{x}(0)/n = x_{ave}$.
- $\forall \varepsilon > 0$, $\mathbb{E}[T_\varepsilon] < \infty$ is a connectivity condition. It implies that the network should be jointly connected in time.

Proof: The first step of this proof is to show that the maximum difference between elements of $\mathbf{x}(T_\varepsilon)$ is strictly smaller than the maximum difference between elements of $\mathbf{x}(0)$. To simplify the notation, define $\mathbf{P} := \prod_{p=0}^{T_\varepsilon - 1} \mathbf{W}(T_\varepsilon - p)$ and denote by p_{ij} the ith row, j^{th} column element of this matrix. Using Condition 2, $\mathbf{P1} = \mathbf{1}$, thus $\sum_j p_{ij} = 1$, for all $i = 1, \ldots, n$. Therefore each element of $\mathbf{x}(T_\varepsilon)$ is a barycentric weighted sum of elements of $\mathbf{x}(0)$, whose weights are all larger than ε by definition of T_ε:

$$\begin{aligned} \max_i x_i(T_\varepsilon) &= \max_i \sum_j p_{ij} x_j(0) \\ &\leq (1-\varepsilon) \max_j x_j(0) + \varepsilon \min_j x_j(0), \end{aligned}$$

since in the worst (max) case, the largest element of $\mathbf{x}(0)$ gets the largest weight, and the smallest element gets the smallest weight in the barycentric weighted sum. Similarly, one can bound the minimum element from below:

$$\min_i x_i(T_\varepsilon) \geq \varepsilon \max_j x_j(0) + (1-\varepsilon) \min_j x_j(0).$$

Taking the difference of the two yields:

$$\max_i x_i(T_\varepsilon) - \min_i x_i(T_\varepsilon) \leq (1-2\varepsilon) \left(\max_j x_j(0) - \min_j x_j(0) \right). \tag{3.21}$$

Now define an increasing sequence of integers $\{k^{(\ell)}\}_{\ell \geq 0}$ as follows:

$$k^{(0)} = 0, k^{(1)} = T_\varepsilon, \ldots, k^{(\ell+1)} = \inf\{t \geq k^{(\ell)} + 1 : \prod_{p=1}^{t-k^{(\ell)}} \mathbf{W}(t-p) \geq \varepsilon\}, \tag{3.22}$$

and an associated counting process

$$N(t) := \max \ell \quad \text{ such that } \quad k^{(\ell)} \leq t, \quad \forall t \geq 0.$$

The matrices $\mathbf{W}(t)$ are grouped in such a way to form a sequence of positive matrices. The counter $N(t)$ denotes the number of such groups up to time t.

As the process $\{\mathbf{W}(t)\}_{t \geqslant 1}$ is stationary (Condition 1), the time intervals $k^{(\ell+1)} - k^{(\ell)}$ are i.i.d. for all $\ell \geq 0$. The process $\{N(t)\}_{t \geq 0}$ is therefore a renewal process [16] with inter-renewal time T_ε.

By the strong law for renewal processes [16],

$$\lim_{t\to\infty} \frac{N(t)}{t} = \frac{1}{\mathbb{E}[T_\varepsilon]} \quad \text{with probability 1.} \tag{3.23}$$

Now iterate Eq. (3.21) ℓ times:

$$\max_i x_i(k^{(\ell)}) - \min_i x_i(k^{(\ell)}) \leq (1-2\varepsilon)^\ell \left(\max_j x_j(0) - \min_j x_j(0) \right), \tag{3.24}$$

and consider the sequence $\{\hat{\mathbf{x}}(t)\}_{t\geq 0}$ obtained by sampling the original process at the renewal times:

$$\hat{\mathbf{x}}(t) := \mathbf{x}\left(k^{(N(t))}\right).$$

Eq. (3.24) can thus be rewritten in terms of t:

$$\max_i \hat{x}_i(t) - \min_i \hat{x}_i(t) \leq (1-2\varepsilon)^{N(t)} \left(\max_j \hat{x}_j(0) - \min_j \hat{x}_j(0) \right). \tag{3.25}$$

Because of Condition 4 and of Eq. (3.23), the right hand side of Eq. (3.25) tends to 0 for large t, hence the left hand side of Eq. (3.25) tends to 0 for large t as well. Therefore $\hat{\boldsymbol{\eta}}(t) = \hat{\mathbf{x}}(t) - \mathbf{J}\hat{\mathbf{x}}(t)$ verifies

$$\begin{aligned} \|\widehat{\boldsymbol{\eta}}(t)\|_\infty &\leq \max_i \hat{x}_i(t) - \min_i \hat{x}_i(t) \\ &\leq (1-2\varepsilon)^{N(t)} \left(\max_j \hat{x}_j(0) - \min_j \hat{x}_j(0) \right). \end{aligned}$$

Hence, $\lim_{t\to\infty} \|\widehat{\boldsymbol{\eta}}\|_\infty = 0$. Furthermore, $\{\max_i \hat{x}_i(t)\}$ is a bounded decreasing sequence and $\{\min_i \hat{x}_i(t)\}$ is a bounded increasing sequence. Let $c_{\max}$ and $c_{\min}$ be their respective limits. Then, for any t,

$$\begin{aligned} \|\widehat{\boldsymbol{\eta}}(t)\|_\infty &\geq \max\left(\left| \max_i \hat{x}_i(t) - \frac{\mathbf{1}^T\hat{\mathbf{x}}(t)}{n} \right|, \left| \min_i \hat{x}_i(t) - \frac{\mathbf{1}^T\hat{\mathbf{x}}(t)}{n} \right| \right) \\ &\geq \frac{\max_i \hat{x}_i(t) - \min_i \hat{x}_i(t)}{2} \\ &\geq \frac{c_{\max} - c_{\min}}{2}. \end{aligned}$$

Since $\lim_{t\to\infty} \|\widehat{\boldsymbol{\eta}}\|_\infty = 0$, $c_{\min} = c_{\max} = c$, and $\lim_{t\to\infty} \hat{\mathbf{x}}(t) = c\mathbf{1}$ with probability 1.

It now remains to show that convergence of $\{\hat{\mathbf{x}}(t)\}_{t\geq 0}$ implies convergence of $\{\mathbf{x}(t)\}_{t\geq 0}$. Define $\mathbf{y}(t) = \mathbf{x}(t) - c\mathbf{1}$ and $\hat{\mathbf{y}}(t) = \hat{\mathbf{x}}(t) - c\mathbf{1}$. Then

$$\begin{aligned} \mathbf{y}(t) &= \mathbf{W}(t)\mathbf{x}(t-1) - c\mathbf{1} && (3.26) \\ &= \mathbf{W}(t)(\mathbf{x}(t-1) - c\mathbf{1}) && (3.27) \\ &= \mathbf{W}(t)\mathbf{y}(t-1). && (3.28) \end{aligned}$$

Therefore, by Condition 3,

$$\begin{aligned}
\|\mathbf{y}(t)\|_2 &= \left\| \prod_{p=0}^{t-k^{(N(t))}+1} \mathbf{W}(t-p)\mathbf{y}(N(t)) \right\|_2 && (3.29)\\
&\leq \left\| \left\| \prod_{p=0}^{t-k^{(N(t))}+1} \mathbf{W}(t-p) \right\| \right\|_2 \left\| \mathbf{y}(k^{(N(t))}) \right\|_2 && (3.30)\\
&\leq \prod_{p=0}^{t-k^{(N(t))}+1} |||\mathbf{W}(t-p)|||_2 \left\| \mathbf{y}(k^{(N(t))}) \right\|_2 && (3.31)\\
&\leq \left\| \mathbf{y}(k^{(N(t))}) \right\|_2 && (3.32)\\
&= \|\hat{\mathbf{y}}(t)\|_2 . && (3.33)
\end{aligned}$$

Since $\hat{\mathbf{y}}(t)$ converges to 0 for large t, $\mathbf{y}(t)$ converges to 0 for large t as well, and

$$\lim_{t\to\infty} \mathbf{x}(t) = c\mathbf{1} \quad \text{with probability 1.}$$

■

The previous theorem and its proof is interesting in the sense that it describes well how the error converges to 0. However, Theorem 3.1.3 being very general, it is not easy to apply on specific algorithms. In particular, choosing the right ε in Condition 4 can be difficult. Using the same idea of contraction of $\max x_i - \min x_i$, a similar almost-sure convergence theorem was stated in [13], which formulation is more specific and more practical than ours. In a few words, Condition 4 is replaced by bounds on the matrices $\mathbf{W}(t)$ directly: $\mathbf{W}_{ii}(t)$ should be larger than a constant α, and whenever $\mathbf{W}_{ij}(t)$ is non-zero, it should be larger than α as well. In addition, the authors state a connectivity condition on the sequence of communications: for any t, the set of communicating edges after t should form a connected graph. Our stationarity condition is replaced by a bound on the intercommunication intervals: any infinitely appearing communication edge should appear at least once in any B consecutive iterations, for some B. Finally, they require the stochasticity of $\mathbf{W}(t)$. Refer to [13] for the detailed conditions and to [75] for the proof.

There is another technique to show that the error converges to 0 with probability 1. It is also more convenient than Th. 3.1.3, but its range of applications is not as broad. It was used in [7] and it is based on martingale theory:

Theorem 3.1.4. Consider a sequence of nonnegative random variables $\{X(t)\}_{t\geqslant 0}$. If $\mathbb{E}[X(0)] < \infty$ and if, for any time t,

$$\mathbb{E}[X(t+1)|X(t),\ldots,X(1),X(0)] \leqslant \lambda X(t),$$

where $\lambda \in (0,1)$, then $X(t)$ converges to 0 almost surely:

$$\mathbb{P}\left[\lim_{t\to\infty} X(t) = 0\right] = 1.$$

Applying this theorem with $X(t) = \|\boldsymbol{\epsilon}(t)\|_2^2$ in the case of i.i.d. matrices $\{\mathbf{W}(t)\}_{t\geqslant 1}$, the sufficient conditions for mean square convergence, described

later in Section 3.1.4, are also sufficient conditions for convergence with probability 1 (see Eq. (3.47) and (3.48), and take $\lambda = \lambda_2(\mathbb{E}[\mathbf{W}^T\mathbf{W}])$).

3.1.3 Convergence in expectation

We are now going to consider the convergence of the expected error. Recall that the error is defined as $\boldsymbol{\epsilon}(t) = \mathbf{x}(t) - x_{ave}\mathbf{1}$.

Definition 3.1.1 (Convergence in expectation). The algorithm converges in expectation if and only if $\lim_{t\to\infty} \mathbb{E}[\boldsymbol{\epsilon}(t)] = 0$.

Theorem 3.1.5. Assume that the four conditions of Theorem 3.1.3 hold, so that $\lim_{t\to\infty} \mathbf{x}(t) = c\mathbf{1}$ with probability 1. Then,

$$\mathbb{E}[c\mathbf{1}] = \lim_{t\to\infty} \mathbb{E}[\mathbf{x}(t)].$$

Proof: This is a direct application of Lebesgue theorem. ■

As a consequence, a distributed averaging algorithm following the four conditions of Theorem 3.1.3 produces a non-biased estimation of the average x_{ave} ($\mathbb{E}[c] = x_{ave}$) if and only if it converges in expectation ($\lim_{t\to\infty} \mathbb{E}[\boldsymbol{\epsilon}(t)] = 0$). This theorem is thus particularly interesting for the analysis of broadcast gossip, which does not conserve the average through iterations.

Theorem 3.1.6 (Convergence in expectation). Consider a distributed average consensus algorithm, in which, at any step t, there is a matrix $\mathbf{W}(t)$ such that $\mathbf{x}(t) = \mathbf{W}(t)\mathbf{x}(t-1)$. Assume that the sequence $\{\mathbf{W}(t)\}_{t\geqslant 1}$ is independently and identically distributed. Let $\mathbb{E}[\mathbf{W}]$ be the expectation of matrix $\mathbf{W}(t)$ at any time t. Then, the algorithm converges in expectation if and only if

$$\mathbb{E}[\mathbf{W}]\mathbf{1} = \mathbf{1}, \tag{3.34}$$
$$\mathbf{1}^T\mathbb{E}[\mathbf{W}] = \mathbf{1}^T, \tag{3.35}$$
$$\rho\left(\mathbb{E}[\mathbf{W}] - \mathbf{J}\right) < 1, \tag{3.36}$$

Proof: For any given initial measurements $\mathbf{x}(0)$,

$$\mathbb{E}[\boldsymbol{\epsilon}(t)] = \mathbb{E}\left[\mathbf{x}(t) - \mathbf{J}\mathbf{x}(0)\right] \tag{3.37}$$
$$= \mathbb{E}\left[\mathbf{x}(t)\right] - \mathbf{J}\mathbf{x}(0) \tag{3.38}$$
$$= \mathbb{E}\left[\mathbf{W}(t)\mathbf{x}(t-1)\right] - \mathbf{J}\mathbf{x}(0) \tag{3.39}$$
$$= \mathbb{E}\left[\mathbf{W}(t)\right]\mathbb{E}\left[\mathbf{x}(t-1)\right] - \mathbf{J}\mathbf{x}(0), \tag{3.40}$$

where the last statement holds because of the independence assumption. Using now the fact that $\{\mathbf{W}(t)\}_{t\geqslant 1}$ is identically distributed,

$$\mathbb{E}[\boldsymbol{\epsilon}(t)] = \mathbb{E}\left[\mathbf{W}\right]\mathbb{E}\left[\mathbf{x}(t-1)\right] - \mathbf{J}\mathbf{x}(0) \tag{3.41}$$
$$= \left(\mathbb{E}\left[\mathbf{W}\right]\right)^t\mathbf{x}(0) - \mathbf{J}\mathbf{x}(0) \tag{3.42}$$
$$= \left(\left(\mathbb{E}\left[\mathbf{W}\right]\right)^t - \mathbf{J}\right)\mathbf{x}(0). \tag{3.43}$$

Therefore $\lim_{t\to\infty} \mathbb{E}[\boldsymbol{\epsilon}(t)] = 0$ for any $\mathbf{x}(0)$ if and only if $\lim_{t\to\infty} \left(\mathbb{E}\left[\mathbf{W}\right]\right)^t = \mathbf{J}$. Applying Theorem 3.1.1 with $\mathbb{E}[\mathbf{W}]$ proves the theorem. ■

Comments: When the network sequence $\{\mathcal{G}(t)\}_{t\geqslant 0}$ is constant or i.i.d. time-varying, this theorem applies to synchronized average consensus algorithms, to all gossip algorithms, and to broadcast gossip.

If the sequence $\{\mathbf{W}(t)\}_{t\geqslant 1}$ is independently and identically distributed, and if Conditions (3.34), (3.35) and (3.36) are fulfilled, then $\mathbb{E}[\boldsymbol{\epsilon}(t)] = (\mathbb{E}[\mathbf{W}] - \mathbf{J})^t \mathbf{x}(0)$, and for any vector norm $||\cdot||$ and its associated matrix norm $|||\cdot|||$,

$$\frac{||\mathbb{E}[\boldsymbol{\epsilon}(t)]||}{||\mathbf{x}(0)||} \leqslant ||| (\mathbb{E}[\mathbf{W}] - \mathbf{J})^t |||.$$

Therefore,

$$\begin{aligned}\lim_{t\to\infty} \sup \left(\frac{||\mathbb{E}[\boldsymbol{\epsilon}(t)]||}{||\mathbf{x}(0)||}\right)^{\frac{1}{t}} &\leqslant \lim_{t\to\infty} \left(||| (\mathbb{E}[\mathbf{W}] - \mathbf{J})^t |||\right)^{\frac{1}{t}} \\ &= \rho(\mathbb{E}[\mathbf{W}] - \mathbf{J}),\end{aligned}$$

where the last equality comes from the fact that $\rho(A) = \lim_{t\to\infty} |||A^t|||^{1/t}$ for any square matrix A and any matrix norm $|||\cdot|||$ ([45], page 299). Therefore the spectral radius $\rho(\mathbb{E}[\mathbf{W}] - \mathbf{J})$ arises for the convergence of the *norm of the expectation* of the error, which is not useful to characterize the speed of convergence of the algorithm. A relevant tool to describe convergence is the *expectation of the norm* of the error, which, as the next section shows, involves a more complicated spectral radius.

3.1.4 Mean square convergence

This section studies the convergence of the *norm* of the error.

Definition 3.1.2 (Mean square convergence). The algorithm converges in mean square if and only if $\lim_{t\to\infty} \mathbb{E}[\|\boldsymbol{\epsilon}(t)\|_2^2] = 0$.

The following theorem is due to Boyd et al. in [15].

Theorem 3.1.7 (Sufficient conditions for mean square convergence). Consider a distributed average consensus algorithm, in which, at any step t, there is a matrix $\mathbf{W}(t)$ such that $\mathbf{x}(t) = \mathbf{W}(t)\mathbf{x}(t-1)$. Assume that the sequence $\{\mathbf{W}(t)\}_{t\geqslant 1}$ is independently and identically distributed. Let $\mathbb{E}[\mathbf{W}^T\mathbf{W}]$ be the expectation of matrix $\mathbf{W}(t)^T\mathbf{W}(t)$ at any time t. Then, the algorithm converges in mean square if

$$\mathbf{W}(t)\mathbf{1} = \mathbf{1} \quad \text{for any time } t, \tag{3.44}$$

$$\mathbf{1}^T\mathbf{W}(t) = \mathbf{1}^T \quad \text{for any time } t, \tag{3.45}$$

$$\lambda_2\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right) < 1, \tag{3.46}$$

where λ_2 denotes the second largest eigenvalue in magnitude (SLEM).

Proof: Assume that Condition (3.44) is fulfilled, then

$$\begin{aligned}\boldsymbol{\epsilon}(t) &= \mathbf{x}(t) - x_{ave}\mathbf{1} \\ &= \mathbf{W}(t)\mathbf{x}(t-1) - x_{ave}\mathbf{W}(t)\mathbf{1} \\ &= \mathbf{W}(t)\boldsymbol{\epsilon}(t-1).\end{aligned}$$

Then, for any given initial measurements $\mathbf{x}(0)$,

$$\begin{aligned}
\mathbb{E}[\|\boldsymbol{\epsilon}(t)\|_2^2 \,|\boldsymbol{\epsilon}(t-1)] &= \mathbb{E}[\boldsymbol{\epsilon}(t)^T\boldsymbol{\epsilon}(t)|\boldsymbol{\epsilon}(t-1)] \\
&= \mathbb{E}[\boldsymbol{\epsilon}(t-1)^T\mathbf{W}(t)^T\mathbf{W}(t)\boldsymbol{\epsilon}(t-1)|\boldsymbol{\epsilon}(t-1)] \\
&= \boldsymbol{\epsilon}(t-1)^T\mathbb{E}[\mathbf{W}(t)^T\mathbf{W}(t)]\boldsymbol{\epsilon}(t-1) \\
&= \boldsymbol{\epsilon}(t-1)^T\mathbb{E}[\mathbf{W}^T\mathbf{W}]\boldsymbol{\epsilon}(t-1). \qquad (3.47)
\end{aligned}$$

For any time t, $\mathbf{W}(t)^T\mathbf{W}(t)$ is a positive definite symmetric matrix, therefore $\mathbb{E}[\mathbf{W}^T\mathbf{W}]$ is also a positive definite symmetric matrix and all its eigenvalues are positive. Assuming Conditions (3.44) and (3.45), $\mathbf{1}$ is a right eigenvector of $\mathbb{E}[\mathbf{W}^T\mathbf{W}]$ associated with the eigenvalue 1. Condition (3.46) thus implies that $\mathbf{1}$ is the largest eigenvalue in magnitude of $\mathbb{E}[\mathbf{W}^T\mathbf{W}]$.

Since $\boldsymbol{\epsilon}(t)$ is orthogonal to $\mathbf{1}$ (Condition (3.45)), we have that

$$\begin{aligned}
\mathbb{E}[\|\boldsymbol{\epsilon}(t)\|_2^2 \,|\boldsymbol{\epsilon}(t-1)] &= \boldsymbol{\epsilon}(t-1)^T\mathbb{E}[\mathbf{W}^T\mathbf{W}]\boldsymbol{\epsilon}(t-1) \\
&\leq \lambda_2\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right)\|\boldsymbol{\epsilon}(t-1)\|_2^2\,. \qquad (3.48)
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathbb{E}[\|\boldsymbol{\epsilon}(t)\|_2^2] &= \mathbb{E}\left[\mathbb{E}[\|\boldsymbol{\epsilon}(t)\|_2^2 \,|\boldsymbol{\epsilon}(t-1)]\right] \\
&\leq \mathbb{E}\left[\lambda_2\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right)\|\boldsymbol{\epsilon}(t-1)\|_2^2\right] \\
&= \lambda_2\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right)\mathbb{E}[\|\boldsymbol{\epsilon}(t-1)\|_2^2] \\
&\leq \lambda_2^t\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right)\|\boldsymbol{\epsilon}(0)\|_2^2\,, \qquad (3.49)
\end{aligned}$$

which implies that $\lim_{t\to\infty}\mathbb{E}[\|\boldsymbol{\epsilon}(t)\|_2^2] = 0$. ■

Comments

- If the network sequence $\{\mathcal{G}(t)\}_{t\geqslant 0}$ is constant or i.i.d. time varying, this theorem applies to synchronized average consensus algorithms and to all gossip algorithms.

- According to Theorem 3.1.4, the sufficient conditions for mean square convergence stated above are also sufficient conditions for **convergence with probability** 1.

- Contrary to Theorem 3.1.6 (convergence in expectation), Theorem 3.1.7 (mean square convergence) gives information about the speed of convergence of the algorithm:

$$\lim_{t\to\infty}\sup\left(\frac{\mathbb{E}[||\boldsymbol{\epsilon}(t)||_2^2]}{||\boldsymbol{\epsilon}(0)||_2^2}\right)^{\frac{1}{t}} \leq \lambda_2\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right). \qquad (3.50)$$

- As a direct consequence, it is now clear why *gossip* algorithms are nice to study (see Definition 2.2.1 of gossip at p. 20). For any gossip algorithm, $\mathbf{W}(t)^T\mathbf{W}(t) = \mathbf{W}$ for any time t, hence

$$\lambda_2\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right) = \lambda_2\left(\mathbb{E}[\mathbf{W}]\right),$$

which simplifies the derivations when computing the upper bound in Eq. 3.50.

- In [15], Boyd et al. develop, in the case of i.i.d. matrices $\mathbf{W}(t)$, some necessary and sufficient conditions for mean square convergence. They are the same as in Theorem 3.1.7, except that Condition 3.46 is replaced by:

$$\rho\left(\mathbb{E}[\mathbf{W}\otimes\mathbf{W}]-\frac{1}{n^2}\mathbf{1}\mathbf{1}^T\right)<1, \tag{3.51}$$

  where $\otimes$ denotes the Kronecker product, and where $\mathbf{1}$ is the all ones vector of size n^2. Although sharper than Condition (3.46), it is more complicated, and thus less practical and well-known. Interestingly, the next Section develops a similar formula, although different, which describes the speed of convergence.

- Finally, when matrices $\{\mathbf{W}(t)\}_{t\geqslant 0}$ are i.i.d, similar calculations give a sufficient condition for $\boldsymbol{\eta}(t)=\mathbf{x}(t)-\mathbf{J}\mathbf{x}(t)$ to mean square converge to 0:

$$\rho\left(\mathbb{E}[\mathbf{W}(t)^T(\mathbf{I}-\mathbf{J})\mathbf{W}(t)]\right)<1. \tag{3.52}$$

  By chance, in broadcast gossip (which is not strictly speaking a gossip algorithm), $\mathbb{E}[\mathbf{W}^T(\mathbf{I}-\mathbf{J})\mathbf{W}]$ is computable, which makes the analysis of the performance of broadcast gossip possible.

In the first part of this chapter, we explained how to prove that a distributed average consensus algorithm converges in expectation, in mean square or with probability 1. In the second part, we will focus on the speed at which the error decays and on the performance of a distributed average consensus algorithm.

MAIN CONVERGENCE THEOREMS

- Deterministic average consensus: $\lim_{t\to\infty} \mathbf{W}^t = \mathbf{J}$ if and only if

$$\begin{aligned} \mathbf{W}\mathbf{1} &= \mathbf{1}, \\ \mathbf{1}^T\mathbf{W} &= \mathbf{1}^T, \\ \rho\left(\mathbf{W} - \mathbf{J}\right) &< 1. \end{aligned}$$

- Convergence in expectation: Assume that $\{\mathbf{W}(t)\}_{t\geqslant 0}$ is i.i.d., then average consensus converges in expectation if and only if

$$\begin{aligned} \mathbb{E}[\mathbf{W}]\mathbf{1} &= \mathbf{1}, \\ \mathbf{1}^T\mathbb{E}[\mathbf{W}] &= \mathbf{1}^T, \\ \rho\left(\mathbb{E}[\mathbf{W}] - \mathbf{J}\right) &< 1. \end{aligned}$$

- Mean square convergence and convergence with probability 1: Assume that $\{\mathbf{W}(t)\}_{t\geqslant 0}$ is i.i.d. If

$$\begin{aligned} \mathbf{W}(t)\mathbf{1} &= \mathbf{1} \quad \text{for any time } t, \\ \mathbf{1}^T\mathbf{W}(t) &= \mathbf{1}^T \quad \text{for any time } t, \\ \lambda_2\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right) &< 1, \end{aligned}$$

  where λ_2 denotes the second largest eigenvalue in magnitude (SLEM), then average consensus not only mean square converges, but also converges with probability 1.

3.2 Convergence Time

This section focuses on distributed averaging algorithms that verify the conditions of Theorem 3.1.3, and thus converge to a consensus with probability 1. In addition, the algorithms are assumed to conserve the average: for any iteration t,

$$\mathbf{1}^T\mathbf{W}(t) = \mathbf{1}^T.$$

As a consequence, in these algorithms, the estimates of all the nodes converge to x_{ave}, and the question is to know how fast and also how reliably fast.

There are two main approaches to this problem. The first approach (described in Section 3.2.1) is probabilistic. It estimates the number of iterations needed to estimate x_{ave} with precision at least ε and with probability at least $1-\eta$. The second approach (described in Section 3.2.2) has deterministic ambitions. It focuses on the characteristics of the behavior of the error $\boldsymbol{\epsilon}(t)$ that are predictable with very high probability or even with probability 1. Although the first approach is well-understood and broadly used in the literature, the second approach still have gray areas with open questions. More precisely, some interesting phenomena observed in simulations remain partially unexplained.

3.2.1 (ε, η)-Averaging time T_{ave}

Averaging time is the most broadly used metric in the literature to measure the speed of convergence of a distributed average consensus algorithm ([15,7]). Recall that $\boldsymbol{\epsilon}(t) = \mathbf{x}(t) - x_{ave}\mathbf{1}$.

Definition 3.2.1 ((ε, η)-averaging time: $T_{ave}(n, \varepsilon, \eta)$). For a given algorithm, for a given graph with n nodes, for any $0 < \varepsilon < 1$ and any $0 < \eta < 1$, the (ε, η)-averaging time $T_{ave}(n, \varepsilon, \eta)$ is defined as:

$$T_{ave}(n, \varepsilon, \eta) = \sup_{\mathbf{x}(0)} \inf_t \mathbb{P}\left[\frac{\|\boldsymbol{\epsilon}(t)\|_2}{\|\boldsymbol{\epsilon}(0)\|_2} \geqslant \varepsilon\right] \leqslant \eta. \tag{3.53}$$

Comments:

- The number n of nodes does not directly appear in the definition, but indirectly as $\boldsymbol{\epsilon}(t)$ depends on n. When studying the performance of an algorithm, the main question is to know how averaging time behaves asymptotically when the network grows. Therefore n is an important parameter and it is included in the definition.

- In the literature, ε is chosen to be equal to η to simplify equations even if there is no substantial reason for it. This section derives the classical theorems on T_{ave} [15] with distinct parameters ε and η, so that their respective effects can be pointed out. The reader will then understand the reasons why it is more practical to have $\varepsilon = \eta$.

Theorem* 3.2.1 (Upper bounding $T_{ave}(n, \varepsilon, \eta)$). Assume that $\{\mathbf{W}(t)\}_{t\geq 0}$ is independently and uniformly distributed and that all the conditions for mean

square convergence are met (see Theorem 3.1.7):

$$\begin{aligned} \mathbf{W}(t)\mathbf{1} &= \mathbf{1} \quad \text{for any time } t, \\ \mathbf{1}^T\mathbf{W}(t) &= \mathbf{1}^T \quad \text{for any time } t, \\ \lambda_2\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right) &< 1, \end{aligned}$$

where λ_2 denotes the second largest eigenvalue in magnitude (SLEM). Then

$$T_{ave}(n,\varepsilon,\eta) \leqslant \frac{3\max\left(\log(\eta^{-1}),\log(\varepsilon^{-1})\right)}{\log\left(\lambda_2^{-1}\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right)\right)}.$$

The following corollary is a well-known result that is frequently used in the literature.

Corollary 3.2.1. With the same assumptions as in Theorem 3.2.1, if $\varepsilon = \beta/n$ and if $\eta = 1/n^\alpha$ for some constants $\beta > 0$ and $\alpha > 0$, then, for large n,

$$\begin{aligned} T_{ave}(n,\varepsilon,\eta) &= O\left(\frac{\log n}{\log\left(\lambda_2^{-1}\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right)\right)}\right) \\ &= O\left(\frac{\log n}{1-\lambda_2\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right)}\right), \end{aligned}$$

where the last equality holds if $\lambda_2\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right)$ tends to 1 when n goes to infinity.

Comments:

- The corollary is a direct application of Theorem 3.2.1.
- Choice of $\varepsilon = \beta/n$: Take an initial signal $\mathbf{x}(0)$ of average x_{ave} and initial error $\boldsymbol{\epsilon}(0)$. If the initial error is equally shared among nodes, then the typical measurement of a node i is

$$x_i(0) = x_{ave} \pm \frac{\|\boldsymbol{\epsilon}(0)\|_2}{\sqrt{n}}. \tag{3.54}$$

  Now, construct a signal $\mathbf{x}'(0)$ such that $\mathbf{x}'(0) = \mathbf{x}(0) + \beta\left(\|\boldsymbol{\epsilon}(0)\|_2/\sqrt{n}\right)\mathbf{e}_j$, where $\mathbf{e}_j$ is the vector of all zeros except for the jth component (equal to 1). Then $\|\mathbf{J}\mathbf{x}'(0) - \mathbf{J}\mathbf{x}(0)\|_2 / \|\boldsymbol{\epsilon}(0)\|_2 = \beta/n$. Therefore, choosing $\varepsilon = \beta/n$ corresponds to estimating the average of $\mathbf{x}(0)$ up to one node precision.
- Choice of $\eta = 1/n^\alpha$: this is a classical choice of probability error. It leads to statements that hold asymptotically "with high probability".
- For any gossip algorithm (see Definition 2.2.1), $\lambda_2\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right) = \lambda_2\left(\mathbb{E}[\mathbf{W}]\right)$, as already noted in Section 3.1.4.
- Finally, taking $\varepsilon = \eta$ is convenient because

$$T_{ave}(n,\varepsilon,\varepsilon) = O\left(\frac{\log \varepsilon^{-1}}{\log\left(\lambda_2^{-1}\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right)\right)}\right). \tag{3.55}$$

Proof:[Theorem 3.2.1] The proof is directly inspired from the proof in [15], where $\varepsilon = \eta$. The idea is to prove the following: For $t \geq K(\varepsilon, \eta)$, where

$$K(\varepsilon, \eta) = \frac{3 \max\left(\log(\eta^{-1}), \log(\varepsilon^{-1})\right)}{\log\left(\lambda_2^{-1}\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right)\right)},$$

and for any initial signal $\mathbf{x}(0)$,

$$\mathbb{P}\left[\frac{\|\boldsymbol{\epsilon}(t)\|_2}{\|\boldsymbol{\epsilon}(0)\|_2} \geqslant \varepsilon\right] \leqslant \eta.$$

Recall that the assumptions of Theorem 3.2.1 imply that $\boldsymbol{\epsilon}(t) = \mathbf{W}(t)\boldsymbol{\epsilon}(t-1)$ (see page 45).

By Markov's inequality,

$$\begin{aligned}
\mathbb{P}\left[\frac{\|\boldsymbol{\epsilon}(t)\|_2}{\|\boldsymbol{\epsilon}(0)\|_2} \geq \varepsilon\right] &= \mathbb{P}\left[\frac{\boldsymbol{\epsilon}(t)^T\boldsymbol{\epsilon}(t)}{\boldsymbol{\epsilon}(0)^T\boldsymbol{\epsilon}(0)} \geq \varepsilon^2\right] \\
&\leq \varepsilon^{-2}\frac{\mathbb{E}[\boldsymbol{\epsilon}(t)^T\boldsymbol{\epsilon}(t)]}{\boldsymbol{\epsilon}(0)^T\boldsymbol{\epsilon}(0)} \\
&\leq \varepsilon^{-2}\left(\lambda_2(\mathbb{E}[\mathbf{W}^T\mathbf{W}])\right)^t,
\end{aligned}$$

where the last inequality comes from Eq. (3.49). If $t \geq K(\varepsilon, \eta)$, then

$$\begin{aligned}
\mathbb{P}\left[\frac{\|\boldsymbol{\epsilon}(t)\|_2}{\|\boldsymbol{\epsilon}(0)\|_2} \geq \varepsilon\right] &\leq \varepsilon^{-2}\left(\lambda_2(\mathbb{E}[\mathbf{W}^T\mathbf{W}])\right)^{K(\varepsilon,\eta)} \\
&= \exp\left[-2\log\varepsilon - K(\varepsilon,\eta)\log\left(\lambda_2^{-1}\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right)\right)\right] \\
&= \exp\left[-2\log\varepsilon - 3\max\left(\log(\eta^{-1}), \log(\boldsymbol{\epsilon}^{-1})\right)\right].
\end{aligned}$$

If $\varepsilon \leqslant \eta$,

$$\begin{aligned}
\mathbb{P}\left[\frac{\|\boldsymbol{\epsilon}(t)\|_2}{\|\boldsymbol{\epsilon}(0)\|_2} \geq \varepsilon\right] &\leq \exp\left[-2\log\varepsilon + 3\log\varepsilon\right] = \varepsilon \\
&\leq \eta.
\end{aligned}$$

If $\varepsilon > \eta$,

$$\begin{aligned}
\mathbb{P}\left[\frac{\|\boldsymbol{\epsilon}(t)\|_2}{\|\boldsymbol{\epsilon}(0)\|_2} \geq \varepsilon\right] &\leq \exp\left[-2\log\varepsilon + 3\log\eta\right] \\
&\leq \exp\left[-2\log\eta + 3\log\eta\right] \\
&= \eta.
\end{aligned}$$

■

Now that we have shown upper bounds, we will focus on lower bounds and extend the result in [15] to distinct parameters ε and η. To prove a lower bound of T_{ave}, an extra condition on $\mathbb{E}[\mathbf{W}]$ is necessary:

Theorem* 3.2.2 (Lower bounding T_{ave}). In addition to the assumptions of Theorem 3.2.1, suppose that $\mathbb{E}[W]$ is a *symmetric* matrix, with second largest

eigenvalue in magnitude (SLEM) smaller than 1. Then,

$$T_{ave}(n,\varepsilon,\eta) \geqslant \frac{\min\left(\log(\eta^{-1}), 2\log(\varepsilon^{-1})\right) - \log 3}{2\log\left(\lambda_2^{-1}\left(\mathbb{E}[\mathbf{W}]\right)\right)}.$$

The following corollary is the one frequently used in the literature on gossip algorithms:

Corollary 3.2.2. If the assumptions of Theorem 3.2.1 hold, if $\varepsilon = \beta/n$, if $\eta = 1/n^{\alpha}$ for some constants $\beta > 0$ and $\alpha > 0$, and if moreover $\mathbf{W}(t)$ is a *gossip* matrix for all t, then, for large n,

$$T_{ave}(n,\varepsilon,\eta) = \Theta\left(\frac{\log n}{\log\left(\lambda_2^{-1}\left(\mathbb{E}[\mathbf{W}]\right)\right)}\right) \tag{3.56}$$

$$= \Theta\left(\frac{\log n}{1-\lambda_2\left(\mathbb{E}[\mathbf{W}]\right)}\right), \tag{3.57}$$

where the last equality holds if $\lambda_2\left(\mathbb{E}[\mathbf{W}]\right)$ tends to 1 when n goes to infinity.

Proof: [Corollary 3.2.2] By definition of a gossip algorithm, $\mathbf{W}(t)$ is symmetric and $\mathbf{W}(t)^2 = \mathbf{W}(t)$. Therefore $\mathbb{E}[\mathbf{W}]$ is symmetric as well, and $\lambda_2(\mathbb{E}[\mathbf{W}]) = \lambda_2\left(\mathbb{E}[\mathbf{W}^T\mathbf{W}]\right) < 1$. The conditions to apply Theorem 3.2.2 are thus fulfilled and the corollary comes from the combination of Theorems 3.2.1 and 3.2.2. ■

Proof: [Theorem 3.2.2] The proof is directly inspired from the proof in [15], where $\varepsilon = \eta$. Let $K'(\varepsilon,\eta)$ be defined as:

$$K'(\varepsilon,\eta) = \frac{\min\left(\log(\eta^{-1}), 2\log(\varepsilon^{-1})\right) - \log 3}{2\log\left(\lambda_2^{-1}\left(\mathbb{E}[\mathbf{W}]\right)\right)}.$$

The goal of this proof is to show that there exists an initial signal $\mathbf{x}(0)$ such that for any $t < K'(\varepsilon,\eta)$

$$\mathbb{P}\left[\frac{\|\boldsymbol{\epsilon}(t)\|_2}{\|\boldsymbol{\epsilon}(0)\|_2} \geqslant \varepsilon\right] > \eta.$$

The process $\{\mathbf{W}(t)\}_{t\geqslant 1}$ is assumed to be independent, and the assumptions of the theorem imply that for any t, $\boldsymbol{\epsilon}(t) = \mathbf{W}(t)\boldsymbol{\epsilon}(t-1)$. Therefore, for any t, $\mathbb{E}[\boldsymbol{\epsilon}(t)] = \mathbb{E}[\mathbf{W}]^t\boldsymbol{\epsilon}(0)$. Since $\mathbb{E}[\mathbf{W}]$ is a symmetric doubly stochastic matrix, it has n real eigenvalues. The largest eigenvalue is 1, and the second largest eigenvalue in magnitude is $\lambda_2(\mathbb{E}[\mathbf{W}])$, which is assumed to be smaller than 1. Let v_2 be the normal eigenvector associated with $\lambda_2(\mathbb{E}[\mathbf{W}])$[2]. Choose

$$\mathbf{x}(0) = \frac{1}{\sqrt{n}}\mathbf{1} + \mathbf{v}_2,$$

Since the eigenvectors of a symmetric matrix are orthogonal,

$$\boldsymbol{\epsilon}(0) = \mathbf{v}_2,$$

[2] If the eigenvalue is negative, $\mathbf{v}_2$ is the normal eigenvector associated with $-\lambda_2(\mathbb{E}[\mathbf{W}])$, and a minus sign should be added in the proof. Note that in gossip algorithms, since $\mathbf{W}(t) = \mathbf{W}(t)^T\mathbf{W}(t)$ for any t, then $\mathbb{E}[\mathbf{W}]$ is in addition a positive semi-definite matrix, implying that the n eigenvalues are non-negative.

and $\|\boldsymbol{\epsilon}(0)\|_2 = 1$. Furthermore

$$\mathbb{E}[\boldsymbol{\epsilon}(t)] = \lambda_2^t(\mathbb{E}[\mathbf{W}])\mathbf{v}_2.$$

As shown in [15], if X is a bounded random variable such that $0 \leqslant X \leqslant B$, then, for any $0 < \varepsilon < B$,

$$\mathbb{P}[X \geqslant \varepsilon] \geqslant \frac{\mathbb{E}[X] - \varepsilon}{B - \varepsilon}. \tag{3.58}$$

As $\{\|\boldsymbol{\epsilon}(t)\|_2\}_{t\geqslant 0}$ is a decreasing sequence, it is upper bounded by $\|\boldsymbol{\epsilon}(0)\|_2 = 1$. Equation (3.58) can thus be applied with $X = \|\boldsymbol{\epsilon}(t)\|_2^2$:

$$\mathbb{P}[\|\boldsymbol{\epsilon}(t)\|_2^2 \geqslant \varepsilon^2] \geqslant \frac{\mathbb{E}[\|\boldsymbol{\epsilon}(t)\|_2^2] - \varepsilon^2}{1 - \varepsilon^2}.$$

Now $\mathbb{E}[\|\boldsymbol{\epsilon}(t)\|_2]$ needs to be lower bounded. By Jensen's inequality,

$$\begin{aligned}
\mathbb{E}[\|\boldsymbol{\epsilon}(t)\|_2^2] &\geqslant \|\mathbb{E}[\boldsymbol{\epsilon}(t)]\|_2^2 \\
&= \mathbb{E}[\boldsymbol{\epsilon}(t)]^T\mathbb{E}[\boldsymbol{\epsilon}(t)] \\
&= \lambda_2^{2t}(\mathbb{E}[\mathbf{W}])\mathbf{v}_2^T\mathbf{v}_2 \\
&= \lambda_2^{2t}(\mathbb{E}[\mathbf{W}]).
\end{aligned}$$

For any $t < K'(\varepsilon, \eta)$,

$$\begin{aligned}
\mathbb{E}[\|\boldsymbol{\epsilon}(t)\|_2^2] &\geqslant \lambda_2^{2K'(\varepsilon,\eta)}(\mathbb{E}[\mathbf{W}]) \\
&= \exp\left(-2K'(\varepsilon,\eta)\log\lambda_2^{-1}(\mathbb{E}[\mathbf{W}])\right) \\
&= \exp\left(-\min\left(\log(\eta^{-1}), 2\log(\varepsilon^{-1})\right) + \log 3\right) \\
&= 3\exp\left(-\min\left(\log(\eta^{-1}), 2\log(\varepsilon^{-1})\right)\right).
\end{aligned}$$

If $\epsilon^2 < \eta$, then $\log(\eta^{-1}) < 2\log(\varepsilon^{-1})$, and

$$\begin{aligned}
\mathbb{P}[\|\boldsymbol{\epsilon}(t)\|_2 \geqslant \varepsilon] &\geqslant \frac{3\eta - \varepsilon^2}{1 - \varepsilon^2} \\
&\geqslant 2\eta \\
&> \eta.
\end{aligned}$$

If $\epsilon^2 \geqslant \eta$, then $\log(\eta^{-1}) \geqslant 2\log(\varepsilon^{-1})$, and

$$\begin{aligned}
\mathbb{P}[\|\boldsymbol{\epsilon}(t)\|_2 \geqslant \varepsilon] &\geqslant \frac{3\varepsilon^2 - \varepsilon^2}{1 - \varepsilon^2} \\
&\geqslant 2\epsilon^2 \\
&\geqslant 2\eta \\
&> \eta.
\end{aligned}$$

Since $\|\boldsymbol{\epsilon}(0)\|_2 = 1$, this concludes the proof. ■

Comments:

- The reader might have noted that using T_{ave} and its bounds is not as

easy as the literature suggests. Usually papers do not justify precisely the bounds they use. First, one should not confuse matrices $\mathbb{E}[\mathbf{W}]$ and $\mathbb{E}[\mathbf{W}^T\mathbf{W}]$, which are equal only in the gossip case. Then, as ϵ is always taken to be equal to η, the meaning of T_{ave} is hard to understand, and part of the intuition disappears in the stated results, which can confuse an non-expert reader.

Finally, one should not forget that these bounds are valid only if the sequence $\{\mathbf{W}(t)\}_{t\geqslant 1}$ is i.i.d. Therefore, if there are some correlations in time for link failures for instance, this analysis does not hold anymore.

- Note that the bounds are loose. The upper bound was obtained with the Markov inequality, which is very loose, and the lower bound uses Eq. (3.58), which is a coarse inequality.

- Assume that the average is not necessarily conserved through iterations ($\mathbf{1}^T\mathbf{W}(t) \neq \mathbf{1}^T$ for some time t), but that conditions to converge to consensus with probability 1 hold (see Th. 3.1.3 in page 40). Define

$$T'_{ave}(n,\varepsilon,\eta) = \sup_{\mathbf{x}(0)} \inf_t \mathbb{P}\left[\frac{\|\boldsymbol{\eta}(t)\|_2}{\|\boldsymbol{\epsilon}(0)\|_2} \geqslant \varepsilon\right] \leqslant \eta. \tag{3.59}$$

If $\mathbb{E}[\mathbf{W}]$ is a symmetric matrix, then, similar calculations lead to the following bounds:

$$\begin{aligned} &\frac{\min\left(\log(\eta^{-1}), 2\log(\varepsilon^{-1})\right) - \log 3}{2\log\left(\lambda_2^{-1}\left(\mathbb{E}[\mathbf{W}]\right)\right)} \\ \leqslant\ & T'_{ave}(n,\varepsilon,\eta) \\ \leqslant\ & \frac{3\max\left(\log(\eta^{-1}), \log(\varepsilon^{-1})\right)}{\log\left(\rho^{-1}\left(\mathbb{E}[\mathbf{W}^T(\mathbf{I}-\mathbf{J})\mathbf{W}]\right)\right)}. \end{aligned}$$

- The first two points motivated the work of the following section, where finer bounds are searched. To achieve this, next section gives a closer look to the behavior of $\{\boldsymbol{\epsilon}(t)\}_{t\geqslant 0}$.

3.2.2 Consensus time T_c

In this section, it is assumed that the conditions of Theorem 3.1.3 hold (i.e. the algorithms converge to a consensus with probability 1), and that, for any t, $\mathbf{1}^T\mathbf{W}(t) = \mathbf{1}^T$ (the consensus value is always x_{ave}).

The work of this section started with a simple but surprising observation on simulations. For any algorithm described in this thesis, and any initial signal $\mathbf{x}(0)$, the temporal function $\log\|\boldsymbol{\epsilon}(t)\|$ looks linear for times t large enough. Furthermore, the empirical slope of $\log\|\boldsymbol{\epsilon}(t)\|$ for large t does not seem to vary with $\mathbf{x}(0)$, nor with the norm $||\cdot||$, nor with the instantiation of the algorithm (see Fig. 3.1).

If this observation were proven to be true, it would offer a powerful new tool to measure the speed of convergence. The error $\|\boldsymbol{\epsilon}(t)\|$ would decay exponentially for large enough times, and its decay rate would be measurable on a few runs of the algorithms. This measure of speed of convergence would be more

practical than averaging time T_{ave}, since T_{ave} cannot be measured empirically. It can only be bounded analytically in a few cases.

This section[3] analyzes the observed phenomenon and tries to quantify it. It starts by stating some theorems on the existence of limits that were found in the 1960's. This will lead to the definition of *consensus time* T_c as the inverse of the limiting slope of $\log(\|\boldsymbol{\epsilon}(t)\|)$ (in absolute value). Second, the theorems will be confronted to simulations, with an example on which we can see that the slope of $\log(\|\boldsymbol{\epsilon}(t)\|)$ does not seem to depend on the norm nor on the initial signal $\mathbf{x}(0)$. Then, we will question whether the slope that we observe at finite large times is close to its limit or not, and the meaning of T_c in practice will be discussed. This will be followed by some upper bounds on the consensus time T_c. Finally, based on a concentration result proved in [31], we will explain how and why we will display the slope of $\log(\|\boldsymbol{\epsilon}(t)\|)$ in all the simulations of this manuscript. We further give an example of algorithm, which is difficult to study with averaging time T_{ave}, but where the display of the observed slopes at finite times is insightful. In summary, we develop here a tool which is very practical to measure empirically the performance of an algorithm.

For the tool to be meaningful, it should give a precise and deterministic description of the convergence of distributed averaging algorithms, despite the randomness of the algorithms and of the network. The observation of simulations suggest the definition of a metric (consensus time), that would be the inverse of the asymptotic slope of $\log \|\boldsymbol{\epsilon}(t)\|$, if theory confirms that it indeed exists:

Definition* 3.2.1 (Consensus time). Whenever the limit:

$$T_c = \lim_{t\to\infty} \left| \frac{1}{t} \log \|\boldsymbol{\epsilon}(t)\| \right|^{-1}.$$

exists, it is called consensus time.

As shown in the following section, this limit exists in a very large framework: whenever the process $\{\mathbf{W}(t)\}_{t\geqslant 1}$ is stationary and ergodic and when $||\cdot||$ is sub-multiplicative. Most importantly, it is possible to show that the limit is almost surely independent of the initial signal $\mathbf{x}(0)$, of the norm $||\cdot||$, and of the instantiation of the algorithm. Therefore, T_c can be interpreted as the asymptotic number of iterations needed to decrease the error $\|\boldsymbol{\epsilon}(t)\|$ by a factor e.

Existence of consensus time T_c

Recall that $\boldsymbol{\epsilon}(t) = \prod_{k=1}^{t-1} (\mathbf{W}(t-k) - \mathbf{J})\, \mathbf{x}(0)$. Therefore, $\boldsymbol{\epsilon}(t)$ involves a random product of matrices, which is a mathematical object that has been studied over several decades.

1-Random product of matrices. It is well known [45, page 299] that for any submultiplicative matrix norm $\|\cdot\|$ (for all A,B, $\|AB\| \leqslant \|A\|\|B\|$), $\lim_{t\to\infty} \|\mathbf{A}^t\|^{1/t} = \rho(\mathbf{A})$. If $\rho(\mathbf{A}) \neq 0$, we can rewrite this as $\lim_{t\to\infty} \frac{1}{t} \log \|\mathbf{A}^t\| =$

[3] This section is joint work with Patrick Denantes (Master Thesis work). It was published in [24]

$\log \rho(\mathbf{A})$. How does this extend to the case where $\mathbf{A}(t)$ varies over time? Is there any way to specify or to compute an asymptotic contraction rate of *matrices*

$$\lim_{t\to\infty} \frac{1}{t} \log \left\| \prod_{k=1}^{t} \mathbf{A}(t-k) \right\|, \tag{3.60}$$

and when is this quantity well-defined in the first place?

Theorem 3.2.1. (Fuerstenberg and Kesten's Theorem. [40]) Let $\{\mathbf{A}(t)\}_{t\geq 1}$ be a stationary and ergodic sequence of $n \times n$ matrices, and let $\|\cdot\|$ be any submultiplicative matrix norm. If

$$\mathbb{E}\left[\max(0, \log \|\mathbf{A}(0)\|)\right] < \infty, \tag{3.61}$$

then, if $\mathbf{M}_t := \prod_{k=1}^{t-1} \mathbf{A}(t-k)$ for any t, the limit

$$\lim_{t\to\infty} \frac{1}{t} \log \|\mathbf{M}_t\|$$

exists, and, is equal with probability 1 to

$$\gamma := \lim_{t\to\infty} \frac{1}{t} \mathbb{E}\left[\log \|\mathbf{M}_t\|\right]. \tag{3.62}$$

Fuerstenberg and Kesten's theorem shows that, as we assume $\{\mathbf{A}(t)\}_{t\geq 1}$ to be stationary and ergodic, the asymptotic contraction rate γ is well-defined and is a deterministic constant with probability 1. Note that in the case of averaging algorithms, $\{\mathbf{W}(t)\}_{t\geq 0}$ are contracting matrices, which implies that $\{\mathbf{A}(t)\}_{t\geq 1} = \{\mathbf{W}(t) - \mathbf{J}\}_{t\geq 1}$ are bounded matrices. Hence condition (3.61) is fulfilled, and the theorem can be applied to the matrices $\{\mathbf{W}(t) - \mathbf{J}\}_{t\geq 1}$.

2-Contraction rate of the average estimates. It is now legitimate to ask how the estimation error *signal* $\|\epsilon(t)\| = \|\mathbf{M}_t \mathbf{x}(0)\|$ behaves (here $\mathbf{A}(t) = \mathbf{W}(t) - \mathbf{J}$). Oseledec's theorem [60, 77, 69, 66] states that the error contracts with the same rate γ as in Theorem 3.2.1 for almost every initial measurement $\mathbf{x}(0)$. In other words, Theorem 3.2.1 can be extended from matrices to signals.

Theorem 3.2.2. (Oseledec's Theorem. [60,66]) Let $\|\cdot\|$ be a norm on $\mathbb{R}^n$ equivalent to a p-norm, with $p \in [1, \infty]$, let $\{\mathbf{A}(t)\}_{t\geq 0}$ be a stationary and ergodic sequence of matrices satisfying condition (3.61), and $\mathbf{M}_t := \prod_{k=1}^{t} \mathbf{A}(t-k)$. Then, with probability 1, there is a proper subspace V of $\mathbb{R}^n$ such that, for all $\mathbf{x} \in \mathbb{R}^n \setminus V$,

$$\lim_{t\to\infty} \frac{1}{t} \log \|\mathbf{M}_t \mathbf{x}\| = \gamma, \tag{3.63}$$

where γ is defined by (3.62) in Theorem 3.2.1.

Theorem 3.2.2 shows therefore that $(\log \|\epsilon(t)\|)/t \to \gamma$ and that $T_c = -1/\gamma$. In the introductory statement, "almost every" starting point $\mathbf{x}$ means all $\mathbf{x} \in \mathbb{R}^n \setminus V$, i.e. all $\mathbf{x}$ outside a proper linear subspace of $\mathbb{R}^n$. Note that if $\mathbf{A}(t) = \mathbf{A}$ is a constant (i.e. degenerate random variable), then e^{γ} is the absolute value of the largest eigenvalue of $\mathbf{A}$ in magnitude. It is also interesting to see that

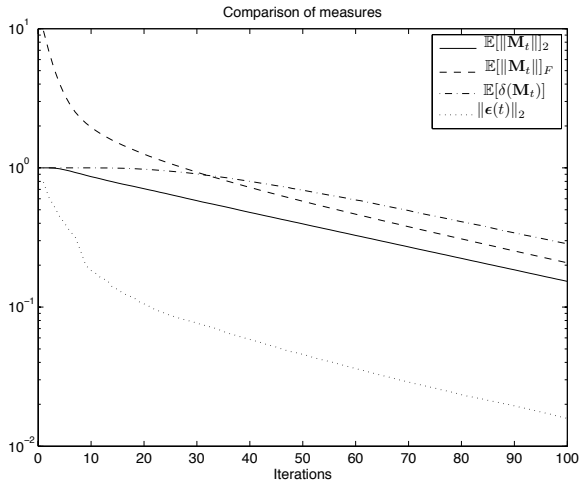


Figure 3.1: Different metrics for the residual error $\mathbf{M}_t = \prod_{k=1}^{t-1}(\mathbf{W}(t-k) - \mathbf{J})$, using the Metropolis weighting algorithm on a random geometric graph (RGG) with 200 nodes, and with link failure probability $p = 0.9$. Note that this figure shows the logarithm of the mean of an error metric $\|\mathbf{M}_t\|$ or $\delta(\mathbf{M}_t)$. Theorems. 3.2.1 and 3.2.2 however consider the expectation of logarithmic error metric $\log \|\mathbf{M}_t\|$ as the value of interest. The difference is very small in practice, so it would not be visible here if $\exp\left(\mathbb{E}\left[\log \|\mathbf{M}_t\|\right]\right)$ would have been plotted instead of $\mathbb{E}\left[\|\mathbf{M}_t\|\right]$.

γ does not depend on the norm $\|.\|$, and that the subspace V is random. This thesis mentions only one coefficient γ, which characterizes the speed of the slowest component of the signal, but a full version of the theorem states the existence of other smaller coefficients, called Lyapunov exponents, which affect the transient phase.

An example

Fig. 3.1 shows, in logarithmic scale, the evolution of selected error metrics over algorithm iterations (time), using the Metropolis weighting algorithm on a random geometric graph (RGG) with 200 nodes, connection radius $r = 0.2$, and link failure probability $p = 0.9$.

The Metropolis weighting algorithm is defined in page 19. The link failure probability p is set to 0.9. This means that a given link is active at time instant t with probability $1 - p = 0.1$, independently of other links and independently of its own activity at previous (or future) time instants.

The plot shows four error metrics, which were defined at the beginning of the manuscript in page xv: the Dobrushin-seminorm, the 2-norm (which is abusively denoted here by $\|\cdot\|_2$ instead of $|||\cdot|||_2$) and the Frobenius norm of the residual matrix M_t, all of them averaged over 400 realizations of link failure/activity sequences. The evolution of the quadratic norm of an error vector $\boldsymbol{\epsilon}(t)$ is also shown, but only one realization for one time sequence, with an initial measurement that was a basis unit vector, i.e. all n nodes had initial value 0, except one node i, which was initialized with $\mathbf{x}_i(0) = 1$.

Choosing to plot this curve for a specific realization instead of an averaged curve shows that the variance of $\boldsymbol{\epsilon}(t)$ is very small, and that the typical behavior is very close to the mean behavior. Moreover, the exponential decay rates (the slopes) of all 4 metrics look the same for large times. In particular, the Frobenius-norm approaches the 2-norm. This illustrates Theorem 3.2.2, which states that $\mathbf{M}_t$ tends to become one-dimensional or, in other words, there tends to be only one dominant singular value, and consequently $\|\mathbf{M}_t\|_F = \sum \sigma_i \to \sigma_1 = |||\mathbf{M}_t|||_2$.

Finally, one should note that for an arbitrary initial vector, the error might, in the beginning, decrease much faster than $|||\mathbf{M}_t|||_2$, which by definition always gives the relative error of the "worst case" starting point $\mathbf{y}(0)$.

By a similar argument, one could expect the ergodicity coefficient δ to have a steeper transient than $|||\mathbf{M}_t|||_2$, because in the case of δ, the maximization goes only over starting vectors of the form $\mathbf{y}(0) = 1/2(\mathbf{e}_i - \mathbf{e}_j)$, $i, j = 1 \ldots n$, and not over all $\mathbf{y}(0) \in \mathbb{R}^n$. However, there is another difference between δ and $|||\mathbf{M}_t|||_2$. δ is defined based on the 1-norm,

$$\delta(\mathbf{M}_t) = \max_{i,j} \frac{1}{2} \|\mathbf{M}_t(\mathbf{e}_i - \mathbf{e}_j)\|_1 ,$$

as opposed to $|||\mathbf{M}_t|||_2$, which is defined based on the 2-norm. Because the effect of changing the norm is stronger than that of the constraint on maximization, we observe $\delta(\mathbf{M}_t) > |||\mathbf{M}_t|||_2$.

The main conclusion from this simulation is that the 4 metrics described above give very similar slope measurements. In addition, the slopes seem to reach their limit in short times. This looks like a direct illustration of Theorem 3.2.2. However, as the next section shows, we cannot rigorously tell that the slopes we observe at finite times are indeed close to their common limit.

Is T_c the inverse of the observed slope?

In this thesis, by definition, the "slope" at time t is the slope of $\log(\|\boldsymbol{\epsilon}(t)\|)$. From the previous analysis, we know that the slope has a limit, but this limit is not observable by a computer. Therefore we introduce the empirical convergence rate, which is the quantity that is observable in practice:

Definition* 3.2.2. *Empirical convergence rate* γ_{t_1}. Let $\varepsilon(t) = \|x(t) - x_{ave}\mathbf{1}\|$ be the estimation error at time t. Given t_1 and $t > t_1$, the empirical convergence rate $\gamma_{t_1}(t)$ is

$$\gamma_{t_1}(t) = \frac{\log(\varepsilon(t)) - \log(\varepsilon(t_1))}{t - t_1}. \tag{3.64}$$

If $\varepsilon(t)$ were exponentially decreasing with negative rate γ', then $\gamma_{t_1}(t)$ would be a constant equal to γ'. The introduction of time t_1 in the definition of the empirical convergence rate allows to get rid of the transient phase by taking t_1 large enough. As Theorem 3.2.2 has shown, $\gamma_{t_1}(t)$ converges to a value γ, and T_c, which is the limit of the inverse of the slope in absolute value exists. T_c and γ do not depend on $\mathbf{x}(0)$ (except for signals $\mathbf{x}(0)$ included in a random proper subspace), nor on the particular instantiation of the algorithm, nor on the norm used to measure $\|\boldsymbol{\epsilon}(t)\|$. However, one question remains: At finite observable times, how close is $\gamma_{t_1}(t)$ to γ? Although the slope appears to

smoothly converge to some value, this smooth regime could be altered later on, and the slope could converge to a completely different value. Thus the observed $\gamma_{t_1}(t)$ at large finite times t might be very different than its limit.

There is one famous example where $-1/T_c$ is *not* approximatively equal to the slope at finite times. On the complete graph, take the simplest gossip algorithm: standard gossip, also called pairwise gossip (see Paragraph 2.2.2). In this algorithm, at any iteration, a random pair of estimates is averaged. Suppose now that there are 2^k nodes. Then, there is at least one special arrangement of matrices that computes exactly x_{ave} in finite time. In other words, with a finite number of iterations, $\boldsymbol{\epsilon}(t) = 0$.

If $k = 1$, this statement is obvious, since one iteration is sufficient to converge exactly. If $k = 2$, there are four nodes A, B, C and D. If, at any point in the algorithm, pairs of nodes that average their values are chosen in the following order: $(A, B), (C, D), (A, C), (B, D)$, then, after this sequence of averages, $\boldsymbol{\epsilon} = 0$. Since this sequence appears with positive probability, it eventually happens with probability 1. Therefore, with probability 1, $\gamma = -\infty$ and $T_c = 0$.

If k is large enough, this phenomenon does not appear in finite time, because it is too rare. Moreover, on simulations, $\frac{1}{t}\log\|\boldsymbol{\epsilon}(t)\|$ seems to concentrate on some value, which is bounded.

Description of the simulations: Pairwise gossip was run on complete graphs of n nodes with Matlab. Because of Matlab rounding errors, pairwise gossip should run on a limited number of iterations. For $n = 4$, exact convergence is observable in finite time. Fig. 3.2 shows the empirical distribution of the exact convergence time, and Fig. 3.3(a) shows the empirical distribution of the slope coefficient of the samples that have not converged yet. This figure exhibits a slope trend that is stable with time and that tends to concentrate. The transition between the smooth error decrease and the exact averaging is sudden. For $n = 8$, the exact convergence phenomenon is already too rare to be observed before the surrender of Matlab. Over 5000 samples, none of them has converged in the simulation time, although they are all bound to converge exactly in finite time. As shown on Fig. 3.3(b), the slopes concentrate as well.

If n is not a power of 2 ($n \neq 2^k$), then it is easy to show that pairwise gossip cannot converge in finite time for any initial signal. Indeed, at any time t, $\mathbf{x}(t) = \mathbf{P}(t)\mathbf{x}(0)$, where $\mathbf{P}(t)$ is product of pairwise gossip matrices. By induction, one can show that any coefficient of the matrix $\mathbf{P}(t)$ has the form:

$$\mathbf{P}_{ij}(t) = \frac{a_{ij}}{2^{b_{ij}}},$$

for some integers $a_{ij} \geqslant 0$ and $b_{ij} > 0$. Therefore, there exists a time t such that $\mathbf{P}(t) = \mathbf{J}$ only if there are integers a and b such that

$$\frac{1}{n} = \frac{a}{2^b}, \tag{3.65}$$

which implies that $an = 2^b$. Hence there exists a time t such that $\mathbf{P}(t) = \mathbf{J}$ only if n divides 2^b, i.e. only if n is a power of 2.

Therefore, if $n = 5$, pairwise gossip cannot converge in finite time, but rigorously speaking, this argument is not sufficient to claim that $-1/T_c$ is equal to the observed slope (see Fig. 3.4(a) for the empirical distribution of the slope when $n = 5$). Indeed some other rare phenomenon we have not thought about

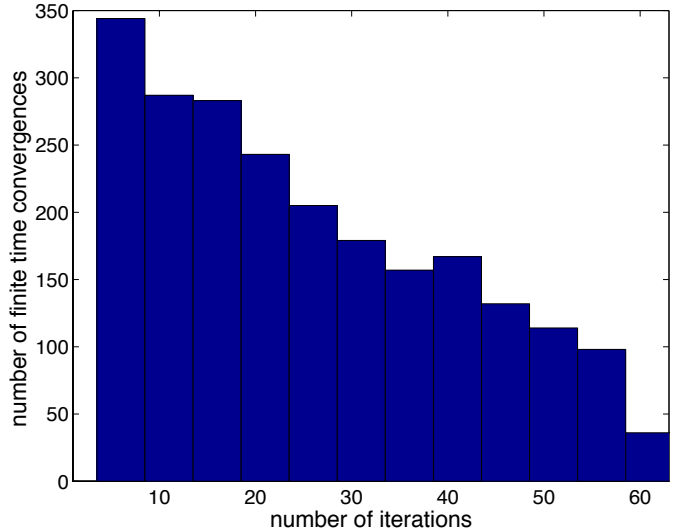


Figure 3.2: $n = 4$. Over 3000 simulations, histogram of the convergence time of simulations that terminated in finite time.

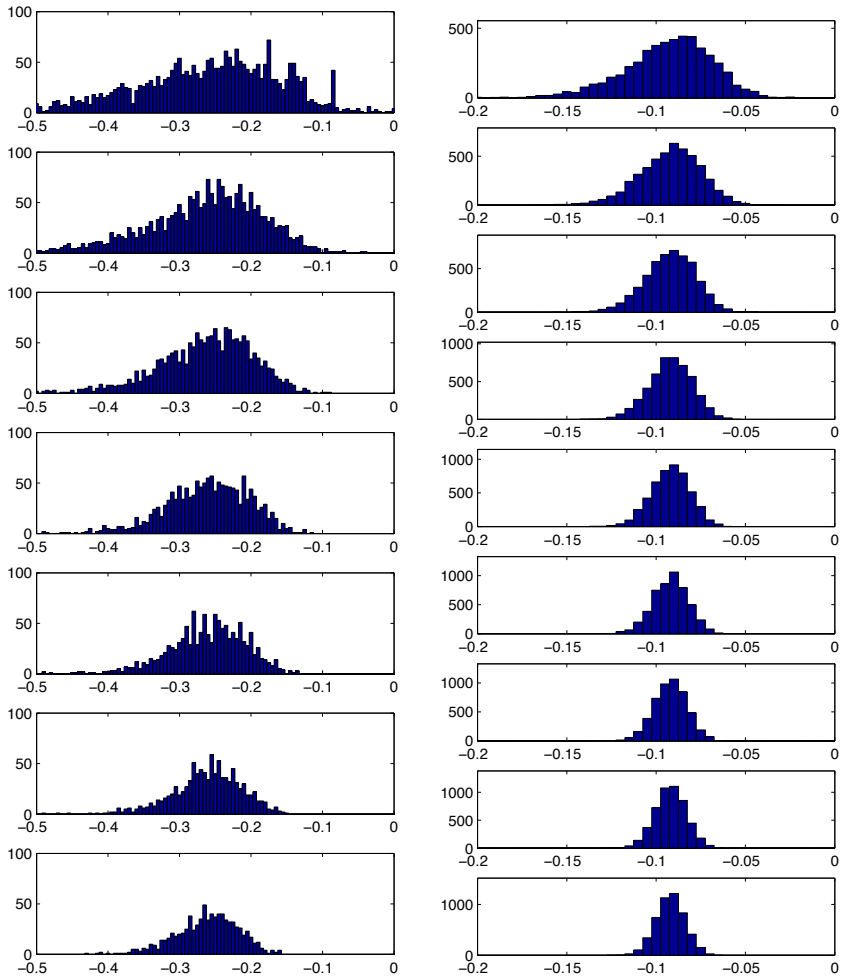
could happen, and disturb the convergence of the slope. For larger networks, the algorithm is slower and it can be run for a longer time. Despite the long running time, the simulations show the same trend, even when n is a power of 2 (for $n = 256 = 2^8$, see Fig. 3.4(b) and 3.5).

In [31], using Azuma's inequality, the authors prove that the error $\boldsymbol{\eta}(t)$ concentrates around its expected behavior for small times. It is delicate to specify exactly how small the concentration time is, but this result explains the concentration observations in the simulations when $n = 2^k$ is a power of 2 and when the connectivity graph is complete. Nevertheless, one question remains open: when $n \neq 2^k$, does the slope stay concentrated on the same value forever, beyond the range of Azuma's inequality? In other words, in pairwise gossip, if $n \neq 2^k$, is $-1/T_c$ equal to the slope that is observed at finite times? What can be said about the other algorithms?

Bounding the contraction rate γ

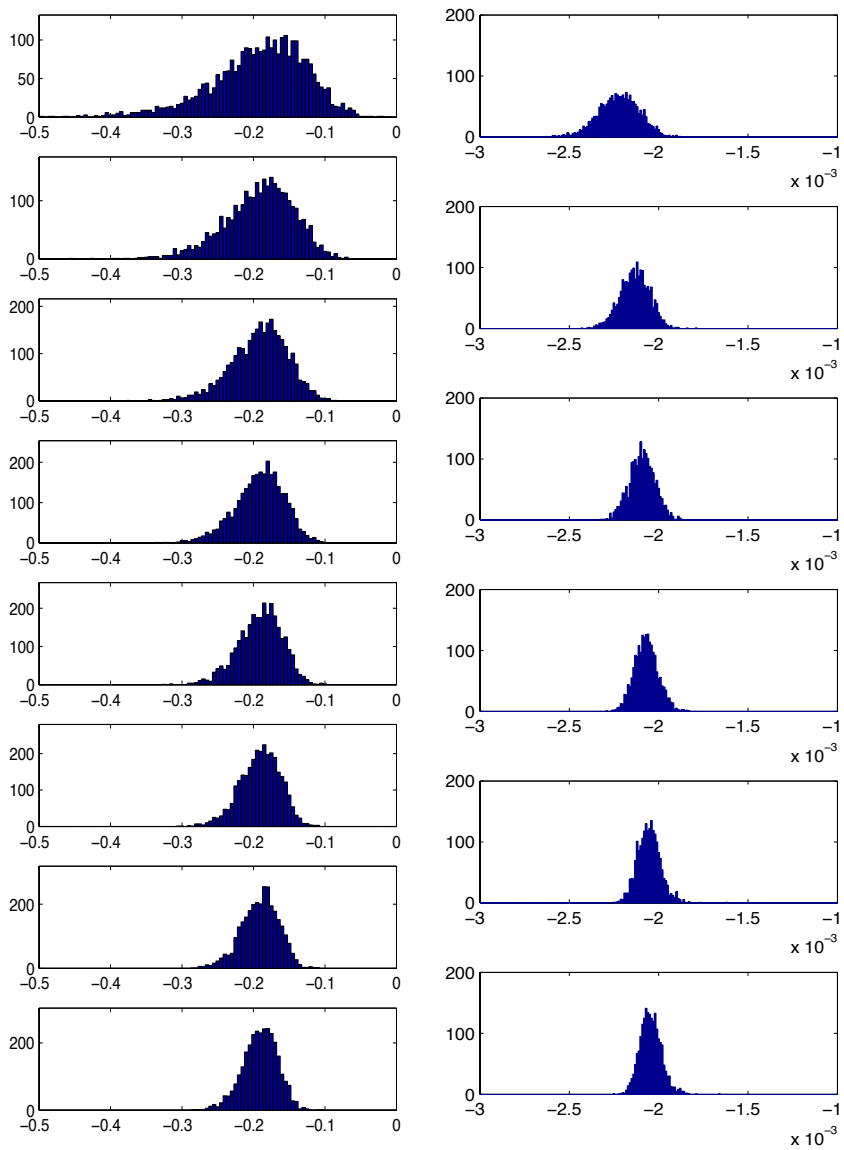
Theorems 3.2.1 and 3.2.2 prove the existence of a deterministic asymptotic contraction rate γ, and of consensus time $T_c := -1/\gamma$, but do not provide any method to compute them, other than evaluating the limit (3.62) or (3.63). Whether the limits γ and T_c are reached rapidly or not, it is interesting to bound these values. Remember that

$$\begin{aligned} \gamma &= \lim_{t\to\infty} \frac{1}{t}\mathbb{E}\left[\frac{1}{2}\log\|\mathbf{M}_t\|^2\right] \\ &\leqslant \lim_{t\to\infty} \frac{1}{2t}\log\mathbb{E}\left[\|\mathbf{M}_t\|^2\right], \end{aligned}$$



(a) $n = 4$ (b) $n = 8$

Figure 3.3: The evolution of the concentration of $\log(\|\boldsymbol{\epsilon}(t)\|_1 / \|\boldsymbol{\epsilon}(0)\|_1)/t$ for pairwise gossip on a complete graph. (3.3(a)) $n = 4$, the number of samples of value $-\infty$ is not shown. One snapshot is displayed every 8 iterations. (3.3(b)) $k = 3$, $n = 8$. Pairwise gossip was run 5000 times. One snapshot is displayed every 30 iterations. Larger time simulations fail because of Matlab precision limit.



(a) $n = 5$ (b) $n = 256$

Figure 3.4: The evolution of the concentration of $\log(\|\boldsymbol{\epsilon}(t)\|_1 / \|\boldsymbol{\epsilon}(0)\|_1)/t$ for pairwise gossip on a complete graph. (3.4(a)) $n = 5$. One snapshot is displayed every 15 iterations. (3.4(b)) $k = 8$, $n = 256$.Pairwise gossip was run 2000 times. One snapshot is displayed every 700 iterations.

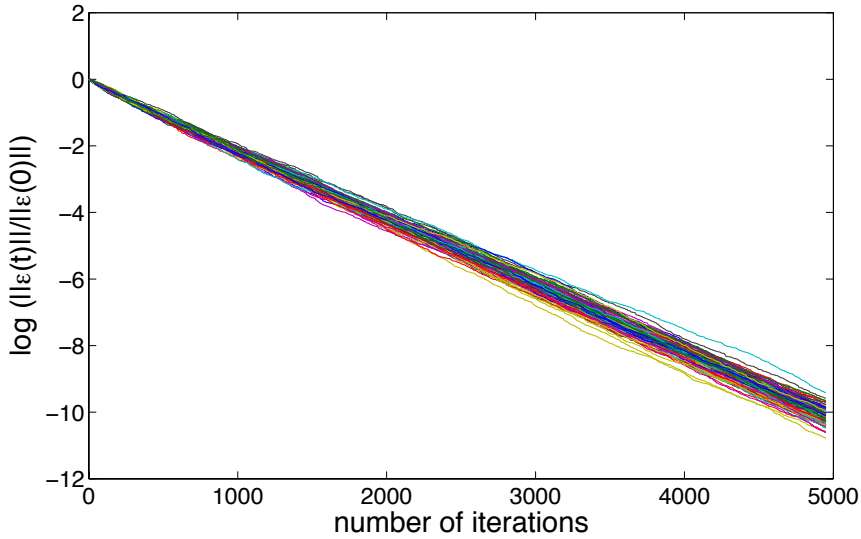


Figure 3.5: $n = 256$ nodes. error in a logarithmic scale. 100 simulations

where the inequality was obtained with Jensen's inequality. Even if $\gamma = -\infty$ because of some rare event that is not observed in finite time, this bound makes practical sense since it focuses on the expected behavior of the error norm. As explained in the previous paragraph, the behavior of the algorithm concentrates around its expected behavior. Bounding it is therefore a sensible approach to describe the behavior in finite times.

The upper bounds on $\gamma = -1/T_c$ described below are valid under the stronger condition that the sequence of averaging matrices $\{\mathbf{W}(t)\}_{t\geqslant 1}$ is i.i.d. The theorems are applied with $\mathbf{A}(t) = \mathbf{W}(t) - \mathbf{J}$, so that:

$$\gamma = \lim_{t\to\infty} \frac{1}{t} \log\left(\frac{\|\boldsymbol{\epsilon}(t)\|}{\|\boldsymbol{\epsilon}(0)\|}\right).$$

A first bound $B_1 := 0.5 \log(\rho(\mathbb{E}[(\mathbf{W} - \mathbf{J}) \otimes (\mathbf{W} - \mathbf{J})]))$.

Theorem* 3.2.3. Assume that $\{\mathbf{W}(t)\}_{t\geq 0}$ is i.i.d. in $\mathbb{R}^{n\times n}$, and satisfies the conditions for almost sure convergence stated in Theorem 3.1.3. Assume moreover that $\mathbf{1}^T\mathbf{W}(t) = \mathbf{1}^T$ for all times t. Let γ be the contraction rate:

$$\gamma = \lim_{t\to\infty} \frac{1}{t} \log\left\|\prod_{k=1}^{t-1}(\mathbf{W}(t-k) - \mathbf{J})\right\|.$$

Then, γ is bounded from above

$$\gamma \leq B_1 := \frac{1}{2} \log \rho\left(\mathbb{E}[(\mathbf{W}(0) - \mathbf{J}) \otimes (\mathbf{W}(0) - \mathbf{J})]\right),$$

where ρ denotes the spectral radius, and $\otimes$ the Kronecker product.

The proof of this theorem can be found in page 64.

A looser but simpler bound $B_2 := 0.5\log(\lambda_2(\mathbb{E}[\mathbf{W}^T\mathbf{W}]))$.
This bound is strongly inspired by the sufficient conditions for mean square convergence.

Theorem* 3.2.4. Under the same assumptions as in Theorem 3.2.3, then γ is bounded from above

$$\gamma \leq B_2 := \frac{1}{2}\log\lambda_2\left(\mathbb{E}[\mathbf{W}(0)^T\mathbf{W}(0)]\right),$$

where λ_2 denotes the second largest eigenvalue.

The proof of this Theorem can be found in page 66.

Bounds when $\mathbf{1}^T\mathbf{W}(t) \neq \mathbf{1}^T$ for some time t.
Since $\boldsymbol{\eta}(t) = (\mathbf{I}-\mathbf{J})\mathbf{W}(t)\boldsymbol{\eta}(t-1)$, the same theorems as before can be stated, with matrices $(\mathbf{I}-\mathbf{J})\mathbf{W}(t)$ instead of $\mathbf{W}(t)-\mathbf{J}$. Note that Theorems 3.2.3 and 3.2.4 are sub-cases of the following more general result:

Theorem* 3.2.5. Under the same assumptions as in Theorem 3.2.3 except for $\mathbf{1}^T\mathbf{W}(t) \neq \mathbf{1}^T$. Let γ be the contraction rate:

$$\gamma = \lim_{t\to\infty}\frac{1}{t}\log\left\|\prod_{k=1}^{t-1}(\mathbf{I}-\mathbf{J})\mathbf{W}(t-k)\right\|,$$

so that there is a random proper subspace V such that for any $\boldsymbol{\eta}(0) = (\mathbf{I}-\mathbf{J})\mathbf{x}(0) \in \mathbb{R}^n \setminus V$,

$$\lim_{t\to\infty}\frac{1}{t}\log\|\boldsymbol{\eta}(t)\| = \gamma.$$

Then, γ is bounded from above by:

$$\gamma \leq B_1' = \frac{1}{2}\log\rho\left(\mathbb{E}[(\mathbf{I}-\mathbf{J})\mathbf{W}(0)\otimes(\mathbf{I}-\mathbf{J})\mathbf{W}(0)]\right),$$

and by:

$$\gamma \leq B_2' = \frac{1}{2}\log\rho\left(\mathbb{E}[\mathbf{W}(0)^T(\mathbf{I}-\mathbf{J})\mathbf{W}(0)]\right).$$

Proofs of the Theorems

Proof:(Theorem 3.2.3) Theorem 3.2.1 gives the choice of the submultiplicative matrix norm and the Frobenius norm is the one chosen for this proof. Denote by m_{ij} the i-th row, j-th column element of $\mathbf{M}_t = \prod_{k=0}^{t-1}\mathbf{A}(t-k)$, where $\mathbf{A}(t) = (\mathbf{I}-\mathbf{J})\mathbf{W}(t)$. Then, by Jensen's inequality (3.66),

$$\begin{aligned}\gamma &= \lim_{t\to\infty}\frac{1}{t}\mathbb{E}\left[\frac{1}{2}\log\|\mathbf{M}_t\|_F^2\right]\\ &\leq \lim_{t\to\infty}\frac{1}{2t}\log\mathbb{E}\left[\|\mathbf{M}_t\|_F^2\right] \qquad (3.66)\\ &= \lim_{t\to\infty}\frac{1}{2t}\log\mathbb{E}\left[\sum_{i=1}^{n}\sum_{j=1}^{n}m_{ij}^2(t)\right].\end{aligned}$$

Now, express $\|\mathbf{M}_t\|_F^2$ as a function $f(\cdot)$ of another matrix, namely $\mathbf{M}_t \otimes \mathbf{M}_t$. Formally, f is thus defined as

$$f : \mathbb{R}^{n^2 \times n^2} \to \mathbb{R} : \mathbf{B} \mapsto \sum_{i=1}^{n} \sum_{j=1}^{n} b_{i+n(i-1),j+n(j-1)} \tag{3.67}$$

where $b_{k,l}$ is the k-th row, l-th column element of $\mathbf{B}$. In other words, $f(\mathbf{B})$ sums n^2 specially chosen elements of the $n^2 \times n^2$ matrix $\mathbf{B}$. Besides the equality

$$f(\mathbf{M} \otimes \mathbf{M}) = \|\mathbf{M}\|_F^2 \,, \tag{3.68}$$

this function has the desirable property of being linear:

$$f\left(\alpha \mathbf{A} + \beta \mathbf{B}\right) = \alpha f(\mathbf{A}) + \beta f(\mathbf{B}), \tag{3.69}$$

where $\mathbf{M}$ is an $n \times n$ matrix, and $\mathbf{A}$ and $\mathbf{B}$ are $n^2 \times n^2$ matrices. Going back to Eq. (3.66), replace $\|\mathbf{M}\|_F^2$ according to (3.68). Then, using property (3.69), invoke linearity of expectation to interchange $f(\cdot)$ and $\mathbb{E}[\cdot]$:

$$\begin{aligned}
\gamma &\leq \lim_{t\to\infty} \frac{1}{2t} \log \mathbb{E}\left[f(\mathbf{M}_t \otimes \mathbf{M}_t)\right] \\
&= \lim_{t\to\infty} \frac{1}{2t} \log f\left(\mathbb{E}\left[\mathbf{M}_t \otimes \mathbf{M}_t\right]\right) \\
&= \lim_{t\to\infty} \frac{1}{2t} \log f\left(\mathbb{E}\left[\prod_{p=0}^{t-1} \mathbf{A}(t-p) \otimes \prod_{p=0}^{t-1} \mathbf{A}(t-p)\right]\right) \\
&= \lim_{t\to\infty} \frac{1}{2t} \log f\left(\mathbb{E}\left[\prod_{p=0}^{t-1} \Big(\mathbf{A}(t-p) \otimes \mathbf{A}(t-p)\Big)\right]\right).
\end{aligned}$$

Note that the distributive property of the Kronecker product was used: $(\mathbf{AC}) \otimes (\mathbf{BD}) = (\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D})$. Now use the i.i.d. property of the matrices $\mathbf{A}(t)$ to interchange the matrix product and expectation

$$\begin{aligned}
\gamma \quad &\leq \quad \lim_{t\to\infty} \frac{1}{2t} \log f\left(\prod_{p=0}^{t-1} \mathbb{E}\big[\mathbf{A}(t-p) \otimes \mathbf{A}(t-p)\big]\right) \\
&= \quad \lim_{t\to\infty} \frac{1}{2t} \log f\big(\mathbb{E}^t\left[\mathbf{A}(0) \otimes \mathbf{A}(0)\right]\big).
\end{aligned}$$

To simplify the notation in the following, define $\mathbf{B} := \mathbb{E}[\mathbf{A}(0) \otimes \mathbf{A}(0)]$. Using linearity of f once again,

$$\begin{aligned}
\gamma &\leq \lim_{t\to\infty} \frac{1}{2t} \log f\left(\mathbf{B}^t\right) \\
&= \lim_{t\to\infty} \frac{1}{2t} \left(\log \left\|\mathbf{B}^t\right\| + \log f\left(\frac{\mathbf{B}^t}{\|\mathbf{B}^t\|}\right)\right) \\
&= \lim_{t\to\infty} \left(\frac{1}{2} \log \left\|\mathbf{B}^t\right\|^{\frac{1}{t}} + \frac{1}{2t} \log f\left(\frac{\mathbf{B}^t}{\|\mathbf{B}^t\|}\right)\right).
\end{aligned}$$

It is a well known fact (Gelfand's formula, see e.g. [45]) that for any p-norm, $p \in \{1, 2, \ldots, \infty\}$,

$$\lim_{t\to\infty} \left\|\mathbf{B}^t\right\|_p^{\frac{1}{t}} = \rho(\mathbf{B}) := \max\{|\lambda| : \lambda \in \lambda(\mathbf{B})\}.$$

To bound the last term, choose $p = \infty$, which yields to $f\left(\frac{\mathbf{B}^t}{\|\mathbf{B}^t\|_\infty}\right) \leq n^2$. The following concludes the proof:

$$\begin{aligned}
\gamma &\leq \lim_{t\to\infty}\left(\frac{1}{2}\log\left\|\mathbf{B}^t\right\|^{\frac{1}{t}} + \frac{1}{2t}\log n^2\right) \\
&= \frac{1}{2}\log\lim_{t\to\infty}\left\|\mathbf{B}^t\right\|^{\frac{1}{t}} + 0 \\
&= \frac{1}{2}\log\rho(\mathbf{B}).
\end{aligned}$$

■

Proof:(Theorem 3.2.4) Let $\boldsymbol{\epsilon}(t) = \mathbf{x}(t) - x_{ave}\mathbf{1}$ and $\mathbf{A}(t) = (\mathbf{I} - \mathbf{J})\mathbf{W}(t)$. For any nonrandom choice of $\boldsymbol{\epsilon}(0) \in \mathbb{R}^n \setminus \{0\}$,

$$\begin{aligned}
&\mathbb{E}\left[\boldsymbol{\epsilon}^T(t)\boldsymbol{\epsilon}(t)\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\boldsymbol{\epsilon}^T(t-1)\mathbf{A}^T(t)\mathbf{A}(t)\boldsymbol{\epsilon}(t-1)|\boldsymbol{\epsilon}(t-1)\right]\right] \\
&= \mathbb{E}\left[\boldsymbol{\epsilon}^T(t-1)\mathbb{E}\left[\mathbf{A}^T(t)\mathbf{A}(t)\right]\boldsymbol{\epsilon}(t-1)\right] \\
&\leq \lambda_1^t\left(\mathbb{E}\left[\mathbf{A}^T(0)\mathbf{A}(0)\right]\right)\boldsymbol{\epsilon}^T(0)\boldsymbol{\epsilon}(0).
\end{aligned}$$

Then, by Jensen's inequality,

$$\begin{aligned}
\mathbb{E}\frac{1}{2t}\left[\log\left(\frac{\boldsymbol{\epsilon}^T(t)\boldsymbol{\epsilon}(t)}{\boldsymbol{\epsilon}^T(0)\boldsymbol{\epsilon}(0)}\right)\right] &\leq \frac{1}{2t}\log\left(\frac{\mathbb{E}\left[\boldsymbol{\epsilon}^T(t)\boldsymbol{\epsilon}(t)\right]}{\boldsymbol{\epsilon}^T(0)\boldsymbol{\epsilon}(0)}\right) \\
&\leq \frac{1}{2}\log\lambda_1\left(\mathbb{E}\left[\mathbf{A}^T(0)\mathbf{A}(0)\right]\right).
\end{aligned}$$

Now, choose $\boldsymbol{\epsilon}(0) = \hat{\boldsymbol{\epsilon}}(0)$ such that $\mathbb{P}\left(\hat{\boldsymbol{\epsilon}}(0) \in \mathbb{R}^n \setminus V\right) = 1$, with V defined as in Theorem 3.2.2. For such a $\hat{\boldsymbol{\epsilon}}(0)$, $\lim_{t\to\infty}\frac{1}{t}\log\frac{\|\hat{\boldsymbol{\epsilon}}(t)\|_2}{\|\hat{\boldsymbol{\epsilon}}(0)\|_2} = \gamma$ with probability 1. Consequently, there is a bounded C such that $\frac{1}{t}\log\frac{\|\hat{\boldsymbol{\epsilon}}(t)\|_2}{\|\hat{\boldsymbol{\epsilon}}(0)\|_2} < C$ for all t. Then, by dominated convergence,

$$\begin{aligned}
\gamma &= \mathbb{E}\lim_{t\to\infty}\frac{1}{t}\log\frac{\|\hat{\boldsymbol{\epsilon}}(t)\|_2}{\|\hat{\boldsymbol{\epsilon}}(0)\|_2} \\
&= \lim_{t\to\infty}\mathbb{E}\frac{1}{t}\log\frac{\|\hat{\boldsymbol{\epsilon}}(t)\|_2}{\|\hat{\boldsymbol{\epsilon}}(0)\|_2} \\
&\leq \frac{1}{2}\log\lambda_1\left(\mathbb{E}[\mathbf{A}^T(0)\mathbf{A}(0)]\right).
\end{aligned}$$

■

Confronting the bounds to simulation results Theorem 3.2.1 showed that

$$\gamma = \lim_{t\to\infty} \frac{1}{t}\mathbb{E}\left[\log \|\mathbf{M}_t\|\right].$$

Since γ is a limit, it cannot be computed on simulations. Therefore, instead of comparing γ to its bounds B_1 and B_2, these bounds are compared to:

$$\gamma(t) = \frac{1}{t}\left\langle \log \|\mathbf{M}_t\| \right\rangle,$$

for some finite time t, where $\langle\cdot\rangle$ denotes the average of empirical samples. Note that $\gamma(t)$ is different from the empirical convergence rate $\gamma_{t_1}(t)$ since it measures the norm of *matrices* whereas $\gamma_{t_1}(t)$ focuses on the *vector* of estimates. The simulations are shown in Fig. 3.6 and 3.7. On these figures, a third value is also shown, called C_3, which is equal to $\log(\lambda_2(\mathbb{E}[\mathbf{W}]))$. C_3 appears very often in the literature [15, 27, 9], and it is therefore interesting to see if it is an accurate approximation of the convergence rate.

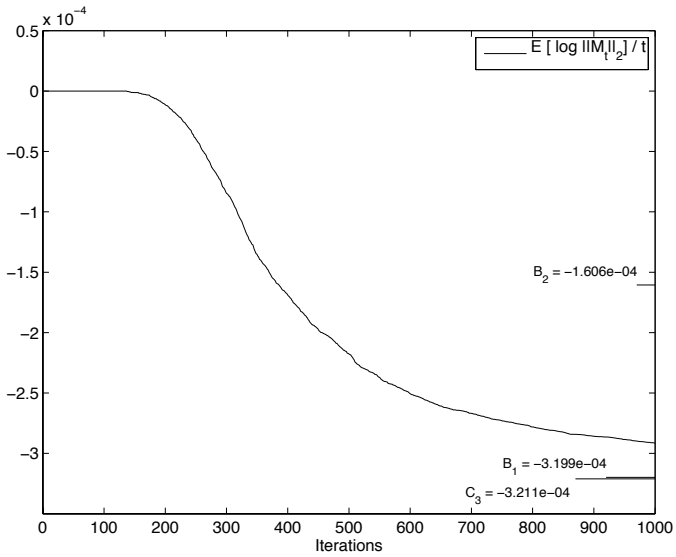
Two algorithms were tested: pairwise gossip on fixed graphs and Metropolis weights on graphs presenting i.i.d. link failures. Both algorithms were run for 1000 iterations on the complete graph of 60 nodes and on an RGG of 60 nodes and connection radius $r = 0.2$. Several interesting observations can be made. First, Metropolis weights is a synchronous algorithm, and the slope of its logarithmic error stabilizes with fewer iterations than gossip. Second, B_1 is indeed a tighter bound than B_2 (there is no formal proof of this claim). Third, C_3 is a good approximation of γ when the network is an RGG, whereas for the fully connected graph, C_3 is far from γ but B_2 becomes a better approximation

One should be careful not to jump to conclusions. The following example shows how tricky the relationship between γ and C_3 is. Consider random geometric graphs (RGGs) with increasing number of nodes n, and link radius $r(n)$ decreasing accordingly, so as to preserve connectivity when scaling the graph. Using the scaling law $r = \sqrt{\alpha \log(n)/n}$, consider two different values of α, namely $\alpha = 1$ and $\alpha = 2$. For better readability, Fig. 3.8 shows the contraction time $-1/\gamma(t)$ and $-1/C_3$, instead of the contraction rate $\gamma(t)$ itself, both when $\alpha = 1$ (Fig. 3.8(a)) and $\alpha = 2$ (Fig. 3.8(b)).

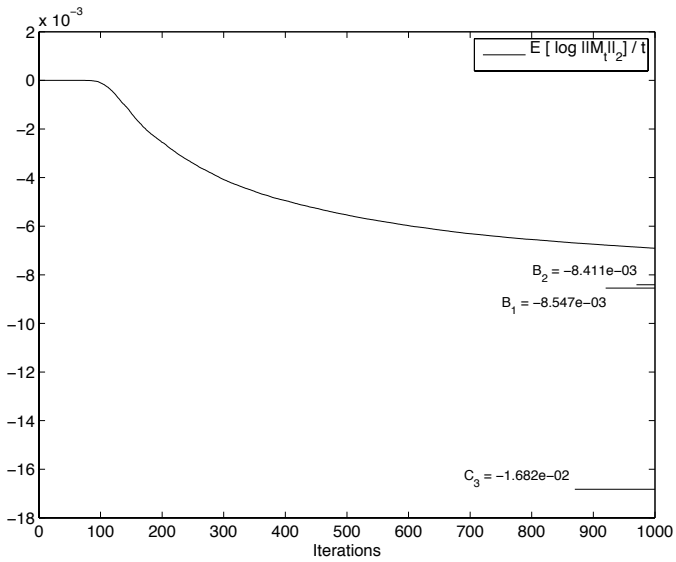
The two pictures are radically different, even though the same order of scaling was used in both cases, except for a multiplicative coefficient. For $\alpha = 1$ the contraction time T_c and $-1/C_3$ remain close together, and the according relative error even tends to vanish. On the other hand, if $\alpha = 2$, C_3 is no longer a valid approximation for the contraction rate γ when n is large.

As a consequence, even though intuition indicates that there must, or should, be some relationship between γ and C_3, it is obviously a very complicated one. Even in very similar special cases, C_3 and γ may in one case be very close, and in the second case behave radically differently. Therefore, at this point C_3 cannot be used to approximate the contraction rate.

As a last remark, one can think of a way of explaining the behavior of C_3. When (and if!) both B_1 and C_3 are asymptotically tight, a possible argument to prove tightness of one of these values, given tightness of the other, could be to show that both converge to the same value. In order to couple B_1 and C_3, define the autocovariance of a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ as the $n^2 \times n^2$ matrix with

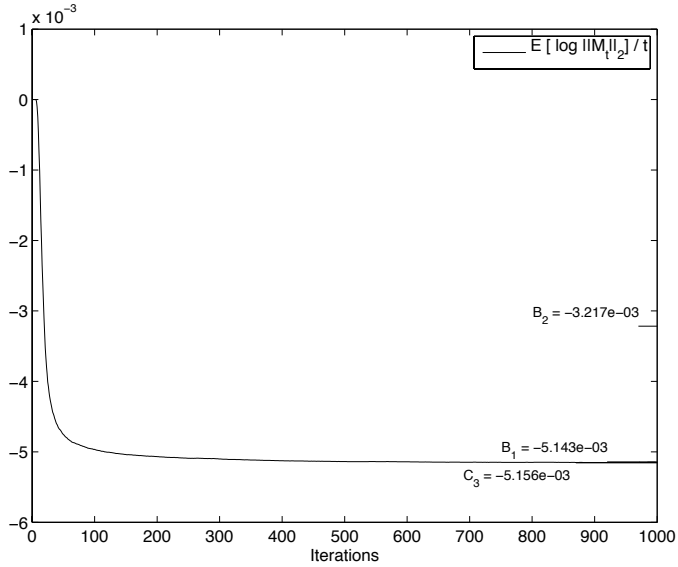


(a) RGG

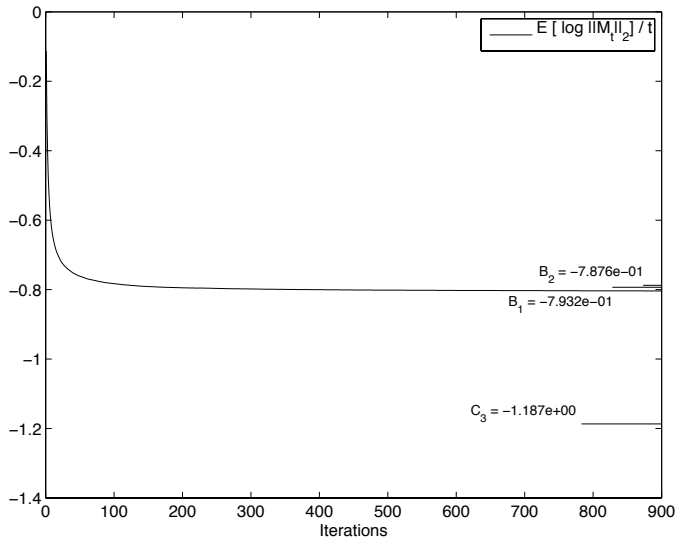


(b) Complete graph

Figure 3.6: Mean logarithmic error slope $\log(|||M_t|||_2)/t$ when running pairwise gossip.

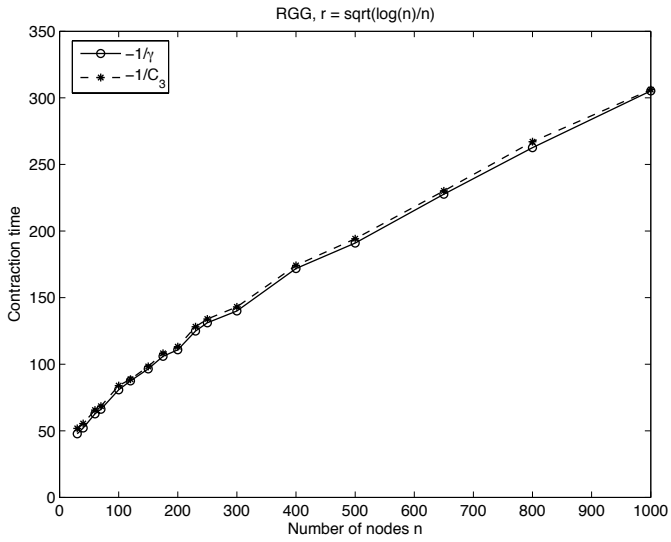


(a) RGG

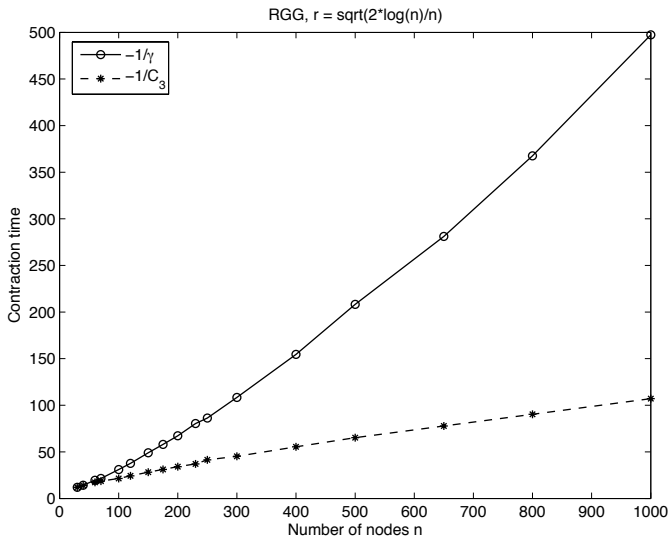


(b) Complete graph

Figure 3.7: Mean logarithmic error slope $\log(|||M_t|||_2)/t$ when running Metropolis weights.



(a) $\alpha = 1$



(b) $\alpha = 2$

Figure 3.8: Contraction Time T_c and its "approximation" based on C_3, as a function of network size n, using Metropolis weights. The graphs are random geometric graphs where the connection radius scales as $r(n) = \sqrt{\alpha \log n / n}$.

elements

$$(\mathrm{cov}\,(\mathbf{A}))_{(i-1)n+k,(j-1)n+l} = \mathrm{cov}\,(a_{i,j}, a_{k,l})\,. \tag{3.70}$$

Then, B_1 can be rewritten as (with $\mathbf{A} = \mathbf{W} - \mathbf{J}$):

$$B_1 = \frac{1}{2}\log\rho\left(\mathbb{E}\left[\mathbf{A}\otimes\mathbf{A}\right]\right) = \frac{1}{2}\log\rho\left(\mathbb{E}\left[\mathbf{A}\right]\otimes\mathbb{E}\left[\mathbf{A}\right] + \mathrm{cov}(\mathbf{A})\right). \tag{3.71}$$

Now, if $\mathrm{cov}(\mathbf{A})$ were 0, then B_1 and C_3 are equal:

$$\frac{1}{2}\log\rho\left(\mathbb{E}\left[\mathbf{A}\right]\otimes\mathbb{E}\left[\mathbf{A}\right]\right) = \frac{1}{2}\log\rho^2\left(\mathbb{E}\left[\mathbf{A}\right]\right) \tag{3.72}$$

$$= \log\rho\left(\mathbb{E}\left[\mathbf{A}\right]\right) \tag{3.73}$$

$$= C_3. \tag{3.74}$$

Therefore, $\mathrm{cov}(\mathbf{A})$ might be regarded as a perturbation term. If $\mathrm{cov}(\mathbf{A})$ is somehow "small" compared to $\mathbb{E}\left[\mathbf{A}\right]\otimes\mathbb{E}\left[\mathbf{A}\right]$, chances are B_1 and C_3 approach each other. However, "small" cannot refer to the magnitude of the entries of $\mathbf{A}$, as these do not vanish for large n. On the other hand, in an RGG, the proportion of nonzero elements in $\mathrm{cov}(\mathbf{A})$ vanishes. But this is not sufficient to prove $\rho\left(\mathbb{E}\left[\mathbf{A}\otimes\mathbf{A}\right]\right) \to \rho\left(\mathbb{E}\left[\mathbf{A}\right]\otimes\mathbb{E}\left[\mathbf{A}\right]\right)$ as n goes to infinity. Thus, it remains to be shown when and whether B_1 and C_3 converge to the same value.

However this remark suggests that the connectivity of the network is an important parameter. Fig. 3.9 studies how the bounds B_1 and B_2, as well as the slope and the value C_3, evolve as a function of graph connectivity. Pairwise gossip was run on an RGG of 30 nodes with an increasing number of edges.

To construct RGG's with a specific number of edges, first pick a sample with n nodes uniformly i.i.d. on the unit square. Then add connections between nodes, one after the other, beginning with an edge between the two nodes which are closest to each other. The next edge to be added is then always between the two nodes which are closest to each other, but not yet directly connected. The bounds are then evaluated when specific numbers of edges are reached, not necessarily after each added edge. To get Figure 3.9, this whole process was repeated 100 times, with randomly sampled node locations.

When the number of edges is too small, the graph is not connected, and therefore the sequence of matrices $\{\mathbf{A}(t)\}$ is not contracting, i.e. the Lyapunov exponent γ is equal to 0, as well as both upper bounds. Also the expectation matrix $\mathbb{E}\left[\mathbf{A}(t)\right]$ is not contracting as long as the graph is not connected, so $C_3 = 0$ as well.

Then, for an increasing number of edges and an increasing connectivity of the graph, the contraction rate $\gamma(t)$ for large time t decreases, and so do both upper bounds. In the beginning, $\gamma(t)$ and B_1 follow the evolution of C_3, but flatten out for very well connected graphs, where B_2 and C_3 drop very rapidly. Remember that in the gossip case, as represented here, $C_3 = 2B_2$. Consequently, for poorly connected graphs, B_2 is only about one-half of the actual contraction rate $\gamma(t)$, and therefore very loose. Observe that B_1 is always smaller than B_2, and is therefore tighter .

While B_2, and also C_3, would suggest a big improvement of the convergence speed when filling in the last connections missing in an almost completely connected graph, B_1 follows more precisely, although not exactly, the actual empirical contraction rate $\gamma(t)$. The gossip algorithm does not perform significantly

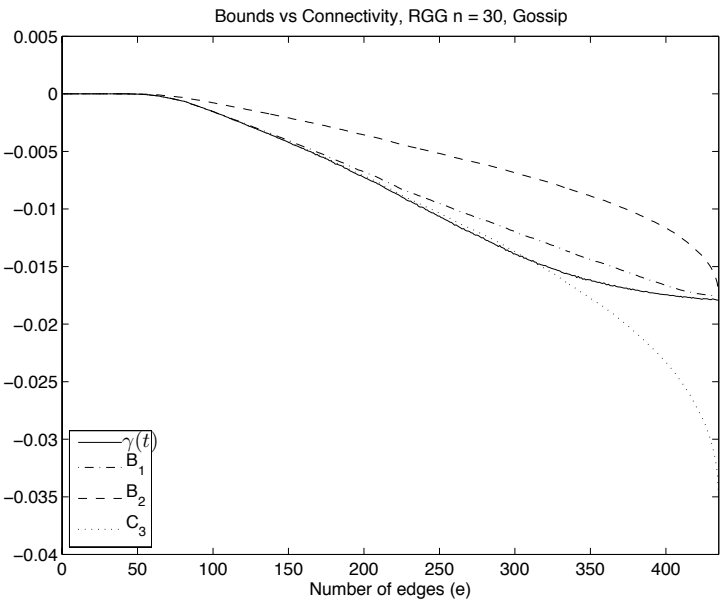


Figure 3.9: Empirical convergence rate for large times, bounds B_1, B_2 and approximation C_3 as a function of the total number of connections, with communications according to the gossip algorithm, for a graph with $n = 30$ nodes.

better on a complete graph than on graphs missing a few edge to complete.

Of the 3 quantities B_1, B_2 and C_3, the last one is easiest to compute and apparently the most accurate approximation of $\gamma(t)$, except for highly connected graphs. However, this is merely an empirical observation, and at this time there are no means to determine exactly where this approximation holds.

Finally, the picture is very similar in "failing links" scenarios using Metropolis weights (not represented). The qualitative evolution is the same as in Figure 3.9, but the gaps between the curves are much smaller.

3.2.3 Consensus cost C_c

Consensus time T_c evaluates the asymptotic number of iterations needed to decrease the error $\boldsymbol{\epsilon}$ by a factor e. Since, for large times t, $\gamma_{t_1}(t)$ does not depend too much on the initial signal and the particular instantiation of the algorithm, a few algorithm runs are sufficient to estimate the performance. However, the number of iterations is not necessarily the most interesting quantity to evaluate. Instead, delay, energy, power consumption or the number of communications might be the quantities of true interest.

- *Delay:* Let $D(t)$ be the duration of the iteration t and define the consensus delay D_c:

$$D_c = \lim_{t\to\infty} \frac{\sum_{k=1}^{t} D(k)}{-\log \|\mathbf{M}_t\|},$$

where $\mathbf{M}_t = \prod_{k=0}^{t-1}(\mathbf{I}-\mathbf{J})\mathbf{W}(t-k)$. Then by ergodicity,

$$\begin{aligned} D_c &= \lim_{t\to\infty} \frac{\sum_{k=1}^{t} D(k)}{t} \frac{t}{-\log \|\mathbf{M}_t\|} \\ &= \mathbb{E}[D]T_c. \end{aligned}$$

Both T_c and D_c measure time, but they do not necessarily scale equally. In a synchronous algorithm for example, all the nodes cannot send their estimates at the same time for interference reasons. Thus, D_c takes into account the delay created by the communication protocol during each iteration.

- *Transmitting energy:* Let $\mathcal{E}(t)$ be the energy spent to send messages in the tth iteration and define the emitting energy consensus cost $\mathcal{E}_c$:

$$\begin{aligned} \mathcal{E}_c &= \lim_{t\to\infty} \frac{\sum_{k=1}^{t} \mathcal{E}(k)}{-\log \|\mathbf{M}_t\|} \\ &= \mathbb{E}[\mathcal{E}]T_c. \end{aligned} \tag{3.75}$$

  In the simulations of this thesis, $\mathcal{E}(t)$ is assumed to be equal to the number of emitted messages.

- *Listening energy:* Similarly, let $\mathcal{M}(t)$ be the energy spent to receive messages in the tth iteration and define the listening consensus cost $\mathcal{M}_c$:

$$\mathcal{M}_c = \lim_{t\to\infty} \frac{\sum_{k=1}^{t} \mathcal{M}(k)}{-\log \|\mathbf{M}_t\|} \tag{3.76}$$

$$= \mathbb{E}[\mathcal{M}]T_c. \tag{3.77}$$

  In the simulations, the listening cost can be assumed to be equal to the number of received messages as a first approximation.

For any cost function $C(t)$ that is a function of $\mathbf{W}(t)$, where $\{\mathbf{W}(t)\}_{t\geqslant 1}$ is stationary and ergodic, one can define the associated consensus cost C_c, which is equal to $\mathbb{E}[C]T_c$. Then C_c is the asymptotic cost needed to reduce the error $\boldsymbol{\epsilon}$ by a factor e.

$\mathbf{T^{emp}}$ and $\mathbf{C^{emp}}$: tools to display empirical performance data. Consensus time T_c and consensus cost C_c, just as averaging time T_{ave}, are abstract notions that are suited for theoretical analysis, but that are not observable in practice. Yet, for many averaging algorithms, T_c, C_c and T_{ave} are not computable in most accurate graph models. Theoretical results often come in complete graphs, or lattices, or graphs embedded on the torus. There is therefore a need for an empirical way of evaluating the performance of algorithms in concrete settings.

Fagnani et al. [31] showed that, when the matrices $\{\mathbf{W}(t)\}_{t\geqslant 1}$ are i.i.d., large deviation theory guarantees the concentration of the typical error $\boldsymbol{\epsilon}(t)$ around its expectation for small enough times. Based on this result, we can argue that $\gamma_{t_1}(t)$ is a random variable of low variance, so that the samples of $\gamma_{t_1}(t)$ *concentrate*. We thus decide to display $\gamma_{t_1}(t)$ in our simulations. We define the empirical consensus time and the empirical consensus cost, which

will be extensively displayed in Chapter 5 on the performance of averaging algorithms.

Definition* 3.2.3. *Empirical consensus time* $T_{t_1}^{emp}$. Given t_1 and $t > t_1$, the empirical consensus time $T_{t_1}^{emp}$ is

$$T_{t_1}^{emp}(t) = -1/\gamma_{t_1}(t).$$

If, for a given t_1 and large times t, $T_{t_1}^{emp}(t)$ concentrates around a value $T_{t_1}^{emp}$ independent of time t and of the initial signal x_0, then the algorithm converges up to ϵ in order $\Theta(T_{t_1}^{emp} \log \epsilon^{-1})$ iterations.

Similarly we define the experimental consensus cost.

Definition* 3.2.4. *Empirical consensus cost* $C_{t_1}^{emp}$. Given t_1 and $t > t_1$, the empirical consensus cost $C_{t_1}^{emp}$ is

$$C_{t_1}^{emp}(t) = -\frac{C(t) - C(t_1)}{\log(\|\boldsymbol{\epsilon}(t)\|) - \log(\|\boldsymbol{\epsilon}(t_1)\|)}.$$

If $t \gg t_1$, and if $\{\mathbf{W}(t)\}_{t \geqslant 0}$ is i.i.d., then

$$\frac{C(t) - C(t_1)}{t - t_1} \simeq \mathbb{E}[C],$$

and $C_{t_1}^{emp}(t) \simeq \mathbb{E}[C] T_{t_1}^{emp}(t)$.

According to Theorem 3.2.2, if $\{\mathbf{W}(t)\}_{t \geqslant 0}$ is stationary and ergodic, then $T_{t_1}^{emp}(t)$ and $C_{t_1}^{emp}(t)$ converge to T_c and C_c when t grows.

How to proceed to display meaningful data: The concentration result in [31] holds some small times only, and it is not easy to know how long the result holds. It is thus important to run the algorithm several times, with different initial conditions, in order to collect a few samples of $T_{t_1}^{emp}(t)$ or $C_{t_1}^{emp}(t)$. If the samples concentrate, then they are valuable data, and their average is a good representative of the sample set. The best representation is to display a boxplot of the samples, showing the median, some quantiles, and the outliers of the sample set.

Then, it is worth choosing a large enough time t_1 so that the samples are not biased by the transition phase, where the algorithm usually converges faster. Finally, choose t as large as possible in order to increase the precision of the estimation. t is usually limited by the quantization refinement of the software on which the algorithm runs, more than by the expiration of the concentration result in [31].

Applications of T^{emp} and C^{emp}. The main application of the consensus cost metrics is a simple way to empirically compare algorithms in order to get an idea of the one that is the best suited to each specific network. Note that, thanks to the concentration of the empirical consensus cost samples, it is sufficient to run each algorithm a few times only to estimate the different consensus costs at finite times. For instance, measuring consensus time for different parameters of an algorithm is a good way to heuristically optimize the parameter when the optimization cannot be done analytically.

We give here an example of optimization problem that is not accessible in theory, but where empirical consensus time allows some interesting intuitive

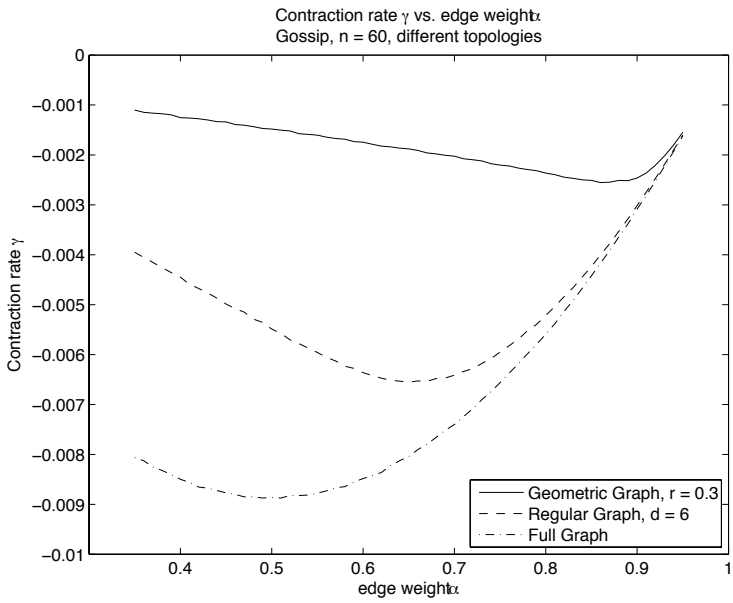


Figure 3.10: γ as a function of the message weight in gossip algorithm, for graphs with $n = 60$ nodes, different topologies.

analysis. The usual update strategy in pairwise gossip is that, when two nodes i and j exchange their current values $x_i(t)$ and $x_j(t)$, they both update their estimates of the global average to the local average of the two (see Section 2.2.2), i.e.

$$x_i(t+1) = x_j(t+1) = 0.5x_i(t) + 0.5x_j(t). \tag{3.78}$$

However, the above update equation is not necessarily the optimal pairwise gossip communication strategy. Instead of using update equation (3.78), consider the following generalization

$$x_i(t+1) = (1-\alpha)x_i(t) + \alpha\ x_j(t) \tag{3.79}$$

$$x_j(t+1) = \alpha\ x_i(t) + (1-\alpha)x_j(t), \tag{3.80}$$

where α is a positive constant denoted in the following as the *edge weight*, and is preferably in the interval $0 < \alpha < 1$. Consider that the edge weight is the same for all links and for all times, so that this generalization does not incur any additional communication costs when compared to the classical gossip setting, which corresponds to the special case $\alpha = 0.5$. From an analytical point of view, the only difference is that the update matrices $\mathbf{W}(t)$ are no longer projection matrices, so that the algorithm is not a gossip algorithm stricto sensu anymore. The fact that the edge weight does not change over time ensures that averaging process is i.i.d., which allows the use of the empirical contraction rate $\gamma_{t_1}^{emp}(t)$ to analyse the convergence speed.

Figure 3.10 shows the empirical contraction rate $\gamma_{t_1}^{emp}(t) = -1/T_{t_1}^{emp}(t)$ as a function of the edge weight α, for different network topologies. The considered

networks are:

1. Full graph (= complete graph): Simplest possible topology model, any node may contact any other node with equal probability.

2. Random geometric graph: Nodes are uniformly i.i.d. on the unit square. A node may communicate with any node within a range r, in the considered case $r = 0.3$.

3. Regular graph: Every node has exactly the same number of neighbors, but the graph has no geometric structure.

For the plot, the networks have size $n = 60$, and $\gamma_{t_1}^{emp}(t)$ was measured and averaged over 500 different graphs according to the corresponding structure, i.e. either random geometric graph (RGG) or regular graph. Averaging over graphs is not necessary for the full graph, as this type of graph has only one realization.

The optimal edge weight is in general *not* equal to 0.5, as can be seen in Figure 3.10. Especially when the graph has a strong geometric structure, i.e. two connected nodes share most of their neighbors, but there are few connections in total, the fastest averaging is achieved at values very close to one, in the considered example around $\alpha = 0.86$. Also in the case where the number of edges is limited but there is no geometric structure, the optimal edge weight is greater than 0.5, even though the effect is much weaker than in geometric graphs. In the example given with degree 6, the best contraction rate is achieved around $\alpha = 0.66$. Finally, if the pairs of nodes to communicate are drawn uniformly upon all pairs of nodes, the intuitive choice $\alpha = 0.5$ is the best strategy for averaging.

The idea behind this phenomenon is that when a network has a large diameter, a node should 'take over" the value provided by its neighbor, and "give away" its own value. Indeed, increasing α accelerates the "flow of information" in the network. In return, increasing α slows down the "averaging power". The local averaging is the most efficient at $\alpha = 0.5$, and the information flow is best when $\alpha = 1$. In the complete graph there is no need of information flow, so the optimum α is 0.5. But as the diameter of the network increases, it is reasonable to expect larger optimal edge weight α.

These interesting thoughts were allowed by the display of Fig. 3.10, which is based on the empirical consensus time. The bottleneck in the computation of Fig. 3.10 was not the complexity of the algorithms nor the complexity of the computation of $\gamma(t)$, but the construction of all the graphs. We believe that the computation of empirical consensus times can guide research to interesting problems on averaging algorithms.

Conclusion

In this chapter, we considered a number of theorems whose purpose is to prove the convergence of distributed averaging algorithms. We paid closer attention to the case of i.i.d. gossip matrices $\{\mathbf{W}(t)\}_{t\geqslant 1}$, which will prove useful in further chapters as we will refer to these results frequently.

We further studied how to estimate the convergence time of distributed averaging algorithms, with again a particular focus on i.i.d. gossip matrices $\{\mathbf{W}(t)\}_{t\geqslant 1}$. This analysis has demonstrated the importance of the second largest eigenvalue in magnitude of stochastic matrices, as this quantity rules the performance of most algorithms. We investigate this important notion in the following chapter.

SPEED OF CONVERGENCE

- There are two main measures for the speed of convergence: T_{ave} and T_c:
  - (ε, η)-averaging time $\mathbf{T_{ave}}(\mathbf{n}, \boldsymbol{\varepsilon}, \boldsymbol{\eta})$: For any $0 < \varepsilon < 1$ and any $0 < \eta < 1$,

$$T_{ave}(n, \varepsilon, \eta) = \sup_{\mathbf{x}(0)} \inf_t \mathbb{P}\left[\frac{\|\boldsymbol{\epsilon}(t)\|_2}{\|\boldsymbol{\epsilon}(0)\|_2} \geqslant \varepsilon\right] \leqslant \eta.$$

  - Consensus time $\mathbf{T_c}$: Whenever the limit:

$$T_c = \lim_{t\to\infty} \left|\frac{1}{t}\log\|\boldsymbol{\epsilon}(t)\|\right|^{-1}.$$

    exists, it is called consensus time. By Th. 3.2.2, the limit exists if $\{\mathbf{W}(t)\}_{t\geqslant 0}$ is a stationary and ergodic process.

- For any *gossip* algorithm, if $\{\mathbf{W}(t)\}_{t\geqslant 0}$ is i.i.d., then both T_{ave} and T_c can be bounded asymptotically in n. Take $\varepsilon = \beta/n$, if $\eta = 1/n^\alpha$ for some constants $\beta > 0$ and $\alpha > 0$,

$$\begin{aligned} T_{ave}(n, \varepsilon, \eta) &= \Theta\left(\frac{\log n}{1 - \lambda_2\left(\mathbb{E}[\mathbf{W}]\right)}\right), \\ T_c &\leqslant \frac{2}{1 - \lambda_2\left(\mathbb{E}[\mathbf{W}]\right)}, \end{aligned}$$

  where λ_2 denotes the second largest eigenvalue in magnitude (SLEM).

- Empirical performance evaluation: For any cost function $C(t)$ (time, energy, number of messages...), and given some times t_1 and $t > t_1$, the empirical consensus cost $C_{t_1}^{emp}$ at time t is defined as:

$$C_{t_1}^{emp}(t) = -\frac{C(t) - C(t_1)}{\log(\|\boldsymbol{\epsilon}(t)\|) - \log(\|\boldsymbol{\epsilon}(t_1)\|)}.$$

  If $\{\mathbf{W}(t)\}_{t\geqslant 0}$ is stationary and ergodic, then $C_{t_1}^{emp}(t)$ converges to $C_c = \mathbb{E}[C(1)]T_c$ when t grows.

  For accurate simulations, t_1 is taken larger than the transient phase in the decay of $\log(\|\boldsymbol{\epsilon}(t)\|)$. t is chosen as large as possible. Several samples are computed and displayed in a box plot, or represented by their average.

Chapter 4

Second Largest Eigenvalue in Magnitude

As the previous chapter has shown, the speed of convergence of distributed averaging algorithms is linked to the second largest eigenvalue in magnitude λ_2 of some matrix $\mathbf{P}$, which depends of the theorem to be used. If matrices $\{\mathbf{W}(t)\}_{t\geqslant 1}$ are i.i.d. and if the average is conserved through iterations, then, according to Theorem 3.2.2, one should compute $\lambda_2(\mathbb{E}[\mathbf{W}])$ to obtain a lower bound on T_{ave}. Theorem 3.2.1 shows that an upper bound is linked to $\lambda_2(\mathbb{E}[\mathbf{W}^T\mathbf{W}])$, which is equal to $\lambda_2(\mathbb{E}[\mathbf{W}])$ if the algorithm is a gossip algorithm. But if the average is not conserved, then the quantity of interest is the largest eigenvalue in magnitude of $\mathbb{E}[\mathbf{W}^T(\mathbf{I}-\mathbf{J})\mathbf{W}]$. Eigenvalues of matrices also appear in the upper bounds of the asymptotic contraction rate γ.

As eigenvalues are useful when studying distributed averaging algorithms, this chapter focuses on eigenvalue computation. The first section of this chapter recalls basic facts about the eigenvalues of stochastic matrices. In particular, all the eigenvalues are between -1 and 1. The second largest eigenvalue in magnitude (SLEM) should be the furthest away from -1 and from 1 so that the algorithm converges fast. Sections 4.2 and 4.3 show how close to -1 or 1 the SLEM tends to be, first on circles and on lattices, then on balanced trees. Circles illustrate the effect of cycles on the performance, whereas trees are emblematic of bottlenecks. Thus these two simple topologies, which allow precise computations, can explain intuitively why average diffusion is actually a slow process. Finally, Section 4.4 explains how the spectral properties of the network can be linked to the performance of the algorithms, through the study of the Laplacian matrix.

4.1 Symmetry, Primitivity and Stochasticity

4.1.1 Definitions and Perron-Frobenius Theorem

The matrices encountered in this thesis generally present a number of nice characteristics that help computing their SLEM λ_2. First, for all the algorithms, $\mathbf{W}(t)\mathbf{1} = \mathbf{1}$. In other words, 1 is an obvious eigenvalue of $\mathbf{W}$. Then, although it is not compulsory, most algorithms are designed with matrix elements $\mathbf{W}_{ij} \geqslant 0$,

which allows to apply all the theories existing around non-negative matrices; the most well-known theorem being the Perron-Frobenius theorem. Furthermore matrices can have other properties as being symmetric, stochastic or doubly-stochastic.

Definition 4.1.1 (Symmetric matrices). A matrix $\mathbf{P}$ is symmetric if and only if $\mathbf{P}^T = \mathbf{P}$.

Definition 4.1.2 (Semi-definite matrices). A matrix $\mathbf{P}$ is semi-definite positive if and only if it is symmetric and for any vector $\mathbf{x}$, $\mathbf{x}^T\mathbf{P}\mathbf{x} \geqslant 0$.

Theorem 4.1.1. The eigenvalues of a symmetric matrix are real and such a matrix admits an orthonormal set of eigenvectors. If furthermore, if $\mathbf{P}$ is semi-definite positive, then all the eigenvalues of $\mathbf{P}$ are non-negative.

This theorem applies to $\mathbb{E}[\mathbf{W}^T\mathbf{W}]$. Indeed, for any $\mathbf{W}$, $\mathbf{W}^T\mathbf{W}$ is symmetric, thus $\mathbb{E}[\mathbf{W}^T\mathbf{W}]$ is symmetric as well. Then, for any vector $\mathbf{x}$, $\mathbf{x}^T\mathbf{W}^T\mathbf{W}\mathbf{x} = \|\mathbf{W}\mathbf{x}\|_2^2 \geqslant 0$. Thus, $\mathbf{x}^T\mathbb{E}[\mathbf{W}^T\mathbf{W}]\mathbf{x} = \mathbb{E}[\mathbf{x}^T\mathbf{W}^T\mathbf{W}\mathbf{x}] \geqslant 0$. As a consequence, all the eigenvalues of $\mathbb{E}[\mathbf{W}^T\mathbf{W}]$ are real non-negative.

In addition note that for all the algorithms described in Chapter 2, $\mathbb{E}[\mathbf{W}]$ is a symmetric matrix.

In the general case, eigenvalues of symmetric $m \times m$ matrices can be negative. We will denote them by the letter ξ and index them in decreasing order:

$$\xi_1 \geqslant \xi_2 \geqslant \cdots \geqslant \xi_m \tag{4.1}$$

In this thesis, we are mostly interested in the magnitude of these eigenvalues. We thus define another indexing of the eigenvalues, which is decreasing in magnitude. In this new order, we denote the eigenvalues by the letter λ:

$$|\lambda_1| \geqslant |\lambda_2| \geqslant \cdots \geqslant |\lambda_m| \geqslant 0. \tag{4.2}$$

Definition 4.1.3 (Non negative matrices). A square matrix $\mathbf{P}$ with real coefficients is called nonnegative if all its entries are nonnegative.

Definition 4.1.4 (Stochastic matrices). A square matrix $\mathbf{P}$ with real coefficients is called stochastic if it is nonnegative and if all its rows sum to 1: $\mathbf{P}\mathbf{1} = \mathbf{1}$.

Spectral properties of stochastic matrices have been extensively studied in Markov chain theory. Since most of the matrices in this thesis are stochastic, some results are derived using tools from Markov chain theory, even though there are no Markov chains in the work.

Definition 4.1.5 (Primitive matrices). A nonnegative square matrix $\mathbf{P}$ is called primitive if there exists an integer k such that $\mathbf{P}^k > 0$ (all entries positive).

Note that this definition resembles Condition 4 for almost sure convergence (Th. 3.1.3).

Take a $m \times m$ nonnegative matrix $\mathbf{P}$, and let $\mathcal{G}$ be the graph of m nodes such that nodes i and j are connected by an edge if and only if $\mathbf{P}_{ij} > 0$. Then $\mathbf{P}$ is primitive only if $\mathcal{G}$ is connected, i.e. only if there is a path between each

pair of nodes. Furthermore, if $\mathcal{G}$ is connected and if $\mathbf{P}_{ii} > 0$ for at least one i, then $\mathbf{P}$ is primitive. Finally, in words borrowed from the Markov chain jargon, if $\mathbf{P}$ is irreducible and aperiodic, then $\mathbf{P}$ is primitive [16].

Theorem 4.1.2 (Perron-Frobenius theorem [16]). Let $\mathbf{P}$ be a nonnegative primitive $m \times m$ matrix. There exists a real eigenvalue λ_1 with algebraic as well as geometric multiplicity one, such that $\lambda_1 > 0$, and $\lambda_1 > |\lambda_j|$ for any other eigenvalue λ_j. Moreover, the left eigenvector $\mathbf{u}_1$ and the right eigenvector $\mathbf{v}_1$ associated with λ_1 can be chosen positive and such that $\mathbf{u}_1^T \mathbf{v}_1 = 1$.

Let $\lambda_2, \lambda_3, \ldots, \lambda_m$ be the eigenvalues of $\mathbf{P}$ other than λ_1 ordered in such a way that

$$\lambda_1 > |\lambda_2| \geqslant \ldots \geqslant |\lambda_m|,$$

and if $|\lambda_2| = |\lambda_j|$ for some $j \geqslant 3$, then $m_2 \geqslant m_j$, where m_j is the algebraic multiplicity of λ_j. Then,

$$\mathbf{P}^k = \lambda_1^k \mathbf{v}_1 \mathbf{u}_1^T + \Theta\left(k^{m_2 - 1} |\lambda_2|^k\right),$$

where $\Theta(f(k))$ represents a function of k such that there exists $\alpha, \beta \in \mathbb{R}$, $0 < \alpha \leqslant \beta < \infty$, such that $\alpha f(k) \leqslant \Theta(f(k)) \leqslant \beta f(k)$ for all k sufficiently large.

If in addition, $\mathbf{P}$ is stochastic, then $\lambda_1 = 1$, $\mathbf{v}_1 = \mathbf{1}$, and $\mathbf{u}_1 = \boldsymbol{\pi}$. If moreover $\mathbf{P}$ is doubly stochastic, then $\boldsymbol{\pi} = \mathbf{1}/n$, and

$$\mathbf{P}^k = \mathbf{J} + \Theta\left(k^{m_2 - 1} |\lambda_2|^k\right).$$

Comments: The first part of Theorem 4.1.2, which states that the largest eigenvalue has multiplicity 1, is useful to apply Theorem 3.1.6 (convergence in expectation) and Theorem 3.1.7 (mean square convergence, with implications on almost-sure convergence). Indeed Theorem 3.1.6 requires that $\rho(\mathbb{E}[\mathbf{W}] - \mathbf{J}) < 1$. But given the other two conditions of the theorem, if $\mathbb{E}[\mathbf{W}]$ is primitive, then $\mathbb{E}[\mathbf{W}]$ is doubly stochastic and $\rho(\mathbb{E}[\mathbf{W}] - \mathbf{J}) = \lambda_2(\mathbb{E}[\mathbf{W}])$[1]. As a consequence, showing that $\mathbb{E}[\mathbf{W}]$ is a primitive doubly stochastic matrix is sufficient to conclude that the algorithm converges in expectation. As to Theorem 3.1.7, the SLEM $\lambda_2(\mathbb{E}[\mathbf{W}^T\mathbf{W}])$ is required to be smaller than 1, and one can try to apply Frobenius theorem directly to the matrix $\mathbb{E}[\mathbf{W}^T\mathbf{W}]$.

The second part of the theorem explains why Markov chain theory contains interesting results on λ_2. Indeed, λ_2 determines the speed of convergence of the state distribution to the stationary distribution $\boldsymbol{\pi}$. The matrices of interest here can thus be interpreted as transition matrices of Markov chains, whose stationary distribution is uniform.

Finally, the last part of the theorem can be applied to synchronous average consensus algorithms in a fixed graph, when $\mathbf{W}$ is primitive doubly stochastic. It shows that, in this case, $\lambda_2(\mathbf{W})$ indeed controls the speed of convergence of the algorithm.

[1] To prove this equality, use Jordan decomposition

4.1.2 Non-negativity and symmetry are not necessary

The non-negativity of matrix $\mathbf{W}$ seems to be a key ingredient. However, one can design a successful matrix $\mathbf{W}$ that does not verify this property. In other words, one can imagine an algorithm that admits some negative coefficients $\mathbf{W}_{ij}$. In [78], the authors design a semidefinite program that finds the optimal symmetric weights $\{\mathbf{W}_{ij}\}_{i,j\in[1,n]^2}$ for synchronous averaging in any given fixed graph. As a result, the optimal solution on some examples have negative weights. Therefore, choosing nonnegative weights only is not the most efficient strategy in terms of convergence speed, but it is definitively a safe choice according to the Perron-Frobenius Theorem, because it guarantees convergence as explained in the comments at the end of Section 4.1.1. Most importantly, each node can check locally that it is using nonnegative coefficients. If negative coefficients are allowed, then it is hard to compute distributively the second largest eigenvalue in magnitude and check that it is smaller than 1 to ensure that the algorithm is converging correctly.

Apart from broadcast gossip and from one-way path averaging, all the algorithms described in Chapter 2 have symmetric matrices $\mathbf{W}(t)$. Symmetry is indeed a very practical feature: if $\mathbf{W}(t)$ is symmetric and if $\mathbf{W}(t)\mathbf{1} = \mathbf{1}$, then necessarily, $\mathbf{1}^T\mathbf{W}(t) = \mathbf{1}^T$. Designing a non-symmetric matrix $\mathbf{W}(t)$ that has both properties $\mathbf{W}(t)\mathbf{1} = \mathbf{1}$ and $\mathbf{1}^T\mathbf{W}(t) = \mathbf{1}^T$ is a hard task if it has to be done in a distributed way. Indeed, for any matrix $\mathbf{W}(t)$,

$$\mathbf{W}(t) = \frac{\mathbf{W}(t) + \mathbf{W}^T(t)}{2} + \frac{\mathbf{W}(t) - \mathbf{W}^T(t)}{2} = \mathbf{W}_1(t) + \mathbf{W}_2(t).$$

In other words, any matrix is the sum of a symmetric matrix $\mathbf{W}_1(t)$ and of an asymmetric matrix $\mathbf{W}_2(t)$. Being able to design a non symmetric matrix $\mathbf{W}(t)$ in a distributed way is equivalent to being able to design a non zero antisymmetric matrix whose rows and columns sum to 0 in a distributed way. Imagine that there is a cycle in the communication graph $\mathcal{G}$. Then, one possible antisymmetric matrix has coefficients equal to $+\alpha$ in one direction of the cycle, and $-\alpha$ in the opposite direction of the cycle (see Fig. 4.1). However detecting a cycle in a distributed way is not an easy task, especially if the network is time-varying. Generally speaking, designing such antisymmetric matrices requires some global knowledge of the network. This is unfortunate since asymmetry would introduce some convection of information in the network, and could therefore accelerate the convergence of the algorithms. There are no detailed theoretical studies on the beneficial effects of convection on the performance of average consensus. Not only the practical implementation of convection defeats the distributivity constraint, but also simulations suggest that convection is a difficult phenomenon to study. For example, Fig. 4.2 and Fig. 4.3 illustrates how convection is sensible to the slightest change of parameter.

4.1.3 Useful results from Markov chain theory

It is good to know that the SLEM λ_2 is smaller than 1 to guarantee convergence, but it is even better to have an estimation of λ_2, since λ_2 appears in the bounds on speed of convergence. They are many ways to compute the second largest eigenvalue in magnitude of a matrix $\mathbf{P}$. First, one can try to compute all the eigenvalues of $\mathbf{P}$ and pick the second largest one in magnitude. A second

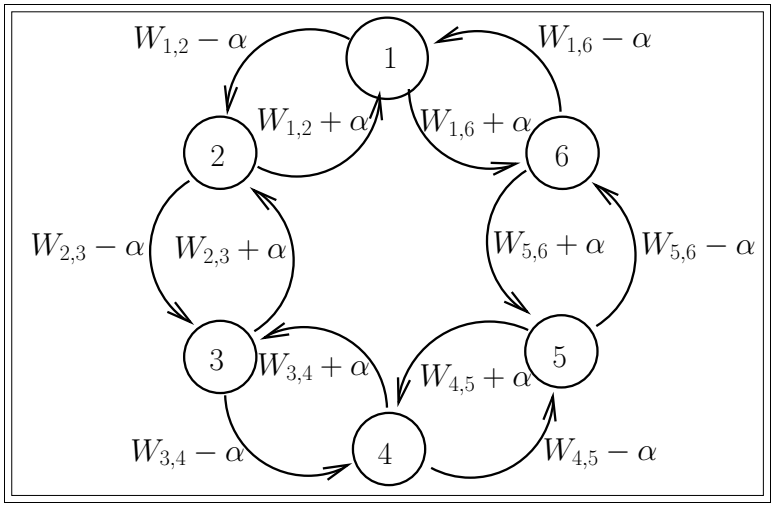


Figure 4.1: Sub-cycle of 6 nodes on which an antisymmetric matrix has been added to the symmetric coefficients $\mathbf{W}_{i,i+1}$, for $i = 1, \ldots, 6$. In this example, coefficient α was added clockwise and subtracted counter-clockwise.

strategy is to study the characteristic polynomial of $\mathbf{P}$, whose roots are the eigenvalues of $\mathbf{P}$, and focus on its behavior only around -1 and 1. These two strategies are applied in the next two sections. A third possibility, which is the topic of this section, is to use theorems developed in the framework of Markov chains.

A good summary of bounds on $\lambda_2(\mathbf{P})$ when $\mathbf{P}$ is an irreducible[2] and reversible[3] stochastic matrix can be found in [16] (matrices are finite in this work). The most famous bound is the conductance bound, which can be found in the literature on distributed average consensus algorithm. It is useful to bound the SLE of random walks on lattices for example. In this thesis however, the conductance bound is not sufficient, and many results on gossip algorithms are proven with the "weighted path upper bound", also called Poincaré's inequality, due to Diaconis and Strook (1991).

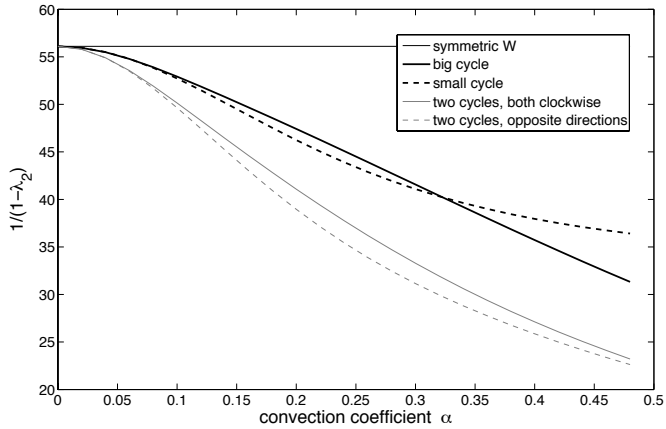
Conductance bound

Theorem 4.1.3. [Jerrum and Sinclair (1989) [47]] Let $\mathbf{P}$ be a finite state space irreducible reversible transition matrix of size $m \times m$, and $\boldsymbol{\pi}$ be its stationary distribution. For any subset B of the states, the capacity of B is defined as $\boldsymbol{\pi}(B) = \sum_{i \in B} \boldsymbol{\pi}(i)$, and the ergodic flow out of B is defined as $F(B) = \sum_{i \in B, j \in \bar{B}} \boldsymbol{\pi}(i)\mathbf{P}_{ij}$. Then the conductance of the pair $(\mathbf{P}, \boldsymbol{\pi})$ is

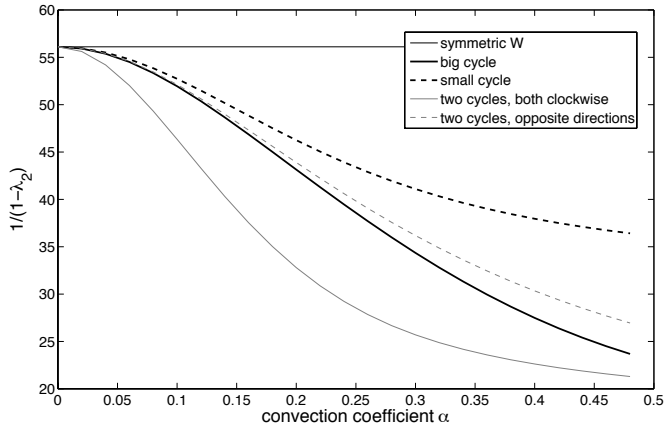
$$\phi(\mathbf{P}) = \inf\left(\frac{F(B)}{\boldsymbol{\pi}(B)}; 0 < |B| < m, \boldsymbol{\pi}(B) \leqslant \frac{1}{2}\right). \tag{4.3}$$

[2]irreducible: for any i and j, there is a path $(i, k_1, \ldots, k_m, j)$ such that $\mathbf{P}_{ik_1}, \ldots, \mathbf{P}_{k_m,j}$ are all positive

[3]reversible: Let $\boldsymbol{\pi}$ be the stationary distribution of $\mathbf{P}$ ($\boldsymbol{\pi}^T\mathbf{P} = \boldsymbol{\pi}^T$); such a $\boldsymbol{\pi}$ exists if $\mathbf{P}$ is irreducible. Then $\mathbf{P}$ is reversible if and only if $\boldsymbol{\pi}(i)\mathbf{P}_{ij} = \boldsymbol{\pi}(j)\mathbf{P}_{ji}$



(a) cycles of radius 2 and 4



(b) cycles of radius 2 and 3

Figure 4.2: Relaxation time $1/(1-\lambda_2) \simeq 1/\log(\lambda_2^{-1})$ of asymmetric matrices $\mathbf{W}$ in a lattice network of 11×11 nodes. Based on a symmetric matrix $\mathbf{W}$ designed with constant weight 0.22, asymmetry is added to some cycles. To clockwise add asymmetry to a cycle, add α to $\mathbf{W}_{ij}$ if node j is clockwise just after node i in the cycle. Substract α if node j is just before node i. A cycle of radius k is the set of nodes situated at L_1-distance k of the central node of the lattice. Four experiments are run in each figure. First add asymmetry on a small cycle, second add asymmetry on a larger cycle. Then add asymmetry to both the small and large cycles, in the same direction first, and in opposite directions second. In the first figure, the two cycles have radii 2 and 4 so that they do not touch each other. In the second figure, they have radii 2 and 3; the two cycles are connected. Note that the two cases lead to very different results. In particular, in the first case, opposite convection is more efficient than convection in the same direction, whereas in the second case the situation is reversed. Coefficient α cannot be chosen arbitrarily large; otherwise eigenvalues larger than 1 in magnitude appear and the algorithm becomes unstable. The curves were stopped just before instability.

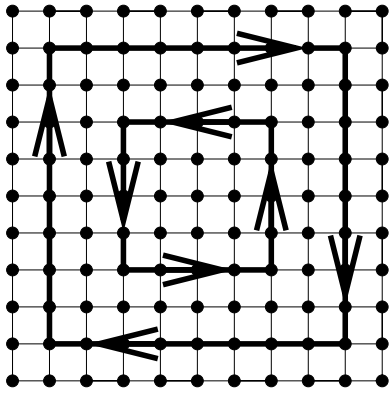


Figure 4.3: 11×11 lattice, with two cycles of radius 2 and 4. This figure illustrates the simulations displayed in Fig. 4.2

Then, the second largest eigenvalue ξ_2 (SLE)[4] verifies:

$$1 - 2\phi(\mathbf{P}) \leqslant \xi_2(\mathbf{P}) \leqslant 1 - \frac{\phi(\mathbf{P})^2}{2}. \tag{4.4}$$

The following theorem may appear as more difficult to use, but it usually gives tighter bounds.

Weighted path upper bound

Theorem 4.1.4 (Poincaré's inequality [25]). Let $\mathbf{P}$ be a finite state space irreducible reversible transition matrix, and let $\boldsymbol{\pi}$ be its stationary distribution ($\boldsymbol{\pi}$ is its left eigenvector associated to the eigenvalue 1 ($\boldsymbol{\pi}^T\mathbf{P} = \boldsymbol{\pi}^T$) such that $\sum_{i=1}^n \boldsymbol{\pi}(i) = 1$). A pair $e = (k, l)$ is called an edge if $\mathbf{P}_{kl} \neq 0$. For each ordered pair (i, j) where $1 \leqslant i, j \leqslant n$, $i \neq j$, choose one and only one path $\gamma_{ij} = (i, i_1, \ldots, i_m, j)$ between i and j such that $\mathbf{P}_{i,i_1}$, $\mathbf{P}_{i_1,i_2}$, …, $\mathbf{P}_{i_m,j}$ are all positive, i.e. such that (i, i_1), (i_1, i_2), …, (i_m, j) are all edges. Define

$$|\gamma_{ij}| = \frac{1}{\boldsymbol{\pi}(i)\mathbf{P}_{ii_1}} + \frac{1}{\boldsymbol{\pi}(i_1)\mathbf{P}_{i_1 i_2}} + \ldots + \frac{1}{\boldsymbol{\pi}(i_m)\mathbf{P}_{i_m j}}. \tag{4.5}$$

The Poincaré coefficient is defined as

$$\kappa = \max_{\text{edge } e} \sum_{\gamma_{ij} \ni e} |\gamma_{ij}| \boldsymbol{\pi}(i)\boldsymbol{\pi}(j). \tag{4.6}$$

Then the second largest eigenvalue ξ_2 (SLE) of $\mathbf{P}$ verifies

$$\xi_2(\mathbf{P}) \leq 1 - \frac{1}{\kappa}. \tag{4.7}$$

Note that Theorems 4.1.3 and 4.1.4 are about the SLE, and thus they are

[4]Here, ξ_2 is the second largest eigenvalue, it is not necessarily the second largest one *in magnitude*.

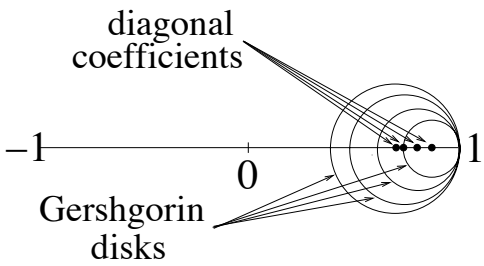


Figure 4.4: The Gershgorin disk argument to claim that the SLE is the SLEM when the diagonal coefficient are close enough to 1. Theorem 4.1.5 states that all the eigenvalues lie in the union of the Gershgorin disks, that are centered in the diagonal coefficients, and that pass by 1.

not sufficient to characterize the SLEM. Since we always consider aperiodic irreducible reversible stochastic matrices, the largest eigenvalue is 1, and the SLEM λ_2 is either equal to ξ_2 or ξ_m. In most asynchronous algorithms, a simple argument quickly shows that the SLE ξ_2 is the SLEM λ_2, in other words, the smallest eigenvalue ξ_m is smaller in magnitude than the SLE ξ_2. This argument is the Gershgorin (also written as Geršgorin) discs theorem [45]:

Theorem 4.1.5 (Gershgorin's bound). Let $\mathbf{P}$ be a finite $m \times m$ matrix. Then, for any eigenvalue ξ, there is a $k \in [1, r]$ such that,

$$|\xi - \mathbf{P}_{kk}| \leqslant \min \left(\sum_{j=1, j\neq k}^{r} |\mathbf{P}_{kj}|, \sum_{j=1, j\neq k}^{r} |\mathbf{P}_{jk}| \right).$$

When this bound is applied to a doubly stochastic matrix, we get that all the eigenvalues lie in disks centered in some diagonal coefficient $\mathbf{P}_{kk}$ passing by 1. In most asynchronous algorithms, the diagonal coefficients of $\mathbf{P}$ are large, because nodes remain quiet in most iterations. Therefore the centers of Gershgorin disks are often close to 1 (they are larger than 0.5) and do not contain negative valued points (see Fig. 4.4). Eigenvalues being positive, for any i, $\xi_i = \lambda_i$, the SLEM λ_2 is equal to the SLE ξ_2.

How to use Poincaré's inequality:

Poincaré's inequality will be used to study the convergence time of gossip algorithms. Remind that for a gossip algorithm, the speed of convergence is bounded with a function of the SLEM $\lambda_2(\mathbb{E}[\mathbf{W}])$. The inequality will thus be applied to matrix $\mathbb{E}[\mathbf{W}]$. It is important to see that the sparsity[5] of $\mathbb{E}[\mathbf{W}]$ is not necessarily equal to the sparsity of the graph $\mathcal{G}$[6], i.e. if $\mathbb{E}[\mathbf{W}]_{ij}$ is positive, then there is not necessarily an edge between i and j in the network. Indeed, for example, in geographic gossip, at any iteration, any pair of nodes can be average thanks to routing. If i and j average their estimates, then $\mathbf{W}_{ij} = 1/2 > 0$. Since this happens with positive probability for any pair of edges, all the entries of $\mathbb{E}[\mathbf{W}]$ are positive. Therefore an "edge" used in Poincaré's inequality does not necessarily correspond to a physical "edge" of the network.

[5]The sparsity of a matrix $\mathbf{A}$ is the set of indices (i, j) such that $\mathbf{A}_{ij} > 0$

[6]The sparsity of a graph $\mathcal{G}$ is the set of indices (i, j) such that edge (i, j) is in $\mathcal{G}$.

In other words, Poincaré's inequality should be seen as an algebraic tool, that can study the spectral properties of $\mathbb{E}[\mathbf{W}]$ independently of the underlying algorithm and network topology. For such a matrix $\mathbb{E}[\mathbf{W}]$, one can define an overlay directed weighted graph $\mathcal{G}'$, that has the same vertices as the network on which the algorithm is running, but that has a possibly different set of edges E'. Each directed edge (i,j) of $\mathcal{G}'$ has a weight equal to $\mathbb{E}[\mathbf{W}]_{i,j}$.

Intuitively, Poincaré's inequality simply states the following: if information can travel fast without any bottleneck between any pair of nodes, then the algorithm converges fast. More precisely, the averaging time T_{ave} scales as $1/(1-\lambda_2(\mathbb{E}[\mathbf{W}]))$, and if $\xi_2 = \lambda_2$, the Poincaré coefficient κ defined in the theorem conveniently verifies:

$$\frac{1}{1-\lambda_2} \leqslant \kappa.$$

The Poincaré coefficient κ is calculated based on paths that need to be designed. Depending on the chosen paths, κ may take different values, but all these values are systematically larger than $1/(1-\lambda_2)$. Therefore, a good paths design leads to a small κ, and thus to a bound which is hopefully tight enough. Interestingly, κ can be interpreted as a measure of the diffusivity of the algorithm. For any pair of vertices (i,j), one should design a path of edges in the overlay graph $\mathcal{G}'$ that goes from i to j. For each directed edge e in $\mathcal{G}'$,

$$\sum_{\gamma_{ij} \ni e} |\gamma_{ij}|\, \boldsymbol{\pi}(i)\boldsymbol{\pi}(j)$$

measures the congestion of the constructed paths on edge e. κ is simply equal to the maximum congestion. In order to get a good bound κ, one should thus design paths that do not unnecessarily overuse some edges. Therefore looking for short paths containing a small number of edges can help reduce congestion. Also note that in

$$|\gamma_{ij}| = \frac{1}{\boldsymbol{\pi}(i)\mathbf{P}_{ii_1}} + \frac{1}{\boldsymbol{\pi}(i_1)\mathbf{P}_{i_1 i_2}} + \ldots + \frac{1}{\boldsymbol{\pi}(i_m)\mathbf{P}_{i_m j}},$$

the inverse of the weights are used. Therefore edges $e = (k,l)$ with large weights $\mathbf{P}_{kl}$ should be privileged in paths γ_{ij}. In other words, edges with large weights are good diffusers and therefore good links, which can help reduce κ and therefore reach a tighter bound.

How Poincaré's inequality helps to design efficient gossip algorithms:

Conversely, one can try to design a gossip algorithm that would lead to small Poincaré coefficients, guaranteeing fast convergence. They are two main ingredients: designing *large coefficients* $\mathbb{E}[\mathbf{W}]_{ij}$, and making *long distance* averagings (through routing), to create long edges in $\mathcal{G}'$, in order to reduce the length of the paths γ_{ij}. In pairwise gossip, $\mathbb{E}[\mathbf{W}]_{ij}$ is very large if i and j are neighbors. However, since there is no routing, some paths γ_{ij} cannot be shorter than the diameter of the network, which increases the value of $|\gamma_{ij}|$ and increases in turn the Poincaré coefficient κ. On the contrary, in geographic gossip, since every pair of nodes can be pairwise averaged thanks to routing, then every path γ_{ij} can be designed with one edge only: $e = (i,j)$. However, in this case, for any pair (i,j), $\mathbb{E}[\mathbf{W}]_{ij}$ is very small, which penalizes the value

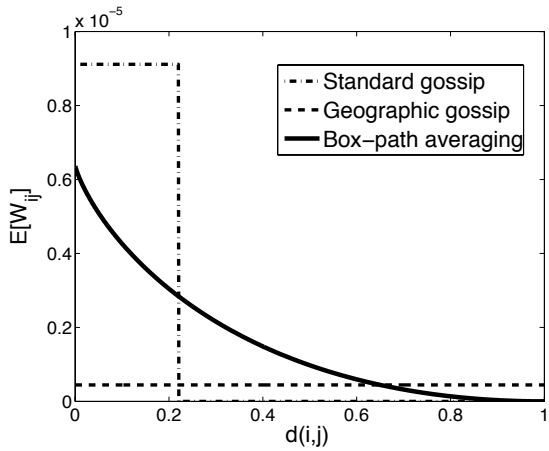


Figure 4.5: Simplified behavior of $\mathbb{E}[W]_{ij}$ as a function of the distance in norm 1 between i and j for standard gossip, geographic gossip and path averaging. In a glance, this figure shows the main differences between the three algorithms.

of κ.

Path averaging is a very good tradeoff between frequent short distance averaging (favoring large coefficients $\mathbb{E}[\mathbf{W}]_{ij}$) and long distance averaging (reducing the size of the paths γ_{ij}). As shown later in Chapter 5, close enough nodes are likely to be on a common route at each iteration. As a consequence, 2-hops paths $\gamma_{ij} = (i,k),(k,j)$, where k is half way between i and j, are good enough to guarantee the use of edges with large enough weights and lengths. Furthermore, edges are used in a fair way, reducing congestion and leading to the optimal bound, where κ is linear in the number n of nodes in the network.

A comparative sketch of behaviors of the coefficients of $\mathbb{E}[\mathbf{W}]$ in standard gossip, geographic gossip and path averaging, is shown in Fig. 4.5. In contrast with geographic gossip, path averaging and standard gossip *concentrate* their averaging effort on close nodes, which leads to larger coefficients $\mathbb{E}[\mathbf{W}]_{i,j}$ when nodes i and j are close enough. However, while standard gossip pays for its excessive concentration with long paths γ_{ij} overusing every existing edge, the diffusion pattern of path averaging operates in 2 steps only without creating any congestion. This intuition will lead to formal proofs in Chapter 5.

4.2 Circles and Lattices

As explained in previous sections, the performance of average consensus algorithms are usually linked to the relaxation time $1/(1-\lambda_2(\mathbf{P}))$ of a Markov chain, whose transition matrix $\mathbf{P}$ is doubly stochastic. This section explores the relaxation time of Markov chains on circles, which leads to the relaxation time of Markov chains on lattices embedded on a torus. The results on lattices in this section will be useful in Chapter 5, where various gossip algorithms on lattices are studied. Some lower bounds of the convergence time of these algorithms on lattices will also be derived in Chapter 5, thanks to results de-

rived in this section on fastest mixing chains, i.e. on chains which minimize the relaxation time.

We will start by recalling some facts on circulant matrices.

4.2.1 Circulant matrices

The transition matrix $\mathbf{P}$ of a Markov chain on a circle of n nodes, where all nodes behave identically, is a $n \times n$ circulant matrix with coefficients $c_{-n}, \ldots, c_{-1}, c_0, c_1, \ldots, c_n$. Furthermore, if each node can reach all k nodes on its right and all k nodes on its left, then $\mathbf{P}$ is circulant and its nonzero coefficients are among $c_{-k}, \ldots, c_0, \ldots, c_k$.

All circulant matrices have the same eigenvectors, which form a discrete Fourier transform basis. For $0 \leqslant m \leqslant n$, their eigenvalues are given by:

$$\xi^{(m)} = \sum_{k=0}^{n-1} c_k \exp\left(\frac{2\pi \jmath m k}{n}\right).$$

$\mathbf{J} = \frac{\mathbf{1}\mathbf{1}^T}{N}$ is also a circulant matrix, hence it is codiagonalisable with $\mathbf{P}$, and its eigenvalues are:

$$\begin{cases} \xi^{(0)} = 1 \\ \xi^{(m)} = 0 \quad \text{for } m \in [1, N-1]. \end{cases}$$

But $\xi^{(0)}(\mathbf{P}) = 1$, thus, even when $\mathbf{1}^T\mathbf{P} \neq \mathbf{1}^T$, the spectral radius involved in the speed of convergence of synchronous algorithms on fixed graphs verifies:

$$\rho(\mathbf{P} - \mathbf{J}) = |\lambda_2(\mathbf{P})|,$$

where λ_2 denotes the SLEM.

A stochastic circulant matrix with minimum second largest eigenvalue in magnitude (SLEM) is necessarily *symmetric*. Indeed, decompose $\mathbf{P}$ as

$$\mathbf{P} = \frac{\mathbf{P} + \mathbf{P}^T}{2} + \frac{\mathbf{P} - \mathbf{P}^T}{2} = \mathbf{S} + \mathbf{A}.$$

$\mathbf{S}$ is a symmetric circulant matrix verifying $\mathbf{1}^T\mathbf{S} = \mathbf{1}^T$ and $\mathbf{S}\mathbf{1} = \mathbf{1}$. $\mathbf{A}$ is antisymmetric and circulant. Since they are both circulant, they are codiagonalisable. The eigenvalues of $\mathbf{P}$ are thus the sum of the eigenvalues of $\mathbf{S}$ and $\mathbf{A}$ corresponding to the same eigenvectors. But the eigenvalues of $\mathbf{S}$ are real and those of $\mathbf{A}$ are purely imaginary, which implies that $|\lambda_2(\mathbf{S})| \leq |\lambda_2(\mathbf{P})|$. For λ_2 to be as small as possible, *the analysis is consequently restricted to symmetrical circulant matrices such that* $\mathbf{P}\mathbf{1} = \mathbf{1}$. Note that these matrices necessarily

verify $\mathbf{1}^T\mathbf{P} = \mathbf{1}^T$. For example, if the connection range $k = 2$, then

$$\mathbf{P} = \begin{pmatrix} c_0 & c_1 & c_2 & 0 & \dots & 0 & c_2 & c_1 \\ c_1 & c_0 & c_1 & c_2 & 0 & \dots & 0 & c_2 \\ c_2 & c_1 & \ddots & \ddots & \ddots & \ddots & \vdots & 0 \\ 0 & c_2 & \ddots & \ddots & \ddots & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \ddots & c_2 & 0 \\ 0 & \vdots & \ddots & \ddots & \ddots & \ddots & c_1 & c_2 \\ c_2 & 0 & \dots & 0 & c_2 & c_1 & c_0 & c_1 \\ c_1 & c_2 & 0 & \dots & 0 & c_2 & c_1 & c_0 \end{pmatrix}.$$

The remainder of this section focuses on the following two questions:

- How does the relaxation time $1/(1 - |\lambda_2(\mathbf{P})|)$ scale as the number n of nodes grows for a given connection radius k? This is a key question that needs to be answered in order to study pairwise gossip on lattices for instance. We specifically need to derive a *lower bound* on the relaxation time, since upper bounds are easily computed thanks to the conductance theorem and to Poincaré's inequality.

- Given n and k, how can one design optimal coefficients $\{c_i\}_{i=-k,\dots,k}$ in order to minimize relaxation time? The answer to this question will allow to design the fastest converging synchronous averaging algorithm on the lattice. It will also indicate how to tune communication probabilities in order to optimize pairwise gossip on a lattice (in a setting where nodes do not necessarily choose a neighbor *uniformly* at random). Section 4.2.3 gives a simple procedure to design near-optimal coefficients $\{c_i\}_{i=-k,\dots,k}$, which are also called filter coefficients in this thesis.

4.2.2 SLEM on cycles and lattices

In this section, we are interested in a lower bound on the SLEM of the fastest mixing chain on toroidal lattices. This result is needed to lower bound the performance of pairwise gossip on lattices and random geometric graphs in Chapter 5.

Impact of the symmetry of a network

For the purpose of this thesis, we will work only on *symmetric* transition matrices, since these are easy to implement in a distributed way, as explained in Section 4.1.2. On the set of symmetric and stochastic matrices, the SLEM becomes a convex function. Indeed, all these matrices share the same eigenvector $\mathbf{1}$ associated with their largest eigenvalue 1. Therefore, for any such matrix $\mathbf{P}$,

$$|\lambda_2(\mathbf{P})| = \max_{\|\mathbf{x}\|=1, \mathbf{x}^T\mathbf{1}=0} \mathbf{x}^T\mathbf{P}\mathbf{x}.$$

Thus, for any $0 < a < 1$, and any symmetric and stochastic matrices $\mathbf{P}_1$ and $\mathbf{P}_2$,

$$\begin{aligned}
|\lambda_2(a\mathbf{P}_1 + (1-a)\mathbf{P}_2)| &= \max_{\|\mathbf{x}\|=1, \mathbf{x}^T\mathbf{1}=0} \mathbf{x}^T(a\mathbf{P}_1 + (1-a)\mathbf{P}_2)\mathbf{x} \\
&\leqslant a \max_{\|\mathbf{x}\|=1, \mathbf{x}^T\mathbf{1}=0} \mathbf{x}^T\mathbf{P}_1\mathbf{x} + (1-a) \max_{\|\mathbf{x}\|=1, \mathbf{x}^T\mathbf{1}=0} \mathbf{x}^T\mathbf{P}_2\mathbf{x} \\
&= a\,|\lambda_2(\mathbf{P}_1)| + (1-a)\,|\lambda_2(\mathbf{P}_2)|\,.
\end{aligned}$$

As a consequence, the matrices with lower SLEM respect the symmetries of the network. Indeed, if the network is invariant under a permutation S of its vertices, then for any symmetric and stochastic matrix $\mathbf{P}$ and its permuted version $\mathbf{P}'$, $0.5\mathbf{P} + 0.5\mathbf{P}'$ will have a smaller SLEM than $\mathbf{P}$. The interesting point is that $0.5\mathbf{P} + 0.5\mathbf{P}'$ is now invariant under S. Therefore, the transitions of the fastest mixing chain should be invariant under any network invariant permutation. This is why, on a cycle for instance, the symmetric fastest mixing chain is circulant.

Cycles are easier to study than lattices. We are thus going to study cycles first, and then we will use the results on cycles to infer results on lattices. We start by giving two different ways to link cycles and lattices.

From cycles to lattices

Argument 4.2.1 (Kronecker product). Let $\mathbf{P}$ be a $n \times n$ symmetric, circulant and stochastic matrix with nonzero coefficients c_i, for $i \in [-k, k]$. $\mathbf{P}$ can be seen as the transition matrix of a Markov chain on the cycle of n nodes and radius k. Based on $\mathbf{P}$, one can construct another symmetric stochastic matrix $\widetilde{\mathbf{P}}$, which is a transition matrix on the lattice of n^2 nodes embedded on a torus, where each pair of node is connected if their distance is less than or equal to k in ℓ_∞ distance (neighborhoods are squares). This matrix $\widetilde{\mathbf{P}}$ is defined thanks to the Kronecker product: $\widetilde{\mathbf{P}} = \mathbf{P} \otimes \mathbf{P}$. Interestingly, the eigenvalues of $\widetilde{\mathbf{P}}$ are obtained as all the pairwise products of eigenvalues of $\mathbf{P}$. Therefore, as $\mathbf{P}$ has eigenvalue 1 with multiplicity 1, 1 is also eigenvalue of $\widetilde{\mathbf{P}}$ with multiplicity 1, and the SLEM of $\widetilde{\mathbf{P}}$ is equal to the SLEM of $\mathbf{P}$ ($\times 1$). Hence, if one has a sequence of matrices $\mathbf{P}_n$ on circles of size n, whose relaxation times scale as $f_k(n)$ as n grows, then the associated matrices $\widetilde{\mathbf{P}}_n$ on toroidal lattices of $n' = n^2$ nodes with radius k have a relaxation time growing as $f_k(\sqrt{n})$.

This is the argument used in [15], a founding paper that computes the performance of pairwise gossip on lattices and random geometric graph. We develop another argument as well.

Argument 4.2.2 (Induced matrix). Consider a Markov chain $\{X(t)\}_{t \geqslant 0}$ on the toroidal lattice of $m = n^2$ nodes and of radius k. Let $\mathbf{P}$ be its $n^2 \times n^2$ transition matrix. Assume that $\mathbf{P}$ respects the symmetry of the lattice: for every pair of nodes (i, j) of coordinates (x_i, y_i) and (x_j, y_j), $\mathbf{P}_{ij}$ is a function f of $d_x = |x_i - x_j|$ and of $d_y = |y_i - y_j|$, such that:

$$f(d_x, d_y) = f(d_y, d_x).$$

The symmetry of the function f ensures that lines and columns behave in the same way. The presence of the absolute value in $|x_i - x_j|$ and $|y_i - y_j|$

guarantees that the transitions are the same to the left and to the right, upwards and downwards. Finally using the distances d_x and d_y implies that P_{ij} does not depend on the location of i, but only on the relative position of i and j. Note that the fastest mixing chain on a toroidal lattice has these symmetries by the convexity of the SLEM function, as explained in the previous paragraph. The idea of our argument is that the time to reach the stationary distribution on lines and columns, should be larger than the relaxation time on columns only. Therefore, the relaxation time on lattices should be lower bounded by the relaxation time on cycles.

Formally, we construct an induced Markov chain $\{Y(t)\}_{t\geqslant 0}$, where $Y(t)$ is the column index of $X(t)$ for any t. Because the Markov chain $\{X(t)\}_{t\geqslant 0}$ has the symmetry properties described above, $\{Y(t)\}_{t\geqslant 0}$ is indeed a homogeneous Markov chain. Furthermore, it is a Markov chain on the cycle of n nodes and of radius k, and its transition matrix $\mathbf{P}'$ is symmetric, circulant and stochastic. The transition matrices $\mathbf{P}$ and $\mathbf{P}'$ are linked by the following formula:

$$\mathbf{P}' = \mathbf{B}^T\mathbf{P}\mathbf{A},$$

where $\mathbf{A}$ and $\mathbf{B}$ are a $m \times n$ matrices such that

- $\mathbf{A}_{ij} = 1$ if and only if node i is in column j. Otherwise $\mathbf{A}_{ij} = 0$
- $\mathbf{B}_{ij} = 1$ if and only if node i is in column j *and* node i is in line 1. Otherwise $\mathbf{B}_{ij} = 0$

Note that $\mathbf{B}^T\mathbf{A} = \mathbf{I}$. We are now going to show that the spectrum of $\mathbf{P}'$ is a subset of the spectrum of $\mathbf{P}$. For any eigenvalue ξ of $\mathbf{P}'$, let $\mathbf{x}$ be one of its associated eigenvectors: $\mathbf{P}'\mathbf{x} = \xi\mathbf{x}$. Then,

$$\mathbf{A}\mathbf{B}^T\mathbf{P}(\mathbf{A}\mathbf{x}) = \xi(\mathbf{A}\mathbf{x}).$$

Now we need to show that $\mathbf{A}\mathbf{B}^T\mathbf{P}(\mathbf{A}\mathbf{x}) = \mathbf{P}(\mathbf{A}\mathbf{x})$ to conclude. For any $\mathbf{y}$, $\mathbf{A}\mathbf{B}^T\mathbf{y} = \mathbf{y}$ if $\mathbf{y}_i = \mathbf{y}_j$ whenever i and j are on the same column. Therefore, we need to show that $(\mathbf{P}\mathbf{A}\mathbf{x})_i$ depends only on the column containing node i. Note that $(\mathbf{A}\mathbf{x})_k = \mathbf{x}_c$, where c is the column of node k. For any node i,

$$\begin{aligned}
(\mathbf{P}\mathbf{A}\mathbf{x})_i &= \sum_{g=1}^{m} \mathbf{P}_{ig}(\mathbf{A}\mathbf{x})_g \\
&= \sum_{\text{col } c} \sum_{g\in\text{col } c} \mathbf{P}_{ig}(\mathbf{A}\mathbf{x})_g \\
&= \sum_{\text{col } c} \sum_{g\in\text{col } c} \mathbf{P}_{ig}x_c \\
&= \sum_{\text{col } c} x_c \left(\sum_{g\in\text{col } c} \mathbf{P}_{ig} \right).
\end{aligned}$$

Since the symmetry of $\mathbf{P}$ implies that $\sum_{g\in\text{col } c}\mathbf{P}_{ig}$ is independent of the location of i in its column, we have proven that $\mathbf{P}(\mathbf{A}\mathbf{x}) = \xi\mathbf{A}\mathbf{x}$. Since $\mathbf{x} \neq 0$, then $\mathbf{A}\mathbf{x} \neq 0$, and ξ is an eigenvalue of $\mathbf{P}$. Therefore the SLEM of the Markov chain on the lattice is larger than or equal to the SLEM of a Markov chain

on the cycle. Hence, it suffices to derive a lower bound on the SLEM of symmetric, circulant and stochastic matrices to get a lower bound on the SLEM of symmetrical Markov chains on lattices.

Having argued that lower bounds on the relaxation time on lattices can be computed thanks to the relaxation time on circles, we now evaluate the latter.

Cycles

Theorem 4.2.1 (Lower bound on the relaxation time on cycles). Let $\mathbf{P}$ be a symmetric and circulant transition matrix of a Markov chain on the cycle of n nodes and of radius k. Its relaxation time is lower bounded by:

$$\frac{1}{1-|\lambda_2(\mathbf{P})|} \geqslant \frac{1}{(1-c_0)\left(1-\cos\left(\frac{2\pi k}{n}\right)\right)} \approx \frac{n^2}{2\pi^2(1-c_0)k^2}, \tag{4.8}$$

where $c_0 = \mathbf{P}_{11}$.

Proof: The eigenvalues of $\mathbf{P}$ are:

$$\begin{aligned}\xi^{(m)} &= \sum_{i=-(n-1)}^{n-1} c_i \exp\left(\frac{2\pi\jmath\, im}{n}\right)\\ &= c_0 + \sum_{i=1}^{k} 2c_i \cos\left(\frac{2\pi im}{n}\right).\end{aligned}$$

For $m=0$, $\xi^{(0)}=1$. For $m=1$,

$$\xi^{(1)} = c_0 + \sum_{i=1}^{k} 2c_i \cos\left(\frac{2\pi i}{n}\right).$$

If $k \leqslant n/4$, then all the cosines are positive and

$$\xi^{(1)} \geqslant c_0 + (1-c_0)\cos\left(\frac{2\pi k}{n}\right),$$

so that the SLEM verifies

$$|\lambda_2(\mathbf{P})| \geqslant c_0 + (1-c_0)\cos\left(\frac{2\pi k}{n}\right).$$

As a consequence,

$$\frac{1}{1-|\lambda_2(\mathbf{P})|} \geqslant \frac{1}{(1-c_0)\left(1-\cos\left(\frac{2\pi k}{n}\right)\right)} \approx \frac{n^2}{2\pi^2(1-c_0)k^2}.$$

■

In [15], Boyd et al. show that if $c_0=0$, $c_1=\delta,\ldots,c_{k-1}=\delta, c_k=0.5-(k-1)\delta$, then, if δ is small enough,

$$\lambda_2(\mathbf{P}) = \cos\left(\frac{2\pi k}{n}\right) + \Theta\left(\frac{k^4}{n^4}\right).$$

In other words, there are coefficients $\{c_i\}_{i\in[-k,k]}$ such that the bound in (4.2.1) is tight. We call these sets of coefficients parametrized by δ: "δ-filters".

Comments:

- Note that the δ-filters contain the case of the natural random walk on the circle ($k = 1$), which thus requires order n^2 transitions to reach the uniform stationary distribution.
- For any of these δ-filters ($c_0 = 0$, $c_1 = \delta, \ldots, c_{k-1} = \delta, c_k = 0.5-(k-1)\delta$), one can design a synchronous averaging algorithm with matrix $\mathbf{W} = \mathbf{P}$. In each round, this synchronous algorithm uses $\Theta(n)$ broadcast messages, which implies that $\Theta(n^3 \log n/k^2)$ messages are needed to reach a level of precision $\|\boldsymbol{\epsilon}\| = n^{-\alpha}$.
- In the case of pairwise gossip algorithms on the cycle of n nodes and of radius k, the expected averaging matrix $\mathbb{E}[\mathbf{W}]$ is of the form of matrix $\mathbf{P}$. Its coefficient c_0 is necessarily close to 1, since any given node i remains silent in most iterations ($\mathbf{W}_{ii} = 1$). Hence, in gossip algorithms, $1-c_0$ becomes small, and it is non negligible in Eq. 4.2.1. More precisely, $1 - c_0 = \Theta(1/n)$. Therefore, even if each iteration demands a number of messages which does not grow with n nor k, an extra factor n is brought in Bound (4.2.1) by the coefficient $1-c_0$. As a conclusion, pairwise gossip on the cycle requires at least $\Theta(n^3 \log n/k^2)$ messages to reach a level of precision $\|\boldsymbol{\epsilon}\| = n^{-\alpha}$ with high probability. Note that the best possible performance of pairwise gossip is the same as the performance of the synchronous algorithm described above.

Lattices We have proven that averaging on a circle takes at least n^3 messages, which proves that the circle is a very slow topology. Due to the arguments described at the beginning of the section, we will show that lattices save an order n compared to the circle. In other words, n^2 messages are necessary to run an average consensus algorithm on a lattice embedded on the torus:

Theorem 4.2.2 (Relaxation time on toroidal lattices). Let $\mathbf{P}$ be a symmetric transition matrix of a Markov chain on the lattice of n nodes and of radius k of the form:

$$\mathbf{P} = \mathbf{P}' \otimes \mathbf{P}',$$

where $\mathbf{P}'$ is an $\sqrt{n} \times \sqrt{n}$ symmetric, circulant and stochastic matrix. The relaxation time of $\mathbf{P}$ is lower bounded by:

$$\frac{1}{1-\lambda_2(\mathbf{P})} \geqslant \frac{1}{(1-c_0)\left(1-\cos\left(\frac{2\pi k}{\sqrt{n}}\right)\right)} \approx \frac{n}{2\pi^2(1-c_0)k^2}, \tag{4.9}$$

where $c_0 = \sqrt{\mathbf{P}_{11}}$

Proof: Based on Argument 4.2.1, it suffices to replace n by $\sqrt{n}$ in Bound (4.2.1) to conclude. ■

Comments: In [15], the authors claim that Theorem 4.2.2 is sufficient to conclude that the fastest mixing chain on lattices has a relaxation time that scales as $\Omega(n/k^2)$. As in the first paragraph of this section, they start by explaining that the fastest mixing chain follows the symmetries of the lattice

(by the convexity of the SLEM). They also conclude that for any i and j, its transition matrix coefficient $\mathbf{P}_{ij}$ should be a function f of $d_x = |x_i - x_j|$ and of $d_y = |y_i - y_j|$ only. Finally they notice that, if $\mathbf{P}$ is a Kronecker product $\mathbf{P}' \otimes \mathbf{P}'$, where $\mathbf{P}'$ is symmetric, circulant and stochastic, then

$$\begin{aligned} \mathbf{P}_{ij} &= (\mathbf{P}' \otimes \mathbf{P}')_{\sqrt{n}(x_i-1)+y_i, \sqrt{n}(x_j-1)+y_j} \\ &= \mathbf{P}'_{x_i,x_j} \mathbf{P}'_{y_i,y_j}, \end{aligned}$$

is indeed a function of the two distances d_x, d_y. They conclude that the transition matrix of the fastest mixing chain is necessarily a Kronecker product. However, up to our knowledge, there is no proof to show that the function f should be a *product* form, i.e. there is no reason to think that $f(d_x, d_y) = f_1(d_x) f_2(d_y)$. Imposing a product form seems very restrictive, and we did not manage to prove that point. This is why we develop the second argument, which leads to the following theorem:

Theorem* 4.2.1 (Smallest relaxation time on toroidal lattices). The relaxation time of the fastest symmetric mixing chain on a toroidal lattice of n nodes and radius k scales as:

$$\tau = \frac{1}{1 - |\lambda_2|} = \Theta\left(\frac{n}{k^2}\right).$$

Proof: Lowerbound: The transition matrix of the fastest mixing chain $\{X(t)\}_{t \geqslant 0}$ on a toroidal torus verifies the symmetry properties described in Argument 4.2.2 because of the convexity of the SLEM. Build now the induced Markov chain $\{Y(t)\}_{t \geqslant 0}$ as in Argument 4.2.2. $\{Y(t)\}_{t \geqslant 0}$ has a symmetric and circulant $\sqrt{n} \times \sqrt{n}$ transition matrix $\mathbf{P}$. By Theorem 4.2.1, the relaxation time of $\mathbf{P}$ scales at best (when $c_0 = 0$) as $\Omega(\sqrt{n}^2/k^2)$. Since the relaxation time of $\{X(t)\}_{t \geqslant 0}$ is lower bounded by the relaxation time of $\{Y(t)\}_{t \geqslant 0}$ by Argument 4.2.2, it is lower bounded by $\Omega(n/k^2)$.

Upperbound: Build the $\sqrt{n} \times \sqrt{n}$ symmetric, circulant and stochastic matrix $\mathbf{P}'$ parametrized by the δ-filter coefficients $c_0 = 0$, $c_1 = \delta, \ldots, c_{k-1} = \delta, c_k = 0.5 - (k-1)\delta$. Argument 4.2.1 applied to $\mathbf{P}'$ produces a Markov chain on the toroidal lattice with transition matrix $\mathbf{P}' \otimes \mathbf{P}'$. Its relaxation time scales as $\Theta(n/k^2)$, if δ is small enough. Hence, the relaxation time of the fastest mixing chain on the toroidal lattice scales as $O(n/k^2)$.

Combining the two points yields the theorem. ■

4.2.3 Fastest mixing chain on cycles

We know now how fast is the fastest mixing chain on a cycle, but we do not know the fastest mixing chain itself. We know that the δ-filter coefficients $c_0 = 0$, $c_1 = \delta, \ldots, c_{k-1} = \delta, c_k = 0.5 - (k-1)\delta$ offer order optimal relaxation times when δ is small enough [15], and it would be interesting to know how close are these coefficients from the optimal ones. This section produces an elegant near-optimal solution to the fastest mixing chain problem, and it intellectually completes the previous section, but it will not be useful in further chapters.

The fastest symmetric mixing chain on a cycle of connection radius k can be computed thanks to well-known convex optimization algorithms. Indeed, when restricted to symmetric matrices, minimizing the spectral radius is equivalent

to minimizing the spectral norm $|||\mathbf{W} - \mathbf{J}|||_2$, which is a convex optimization problem.

However, this section proposes a different approach, that makes a link between fastest mixing chain design and a famous algorithm in signal processing: the Remez algorithm.

A near-optimal solution using Chebyshev polynomials.

Problem statement: Given a cycle's length n and a connection radius k, what is the symmetric circulant matrix $\mathbf{W}$ verifying $\mathbf{W1} = \mathbf{1}$ that minimizes $\lambda_2(\mathbf{W})$, its second largest eigenvalue in magnitude, the largest one being 1?

Sketch of the proposed near-optimal solution: Our solution has two steps:

1. Run Remez algorithm (Alg. 4.1) as explained in page 99 until it outputs the coefficients $p_0, p_1, \ldots, p_k$ of a polynomial P.

2. Compute the near-optimal coefficients $c_0, \ldots, c_k$ by multiplying the inverse of a square matrix $\mathbf{T}$ containing Chebychev polynomial coefficients with the vector of the coefficients of P (Eq. 4.11).

As the reader can see, the solution is short, but it uses tools that seem unrelated with the problem. In order to slowly introduce the concepts, we propose a step by step constructive approach of our solution:

Let T_k be the kth Chebyshev polynomial: by definition, $T_k(\cos(\theta)) = \cos(k\theta)$. As seen in the previous section, the eigenvalues of $\mathbf{W}$ are:

$$\begin{aligned}
\xi^{(m)} &= \sum_{i=0}^{n-1} c_i \exp\left(\frac{2\pi jim}{n}\right) \\
&= c_0 + \sum_{i=1}^{k} 2c_i \cos\left(\frac{2\pi im}{n}\right) \\
&= c_0 + 2\sum_{i=1}^{k} c_i T_i\left(\cos\left(\frac{2\pi m}{n}\right)\right) \\
&= \varphi(\mathbf{W})\left(\cos\left(\frac{2\pi m}{N}\right)\right),
\end{aligned}$$

where $\varphi(\mathbf{W})$ is a polynomial of degree k. In other words, for any matrix $\mathbf{W}$, there is a polynomial $\varphi(\mathbf{W})$ of degree k such that the eigenvalues of $\mathbf{W}$ are the values taken by $\varphi(\mathbf{W})(x)$ at $x = 1, \cos(2\pi/n), \cos(4\pi/n), \ldots, \cos(2\pi(n-1)/n)$. Furthermore,

$$\varphi(\mathbf{W})(\cdot) = c_0 + 2\sum_{i=1}^{k} c_i T_i(\cdot).$$

The Chebyshev polynomials form a basis of the polynomial space because the degree of T_k is k for any non-negative integer k. Thus, φ is a bijective mapping between the set of symmetric circulant matrices such that $\mathbf{W1} = \mathbf{1}$ and $c_{i>k} = 0$ and the set of polynomials P of degree k verifying $P(1) = 1$. In order to compute the optimal polynomial $\widetilde{P}$, which gives the optimal matrix $\widetilde{\mathbf{W}}$ by using the inverse transform $\widetilde{\mathbf{W}} = \varphi^{-1}(\widetilde{P})$, one should solve the following optimization

program:

$$\widetilde{P} = \underset{\left\{\begin{array}{l} P \in \mathbb{R}_k[X] \\ P(1) = 1 \end{array}\right.}{\arg\min} \max_{m \in \{1,\ldots,n-1\}} |P(\cos(\frac{2\pi m}{n}))|. \tag{4.10}$$

The inverse transform φ^{-1} is easily obtained, without making any evolved considerations on the orthogonality of Chebyshev polynomials. These polynomials have been computed for a long time: $T_j(x) = \sum_{i=0}^{j} t_i^{(j)} x^i$, where the coefficients $t_i^{(j)}$ can be found in tables. For any connection radius k, build the $(k+1)\times(k+1)$ Chebyshev matrix $\mathbf{T}$ such that $\mathbf{T}_{ij} = t_i^{(j)}$. Then, if $\widetilde{P} = \sum_{i=0}^{k} p_i x^i$ is known, the corresponding matrix $\widetilde{\mathbf{W}} = \varphi^{-1}(\widetilde{P})$ has coefficients $c_0, c_1, \ldots, c_k$ such that:

$$\mathbf{T} \cdot \begin{pmatrix} c_0 \\ 2c_1 \\ \vdots \\ 2c_k \end{pmatrix} = \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_k \end{pmatrix}.$$

Therefore the coefficients can be computed with the following expression:

$$\begin{pmatrix} c_0 \\ 2c_1 \\ \vdots \\ 2c_k \end{pmatrix} = \mathbf{T}^{-1} \cdot \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_k \end{pmatrix}, \tag{4.11}$$

where $p_0, p_1, \ldots, p_k$ are the coefficients of the polynomial solving Eq. (4.10). For instance, here are the Chebyshev coefficients up to $k = 9$ gathered in the matrix $\mathbf{T}$:

$$\mathbf{T} = \left|\begin{array}{cccccccccc} 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -3 & 0 & 5 & 0 & -7 & 0 & 9 \\ 0 & 0 & 2 & 0 & -8 & 0 & 18 & 0 & -32 & 0 \\ 0 & 0 & 0 & 4 & 0 & -20 & 0 & 56 & 0 & -120 \\ 0 & 0 & 0 & 0 & 8 & 0 & -48 & 0 & 160 & 0 \\ 0 & 0 & 0 & 0 & 0 & 16 & 0 & -112 & 0 & 432 \\ 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & -256 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 64 & 0 & -576 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 128 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 256 \end{array}\right|. \tag{4.12}$$

The problem now is to solve the optimization program of Eq. (4.10). We start by considering the complete graph (k very large), where the solution of this optimization program is obvious.

A special case: the complete graph ($k \geqslant \lfloor n/2 \rfloor$) Suppose that n is odd. Gathering all the estimates in one step and computing the global average corresponds to taking $k = \lfloor n/2 \rfloor$ and $c_i = 1/n$ for all i. In other words, $\mathbf{W} = \mathbf{J}$ and the SLEM $\lambda_2(\mathbf{W}) = 0$. The polynomial mapped to this filter is obtained

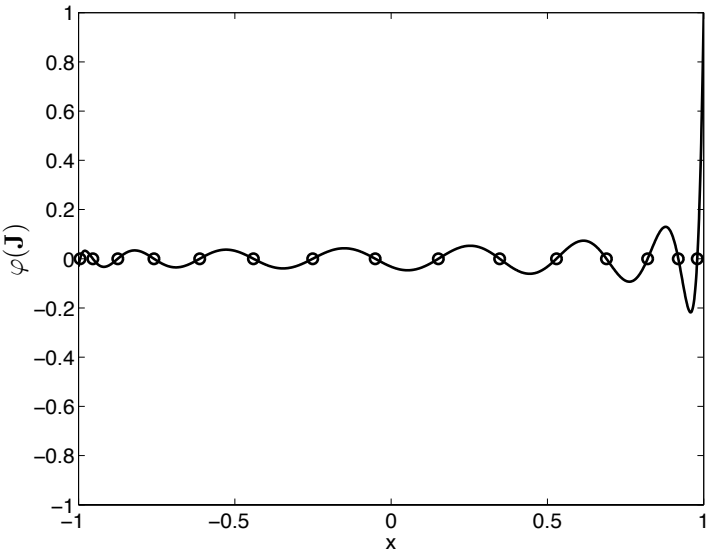


Figure 4.6: Polynomial $\varphi(\mathbf{J})$ associated with the exact filtering $\mathbf{W} = \mathbf{J}$. The circles are placed at $\cos(\frac{2\pi i}{n})$, $i = 1, \ldots, n-1$ and indicate where to read the eigenvalues. Here all the eigenvalues are equal to 0, except the one obtained for $i = 0$, which is equal to 1.

by a Lagrange interpolation (see Fig. 4.6):

$$P_{\lfloor \frac{n}{2} \rfloor}(x) = \frac{\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor}(x - \cos(\frac{2\pi i}{n}))}{\prod_{i=1}^{\lfloor \frac{n}{2} \rfloor}(1 - \cos(\frac{2\pi i}{n}))}. \tag{4.13}$$

The general case ($k < \lfloor n/2 \rfloor$) If $k < \lfloor n/2 \rfloor$, then, the SLEM $\lambda_2(\mathbf{W})$ is strictly positive. Indeed, with a degree smaller than $\lfloor n/2 \rfloor$, a polynomial cannot have $\lfloor n/2 \rfloor$ zeros and verify the constraint $P(1) = 1$.

In addition, the polynomials of degree $k \ll \lfloor n/2 \rfloor$ have very few oscillations around 0 compared to the number $\lfloor n/2 \rfloor$ of eigenvalues. Therefore, since the cosine values $\{\cos(2\pi i/n)\}_{i=1,\ldots,n-1}$ are fairly well spread out between -1 and $\cos(2\pi/n)$, there are high chances that these cosine values are not concentrated around the roots of the optimal polynomial, like in Fig. 4.6, and that some large eigenvalues are located around the top of the polynomial oscillations.

As a consequence, for $k \ll \lfloor n/2 \rfloor$, the discrete optimization program (4.10) is relaxed into a continuous optimization program, which minimizes the amplitude of the polynomial oscillations in $[-1, \cos(\frac{2\pi}{n})]$:

$$\widetilde{P} = \underset{\left\{\begin{array}{l} P \in \mathbb{R}_k[X] \\ P(1) = 1 \end{array}\right.}{\arg\min} \max_{x \in [-1, \cos(\frac{2\pi}{n})]} |P(x)|. \tag{4.14}$$

As a result, the solution of Program (4.14) is an approximation of the optimal polynomial of the discrete Program (4.10), especially when $k \ll \lfloor n/2 \rfloor$. By the Chebyshev alternation theorem [59, page 489], the optimal polynomial $\widetilde{P}$ of the continuous min-max program (4.14) is equiripple[7]. In other words, $\widetilde{P}(x)$ exhibits $k+1$ alternations: there are at least $k+1$ increasingly ordered values y_i in $[-1, \cos(\frac{2\pi}{n})]$ such that $\widetilde{P}(y_i) = -\widetilde{P}(y_{i+1}) = \pm \max_{x \in [-1, \cos(\frac{2\pi}{n})]} |\widetilde{P}(x)|$ for $i = 1, \ldots, k$.

The continuous min-max program can be algorithmically solved by a modified version of Remez's algorithm. The solution output by Remez algorithm is indeed an equiripple polynomial around 0 (see Fig. 4.7(b) and 4.7(d)). Interestingly, when using the inverse transform φ^{-1} on the Remez polynomial, the resulting coefficients $c_0, c_1, \ldots, c_{k-1}$ are not only small but also fairly identical (see caption of Fig. 4.7). This can be related to a variant of the δ-filter coefficients, where $\mathbf{c_0}, c_1, \ldots, c_{k-1}$ are *all* taken equal to a small constant δ. For $n = 20$ and $k = 3$, Fig. 4.7(a) shows that $\log(\|\boldsymbol{\epsilon}(t)\|)$ converges more slowly with $\delta = 0.025$ than with the Remez coefficients, which are all close to 0.05 except for c_k and c_{-k}. In another example (Fig. 4.7(c)), $n = 50$ and $k = 4$ give Remez coefficients all approximatively equal to 0.014 except for c_k and c_{-k}. Even a slight change of coefficients with $\delta = 0.010$ slows clearly down the convergence speed of the logarithmic error.

Description of Remez's algorithm Remez algorithm is described in Alg. 4.1. In a few words, Remez algorithm proceeds as follows: It starts by defining $x_0 = \cos(\frac{2\pi}{n})$ and by initializing $\alpha_i = -1 + 2i/k$ for $i = 1, \ldots, k-1$ (for k small enough, $\alpha_{k-1} < x_0$). These α_i values are updated in the iterations of the algorithm. They will converge to the optimal "ripple" position values y_i which were discussed in the Chebyshev alternation theorem. Note that only $k-1$ such α_i values are defined while the alternation theorem predicts $k+1$ ripples. Indeed 2 ripple positions are already known: $y_1 = -1$ and $y_{k+1} = \cos(2\pi/n) = x_0$. Finally, Remez algorithm outputs the coefficients of the polynomial with optimal ripple positions y_i, for $i = 1, \ldots, k-1$.

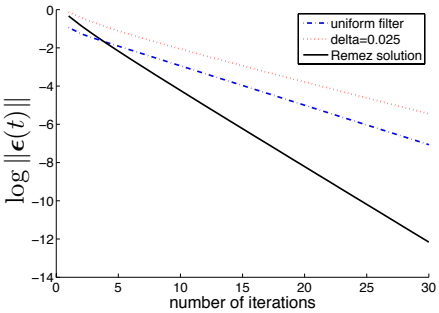
Comments:

- The equiripple polynomial that is obtained at convergence has oscillations of size Δ. Since $P(x_0) = \Delta$ and $P(x) \leq \Delta$ for all $x \in [-1, x_0]$, then $\lambda_2(\mathbf{W}) = \Delta$. This value Δ computed by Remez's algorithm is thus an upper bound of the best $\lambda_2(\mathbf{W})$ a filter of range k can achieve.

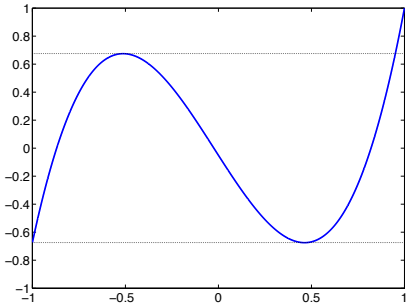
- It is interesting to notice that Remez's algorithm is extremely simple and that it converges in very few iterations (less than 8 in the examples shown in Fig. 4.7).

- We ran Remez algorithm on a few examples, and we computed the coefficients $c_0, c_1, \ldots, c_k$ using Eq. (4.11). When the connection radius k is fixed, but when the number n of nodes in the cycle increases, it was observed on simulations (Fig. 4.8) that the coefficients $c_0, c_1, \ldots, c_{k-1}$ decrease while c_k approaches 0.5. Furthermore, the coefficients different than $c_k = c_{-k}$ tend to equalize. This observation recalls the coefficients

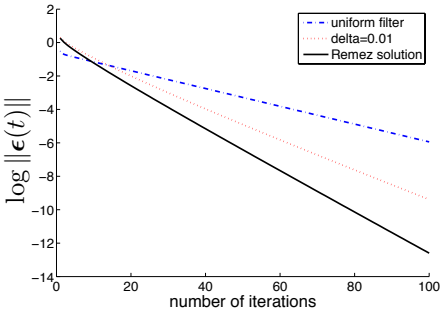
[7] In the present case, the theorem is adapted to the loss of one degree of freedom in the polynomial due to the constraint $P(1) = 1$.



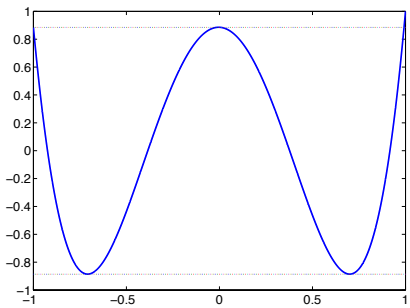
(a) Simulations on the 20 nodes cycle, $k = 3$



(b) Remez polynomial, $n = 20$, $k = 3$



(c) Simulations on the 50 nodes cycle, $k = 4$



(d) Remez polynomial, $n = 50$, $k = 4$

Figure 4.7: Remez's algorithm. Figures 4.7(a) and 4.7(c) compare the convergence speed of the logarithmic error $\log(\|\epsilon(t)\|)$ for the uniform filter ($c_0 = c_1 = \ldots = c_k = 1/(2k+1)$), for some δ-filters ($c_0 = c_1 = \ldots = c_{k-1} = \delta$), and for the Remez coefficients. In Fig. 4.7(a), the Remez coefficients are $c_0 = 0.05597$, $c_1 = 0.05531$, $c_2 = 0.05335$, $c_3 = 0.363355$, and in Fig. 4.7(c), the Remez coefficients are $c_0 = 0.01444$, $c_1 = 0.01441$, $c_2 = 0.01433$, $c_3 = 0.01419$, $c_4 = 0.44985$. Figures 4.7(b) and 4.7(d) show the equiripple polynomials output by Remez algorithm.

Algorithm 4.1: Remez algorithm

Let $x_0 = \cos(\frac{2\pi}{n})$ and initialize $\alpha_i = -1 + 2i/k$ for $i = 1, \ldots, k-1$. Let $(p_0, p_1, \ldots, p_k)$ be the coefficients of the searched polynomial and let Δ be the amplitude of its oscillations ("ripples") around 0 between $x = -1$ and $x = x_0$. At each iteration of the algorithm do:

1: Construct the $(k+2) \times (k+2)$ matrix M:

$$M = \begin{pmatrix} 1 & 1 & 1 & \ldots & \ldots & 1 & 0 \\ 1 & -1 & 1 & \ldots & \ldots & (-1)^k & (-1)^{k+1} \\ 1 & \alpha_1 & \alpha_1^2 & \ldots & \ldots & \alpha_1^k & (-1)^{k+2} \\ 1 & \alpha_2 & \alpha_2^2 & \ldots & \ldots & \alpha_2^k & (-1)^{k+3} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha_{k-1} & \alpha_{k-1}^2 & \ldots & \ldots & \alpha_{k-1}^k & 1 \\ 1 & x_0 & x_0^2 & \ldots & \ldots & x_0^k & -1 \end{pmatrix}.$$

2: Compute

$$\begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_k \\ \Delta \end{pmatrix} = M^{-1} \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

3: Define P as the polynomial $P(x) = p_0 + p_1 x + \ldots + p_k x^k$.

4: Find the new α_i, $i = 1, \ldots, k-1$ such that $P(\alpha_i)$ is a local minimum or maximum of $P_{|[-1,x_0]}$. End of the iteration, go back to the step 1.

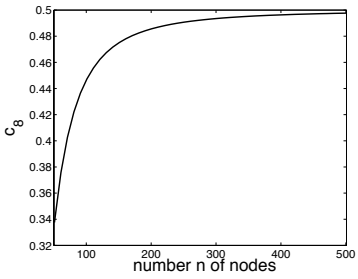
After a number of iterations (8 for example), the algorithms outputs $(p_0, p_1, \ldots, p_k, \Delta)$.

chosen in [15], where $c_1, \ldots, c_{k-1}$ were taken equal to a small δ, and c_k was taken close to 0.5. The intuition behind the shape of the coefficients is that long distance transitions speed up the convergence to the stationary distribution. In the distributed averaging setting, it means that averaging estimates with distant nodes in more efficient than averaging estimates with the closest neighbors.

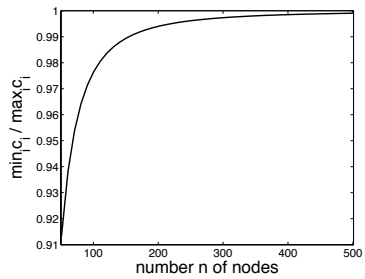
- Another interest of the Remez method is pedagogical. It helps understanding in an intuitive way why it is so difficult to design an efficient distributed averaging algorithm: in the case of a circular connectivity graph, it boils down to squeezing a polynomial of fixed degree in a tube as small as possible.

4.3 Trees

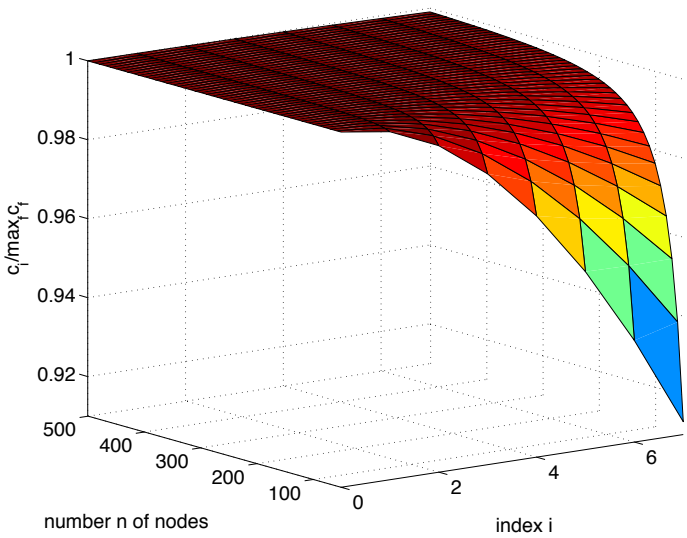
Just as in the previous one, this section is about the second largest eigenvalue of a stochastic matrix which sparsity forms a basic graph topology. While the previous section studied cycles, this section studies a specific class of trees called *balanced trees*. Studying trees is a natural complement to the study of



(a) Filter coefficient c_8



(b) $\min_{i=0,\ldots,7} c_i / \max_{i=0,\ldots,7} c_i$.



(c) Filter shapes: $\{c_0, c_1, \ldots, c_7\} / \max_{f=0,\ldots,7} c_f$

Figure 4.8: Remez filter coefficients $c_0, c_1, \ldots, c_8$ when $k = 8$ for varying circle sizes n. Figure (a) shows that an increasing weight is given to the furthest coefficient c_8 as n increases. At the same time, all coefficients $c_0, \ldots, c_7$ decrease as n increases (coefficients sum to 1). Not only these coefficients decrease, but they also tend to become identical, as shown in Figure (b). Finally Figure (c) shows the shape of the consecutive filters $c_0, c_1, \ldots, c_7$ (without c_8) normalized with their respective maximum coefficient. It appears that Remez coefficients are ordered in the following way: $c_0 > c_1 > \ldots > c_7 \lll c_8$.

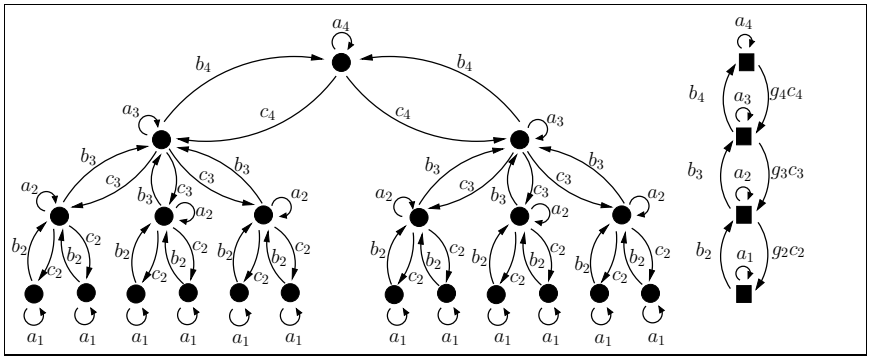


Figure 4.9: Balanced Markov chain on a balanced tree, and its induced Markov chain. In this example, $H = 4$, $g_2 = 2$, $g_3 = 3$ and $g_4 = 2$.

cycles, since by definition, a tree is a graph without cycles. Unlike the study of cycles, which focused only on double stochastic matrices, this section deals with stochastic matrices in general; it computes their stationary distributions, and more interestingly, it gives an upper bound on their relaxation times. Different transition matrices $\mathbf{P}$ can have the same stationary distribution $\boldsymbol{\pi}$, but their corresponding Markov chains do not converge to stationarity at the same rate. In many problems, including the object of this thesis, it is interesting to maximize the convergence rate, which is related to the relaxation time. It is usually admitted that a simple way to design an efficient Markov chain for a given stationary distribution $\boldsymbol{\pi}$ consists in constructing the transition coefficients starting with the root of the tree, and finishing with the leaves. In other words, taking care of the most problematic bottlenecks first is a good strategy. The bound on relaxation time developed here gives a formal justification to this intuition.

4.3.1 Markov Chain on Trees

In trees, communications between nodes i and j ($\mathbf{P}_{ij} > 0$) are allowed if and only if i and j form a parent/child pair. In order to derive analytical results, only a specific and highly symmetrical sub-class of trees are studied, and the stochastic matrices are assumed to follow these same symmetries (see Fig. 4.9). In this case, where the transitions of a Markov chain are invariant under the symmetries of the graph, we then say that the Markov chain is *balanced*. To our knowledge, previous work [2] on relaxation times on trees has only considered n-ary trees, which are a sub-case of the trees considered in this section, called balanced trees.

Definition* 4.3.1 (Balanced tree). A tree $\mathcal{T}$ is a balanced tree if and only if:

- All leaves are at the same distance from the root.
- All leaves of equal height have equal degree as well.

This section is articulated in 5 steps:

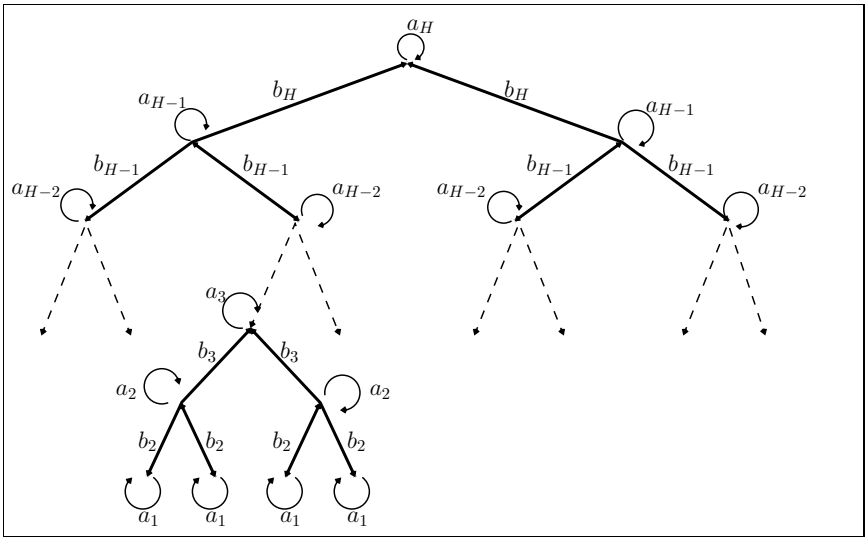


Figure 4.10: Doubly stochastic balanced Markov chain example (for any k, $b_k = c_k$).

1. For any balanced Markov chain, we compute an upper bound on the pseudo-relaxation time $1/(1-\xi_2)$.
2. We specify the bound in the case of doubly stochastic matrices, which are the matrices encountered in the analysis of distributed averaging algorithms.
3. To minimize the upper bound in the doubly stochastic case, we show that all the self-loops should be equal to zero, except at the leaves. This implies that the transition coefficients of the resulting Markov chain should be calculated starting from the root of the tree, i.e. from the main bottleneck of the graph.
4. We apply step 3 to p-ary trees by setting all the self-loops equal to 0, and we compute the resulting upper bound on the pseudo-relaxation time. This gives that the pseudo-relaxation time of the fastest mixing chain on a p-ary tree scales linearly in n.
5. We attempt to distinguish the case where the pseudo-relaxation time is equal to the relaxation time and the case where it is not, i.e. the case where $\xi_2 = \lambda_2$ and the case where $\xi_n = \lambda_2$.

Notations and assumptions Nodes of a balanced tree $\mathcal{T}$ are labelled in depth-first search order (this order is used to index the entries of the transition matrices $\mathbf{T}$). For any node i, let $h(i)$ be its height. The height of the root is denoted by H, and, naturally, the height of the leaves is 1. All the nodes of height $k \geq 2$ are assumed to have $g_k \geq 2$ children.

Let $\mathbf{T}$ (instead of $\mathbf{P}$) be a stochastic matrix corresponding to the transition matrix of a Markov chain on the tree $\mathcal{T}$. The transition probability between a

node i of height $h(i)$ and a node j of height $h(j)$ is assumed to be a function of $h(i)$ and $h(j)$ only. In particular, for any node i, $\mathbf{T}_{ii} = a_{h(i)}$ is called the self loop probability of node i. Note that all the nodes at a common height are assumed to share the same self loop value. For any node i at height $h(i)$, $b_{h(i)+1}$ is its transition probability to its parent, and $c_{h(i)}$ is its transition probability to any of its g_k children. A Markov chain with such a probability transition matrix $\mathbf{T}$ is called a *balanced Markov chain*. See Fig. 4.9 for an example of balanced Markov chain. These notations will be used until the end of this chapter.

4.3.2 Stationary distribution and induced Markov chain.

Theorem* 4.3.1 (Stationary distribution $\boldsymbol{\pi}_{\mathcal{T}}$). Let $\mathcal{T}$ be a balanced tree with n nodes, on which is defined a balanced Markov chain as described above. Define the following weights $\tilde{w}_i$, for $i \in [1, n]$:

$$\tilde{w}_i = \prod_{k=h(i)+1}^{H} \frac{c_k}{b_k}.$$

Then, the stationary distribution of the balanced Markov chain on $\mathcal{T}$ is:

$$\boldsymbol{\pi}_{\mathcal{T}}(i) = \frac{\tilde{w}_i}{\sum_{k=1}^{n} \tilde{w}_k}.$$

The proof of Theorem 4.3.1 introduces an induced Markov chain with a transition matrix denoted by $\mathbf{A}_H$, which will be used again in the next section.
Proof: Because of the symmetry of the Markov chain, a simpler chain can be introduced, which allows to derive in a simple way many characteristics of the original chain. The H states of this new chain are the H disjoint sets of nodes having equal depth in the tree $\mathcal{T}$. Clearly, the state representing the nodes with height k in $\mathcal{T}$ is denoted by k. This new chain is also a Markov chain and will be called the induced Markov chain. It simply describes the depth dynamics of the original Markov chain without differentiating the branches of the tree.

The transition matrix $\mathbf{P}$ of the induced Markov chain is defined as follows:

- $p_{k,k} = a_k$ for $k = 1, \ldots, H$.
- $p_{k-1,k} = b_k$ for $k = 2, \ldots, H$.
- $p_{k,k-1} = g_k c_k$ for $k = 2, \ldots, H$.

In order to determine the stationary distributions $\boldsymbol{\pi}_{induced}$ and $\boldsymbol{\pi}_{\mathcal{T}}$ and to derive further computations, define the following sequence of matrices:

$$\begin{aligned}
\mathbf{A}_1 &= a_1, \\
\mathbf{A}_k &= \left(\begin{array}{cccc|c}
 & & & & 0 \\
 & & \mathbf{A}_{k-1} & & \vdots \\
 & & & & 0 \\
 & & & & b_k \\
\hline
0 & \cdots & 0 & g_k c_k & a_k
\end{array}\right),
\end{aligned}$$

for any $k \in [2, H]$. Observe that $\mathbf{A}_H = \mathbf{P}$. Next, define a diagonal matrix $\mathbf{W}$ of weights such that $\mathbf{W}^{\frac{1}{2}}\mathbf{P}\mathbf{W}^{-\frac{1}{2}} = \mathbf{S}$ is symmetric:

$$\mathbf{W} = \operatorname{diag}\left(\prod_{k=2}^{H} \frac{g_k c_k}{b_k}, \prod_{k=3}^{H} \frac{g_k c_k}{b_k}, \ldots, \frac{g_H g_{H-1} c_H c_{H-1}}{b_H b_{H-1}}, \frac{g_H c_H}{b_H}, 1\right). \tag{4.15}$$

Let w_i be the ith element of $\mathbf{W}$. We show now that, for any $i = 1, \ldots, H$, w_i is actually the weight of state i in the induced stationary distribution $\boldsymbol{\pi}_{induced}$:

$$\boldsymbol{\pi}_{induced}(i) = \frac{w_i}{\Sigma_{k=1}^{H} w_k}. \tag{4.16}$$

Indeed, $(\mathbf{W1})^T\mathbf{P} = (\mathbf{1}^T\mathbf{W})(\mathbf{W}^{-\frac{1}{2}}\mathbf{S}\mathbf{W}^{\frac{1}{2}}) = \mathbf{1}^T\mathbf{W}^{\frac{1}{2}}\mathbf{S}\mathbf{W}^{\frac{1}{2}} = (\mathbf{W}^{\frac{1}{2}}\mathbf{S}\mathbf{W}^{\frac{1}{2}}\mathbf{1})^T = (\mathbf{WP1})^T = (\mathbf{W1})^T$.

The stationary distribution of the Markov chain on the tree $\mathcal{T}$ can now be deduced by noticing that multiplying the number of nodes at height i by $\boldsymbol{\pi}_{\mathcal{T}}(i)$ gives $\boldsymbol{\pi}_{induced}(i)$. Indeed all the nodes at the same height in $\mathcal{T}$ are equiprobable in the stationary distribution (by the symmetry of $\mathcal{T}$).

$$\begin{aligned} \boldsymbol{\pi}_{\mathcal{T}}(i) &= \frac{\boldsymbol{\pi}_{induced}}{\prod_{k=i+1}^{H} g_k} \\ &= \frac{1}{\Sigma_{k=1}^{H} w_k} \prod_{k=i+1}^{H} \frac{c_k}{b_k}. \end{aligned}$$

Simplifying this equation leads to the theorem. ■

4.3.3 Upper Bound on the Pseudo-Relaxation Time

Let $\xi_2(\mathbf{T})$ be the second largest eigenvalue (SLE) of the transition matrix $\mathbf{T}$, and let $\tau' = \frac{1}{1-\xi_2(\mathbf{T})}$ be the *pseudo-relaxation time* of $\mathbf{T}$. The goal of this section is to derive an upper bound on τ', which is tighter than the bound that can be computed with the Poincaré inequality.

Let S_k be any subtree of $\mathcal{T}$ whose root is at height k in $\mathcal{T}$; S_k contains its root's complete line of descent in $\mathcal{T}$. This section shows that the pseudo-relaxation time τ' of a balanced Markov chain on $\mathcal{T}$ can be upper bounded as follows:

Theorem* 4.3.2 (Pseudo-relaxation time on a balanced tree). The pseudo-relaxation time τ' of a balanced Markov chain on a balanced tree is upper bounded as follows:

$$\tau' = \frac{1}{1-\xi_2} \leqslant \sum_{k=1}^{H-1} \frac{\sum_{i \in S_k} \tilde{w}_i}{\tilde{w}_k b_{k+1}}, \tag{4.17}$$

where the weights $\tilde{w}_k$ were defined in Theorem 4.3.1.

This theorem illustrates the importance of the root of $\mathcal{T}$ as a bottleneck in the relaxation time. The coefficients b_k have more impact on the relaxation time for large k's because the subtrees S_k have more nodes and thus larger total weight when k is large. In addition, the relaxation time is relatively

small when the stationary distribution has heavy weights close to the root and lighter weights close to the leaves, compared to stationary distributions with large probabilities around the leaves.

Finally, note that Theorem 4.3.2 considers the SLE and not the SLEM. For the theorem to make practical sense, it is necessary that $|\xi_n| < \xi_2$. Simulations and discussions in the last section of this chapter show that as long as $\mathbf{T}$ does not have negative coefficients, then the inequality holds in most cases. However this particular point remains to be clarified and proven. In addition, simulations in the last section will show that, as the self loop coefficients decrease and become negative, ξ_n becomes rapidly smaller than -1. The question "$\xi_2 = \lambda_2$?" being delicate, we postpone it to the end of the chapter. Until then we will focus on the SLE ξ_2 only.

To prove Theorem 4.3.2, we need a succession of 4 lemmas:

Lemma* 4.3.1 (Eigenvalues of T). The eigenvalue of $\mathbf{T}$ are the eigenvalues of $\mathbf{A}_k$, $k = 1, \ldots, H$.

With the same proof as for the lattice in Argument 4.2.2, we can show that the eigenvalues of the induced chain, i.e. the eigenvalues of $\mathbf{P} = \mathbf{A}_H$ are also eigenvalues of $\mathbf{T}$. Lemma 4.3.1 thus states a stronger result, since it characterizes *all* the eigenvalues of $\mathbf{T}$ based on the transition matrix $\mathbf{A}_H$ of the induced Markov chain and on its sub-matrices $\mathbf{A}_k$, for $k = 1, \ldots, H-1$.

Proof: We can construct a recursive sequence of matrices $\{\mathbf{T}_i\}_{i=1,\ldots,H}$ such that $\mathbf{T} = \mathbf{T}_H$. For example, in a binary tree ($g_k = 2$):

$$\mathbf{T}_1 = a_1,$$

$$\mathbf{T}_k = \left(\begin{array}{c|ccc|ccc}
a_k & c_k & 0 \cdots & 0 & c_k & 0 \cdots & 0 \\
\hline
b_k & & & & & & \\
0 & & & & & & \\
\vdots & & \mathbf{T}_{k-1} & & & 0 & \\
0 & & & & & & \\
\hline
b_k & & & & & & \\
0 & & & & & & \\
\vdots & & 0 & & & \mathbf{T}_{k-1} & \\
0 & & & & & &
\end{array}\right),$$

for $k \in [2, H]$.

Let $\chi_k(x) = \det(\mathbf{T}_k - xI)$ be the characteristic polynomial of $\mathbf{T}_k$. Note that the eigenvalues of $\mathbf{T}$ are the roots of χ_H. Developing the determinant gives a second order recursive equation:

$$\chi_{k+1}(x) = \chi_k^{g_{k+1}-1}(x) \left[(a_{k+1} - x)\chi_k(x) - g_{k+1}\ b_{k+1}\ c_{k+1}\ \chi_{k-1}^{g_k}(x)\right]. \quad (4.18)$$

It is helpful to introduce the following Sturm polynomial sequence:

$$\begin{cases} P_0(x) = 1. \\ P_1(x) = a_1 - x. \\ P_k(x) = (a_k - x)P_{k-1} - g_k\ b_k\ c_k\ P_{k-2}(x). \end{cases} \quad (4.19)$$

Note that for any k, P_k is the characteristic polynomial of A_k. The next part of the proof shows by induction that for any k, χ_k can be factorized with all the Sturm polynomials P_i, $i = 1, \ldots, k$. Therefore, the roots of χ_H are the roots of the polynomials P_i, $i = 1, \ldots, H$, which are the eigenvalues of the matrices A_i, $i = 1, \ldots, H$.
The induction hypothesis at step k is stated as following: "There is a polynomial Q_k which is a product of all P_i for $i = 1, \ldots, k-1$ such that:

$$\begin{aligned} \chi_k &= P_k Q_k \\ \chi_{k-1}^{g_k} &= P_{k-1} Q_k." \end{aligned}$$

This is true for $n = 2$ with $Q_2 = P_1^{g_2 - 1}$. Indeed,

$$\begin{aligned} \chi_2 &= P_2 Q_2 \\ \chi_1^{g_2} &= P_1 Q_2. \end{aligned}$$

Suppose that the hypothesis is true at step k. Then, the following equations show that the hypothesis is true at step $k+1$ by taking $Q_{k+1} = \chi_k^{g_{k+1}-1} Q_k$, which is indeed a product of all Sturm polynomials until P_k:

$$\begin{aligned} \chi_{k+1}(x) &= \chi_k^{g_{k+1}-1}(x) \left[(a_{k+1} - x) P_k(x) Q_k(x) - g_{k+1}\ b_{k+1}\ c_{k+1}\ P_{k-1}(x) Q_k(x) \right]. \\ &= Q_{k+1}(x) P_{k+1}(x). \\ Q_{k+1} P_k &= \chi_k^{g_{k+1}-1} Q_k P_k \\ &= \chi_k^{g_{k+1}}. \end{aligned}$$

This concludes the proof of Lemma 4.3.1 ■

Lemma 4.3.1 and its proof have many implications thanks to various properties of Sturm polynomials. First it is easy to notice that the characteristic polynomial χ_H has degree n whereas a Sturm polynomial P_k has degree $k \leqslant \log_2 n$; indeed, $k = H$ is the Sturm polynomial highest degree and each internal node in $\mathcal{T}$ is assumed to have at least two children ($H \leqslant \log_2 n$). The factorization of χ_H with polynomials of small degree is already a great simplification. However, the most interesting property of a Sturm sequence of polynomials is that their zeros interlace. This implies that for any k, P_k has k distinct roots. Furthermore the eigenvalues $\xi_1 = 1$ and ξ_2 can be localized in the following way:

Lemma* 4.3.2. The Sturm polynomials verify

$$\begin{aligned} P_H(1) &= 0. \\ P_{H-1}(\xi_2) &= 0. \end{aligned}$$

Proof: Since the zeros of Sturm polynomials interlace, the largest root of P_H is the largest root of all the roots of the polynomials $P_0, P_1, \ldots, P_H$. But these zeros are the eigenvalues of $\mathbf{T}$ as shown in Lemma 4.3.1. Thus, the largest eigenvalue of $\mathbf{T}$ being equal to 1, 1 is the largest zero of P_H: $P_H(1) = 0$.

For the same reason, the second largest root of the set of polynomials $P_0, P_1, \ldots, P_H$ is equal to ξ_2. Since the zeros interlace, the second largest root of the whole set is the largest root of P_{H-1}: $P_{H-1}(\xi_2) = 0$. ■

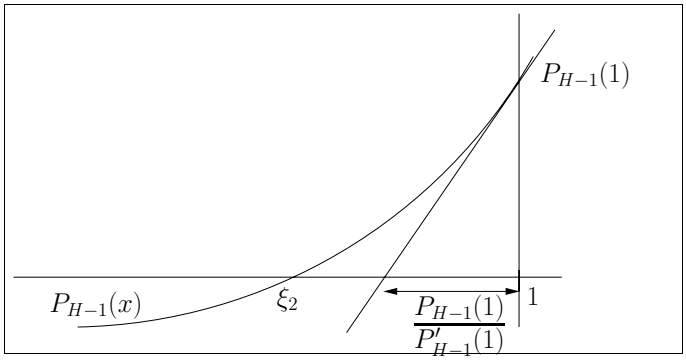


Figure 4.11: Tangent argument.

The following lemma is a key remark to prove Theorem 4.3.2. Because the Sturm polynomial P_{H-1} has degree $H-1$ and has $H-1$ distinct roots smaller than 1, its tangent in $x=1$ hits the x-axis *after* ξ_2 (see Fig. 4.11). In other words,

Lemma* 4.3.3.

$$\begin{aligned} 1-\xi_2 &\geqslant \frac{P_{H-1}(1)}{P'_{H-1}(1)}. \\ \tau' &\leqslant \frac{P'_{H-1}(1)}{P_{H-1}(1)}. \end{aligned}$$

Proof: The tangent argument explained just before the statement of the Lemma is sufficient to prove the lemma (see Fig. 4.11). ■

To prove Theorem 4.3.2, we thus need to derive P'_{H-1}/P_{H-1}, which will be done by induction. For that purpose a last lemma is needed:

Lemma* 4.3.4. For any $k=1,\ldots,H-1$, $P_k(1)=\prod_{j=2}^{k+1}(-b_j)$.

Proof: (by induction) For $k=1$: $P_1(1)=a_1-1=-b_2$ because $\mathbf{T}$ is a stochastic matrix (the sum of its rows is equal to 1, thus $a_1+b_2=1$).

Assume now that the property is true until rank $k-1$. The stochasticity of $\mathbf{T}$ implies that for any $1<k<H-1$, $a_k+b_{k+1}+g_kc_k=1$. Then, by the recursive definition of Sturm polynomials,

$$\begin{aligned} P_k(1) &= (a_k-1)P_{k-1}(1)-g_kb_kc_kP_{k-2}(1) \\ &= (-b_{k+1}-g_kc_k)\prod_{j=2}^{k}(-b_j)-g_kb_kc_k\prod_{j=2}^{k-1}(-b_j) \\ &= \prod_{j=2}^{k+1}(-b_j). \end{aligned}$$

■

We are now ready to prove Theorem 4.3.2.

Proof:[of Theorem 4.3.2] Differentiating Eq. (4.19), using Lemma 4.3.4, and because for any $1 < i < H-1$, $a_i + b_{i+1} + g_i c_i = 1$, we get that for any $1 \leqslant k \leqslant H-2$,

$$\begin{aligned}
\frac{P'_{k+1}(1)}{P_{k+1}(1)} &= -\frac{P_k(1)}{P_{k+1}(1)} + \frac{(a_{k+1}-1)P'_k(1)}{P_{k+1}(1)} - \frac{g_{k+1}b_{k+1}c_{k+1}P'_{k-1}(1)}{P_{k+1}(1)} \\
\frac{P'_{k+1}(1)}{P_{k+1}(1)} &= \frac{1}{b_{k+2}} + \frac{-b_{k+2}-g_{k+1}c_{k+1}}{-b_{k+2}}\frac{P'_k(1)}{P_k(1)} - \frac{g_{k+1}b_{k+1}c_{k+1}}{b_{k+1}b_{k+2}}\frac{P'_{k-1}(1)}{P_{k-1}(1)} \\
\frac{P'_{k+1}(1)}{P_{k+1}(1)} - \frac{P'_k(1)}{P_k(1)} &= \frac{1}{b_{k+2}} + \frac{g_{k+1}c_{k+1}}{b_{k+2}}\left(\frac{P'_k(1)}{P_k(1)} - \frac{P'_{k-1}(1)}{P_{k-1}(1)}\right). \qquad (4.20)
\end{aligned}$$

For $k \in [1, H-1]$, define $x_k = \frac{P'_k(1)}{P_k(1)} - \frac{P'_{k-1}(1)}{P_{k-1}(1)}$. Then, simple inductive computations on Eq.(4.20) give:

$$\begin{aligned}
x_1 &= \frac{1}{b_2} \\
x_k &= \frac{y_k}{b_{k+1}}, \text{for } k \in [1, H-1]. \qquad (4.21)
\end{aligned}$$

where the sequence y_n is defined by induction as

$$\begin{aligned}
y_1 &= 1 \\
y_k &= 1 + \frac{g_k c_k}{b_k} y_{k-1} \\
&= 1 + \frac{g_k c_k}{b_k} + \frac{g_k g_{k-1} c_k c_{k-1}}{b_k b_{k-1}} + \ldots + \prod_{i=2}^{k} \frac{g_i c_i}{b_i} \\
&= \frac{\sum_{i \in S_n} \tilde{w_i}}{\tilde{w_n}}. \qquad (4.22)
\end{aligned}$$

Finally, note that $\frac{P'_0(1)}{P_0(1)} = 0$. Thus,

$$\begin{aligned}
\frac{P'_{H-1}(1)}{P_{H-1}(1)} &= \sum_{k=1}^{H-1}\left(\frac{P'_k(1)}{P_k(1)} - \frac{P'_{k-1}(1)}{P_{k-1}(1)}\right) \\
&= \sum_{k=1}^{H-1} x_k. \qquad (4.23)
\end{aligned}$$

Combining Eq. 4.23 with Eq. 4.21, Eq. 4.22, and Lemma 4.3.3 ends the proof of Theorem 4.3.2. ■

4.3.4 Applications to Doubly Stochastic Matrices

The case where $\mathbf{T}$ is doubly stochastic is the one of interest in this thesis. This is why we explore the consequences of Theorem 4.3.2 in that special case. This will allow us to derive for example the best performance of synchronized average consensus on balanced trees. Most importantly, it will show that the price to pay for the presence of a bottleneck is higher than one can imagine. In particular we will show that the speed of convergence of pairwise gossip scales

identically on a balanced tree and on a lattice, even though the diameter of the lattice scales as $\Theta(\sqrt{n})$, whereas the diameter of a balanced tree scales as $\Theta(\log n)$ only.

Upperbound on the pseudo-relaxation time

Note that a *balanced* Markov chain with uniform stationary distribution necessarily has a symmetric and doubly stochastic transition matrix $\mathbf{T}$ (see the recursive definition of $\mathbf{T}$ at page 107). In other words, for any height k, $b_k = c_k$ and the weights

$$\tilde{w_i} = \prod_{k=h(i)+1}^{H} \frac{c_k}{b_k}.$$

are all equal to 1 (Fig. 4.10 for a general illustration of a doubly stochastic and balanced Markov chain). Theorem 4.3.2 can thus be restated as follows:

Theorem* 4.3.3 (Pseudo-relaxation time on a balanced tree: doubly stochastic case). The pseudo-relaxation time of a symmetric balanced Markov chain on a balanced tree is upper bounded as follows:

$$\tau' = \frac{1}{1-\xi_2} \leqslant \sum_{k=1}^{H-1} \frac{|S_k|}{b_{k+1}}, \tag{4.24}$$

where $|S_k|$ is the number of nodes of a subtree that has a root at height k.

Tightness: In Fig. 4.12, we compare the spectral gap $1-\xi_2 = 1/\tau'$ to its lower bound (the inverse of the right hand side of Eq. 4.24) by displaying their relative error in a logarithmic scale. The computations were done on ternary trees of growing heights H. Figure 4.12 suggests that the error decreases at least exponentially in H. Showing the tightness of the bound remains to be proved in further work.

Example 1*:* Theorem 4.3.3 is very simple to use. Take for example the tree drawn in Fig. 4.9 at page 103. Counting the number of nodes in each subtree gives: $|S_3| = 10$, $|S_2| = 3$ and $|S_1| = 1$ ($|S_1|$) is always equal to 1, by definition of a leaf). Therefore, for this tree,

$$\tau' \leqslant \frac{10}{b_4} + \frac{3}{b_3} + \frac{1}{b_2}.$$

Example 2*:* A crude upper bound can easily be derived from Theorem 4.3.3, using the fact that each generation has at least $p_k \geqslant 2$ children (see Fig. 4.13):

$$\tau' \leqslant \frac{n}{\min_{k=2,\ldots,H} b_k}.$$

Example 3*:* If the tree is p-ary with $p \geqslant 2$, then the inequality can be refined:

$$\tau' \leqslant \frac{n}{(p-1)\min_{k=2,\ldots,H} b_k}.$$

Indeed, note that for any k, $|S_k| \leqslant n/p^{H-k}$ because there are p^{H-k} nodes at height k, and each of these nodes generate a subtree S_k. The union of these

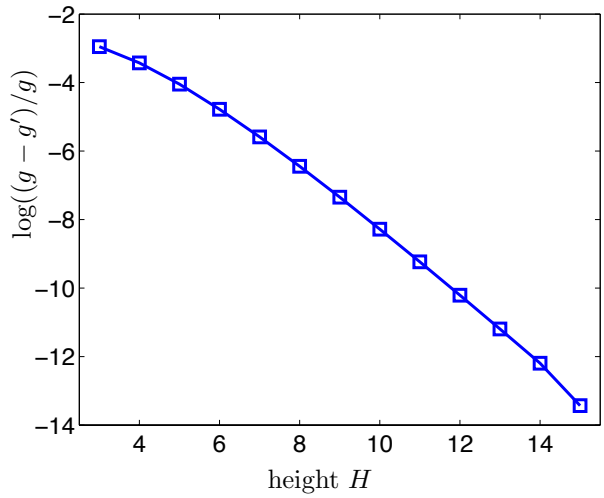


Figure 4.12: The logarithmic relative error between the spectral gap $g = 1 - \xi_2$ and its lower bound $g' = 1/(\sum_{k=1}^{H-1} |S_k| / b_{k+1})$ as a function of the tree height H on ternary trees.

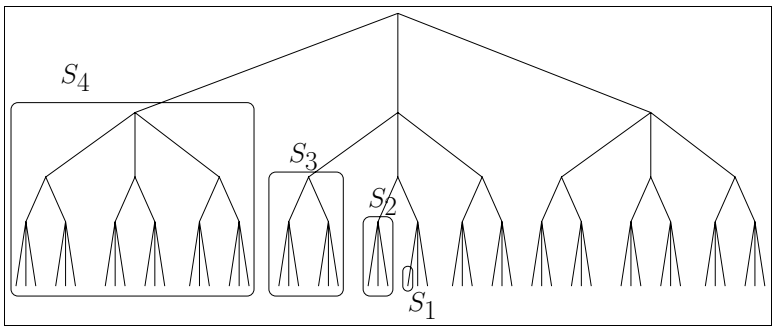


Figure 4.13: Visual argument to show that, if $p_k \geqslant 2$ for any height k, then $\sum_{k=1}^{H-1} |S_k| < N$.

subtrees contains $p^{H-k}\,|S_k| < n$ nodes. As a consequence,

$$\begin{aligned}
\tau' &\leqslant \frac{1}{\min_{k=2,\dots,H} b_k} \sum_{k=1}^{H-1} |S_k| \\
&\leqslant \frac{1}{\min_{k=2,\dots,H} b_k} \sum_{k=1}^{H-1} \frac{n}{p^{H-k}} \\
&= \frac{n}{\min_{k=2,\dots,H} b_k} \sum_{k=1}^{H-1} \frac{1}{p^k} \\
&= \frac{n}{\min_{k=2,\dots,H} b_k} \frac{1-1/p^{H-1}}{p-1} \\
&\leqslant \frac{n}{(p-1)\min_{k=2,\dots,H} b_k}.
\end{aligned}$$

As a consequence, relaxation times linear in n are achievable. If the bound is tight, then n^2 messages are necessary to run synchronized distributed average consensus on a balanced tree. Note that the lattice requires the same number of messages (see Th. 4.2.1).

Fastest symmetric mixing chain on balanced trees

We minimize here the upper bound stated in Theorem 4.3.3, which gives an upper bound on the optimal pseudo-relaxation time τ'. In order to guarantee that $\xi_n > -1$, it is sufficient, and thus practical, to forbid negative coefficients in $\mathbf{T}$ (Perron-Frobenius Theorem). This section shows that, under this constraint, and in the case of *doubly stochastic* matrices at least, **the chain that minimizes the upper bound is the unique chain with all self-loops $\{\mathbf{a_k}\}_{\mathbf{k>1}}$ set equal to 0**. In other words, regarding the upper bound, the best coefficients b_k are easily computable, starting from the root of the tree and ending with its leaves, in the following way:

$$\begin{aligned}
b_H &= \frac{1}{g_H} \\
b_{H-1} &= \frac{1-\frac{1}{g_H}}{g_{H-1}} \\
&= \frac{1}{g_{H-1}} - \frac{1}{g_{H-1}g_H} \\
b_{H-2} &= \frac{1}{g_{H-2}} - \frac{1}{g_{H-2}g_{H-1}} + \frac{1}{g_{H-2}g_{H-1}g_H} \\
&\vdots \qquad (4.25) \\
b_{H+1-n} &= -\sum_{k=1}^{n}\prod_{j=1}^{k} \frac{-1}{g_{H-n+j}} \quad \text{for} \quad n = 1,\dots,H-1, \qquad (4.26)
\end{aligned}$$

where the last equation is obtained by straightforward top-down induction. It is interesting to see that we need to start to compute the coefficients b_k starting from the root coefficient b_H down the leaf coefficient b_2. In other words, the

priority is to optimize the coefficients close to the bottleneck of the tree first. Note that it is recommended in [68] to keep this priority in mind when designing a Markov chain on any graph that has a bottleneck. The reader can find in [68] some algorithms that help reducing the relaxation time by precisely increasing the flow through bottlenecks.

To prove that the coefficients in Eq. (4.26) minimize the upperbound, it is sufficient to apply Lagrangian optimization techniques. The function to be minimized is:

$$f(b_2, \ldots, b_H) = \sum_{k=1}^{H-1} \frac{|S_k|}{b_{k+1}},$$

under the constraints (equivalent to $a_k \geqslant 0$ for $k = 2, \ldots, H$):

$$\begin{aligned} g_H b_H &\leqslant 1 && (4.27)\\ b_H + g_{H-1} b_{H-1} &\leqslant 1 && (4.28)\\ &\vdots \\ b_3 + g_2 b_2 &\leqslant 1. && (4.29) \end{aligned}$$

Theorem* 4.3.4 (Optimal transition coefficients). The above optimization program is minimized for coefficients $a_k = 0$, for $k = 2, \ldots, H$, which lead to the coefficients $\{b_k\}_{k=2,\ldots,H}$ defined in Eq. (4.26), and to a coefficient a_1 which is not necessarily equal to zero.

Proof: The Lagrangian of this convex optimization program is:

$$L(f, \mu_2, \ldots, \mu_H) = \sum_{k=1}^{H-1} \frac{|S_k|}{b_{k+1}} + \sum_{k=2}^{H-1} \mu_k (g_k b_k + b_{k+1} - 1) + \mu_H (g_H b_H - 1), \quad (4.30)$$

where $\mu_2, \ldots, \mu_H$ are the lagrangian multipliers. Differentiating the lagrangian and setting the partial derivatives to 0 gives:

$$\begin{aligned} \frac{-|S_{H-1}|}{b_H^2} + \mu_{H-1} + g_H \mu_H &= 0 && (4.31)\\ &\vdots \\ \frac{-|S_2|}{b_3^2} + \mu_2 + g_3 \mu_3 &= 0 \\ \frac{-1}{b_2^2} + g_2 \mu_2 &= 0. && (4.32) \end{aligned}$$

To show that the announced coefficients $\{b_k\}, k = 2, \ldots, H$ are indeed optimal, one should check that the KKT conditions are satisfied. First notice that the announced coefficients are the unique ones for which all the constraints from (4.27) to (4.29) are tight. Then, note that for any given set of coefficients $\{b_k\}, k = 2, \ldots, H$, there is only one set of Lagrange multipliers $\{\mu_k\}, k = 2, \ldots, H$ such that all equations from (4.31) to (4.32) hold. Therefore, what is left to show is that the Lagrange multipliers obtained by reversing the system of equations (4.31) to (4.32) with the coefficients $\{b_k\}, k = 2, \ldots, H$

defined by tight constraints, are all positive:

$$\forall k = 2, \ldots, H, \mu_k > 0.$$

This can be proven by induction on k. First notice that μ_2 is indeed positive:

$$\mu_2 = \frac{1}{g_2 b_2^2} > 0.$$

Then, since each coefficient b_k is equal to a decaying alternate sum, each b_k verifies:

$$\forall k = 2, \ldots, H-1, \quad \frac{1}{g_k} - \frac{1}{g_k g_{k+1}} \leqslant b_k \leqslant \frac{1}{g_k}. \tag{4.33}$$

Suppose now that $\mu_i > 0$ for all $i = 2, \ldots, k$, with $k < H$. The goal is to prove that $\mu_{k+1} > 0$. Note that $\mu_{k-1} \geqslant 0$ implies that:

$$\mu_k = \frac{1}{g_k}\left(\frac{|S_{k-1}|}{b_k^2} - \mu_{k-1}\right) \leqslant \frac{|S_{k-1}|}{g_k b_k^2},$$

with the convention $\mu_1 = 0$ in order to handle the case $k = 2$. Therefore,

$$\begin{aligned} \mu_{k+1} &= \frac{1}{g_{k+1}}\left(\frac{|S_k|}{b_{k+1}^2} - \mu_k\right) \\ &\geqslant \frac{1}{g_{k+1}}\left(\frac{|S_k|}{b_{k+1}^2} - \frac{|S_{k-1}|}{g_k b_k^2}\right). \end{aligned} \tag{4.34}$$

Thus, it remains to prove that

$$\frac{|S_k|}{b_{k+1}^2} - \frac{|S_{k-1}|}{g_k b_k^2} > 0. \tag{4.35}$$

Equation (4.35) is equivalent to (remember that constraints are assumed to be tight: $b_{k+1} + g_k b_k = 1$):

$$\begin{aligned} \frac{|S_k|}{|S_{k-1}|} > \frac{b_{k+1}^2}{g_k b_k^2} \quad &\Leftrightarrow \quad \frac{1 + g_k |S_{k-1}|}{|S_{k-1}|} > \frac{1}{g_k}\left(\frac{1}{b_k} - g_k\right)^2 \\ &\Leftrightarrow \quad g_k + \frac{1}{|S_{k-1}|} > \frac{1}{g_k}\left(\frac{1}{b_k} - g_k\right)^2. \end{aligned} \tag{4.36}$$

By Eq. (4.33),

$$\begin{aligned} b_k &\geqslant \frac{1}{g_k}\left(1 - \frac{1}{g_{k+1}}\right) \\ \frac{1}{b_k} &\leqslant g_k \frac{g_{k+1}}{g_{k+1} - 1}. \end{aligned}$$

Using this last inequality, the right hand side of Eq. 4.36 is lower bounded by:

$$\begin{aligned}\frac{1}{g_k}\left(\frac{1}{b_k}-g_k\right)^2 &\leqslant \frac{1}{g_k}\left(\frac{g_k}{g_{k+1}-1}\right)^2\\ &= \frac{g_k}{(g_{k+1}-1)^2}.\end{aligned}$$

Since $g_{k+1} \geqslant 2$,

$$\begin{aligned}\frac{1}{g_k}\left(\frac{1}{b_k}-g_k\right)^2 &\leqslant g_k\\ &< g_k+\frac{1}{|S_{k-1}|},\end{aligned}$$

which proves Eq. 4.36, which implies Eq. 4.35, which leads to $\mu_{k+1} > 0$. ■

Pseudo-relaxation time on p-ary trees

For p-ary trees, coefficients g_k are all equal to a constant p, which simplifies equations and allows to compute the "optimal"[8] pseudo-relaxation time as a function of the number n of nodes in the tree. Remember that the previous sections showed that Markov chains on cycles (and thus paths) present relaxation times of order $\Omega(n^2)$ and that the relaxation time of lattices scales in $\Omega(n)$.

Corollary* 4.3.1 (Upper bound on the optimal pseudo-relaxation time). The optimal pseudo-relaxation time $\widetilde{\tau}$ of a doubly stochastic matrix on a balanced p-ary tree is upper bounded as follows:

$$\begin{aligned}\widetilde{\tau} &\leq \alpha(p,H)n-\frac{p+1}{p-1}H+6\\ &\leq \alpha(p)n-\frac{p+1}{p-1}H+6,\end{aligned}$$

where

$$\begin{aligned}\alpha(p,H) &= \sum_{k=1}^{H-1}\frac{p+1}{p^k-(-1)^k}\\ &\leq \lim_{H\to\infty}\alpha(p,H)=\alpha(p).\end{aligned}$$

Proof: It is easy to show that in a p-ary tree,

$$|S_k| = \sum_{k=1}^{H} p^{H-k} = \frac{p^H-1}{p-1}.$$

As seen in the previous paragraph, minimizing $\sum_{k=2}^{H}|S_{k-1}|/b_k$ over b_k's veri-

[8] In this section, the "optimal" coefficients are obtained by minimizing the *upper bound* of the relaxation time, not the relaxation time itself.

fying the a_k non-negativity constraints

$$\begin{cases} p\ b_k + b_{k+1} \leq 1 & \text{if } \ k = 2, \ldots, H-1. \\ p\ b_H \leq 1 \end{cases}$$

leads to zero self-loops ($a_k = 0$ for $k = 2, \ldots, H$), and, for $k = 1, \ldots, H-1$:

$$b_{H+1-k} = \frac{1 - (-\frac{1}{p})^k}{1+p}.$$

An analytical upperbound for the optimal relaxation time can then be computed:

$$\begin{aligned}
\widetilde{\tau} &\leq \sum_{k=2}^{H} \frac{p^{k-1}-1}{p-1} \frac{p+1}{1-(-\frac{1}{p})^{H-k+1}} \\
&= \frac{p+1}{p-1}\left[p^H \sum_{k=1}^{H-1} \frac{1}{p^k-(-1)^k} - \sum_{k=1}^{H-1} \frac{1}{1-(-\frac{1}{p})^k}\right] \\
&= \frac{p^H-1}{p-1}\left(\sum_{k=1}^{H-1} \frac{p+1}{p^k-(-1)^k}\right) + \frac{p+1}{p-1}\left(\sum_{k=1}^{H-1} \frac{1}{p^k-(-1)^k} - \sum_{k=1}^{H-1} \frac{1}{1-(-\frac{1}{p})^k}\right) \\
&= n\alpha(p,H) - \frac{p+1}{p-1} \sum_{k=1}^{H-1} \frac{p^k-1}{p^k-(-1)^k},
\end{aligned}$$

where $\alpha(p,H)$ was defined in the theorem statement. Finally, note that

$$\begin{aligned}
\sum_{k=1}^{H-1} \frac{p^k-1}{p^k-(-1)^k} &= \sum_{k=1,k \text{ odd}}^{H-1} \frac{p^k-1}{p^k+1} + \sum_{k=1,k \text{ even}}^{H-1} \frac{p^k-1}{p^k-1} \\
&= \sum_{k=1,k \text{ odd}}^{H-1} \frac{p^k+1-2}{p^k+1} + \sum_{k=1,k \text{ even}}^{H-1} 1 \\
&= \sum_{k=1,k \text{ odd}}^{H-1} \frac{-2}{p^k+1} + \sum_{k=1}^{H-1} 1 \\
&= \sum_{k=1,k \text{ odd}}^{H-1} \frac{-2}{p^k+1} + H - 1.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\widetilde{\tau} &\leq n\alpha(p,H) - \frac{p+1}{p-1}\left(\sum_{k=1,k \text{ odd}}^{H-1} \frac{-2}{p^k+1} + H - 1\right) && (4.37)\\
&= n\alpha(p,H) - \frac{p+1}{p-1}H + \frac{p+1}{p-1}\left(\frac{2}{p+1} + \sum_{k=3,k \text{ odd}}^{H-1} \frac{2}{p^k+1} + 1\right) && (4.38)\\
&\leq n\alpha(p,H) - \frac{p+1}{p-1}H + \frac{p+3}{p-1} + \frac{p+1}{p(p-1)}\sum_{k=1}^{\infty}\frac{2}{(p^2)^k} && (4.39)\\
&\leq n\alpha(p,H) - \frac{p+1}{p-1}H + \frac{p+3}{p-1} + \frac{2}{p(p-1)^2} && (4.40)\\
&\leq n\alpha(p,H) - \frac{p+1}{p-1}H + 6, && (4.41)
\end{aligned}$$

as in the worst case, $p = 2$ gives $\frac{p+3}{p-1} + \frac{2}{p(p-1)^2} = 6$. ■

Comments:

- Under the conjecture $|\xi_n| < \xi_2$ (see Fig. 4.14(a) and 4.14(b) for some empirical evidence, and the Section 4.3.5 for a discussion), then the relaxation time τ describes the speed of convergence of the Markov chain to its stationary distribution.
- Note that $\alpha(p,H) \leqslant \lim_{H\to\infty} \alpha(p,H) = \alpha(p)$, which is independent of H.
- $\alpha(p)$ is larger than 1 but smaller than 3. In addition, $\alpha(p)$ decreases with p (see Fig. 4.14).
- Exact computations are possible when the graph is a star ($H = 2$):

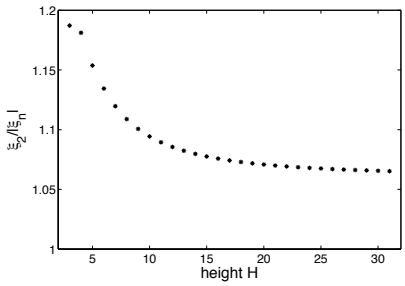
$$\tilde{\tau} = n - 1.$$

- For balanced trees, the optimal pseudo-relaxation time scales as $O(n)$. Starting from cycles, the structure of a tree introduces a breakdown of the graph diameter sufficient to consistently reduce the relaxation time. However symmetric balanced Markov chains on a balanced tree do not reach stationarity faster than Markov chains on the lattice, although the lattice has a much larger diameter than the tree. This is due to the presence of a bottleneck in the tree.

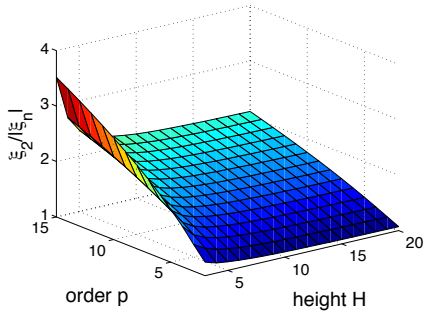
4.3.5 Is the SLE ξ_2 equal to the SLEM λ_2?

Having a small pseudo-relaxation time τ' is not sufficient to guarantee fast convergence time, since τ' is related to the SLE ξ_2, which is not necessarily equal to the SLEM λ_2. The value of actual interest is the relaxation time:

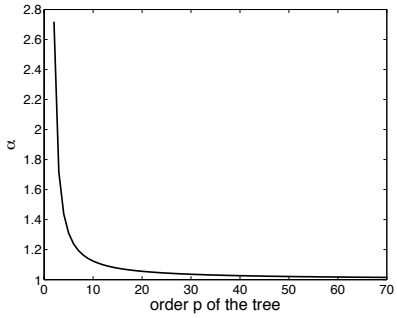
$$\tau = \max\left(\frac{1}{1-\xi_2}, \frac{1}{1-|\xi_n|}\right).$$



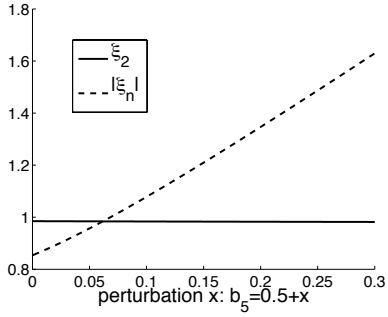
(a) $\xi_2/|\xi_n|$ for $p = 2$ as a function of H.



(b) $\xi_2/|\xi_n|$ as a function of H and p.



(c) $\alpha(p)$ as a function of p



(d) Perturbation of ξ_2 and $|\xi_n|$ for $p = 2$ and $H = 5$

Figure 4.14: Optimal upper bound in p-ary trees and symmetric balanced Markov chains. Fig. 4.14(a) and 4.14(b) show that ξ_2 is indeed the SLEM when all the self-loops are set to 0. Fig. 4.14(c) shows that the coefficient $\alpha(p)$ is small. In particular, $\alpha(p)$ approaches 1 as p increases. In the experiment of Fig. 4.14(d), the solution obtained by optimizing the relaxation time upper bound is perturbed by a coefficient x, such that $b_H = 1/p + x$, and therefore $a_H = -px$. The existence of one negative coefficient is enough to destabilize the convergence of an averaging algorithm that would use these coefficients.

Therefore, for the bound that has been computed to have some meaning, it is necessary to ensure $\xi_2 > |\xi_N|$, or to bound ξ_N as well.

Some reasons to think that $\boldsymbol{\xi}_2 > |\boldsymbol{\xi}_n|$ In Fig. 4.14(a) and 4.14(b), simulations suggest that when $b_k = c_k$ and $a_k = 0, k = 2, \ldots, H$, ξ_2 is not only the SLE but also the SLEM ($\xi_2 > |\xi_n|$). However, as shown in Fig. 4.14(d), if some coefficients a_k can be negative, then ξ_n quickly goes to -1 whereas the benefit in ξ_2 is small (the benefit in ξ_2 is non-zero since the proof of Th. 4.3.4 showed that the lagrange multipliers are strictly positive). It is therefore not worth gaining some gap on ξ_2 with some negative coefficients a_k because the risk of getting a diverging system with $\xi_n < -1$ is present and very coefficient sensitive.

Intuitively, the larger the self-loop coefficients $\{a_i\}_{i=1,\ldots,H}$ are, the more stable the Markov chain is, i.e. the larger ξ_n is. Gershgorin theorem (Th. 4.1.5) is a useful tool to understand this phenomenon. When the self-loops increase, Gershgorin disks shift right. In addition larger self-loops imply smaller off diagonal coefficients, since rows sum to 1. Thus, larger self-loops also reduce the diameter of the disks. Overall, the disks, which contain the eigenvalues, shift right and shrink. Therefore the eigenvalues should have a tendency to increase, ξ_2 approaching 1, and ξ_n moving away from -1.

Another way to understand this tendency is the following. Take a stochastic matrix $\mathbf{P}$, and a positive parameter $a > 0$. Then, build another stochastic matrix $\mathbf{P}'$:

$$\mathbf{P}' = \frac{\mathbf{P} + a\mathbf{I}}{1 + a}.$$

If $\boldsymbol{\pi}$ is the stationary distribution of $\mathbf{P}$, then it is also the stationary distribution of $\mathbf{P}'$:

$$\boldsymbol{\pi}^T\mathbf{P}' = \frac{\boldsymbol{\pi}^T\mathbf{P} + a\boldsymbol{\pi}^T}{1 + a} = \boldsymbol{\pi}^T.$$

Interestingly, this transformation changes all the eigenvalues except the largest one: 1. For each eigenvalue ξ of $\mathbf{P}$, $\mathbf{P}'$ has an eigenvalue ξ' such that:

$$\xi' = \frac{\xi + a}{1 + a} > \xi.$$

Therefore, this transformation, which consists in increasing the self-loop coefficients by a constant a, increases all eigenvalues, except eigenvalue 1.

As a conclusion, simulations showing that $\xi_2 > |\xi_n|$ for the "optimal" chain with self-loops set to 0 are a strong suggestion to generalize this property to all *doubly* stochastic matrices on balanced trees. However, when a balanced Markov chain is not *doubly* stochastic, the situation is less clear and there are examples where $|\xi_n| > \xi_2$. It is therefore important not to jump to early conclusions, and it is worth studying ξ_n in the general case.

A lower bound for ξ_n. The weighted path lower bound theorem gives lower bounds on the smallest eigenvalue ξ_n of the transition matrix $\mathbf{P}$ of an aperiodic and irreducible Markov chain on a finite space.

Theorem 4.3.1 (Weighted path lower bound ([16], page 213)). Let $\mathbf{P}$ be an irreducible and aperiodic transition matrix on a finite state space of size

N, with stationary distribution $\boldsymbol{\pi}$. For each state, choose one closed path σ_i from i to i that does not pass twice by the same edge, and that has an odd number of edges. Then for any selected path σ_i, let

$$|\sigma_i|_Q = \sum_{e \in \sigma_i} \frac{1}{Q(e)},$$

where, if $e = (i, j)$, then

$$Q(e) = \boldsymbol{\pi}(i)\mathbf{P}_{ij}.$$

Define

$$\alpha = \max_e \sum_{\sigma_i \ni e} |\sigma_i|_Q \, \boldsymbol{\pi}(i).$$

Then,

$$\begin{aligned} \xi_n &\geqslant -1 + \frac{2}{\alpha}, \\ \tau'' = \frac{1}{\xi_n - (-1)} &\leqslant \frac{\alpha}{2}. \end{aligned}$$

The pertinent relaxation time to consider is $\max(\tau', \tau'')$. This theorem offers an easy upper bound on τ'':

Theorem* 4.3.5. For any irreducible and aperiodic balanced Markov chain on a balanced tree,

$$\tau'' = \frac{1}{\xi_n - (-1)} \leqslant \frac{1}{2a_1}\left(1 + \frac{1}{\tilde{w}_1}\right) + \sum_{k=1}^{H-1} \frac{1}{\tilde{w}_k b_{k+1}}. \tag{4.42}$$

It is interesting to see that the bound in Eq. 4.42 is very similar to the upper bound of τ':

$$\tau' \leqslant \sum_{k=1}^{H-1} \frac{\sum_{i \in S_k} \tilde{w}_i}{\tilde{w}_k b_{k+1}}, \tag{4.43}$$

Proof: For any state i, build a path starting from i, which goes all the way to a leaf, uses the self-loop at that leaf, and then goes back up to i. For all the states that are not leaves, it is easy to build these paths so that they all use different edges, because there are more leaves than internal nodes in a tree with branching degrees systematically larger than or equal to 2. For every leaf, build a path that uses only its self-loop. It is easy to see that α should be computed with the self-loop of the leaf ℓ that is on the path σ which starts at the root r of the tree:

$$\begin{aligned} \alpha &= |\sigma_\ell|_Q \boldsymbol{\pi}(\ell) + |\sigma_r|_Q \boldsymbol{\pi}(r) \\ &= \frac{1}{\boldsymbol{\pi}(\ell) a_1}\boldsymbol{\pi}(\ell) + \left(\sum_{k=2}^{H} \frac{\boldsymbol{\pi}(r)}{c_k \boldsymbol{\pi}(k)} + \frac{\boldsymbol{\pi}(r)}{\boldsymbol{\pi}(\ell) a_1} + \sum_{k=2}^{H} \frac{\boldsymbol{\pi}(r)}{b_k \boldsymbol{\pi}(k-1)}\right). \end{aligned}$$

Using Theorem 4.3.1 on the stationary distribution of balanced Markov chains

on balanced trees,

$$\begin{aligned}
\alpha &= \frac{1}{a_1} + \left(\sum_{k=2}^{H} \frac{b_{k+1} \dots b_H}{c_k c_{k+1} \dots c_H} + \frac{b_2 \dots b_H}{c_2 \dots c_H a_1} + \sum_{k=2}^{H} \frac{b_k \dots b_H}{b_k c_k \dots c_H} \right). \\
&= \frac{1}{a_1} \left(1 + \frac{1}{\tilde{w}_1} \right) + 2 \sum_{k=2}^{H} \frac{1}{\tilde{w}_{k-1} b_k} \\
&= \frac{1}{a_1} \left(1 + \frac{1}{\tilde{w}_1} \right) + 2 \sum_{k=1}^{H-1} \frac{1}{\tilde{w}_k b_{k+1}}. \qquad (4.44)
\end{aligned}$$

Therefore,

$$\tau'' \leqslant \frac{1}{2a_1} \left(1 + \frac{1}{\tilde{w}_1} \right) + \sum_{k=1}^{H-1} \frac{1}{\tilde{w}_k b_{k+1}}.$$

■

Comments:

- For doubly stochastic matrices, all $\tilde{w}_{(\cdot)}$'s are equal to 1. Thus, in that case,

$$\begin{aligned}
\tau' &\leqslant \sum_{k=1}^{H-1} \frac{|S_k|}{b_{k+1}}, \\
\tau'' &\leqslant \frac{1}{a_1} + \sum_{k=1}^{H-1} \frac{1}{b_{k+1}}.
\end{aligned}$$

  If negative transitions are forbidden, then a_1 is larger than 0.5, and $1/a_1$ is smaller than 2. For sufficiently large trees, the upper bound on $\tau' = 1/(1-\xi_2)$ is thus much larger than the upperbound on $\tau'' = 1/(1-|\xi_n|)$. Assuming that the bound on τ is tight, it is then clear why $\xi_2 > |\xi_n|$ in doubly stochastic balanced Markov chains.

- When a Markov chain is not doubly stochastic, the bound on τ'' can be larger than the bound on τ'. This happens especially when the coefficients $\tilde{w}$ are smaller than 1, i.e. when the stationary distribution tends to concentrate around the root of the tree. The following example illustrates this idea. On a p-ary tree, take two different sets of balanced Markov chains parametrized by a real number $\beta \in (0,1)$:

  - Chains A: For $k = 2, \dots, H$, $b_k = \beta$. For $k = 2, \dots, H-1$, $c_k = (1-\beta)/p$. In order to keep all self-loops to 0, even at the root, $c_H = 1/p$.
  - Chains B: For $k = 2, \dots, H$, $b_k = \beta$. For $k = 2, \dots, H$, $c_k = (1-\beta)/p$. Chain B has a positive self-loop: $a_H = \beta$.

  In both chains, if $\beta > 0.5$, then transitions are larger in the direction of the root than in the direction of the leaves. If $\beta < 0.5$, then it's the contrary, i.e. transitions are larger towards the leaves than towards the root.

By the Sturm polynomial construction, it is easy to see that both chains A and B share the same P_{H-1} polynomial, so that they share the same SLE ξ_2. However, their polynomials P_H, whose smallest roots are their smallest eigenvalue ξ_n, are different. Chain A has a zero self-loop at the root ($a_H = 0$), and Fig. 4.15(a) shows that if $\beta > 0.5$, i.e. if chain A has a tendency to "go up to the root", then $|\xi_n| > \xi_2$. Reversely, if $\beta < 0.5$, i.e. if chain A has a tendency to "go down to the leaves", then $|\xi_n| < \xi_2$. Interestingly, in chain B, the non-zero self-loop at the root is sufficient to recover $|\xi_n| < \xi_2$ for any parameter $\beta \in (0, 1)$ (see Fig. 4.15(b)).

4.4 Laplacian

4.4.1 Definitions

The Laplacian of a graph $(V, \mathcal{E})$ is a $|V| \times |V|$ matrix that indicates which pairs of nodes are connected by an edge in $\mathcal{E}$. It is well-known that the spectrum of the Laplacian describes the level of connectivity of the graph. Interestingly, Laplacians recurrently appear in the study of distributed averaging algorithms, and the second smallest eigenvalue of the Laplacian (also called the algebraic connectivity of the graph) can be related to the SLEM dictating the speed of convergence of the algorithms. In other words, Laplacians are useful to explain the following natural statement: "The better connected a graph is, the faster distributed averaging algorithms run". This Section is a quick summary of the properties of Laplacians that are useful in this thesis. A more detailed description of Laplacians and their spectrum can be found in [3, 19].

Definition 4.4.1 (Laplacian). Let $\mathcal{G} = (V, \mathcal{E})$ be a graph without loops nor multiple edges, and let d_i be the degree of node i. The Laplacian matrix $\mathbf{L}$ is defined by:

$$\mathbf{L}(i,j) = \begin{cases} d_i & \text{if } i = j, \\ -1 & \text{if } i \text{ and } j \text{ are adjacent}, \\ 0 & \text{otherwise.} \end{cases}$$

Then, let $\mathbf{T}$ be the diagonal matrix such that for any $i = 1, \ldots, |V|$, $\mathbf{T}(i.i) = d_i$. The normalized Laplacian $\boldsymbol{\mathcal{L}}$ of $\mathcal{G}$ is defined as:

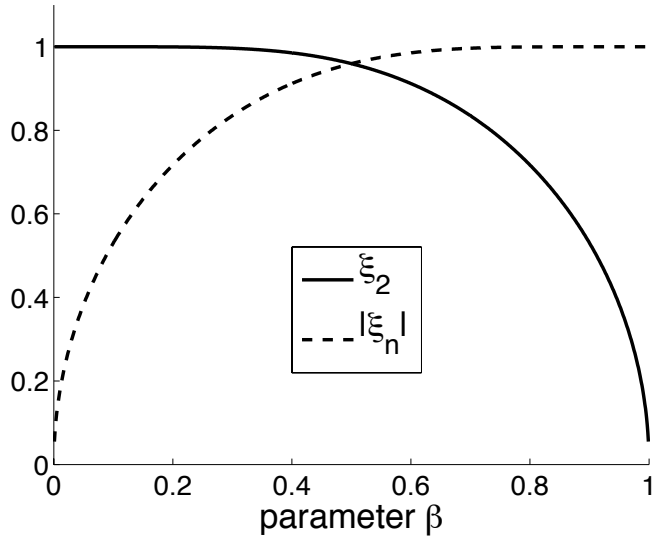
$$\boldsymbol{\mathcal{L}}(i,j) = \begin{cases} 1 & \text{if } i = j \text{ and } d_i \neq 0, \\ -\frac{1}{\sqrt{d_i d_j}} & \text{if } i \text{ and } j \text{ are adjacent} \\ 0 & \text{otherwise.} \end{cases}$$

Matrices $\mathbf{L}$ and $\boldsymbol{\mathcal{L}}$ are linked by the following formula:

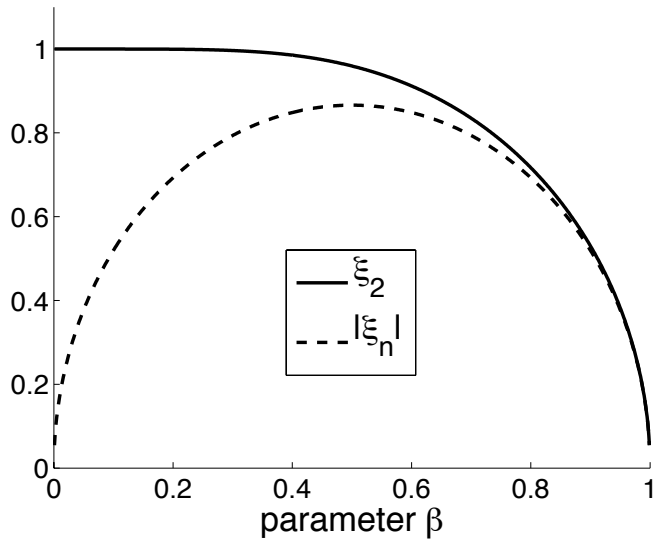
$$\boldsymbol{\mathcal{L}} = \mathbf{T}^{-\frac{1}{2}} \mathbf{L} \mathbf{T}^{-\frac{1}{2}},$$

with the convention $\mathbf{T}^{-1}(i,i) = 0$ if $d_i = 0$.

Let $\boldsymbol{\Phi}$ be the adjacency matrix of graph $\mathcal{G}$: $\boldsymbol{\Phi}(i,j) = 1$ if $i \neq j$ and if i and j are connected; $\boldsymbol{\Phi}(i,j) = 0$ otherwise. Then, $\mathbf{L} = \mathbf{T} - \boldsymbol{\Phi}$ and $\boldsymbol{\mathcal{L}} = \mathbf{I} - \mathbf{T}^{-1/2} \boldsymbol{\Phi} \mathbf{T}^{-1/2}$. Both the Laplacian $\mathbf{L}$ and the normalized Laplacian $\boldsymbol{\mathcal{L}}$ have been extensively studied. Note that in studies on normalized Laplacians,



(a) Chain A



(b) Chain B

Figure 4.15: Eigenvalues on balanced trees of degree $p = 3$ and height $H = 6$. The parameter β is the transition probability from any height h to height $h + 1$. In chain A, the self-loop at the root is 0, implying that the chain cannot stop at the root for two consecutive times. On the contrary, in chain B, the root has a self-loop equal to β, which, as the figures show, stabilizes the Markov chain.

authors rename $\boldsymbol{\mathcal{L}}$ as the "Laplacian". In this thesis, only the Laplacian $\mathbf{L}$ is useful, and one should be careful not to confuse it with $\boldsymbol{\mathcal{L}}$, since $\mathbf{L}$ and $\boldsymbol{\mathcal{L}}$ do not share the same eigenvalues.

Property 4.4.1. An $n \times n$ Laplacian matrix is always positive-semidefinite: its eigenvalues are non-negative:

$$0 = \lambda_0^{\mathbf{L}} \leqslant \lambda_1^{\mathbf{L}} \leqslant \ldots \leqslant \lambda_{n-1}^{\mathbf{L}}.$$

The number of times 0 appears as an eigenvalue of $\mathbf{L}$ is equal to the number of connected components in the graph. $\lambda_1^{\mathbf{L}}$ is called the algebraic connectivity of the graph. For a connected graph, it is positive and it is also called the spectral gap or the Fiedler value.

Note that the eigenvalues are indexed here in the opposite order than the usual one in this thesis, because it is the classical ordering in the literature on Laplacians.

4.4.2 Examples

1. *Synchronous average consensus with uniform weights:* For any iteration t, let $\mathbf{L}(t)$ be the Laplacian of the network at instant t, and let α be the constant weight used in the algorithm (see Algorithm in p. 19)

$$\mathbf{W}(t) = \mathbf{I} - \alpha \mathbf{L}(t).$$

Then, if the graph is non time-varying, the SLEM verifies:

$$\lambda_2(\mathbf{W}) = \max\left(1 - \alpha\lambda_1^{\mathbf{L}}, \left|1 - \alpha\lambda_{n-1}^{\mathbf{L}}\right|\right).$$

Therefore, in the non time-varying case, the best coefficient $\tilde{\alpha}$ (minimizing $\lambda_2(\mathbf{W})$) is equal to:

$$\tilde{\alpha} = \frac{2}{\lambda_1^{\mathbf{L}} + \lambda_{n-1}^{\mathbf{L}}}. \tag{4.45}$$

2. *Pairwise gossip:* Consider the generalization of pairwise gossip that was described in p. 75, where nodes update with weights α not necessarily equal to 0.5. Then, if at iteration t, if nodes i and j communicate and update their states:

$$\begin{aligned} \mathbf{W}(t)^T\mathbf{W}(t)(i,i) = \mathbf{W}(t)^T\mathbf{W}(t)(j,j) &= 1 - 2\alpha(1-\alpha) \\ \mathbf{W}(t)^T\mathbf{W}(t)(i,j) = \mathbf{W}(t)^T\mathbf{W}(t)(j,i) &= 2\alpha(1-\alpha), \end{aligned}$$

so that, in expectation, on a non time-varying *k-regular* graph:

$$\mathbb{E}[\mathbf{W}^T\mathbf{W}] = \mathbf{I} - \frac{4\alpha(1-\alpha)}{nk}\mathbf{L}. \tag{4.46}$$

Therefore, the eigenvalues of $\mathbb{E}[\mathbf{W}^T\mathbf{W}]$ can be computed as follows:

$$\lambda_k(\mathbf{W}^T\mathbf{W}) = 1 - \frac{4\alpha(1-\alpha)}{nk}\lambda_{k-1}^{\mathbf{L}}. \tag{4.47}$$

3. *Broadcast gossip* Consider the broadcast gossip algorithm described in p. 27, in which one node wakes up at each iteration and sends its state to its neighbors. The neighbors that received the state update their states according to Eq. (2.12), which is tuned by a parameter γ. This algorithm does not require the neighbors to send a message back to the sender, which does not update its own state.

   Take an undirected graph $\mathcal{G}$ (with Laplacian $\mathbf{L}_{\mathcal{G}}$), and model it as a time-varying directed graph $\mathcal{G}(t)$, which directed links fail identically and independently of each other with probability p. Laplacians were defined on undirected graphs only. Therefore, here, a generalization of the Laplacian is needed. For any node i, let $d(i)$ be its out-degree, and define the directed graph Laplacian as:

$$\mathbf{L}(i,j) = \begin{cases} d_i & \text{if } i = j, \\ -1 & \text{if there is a link from } i \text{ to } j, \\ 0 & \text{otherwise.} \end{cases}$$

   Then, tedious and careful computations, which are omitted in this text, allow to generalize the result, which was proved in [7] in the undirected case only. Since broadcast gossip does not conserve the average, only the error $\boldsymbol{\eta}(t)$ converges to 0, and the pertinent matrix to study is $\mathbb{E}[\mathbf{W}^T(\mathbf{I}-\mathbf{J})\mathbf{W}]$.

$$\begin{aligned} \mathbb{E}[\mathbf{W}^T(\mathbf{I}-\mathbf{J})\mathbf{W}] &= \mathbf{I}-\mathbf{J}-\frac{2\gamma(1-\gamma)}{n}\mathbb{E}[\mathbf{L}]-\frac{(1-\gamma)^2}{n^2}\mathbb{E}[\mathbf{L}^T\mathbf{L}] \\ &= \mathbf{I}-\mathbf{J}-2\left(\frac{p\gamma(1-\gamma)}{n}-\frac{p(1-p)(1-\gamma)^2}{n^2}\right)\mathbf{L}_{\mathcal{G}} \\ &\quad -\frac{p^2(1-\gamma)^2}{n^2}\mathbf{L}_{\mathcal{G}}^2. \end{aligned} \tag{4.48}$$

   Since $\lambda_1^{\mathbf{L}_{\mathcal{G}}^2} = (\lambda_1^{\mathbf{L}_{\mathcal{G}}})^2$, the largest eigenvalue $\lambda_1(\mathbb{E}[\mathbf{W}^T(\mathbf{I}-\mathbf{J})\mathbf{W}])$ can be computed as a function of second smallest eigenvalue of the Laplacian $\lambda_1^{\mathbf{L}_{\mathcal{G}}}$:

$$\lambda_1(\mathbb{E}[\mathbf{W}^T(\mathbf{I}-\mathbf{J})\mathbf{W}]) = 1-2\left(\frac{p\gamma(1-\gamma)}{n}-\frac{p(1-p)(1-\gamma)^2}{n^2}\right)\lambda_1^{\mathbf{L}_{\mathcal{G}}}-\frac{p^2(1-\gamma)^2}{n^2}(\lambda_1^{\mathbf{L}_{\mathcal{G}}})^2. \tag{4.49}$$

   The larger $\lambda_1^{\mathbf{L}_{\mathcal{G}}}$ is, the smaller $\lambda_1(\mathbb{E}[\mathbf{W}^T(\mathbf{I}-\mathbf{J})\mathbf{W}])$ is, and the faster the algorithm converges (in the worst initial signal case). The optimal parameter γ^*, which minimizes $\lambda_1(\mathbb{E}[\mathbf{W}^T(\mathbf{I}-\mathbf{J})\mathbf{W}])$ is equal to:

$$\gamma = \frac{n+2(1-p)-p\lambda_1^{\mathbf{L}_{\mathcal{G}}}}{2n+2(1-p)-p\lambda_1^{\mathbf{L}_{\mathcal{G}}}}. \tag{4.50}$$

4.4.3 Properties of the algebraic connectivity

This section briefly lists a few well-known properties of the algebraic connectivity $\lambda_1^{\mathbf{L}}$, which the second smallest eigenvalue of the Laplacian $\mathbf{L}$, and of $\lambda_{n-1}^{\mathbf{L}}$, which is the largest eigenvalue of $\mathbf{L}$. The latter eigenvalue can be useful to

ensure that some SLEs are SLEMs, or to optimize a parameter, as in Eq. 4.45. The properties cited below come from the works of Mohar [55] and of Alon and Milman [3]. They hold for undirected graph $\mathcal{G} = (V, \mathcal{E})$ with n nodes, no self-loops and no multiple edges.

Property 4.4.2 (largest eigenvalue-1). [55]

$$\lambda_{n-1}^{\mathbf{L}} \leqslant n.$$

Property 4.4.3 (largest eigenvalue-2). [55, 4]

$$\lambda_{n-1}^{\mathbf{L}} \leqslant \max\{d(i) + d(j) : (i,j) \in \mathcal{E}\},$$

where $d(i)$ denotes the degree of node i.

Property 4.4.4 (largest eigenvalue-3). [55, 35]

$$\lambda_{n-1}^{\mathbf{L}} \geqslant \frac{n}{n-1} \max\{d(i) : i \in V\}.$$

Property 4.4.5 (smallest eigenvalue-1). [55, 35]

$$\lambda_{1}^{\mathbf{L}} \leqslant \frac{n}{n-1} \min\{d(i) : i \in V\}.$$

As a consequence, the optimal coefficient γ^* computed in Eq. (4.50) tends to 1/2 in lattices or random geometric graphs.

Property 4.4.6 (smallest eigenvalue-2). [3]

$$\lambda_{1}^{\mathbf{L}} \leqslant \frac{8 \max\{d(i) : i \in V\}}{\mathrm{diam}(\mathcal{G})^2} \log_2^2 n,$$

where $\mathrm{diam}(\mathcal{G})$ denotes the diameter of graph $\mathcal{G}$.

Property 4.4.7 (smallest eigenvalue-3). [55]

$$\lambda_{1}^{\mathbf{L}} \geqslant \frac{4}{n \ \mathrm{diam}(\mathcal{G})}.$$

Conclusion

Second largest eigenvalue in magnitude (SLEM) is an important quantity that rules the speed of convergence of most distributed average consensus algorithms. In this chapter, we first stated a number of useful results that bound this eigenvalue. Then we focused on the SLEM of transition matrices of Markov chains on simple graphs such as circles, toroidal lattices and balanced trees. We derived a lower bound on the relaxation time of symmetric Markov chains on toroidal lattices, which will be useful to compute the best performance possible of pairwise gossip in random geometric graphs.

In addition, we showed that long distance transitions help Markov chains to reach stationarity faster on the cycle. This is interesting as we will show later that long distance averaging on random geometric graphs similarly accelerates the convergence of distributed averaging algorithms. Finally the study

of balanced trees allowed a better understanding of the impact of bottlenecks on the convergence speed of our algorithms. In particular, averaging coefficients around bottlenecks should be as large as possible in order to increase the information flow through these bottlenecks.

SECOND LARGEST EIGENVALUES

- **Perron-Frobenius** theorem: for any irreducible, aperiodic, and stochastic matrix $\mathbf{P}$, $\lambda_1(\mathbf{P}) = 1$ and the SLEM $\lambda_2 < 1$.
- The **conductance bound** (Th. 4.1.3) and the **Poincaré inequality**, also called the weighted path upper bound (Th. 4.1.4), allow to compute upper bounds on the SLE ξ_2 of irreducible reversible stochastic matrices.
- **Gershgorin's bound** (Th. 4.1.5) is useful to show that the SLE ξ_2 is equal to the SLEM λ_2. In particular, for any stochastic matrix $\mathbf{P}$, $\xi_2(\mathbf{P}) = \lambda_2(\mathbf{P})$ if $\mathbf{P}_{ii} \geqslant 1/2$ for all $i \in [1, n]$.
- On the **toroidal lattice** of n nodes and radius k, the relaxation time of the fastest symmetric mixing chain scales as:

$$\frac{1}{1 - |\lambda_2|} = \Theta\left(\frac{n}{k^2}\right).$$

- On a **balanced tree** of height H, the pseudo-relaxation time of a symmetric Markov chain is upper bounded by:

$$\frac{1}{1 - \xi_2} \leqslant \sum_{k=1}^{H-1} \frac{|S_k|}{b_k}, \tag{4.51}$$

  where $|S_k|$ is the number of nodes of a subtree which root is at height k, and where b_k is the transition probability between a node at height $k-1$ and its parent.

  As far as the smallest eigenvalue is concerned:

$$\frac{1}{1 - |\xi_n|} \leqslant \frac{1}{1 - b_2} + \sum_{k=1}^{H-1} \frac{1}{b_k}.$$

- As a consequence, the relaxation time of the fastest mixing chain on a **p-ary tree** scales in $O(n)$.
- To **minimize the relaxation time** on a cycle, one should favor long distance transitions. To minimize the relaxation time on a balanced tree, i.e. to minimize the bound in Eq. 4.51, all the self-loops should be set to 0, except at the leaves. This implies that the transition probabilities should be computed starting from the root. These two observations make a good guideline in the conception of averaging algorithms: favor **long-distance** averaging, and design carefully the averaging coefficients around network **bottlenecks** first, by making them as large as possible.

Chapter 5

Performance of Various Average Consensus Algorithms

All the tools needed to study the performance of average consensus algorithms have now been derived. This section applies previous chapters to the algorithms described in Chapter 2, starting by pairwise gossip.

5.1 Pairwise Gossip

In pairwise gossip, at each iteration t, a random i.i.d. node i wakes up and calls a neighbor j uniformly at random. Nodes i and j exchange their estimates $\mathbf{x}_i$ and $\mathbf{x}_j$, and both nodes update to $(x_i + x_j)/2$. In this section, we first explain in which conditions pairwise gossip converges. Then we derive its performance on various graphs of gradually increasing complexity. We start with simple topologies as the complete graph and lattices, we then continue with lattices of radius r and we finish with random geometric graphs.

5.1.1 Convergence

As seen in Chapter 2, in order to prove that a gossip algorithm converges with probability 1 in the case where $\{\mathbf{W}(t)\}_{t\in\mathbb{N}}$ is an i.i.d. sequence, one has to check that for any iteration t, $\mathbf{W}(t)\mathbf{1} = \mathbf{1}$ and $\mathbf{1}^T\mathbf{W}(t) = \mathbf{1}^T$, and that $\lambda_2(\mathbb{E}[\mathbf{W}^T\mathbf{W}]) = \lambda_2(\mathbb{E}[\mathbf{W}]) < 1$. Note that the time term"(t)" drops in $\mathbb{E}[\mathbf{W}]$, because $\{\mathbf{W}(t)\}_{t\in\mathbb{N}}$ is taken i.i.d. In pairwise gossip, at any time t, the first two conditions are verified. Indeed, if nodes i and j communicate at time t, then $\mathbf{W}(t) = \mathbf{W}^{(ij)}$, where $\mathbf{W}^{(ij)}$ is a matrix whose diagonal entries are 1 and non-diagonal entries are 0, except for $\mathbf{W}^{(ij)}_{ii} = \mathbf{W}^{(ij)}_{jj} = \mathbf{W}^{(ij)}_{ij} = \mathbf{W}^{(jj)}_{ii} = 1/2$.

$\mathbf{W}^{(ij)}$ thus reads:

$$\mathbf{W}^{(ij)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \ddots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{pmatrix}.$$

In the above realation, coefficients 0.5 are located at positions $(i,i),(j,j),(i,j)$ and (j,i) in the matrix. If, for example, nodes wake up uniformly at random and choose a neighbor uniformly at random, then,

$$\mathbb{E}[\mathbf{W}_{ij}] = \frac{1}{2n}\left(\frac{1}{d_i} + \frac{1}{d_j}\right), \tag{5.1}$$

where d_i and d_j are the degrees of nodes i and j, with $i \neq j$. As at any iteration t, $\mathbf{W}(t)$ is doubly stochastic, $\mathbb{E}[\mathbf{W}]$ is doubly stochastic as well. It is thus not necessary to compute the diagonal terms to fully characterize $\mathbb{E}[\mathbf{W}]$.

In order to prove convergence of pairwise gossip, it remains to prove that $\lambda_2(\mathbb{E}[\mathbf{W}]) < 1$. According to the Perron-Frobenius theorem, it is then sufficient to show that $\mathbb{E}[\mathbf{W}]$ is a primitive and stochastic matrix. It has already been established that $\mathbb{E}[\mathbf{W}]$ is stochastic. Remember that the convergence theorem, which is used here, holds only if $\{\mathbf{W}(t)\}_{t\in\mathbb{N}}$ is an i.i.d. sequence. One way to guarantee this and to ensure that $\mathbb{E}[\mathbf{W}]$ is primitive is the following. If nodes wake up i.i.d. and call a neighbor in an i.i.d fashion as well, and if the network is jointly connected in time (see Def. 5.1.1), then $\mathbb{E}[\mathbf{W}]$ is primitive.

Definition 5.1.1 (Jointly connected network). Let $\{\mathcal{G}(t) = (V, \mathcal{E}(t))\}_{t\in\mathbb{N}}$ be the sequence of graphs modeling a given network in discrete time $t \in \mathbb{N}$. If $\{\mathcal{G}(t)\}_{t\in\mathbb{N}}$ is *i.i.d.* and if the following graph $\tilde{\mathcal{G}}$ is connected:

$$\tilde{\mathcal{G}} = \left(V, \tilde{\mathcal{E}}\right),$$

where

$$\tilde{\mathcal{E}} = \{e \in (V \times V) : \mathbb{P}[e \in \mathcal{E}] > 0\},$$

then the network is jointly connected in time.

Indeed, if the algorithm is i.i.d. and if the network is jointly connected, then $\mathbb{E}[\mathbf{W}]$ has the same sparsity pattern as the connected graph $\tilde{\mathcal{G}}$, which implies that it is primitive.

For network models where $\{\mathcal{G}(t)\}_{t\in\mathbb{N}}$ is not an i.i.d. sequence, one can use Theorem 3.1.3, where only stationarity and ergodicity are required from the sequence of graphs $\{\mathcal{G}(t)\}_{t\in\mathbb{N}}$ and matrices $\{\mathbf{W}(t)\}_{t\in\mathbb{N}}$.

5.1.2 Speed of Convergence

Using Corollary 3.2.1 for gossip algorithms, where $\epsilon = \beta/n$ and $\eta = 1/n^\alpha$,

$$T_{ave}(n, \varepsilon, \eta) = \Theta\left(\frac{\log n}{1 - \lambda_2\left(\mathbb{E}[\mathbf{W}]\right)}\right). \tag{5.2}$$

The goal of this section is thus to derive bounds on $\lambda_2(\mathbb{E}[\mathbf{W}])$ over different network topologies: complete graphs, lattices, lattices of radius r and random geometric graphs. This will be achieved thanks to the tools developed in Section 4.

Performance of pairwise gossip on complete graphs

We first consider the performance of pairwise gossip on complete graphs, as we expect that pairwise gossip performs the best on this topology.

Theorem 5.1.1. The averaging time $T_{ave}(n, \epsilon, \eta)$ of pairwise gossip on a complete graph, for $\epsilon = \beta/n$ and $\eta = 1/n^\alpha$, scales as:

$$T_{ave}(n, \epsilon, \eta) = \Theta\left(n \log n\right).$$

The total expected cost in $T_{ave}(n, \epsilon, \eta)$ iterations scales as $\Theta\left(n \log n\right)$. In addition, the consensus cost $\mathcal{E}_c$ (see Section 3.2.3), i.e. the number of sent messages asymptotically needed to reduce the error by a factor e, scales as $O(n)$.

Proof: Applying Eq. (5.1) in the case of a complete graph with n nodes gives:

$$\mathbb{E}[\mathbf{W}] = \left(1 - \frac{1}{n-1}\right)\mathbf{I} - \frac{1}{n-1}\mathbf{J}.$$

Since $\mathbf{J}$ has only one non-zero eigenvalue, which is 1, associated with the eigenvector $\mathbf{1}$, and since any vector is an eigenvector of $\mathbf{I}$ with eigenvalue 1, the second largest eigenvalue in magnitude (SLEM) and the relaxation time of $\mathbb{E}[\mathbf{W}]$ are respectively

$$\begin{aligned} \lambda_2(\mathbb{E}[\mathbf{W}]) &= 1 - \frac{1}{n-1}, \\ \frac{1}{1 - \lambda_2(\mathbb{E}[\mathbf{W}])} &= n - 1. \end{aligned}$$

■

Note that one cannot expect a better performance for a distributed averaging algorithm, since at least n messages should be sent to average the signal.

Performance of pairwise gossip on lattices

We now consider the performance of pairwise gossip on lattices, which is a first step towards the analysis on random geometric graphs.

Theorem 5.1.2. The averaging time $T_{ave}(n, \epsilon, \eta)$ of pairwise gossip on a lattice with n nodes, for $\epsilon = \beta/n$ and $\eta = 1/n^\alpha$, scales as:

$$T_{ave}(n, \epsilon, \eta) = \Theta\left(n^2 \log n\right).$$

The total expected cost in $T_{ave}(n, \epsilon, \eta)$ iterations scales as $\Theta\left(n^2 \log n\right)$. In addition, the consensus cost $\mathcal{E}_c$ (see Section 3.2.3), i.e. the number of sent messages asymptotically needed to reduce the error by a factor e, scales as $O(n^2)$.

This theorem can be proved either with the conductance bound (Th. 4.1.3), either with the weighted path bound (Th. 4.1.4). Both proofs are written here.
Proof: (Conductance bound technique) Consider a lattice with n nodes, where each node has 4 neighbors, except for border nodes that have 3 neighbors and for corner nodes that have only 2 neighbors. To ease the proof, assume in addition that n is even. Applying Eq. (5.1), if i and j are neighbors, then

$$\begin{aligned} \mathbb{E}[\mathbf{W}_{ij}] &\geqslant \frac{1}{2n}\left(\frac{1}{4}+\frac{1}{4}\right) \\ &= \frac{1}{4n}. \end{aligned} \tag{5.3}$$

Upper bound: The conductance bound (Th. 4.1.3) is the one used in this proof. This bound gives tighter results on the following matrix $\mathbf{W}'$:

$$\mathbf{W}' = \frac{\mathbb{E}[\mathbf{W}] - a\mathbf{I}}{1-a}, \tag{5.4}$$

where $a = \min_{1 \leqslant i \leqslant n} \mathbb{E}[\mathbf{W}_{ii}]$. One can show that $\mathbf{W}'$ is also a doubly stochastic matrix, on which the conductance bound can be applied, and its second largest eigenvalue is linked to the second largest eigenvalue of $\mathbb{E}[\mathbf{W}]$:

$$\lambda_2(\mathbf{W}') = \frac{\lambda_2(\mathbb{E}[\mathbf{W}]) - a}{1-a}.$$

Therefore the relaxation times of $\mathbf{W}$ and $\mathbf{W}'$ are related by:

$$\frac{1}{1-\lambda_2(\mathbb{E}[\mathbf{W}])} = \frac{1}{1-a} \cdot \frac{1}{1-\lambda_2(\mathbf{W}')}.$$

Here, $a = 1 - k/n$, where k is a constant slightly larger than 1. Indeed, the node i for which $\mathbb{E}[\mathbf{W}]_{ii}$ is minimum is a border node, which has a corner node as a neighbor, because of Eq. (5.1), and one can compute that $k = 12.5/12$. Therefore,

$$\frac{1}{1-\lambda_2(\mathbb{E}[\mathbf{W}])} = \frac{n}{k} \cdot \frac{1}{1-\lambda_2(\mathbf{W}')}. \tag{5.5}$$

Now, it is time to apply the conductance bound on $\mathbf{W}'$. The minimum ergodic flow is obtained when cutting the network in two parts B and $\bar{B}$, as shown in Fig. 5.1, which yields:

$$\begin{aligned} \phi(\mathbf{W}') &= \frac{\sum_{i \in B, j \in \bar{B}} \boldsymbol{\pi}(i) \mathbf{W}'_{ij}}{\boldsymbol{\pi}(B)} \\ &= \frac{2 \sum_{i \in B, j \in \bar{B}} n \mathbb{E}[\mathbf{W}]_{ij}/k}{n}, \end{aligned}$$

because here $\boldsymbol{\pi}(B) = 1/2$. The cutset separating B from $\bar{B}$ is made of two edges between nodes of degree 3, and of $\sqrt{n} - 2$ edges between nodes of degree

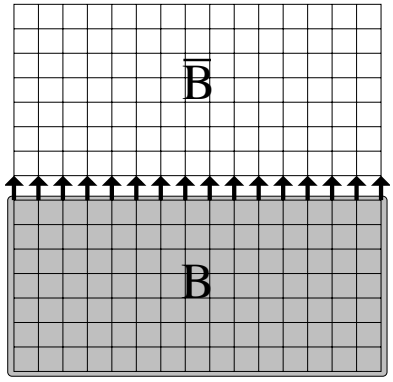


Figure 5.1: Minimum ergodic flow. The nodes are situated at the intersection of each line.

4. Therefore, using Eq. (5.1)

$$\begin{aligned}\phi(\mathbf{W}') &= \frac{2\left(\frac{2}{3}+\frac{\sqrt{n}-2}{4}\right)}{kn} \\ &= \Theta\left(\frac{1}{\sqrt{n}}\right).\end{aligned}$$

According to the conductance bound,

$$\begin{aligned}\frac{1}{1-\lambda_2(\mathbf{W}')} &\leqslant \frac{2}{\phi^2(\mathbf{W}')} \\ &= O\left(n\right). \end{aligned} \tag{5.6}$$

Combining Eqs. (5.5) and (5.6) gives

$$\frac{1}{1-\lambda_2(\mathbb{E}[\mathbf{W}])} = O\left(n^2\right).$$

Lower bound: The relaxation time $1/(1-\lambda_2(\mathbf{W}'))$ of $\mathbf{W}'$ is necessarily larger than the relaxation time of the fastest mixing chain on the lattice embedded on the torus, which, according to Theorem 4.2.1 in Section 4.2.2, scales as $\Omega(n)$. Therefore,

$$\frac{1}{1-\lambda_2(\mathbb{E}[\mathbf{W}])} = \Omega(n^2)$$

Combining the upper and lower bounds, and using Eq. 5.2, yield the theorem. ■

Proof: (Weighted path bound) In this proof, the upper bound of Theorem 5.1.2 is proven a second time, using the weighted path upper bound (page 85, Theorem 4.1.4).

In order to apply Theorem 4.1.4, one path γ_{ij} between each pair of nodes (i,j) should be designed. For any pair of nodes (i,j), let γ_{ij} be the $(\leftrightarrow,\updownarrow)$-route from i to j (see Alg. A.1) and let ℓ_{ij} be the number of edges in this path. Then,

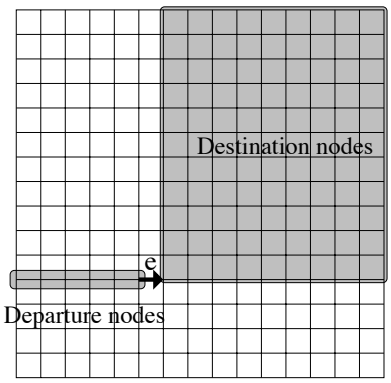


Figure 5.2: The possible departure and destination nodes for horizontal-vertical paths using edge e on the lattice. This figure proves that $\mathcal{C}_e \leqslant n^{1.5}/4$. Indeed, for a horizontal edge for instance, if there are $\alpha\sqrt{n}$ possible originating nodes $(0 \leqslant \alpha \leqslant 1)$, there are at most $(1-\alpha)n$ possible destination nodes. Noting that $\alpha(1-\alpha) \leqslant 1/4$ gives the announced bound on $\mathcal{C}_e$.

according to Eq. (5.3) and to Eq. (4.5) applied with $\mathbf{P} = \mathbb{E}[\mathbf{W}]$,

$$\begin{aligned} |\gamma_{ij}| &\leqslant 4\ell_{ij}n^2 \\ &\leqslant 8n^{2.5}, \end{aligned} \tag{5.7}$$

because the longest path is the one from one corner to the diagonally opposite corner of the network, so that $\ell_{ij} \leqslant 2\sqrt{n}$.

For any edge e, Eq. 5.7 implies that

$$\sum_{\gamma_{ij} \ni e} |\gamma_{ij}| \boldsymbol{\pi}(i)\boldsymbol{\pi}(j) \leqslant \frac{8n^{2.5}}{n^2}\mathcal{C}_e,$$

where $\mathcal{C}_e$ is the number of paths γ_{ij} that contain edge e. A horizontal-vertical path using a given *horizontal* edge e can be originated by at most $\sqrt{n}$ nodes, and can end in at most n nodes. Similarly, a horizontal-vertical path using a given *vertical* edge e can be originated by at most n nodes, and can end in at most $\sqrt{n}$ nodes (see Fig. 5.2). Therefore, any given edge e is contained in at most $n^{1.5}$ paths γ_{ij}. In other words, $\mathcal{C}_e \leqslant n^{1.5}$ for all edges e. More precisely, one can show that $\mathcal{C}_e \leqslant n^{1.5}/4$. Indeed, for a horizontal edge for instance, if there are $\alpha\sqrt{n}$ possible originating nodes $(0 \leqslant \alpha \leqslant 1)$, there are at most $(1-\alpha)n$ possible destination nodes. Noting that $\alpha(1-\alpha) \leqslant 1/4$ gives the finest bound on $\mathcal{C}_e$. As a consequence,

$$\sum_{\gamma_{ij} \ni e} |\gamma_{ij}| \boldsymbol{\pi}(i)\boldsymbol{\pi}(j) \leqslant 8\sqrt{n} \cdot n^{1.5}/4 = 2n^2.$$

Therefore, κ, which is defined in Eq. (4.6), verifies $\kappa \leqslant 2n^2$. Finally, by Theorem 4.1.4,

$$\frac{1}{1-\lambda_2(\mathbb{E}[\mathbf{W}])} \leqslant 2n^2.$$

Lower bound: it is the same argument as in the first proof. ■

Performance of pairwise gossip on lattices of radius r.

Now that we have studied the performance of pairwise gossip on lattices, we are going to extend the study to lattices admitting a connection radius r.

Theorem 5.1.3. The averaging time $T_{ave}(n, \epsilon, \eta)$ of pairwise gossip on a lattice of radius $r \in [1, \sqrt{n}]$ with n nodes, for $\epsilon = \beta/n$ and $\eta = 1/n^{\alpha}$, scales as:

$$T_{ave}(n, \epsilon, \eta) = \Theta\left(\frac{n^2}{r^2} \log n\right).$$

The total expected cost in $T_{ave}(n, \epsilon, \eta)$ iterations scales as $\Theta\left(n^2 \log n / r^2\right)$. In addition, the consensus cost $\mathcal{E}_c$ (see Section 3.2.3), i.e. the number of sent messages asymptotically needed to reduce the error by a factor e, scales as:

$$\mathcal{E}_c = O\left(\frac{n^2}{r^2}\right).$$

Proof: Both proofs with conductance bound and weighted path bound in the case of the simple lattice can easily be extended to the case of lattices with radius r. We choose to extend here the proof that uses the weighted path bound. In order to make the proof shorter, only order of magnitudes are going to be derived.

Any node in the lattice have order $\Theta\left(r^2\right)$ neighbors. Therefore, for any neighbor nodes i and j,

$$\begin{aligned} \mathbb{E}[\mathbf{W}_{ij}] &= \frac{1}{2n}\left(\frac{1}{d_i} + \frac{1}{d_j}\right) \\ &= \Omega\left(\frac{1}{r^2 n}\right). \end{aligned} \tag{5.8}$$

Upper bound: Divide the lattice in boxes of size $r/2 \times r/2$ (in each box there are $r^2/4$ nodes), so that nodes are connected to all the nodes in adjacent boxes. For any nodes i and j, define a path γ_{ij}, such that the successive intermediate nodes are located on the $(\leftrightarrow, \updownarrow)$-box route between the box of i and the box of j (see Alg. A.2). It is important to notice that it is possible to design these paths such that the set of paths $\{\gamma_{ij}\}_{i \in b_1, j \in b_2}$ do not use twice the same edge. Indeed, there are $(r^2/4)^2$ paths to design between nodes of box b_1 and nodes of box b_2, but there are also $(r^2/4)^2$ edges between each box. Fig. 5.3 shows an example of paths of length 4, where boxes have 3 nodes each. The paths that have just been designed have a length of order $O(\sqrt{n}/r)$. Therefore,

$$\begin{aligned} |\gamma_{ij}| &= O\left(\frac{\sqrt{n}}{r} \cdot n \cdot r^2 n\right) \\ &= O\left(n^{2.5} r\right). \end{aligned}$$

Therefore, for any edge e (not a physical edge, but an edge in the sense of

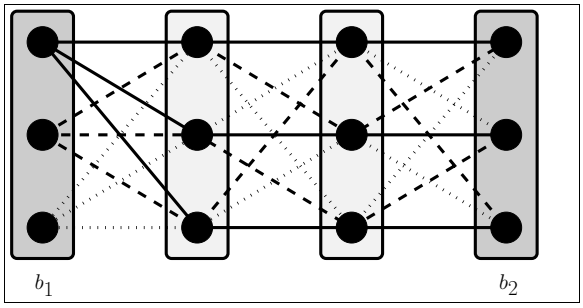


Figure 5.3: Paths do not overlap

Theorem 4.1.4),

$$\sum_{\gamma_{ij} \ni e} |\gamma_{ij}| \boldsymbol{\pi}(i)\boldsymbol{\pi}(j) = O\left(\frac{n^{2.5} r}{n^2} \mathcal{C}_e\right) = O\left(\sqrt{n} r \mathcal{C}_e\right),$$

where $\mathcal{C}_e$ is the number of paths γ_{ij} that contain edge e. Using the same argument as in the second proof of Th. 5.1.2, based on Fig. 5.2, for any edge e,

$$\mathcal{C}_e = O\left(\left(\frac{\sqrt{n}}{r}\right)^3\right).$$

As a consequence,

$$\begin{aligned} \sum_{\gamma_{ij} \ni e} |\gamma_{ij}| \boldsymbol{\pi}(i)\boldsymbol{\pi}(j) &= O\left(\sqrt{n} r \left(\frac{\sqrt{n}}{r}\right)^3\right) \\ &= O\left(\frac{n^2}{r^2}\right). \end{aligned}$$

Thus, $\kappa = \max_{\text{edge } e} \sum_{\gamma_{ij} \ni e} |\gamma_{ij}| \boldsymbol{\pi}(i)\boldsymbol{\pi}(j) = O(n^2/r^2)$, and

$$\frac{1}{1 - \lambda_2(\mathbb{E}[\mathbf{W}])} = O\left(\frac{n^2}{r^2}\right).$$

Lower bound: Define the stochastic matrix $\mathbf{W}'$ just as done in the case of the simple lattice (Eq. 5.4). The relaxation time of $\mathbf{W}'$ is necessarily larger than the relaxation time of the fastest mixing chain on the lattice embedded on the torus, which, according to Section 4.2.2 at page 94, scales as $\Omega(n/r^2)$. Since the relaxation time of $\mathbb{E}[\mathbf{W}]$ is order $\Theta(n)$ times larger than the relaxation time of $\mathbf{W}'$,

$$\frac{1}{1 - \lambda_2(\mathbb{E}[\mathbf{W}])} = \Omega\left(\frac{n^2}{r^2}\right).$$

Finally, since there are at most $\mathcal{E} = 3$ messages exchanged at each iteration,

$$\mathcal{E}_c = \mathbb{E}[\mathcal{E}] T_c = O\left(\frac{1}{1 - \lambda_2(\mathbb{E}[\mathbf{W}])}\right),$$

which concludes the proof. ■

Performance of pairwise gossip on random geometric graphs

Pairwise gossip on a lattice of radius r converges in $O(n^2/r^2)$, as it has just been proven. In this section, the result is generalized to Random Geometric Graphs (RGG), in the sense that the same formula holds, but with the number of neighbors r^2 changed into $\log n$, which the order of magnitude of any neighborhood size in a RGG.

Results on random geometric graphs very often come from refinements of proofs on lattices. Indeed, according to Theorem A.3.1, if the connection radius is large enough, one can split a random geometric graph into virtual boxes of size $\sqrt{\alpha \log n/n} \times \sqrt{\alpha \log n/n}$, such that every box contains between $a \log n$ and $b \log n$ nodes with high probability, and such that any node is connected to any other node in an adjacent box. The constants a, b and α are all $O(1)$, and thus independent of n, and they verify $a \leqslant \alpha \leqslant b$.

In other words, a random geometric graph can be embedded in a lattice of boxes, which play the role of super nodes, and which contain between $a \log n$ and $b \log n$ nodes each. This graph structure is the key of the analysis on random geometric graphs.

Theorem 5.1.4. The averaging time $T_{ave}(n, \epsilon, \eta)$ of pairwise gossip on a random geometric graph with n nodes and connection radius $r(n) \geqslant \sqrt{10 \log n/n}$, for $\epsilon = \beta/n$ and $\eta = 1/n^{\beta'}$, scales w.h.p. as:

$$T_{ave}(n, \epsilon, \eta) = \Theta\left(n^2\right).$$

The total expected cost in $T_{ave}(n, \epsilon, \eta)$ iterations scales as $\Theta\left(n^2\right)$. In addition, the consensus cost $\mathcal{E}_c$ (see Section 3.2.3), i.e. the number of sent messages asymptotically needed to reduce the error by a factor e, scales as $O(n^2/\log n)$.

Both proofs on lattices (conductance bound (Th. 4.1.3) first, and weighted path bound (Th. 4.1.4) second) are refined to random geometric graphs, in order to introduce the reader to the refinement techniques. The first proof with the conductance bound will be less rigorous than the second one, but the proof technique is interesting enough to justify its presence in the manuscript.
Proof:(conductance bound) Since $c > 10$, the RGG is log-balanced with respect to some values d_1 and d_2 with high probability (see Def. A.3.1 and Th. A.3.2). Therefore, if i and j are neighbors,

$$\begin{aligned} \mathbb{E}[\mathbf{W}_{ij}] &\geqslant \frac{1}{2n}\left(\frac{1}{d_2 \log n} + \frac{1}{d_2 \log n}\right) \\ &= \frac{1}{d_2 n \log n}. \end{aligned} \tag{5.9}$$

Upper bound: As in the proof on lattices using the conductance bound, one should define a matrix $\mathbf{W}'$ based on matrix $\mathbb{E}[\mathbf{W}]$, apply the bound on $\mathbf{W}'$, and finally deduce a bound on the SLEM of $\mathbb{E}[\mathbf{W}]$. Follow the example of the matrix $\mathbf{W}'$ defined in Eq. (5.4). Here,

$$a = \min_{1 \leqslant i \leqslant n} \mathbb{E}[\mathbf{W}_{ii}] \geqslant 1 - \frac{d_2}{d_1 n},$$

since for any node i,

$$\begin{aligned} 1 - \mathbb{E}[\mathbf{W}_{ii}] = \sum_{j \neq i, j \in \mathcal{N}(i)} \mathbb{E}[\mathbf{W}_{ij}] &\leqslant \sum_{j \neq i, j \in \mathcal{N}(i)} \frac{1}{d_1 n \log n} \\ &\leqslant \frac{d_2 \log n}{d_1 n \log n} \\ &= \frac{d_2}{d_1 n}. \end{aligned}$$

Therefore, $\mathbf{W}'$ is defined here as:

$$\mathbf{W}' = \frac{\mathbb{E}[\mathbf{W}] - \left(1 - \frac{d_2}{d_1 n}\right) \mathbf{I}}{\frac{d_2}{d_1 n}}. \tag{5.10}$$

In this context, $\mathbf{W}'$ is a doubly stochastic matrix such that, for $i \neq j$,

$$\begin{aligned} \mathbf{W}'_{ij} &= \frac{d_1 n}{d_2} \mathbb{E}[\mathbf{W}]_{ij} && (5.11) \\ &\geqslant \frac{d_1 n}{d_2} \cdot \frac{1}{d_2 n \log n} \\ &= \frac{d_1}{d_2^2 \log n}. && (5.12) \end{aligned}$$

It is now time to apply the conductance bound, and to look at the ergodic flows. In contrast to lattices, it is difficult to precisely define the best set B in a random geometric graph. However, since only orders of magnitude are sought, an approximation of B is sufficient to derive the bound. This proof will thus be confirmed with the more rigorous second proof, which uses the weighted path bound. Divide the unit square in boxes of size $\sqrt{\alpha \log n/n} \times \sqrt{\alpha \log n/n}$, such that $\sqrt{5\alpha \log n/n} \leqslant r(n)$, which ensures that any pair of nodes in adjacent boxes are connected. Define B as the set of nodes that are in the lower or upper half of boxes. More precisely, B is the set of nodes located in the less populated half, and $\bar{B}$ is the set of nodes located in the most populated half. Next, lower bound the ergodic flow from B to its complement $\bar{B}$, by considering only the edges between nodes that are in adjacent boxes (communications between diagonal boxes are ignored). Call B^f the row of boxes in B that border $\bar{B}$. Now, each box contains at least $d_1 \log n$ nodes, and there are $\sqrt{n/\alpha \log n}$ boxes in B^f, hence the ergodic flow defined in the conductance bound theorem (Th. 4.1.3)

is lower bounded as

$$\begin{aligned} F(B) &\geqslant \sum_{i\in B^f} d_1 \log n \left(\frac{1}{n}\cdot\frac{d_1}{d_2^2\log n}\right) \\ &= \sum_{i\in B^f} \frac{d_1^2}{d_2^2 n} \\ &\geqslant \sqrt{\frac{n}{\alpha\log n}}\cdot(d_1\log n)\cdot\frac{d_1^2}{d_2^2 n} \\ &= \frac{d_1^3}{d_2^2\sqrt{\alpha}}\sqrt{\frac{\log n}{n}}. \end{aligned}$$

Now comes the part of the proof, which is not rigorous. We will assume that $(B,\bar{B})$ is the cut-set which minimizes $F(B)/\boldsymbol{\pi}(B)$. This assumption is probably wrong, but it is reasonable to think that it will produce the correct bound. Since B was constructed such that $\boldsymbol{\pi}(B)\leqslant 1/2$,

$$\phi(\mathbf{W}')\geqslant\frac{2d_1^3}{\sqrt{\alpha}d_2^2}\sqrt{\frac{\log n}{n}}.$$

Thus,

$$\begin{aligned} \frac{1}{1-\lambda_2(\mathbf{W}')} &\leqslant \frac{2}{\phi^2(\mathbf{W}')} \\ &\leqslant \frac{\alpha d_2^4}{2d_1^6}\cdot\frac{n}{\log n}. \end{aligned}$$

Since

$$\frac{1}{1-\lambda_2(\mathbb{E}[\mathbf{W}])}=\frac{d_1 n}{d_2}\cdot\frac{1}{1-\lambda_2(\mathbf{W}')}, \tag{5.13}$$

then

$$\frac{1}{1-\lambda_2(\mathbb{E}[\mathbf{W}])}\leqslant\frac{\alpha d_2^3}{2d_1^5}\cdot\frac{n^2}{\log n},$$

which, according to Eq. 5.2, finishes the proof of the upper bound.

Lower bound: A proof for the lower bound can be found in [15] in the paragraph called "Optimal walk on $G^2(n,r)$". The main idea is that random geometric graphs can somehow be embedded in a lattice of some radius k, on which lower bounds are known. Since the proof is rather long and technical, only a summary of the proof is given here. In addition, a "hand-wavy" proof is given first, so that the reader can understand in a glance why the lower bound holds.

Hand-wavy proof: Imagine that the nodes of the RGG are slightly moved so that their new positions match the positions of a lattice, two different nodes being necessarily in different positions. In addition, embed the unit square on a torus. Then there is a radius k on the lattice such that any edge in the RGG is included in the set of edges of the toroidal lattice of radius k. According to Theorem 4.2.1, the relaxation time of the fastest symmetric mixing chain on

the toroidal lattice of n nodes and radius k scales as:

$$\tau = \frac{1}{1-|\lambda_2|} = \Theta\left(\frac{n}{k^2}\right).$$

Therefore, the relaxation time of $\mathbb{E}[\mathbf{W}']$ (see Eq. 5.10), which is the transition matrix of a symmetric Markov chain on the RGG, is necessarily larger than the relaxation time of the fastest symmetric mixing chain on the toroidal lattice of n nodes and radius k. In other words, $1/(1-\lambda_2(\mathbb{E}[\mathbf{W}']))$ scales as $\Omega\left(n/k^2\right)$. Hence, according to Eq. 5.13, the mixing time of $\mathbb{E}[\mathbf{W}]$ scales as $\Omega\left(n^2/k^2\right)$. By extrapolating the above formula, k^2, which is proportional to the number of neighbors of each node in the lattice of radius k, can be extrapolated to $\log n$, which is the number of neighbors in the RGG. This extrapolation gives the correct bound: the relaxation time is larger than order $n^2/\log n$.

Summary of rigorous proof: Making this intuitive argument rigorous is not easy. The main steps of the proof in [15] are the following. Divide the unit square in boxes of size $r(n) \times r(n)$. Then, there is an edge between two nodes i and j in the RGG $\mathcal{G}$ only if i and j are in adjacent boxes (including diagonal boxes, so that each box has 8 adjacent boxes), or if they are in the same box. The first step of the proof consists in adding edges to the RGG graph $\mathcal{G}$, such that the fastest mixing time on the resulting better connected new graph $\mathcal{G}'$ lower bounds the fastest mixing time on the original RGG graph $\mathcal{G}$. By construction, there is an edge between two nodes i and j in the new graph $\mathcal{G}'$ if and only if i and j are in the same box or in adjacent boxes, where adjacency is considered horizontally, vertically, in diagonal and on the torus.

The second step of the proof is to show that there is a fastest mixing chain $\mathbf{P}$ on $\mathcal{G}'$ which has the following property: if nodes i and j are in the same box, then for any k, $\mathbf{P}_{ik} = \mathbf{P}_{jk}$, and $\mathbf{P}_{ii} = \mathbf{P}_{jj}$ as well. This allows to build an induced Markov chain. Indeed, if $\{X(t)\}_{t\in\mathbb{N}}$ is a Markov chain with a transition matrix $\mathbf{P}$ as described above, then the sequence $\{B(t)\}_{t\in\mathbb{N}}$, where $B(t)$ is the box that contains $X(t)$, is also a Markov chain. In other words, box transitions do not depend on the specific nodes $X(t)$, but only on the boxes $B(t)$.

The induced Markov chain $\{B(t)\}_{t\in\mathbb{N}}$ is defined on a lattice of $r^{-2}(n)$ super nodes with radius $k = 1$. In addition, it necessarily mixes faster than $\{X(t)\}_{t\in\mathbb{N}}$. Therefore, the fastest mixing time on the lattice of $r^{-2}(n)$ nodes and radius $k = 1$ lower bounds the fastest mixing time on $\mathcal{G}'$, which lower bounds the fastest mixing time on the initial RGG, which lower bounds the relaxation time of $\mathbf{W}'$, which is linked to $\mathbb{E}[\mathbf{W}]$ by Eq. (5.10). As Theorem 4.2.1 shows, this relaxation time is thus lower bounded by

$$\frac{1}{1-\lambda_2(\mathbf{W}')} = \Omega\left(r^{-2}(n)\right),$$

so that

$$\begin{aligned}\frac{1}{1-\lambda_2(\mathbb{E}[\mathbf{W}])} &= \Omega\left(nr^{-2}(n)\right)\\ &= \Omega\left(\frac{n^2}{\log n}\right).\end{aligned}$$

■

The upper bound of Theorem 5.1.4 can be proven is a more rigorous way[1], by using the weighted path bound (page 85, Theorem 4.1.4).

Proof: (Weighted path bound) In this proof, one path γ_{ij} should be designed between each pair of nodes (i,j). To design these paths, the unit square is divided in boxes of size $\alpha \log n/n$, with $\alpha > 2$, such that $r(n) \geqslant \sqrt{5\alpha \log n/n}$. Remember that this implies that a node can communicate with any node in any adjacent box (now adjacency is defined horizontally and vertically, but not in diagonal).

The first step of this proof is to build the paths γ_{ij} at the box level. More precisely, we impose that if $\gamma_{ij} = (i,i_1),(i_1,i_2),\ldots,(i_m,j)$, and if $b(i)$ denotes the box that contains node i, then the box-path $\Gamma_{ij} = (b(i),b(i_1)),(b(i_1),b(i_2)),\ldots,(b(i_m),b(j))$ should be the $(\leftrightarrow,\updownarrow)$-route between box $b(i)$ and box $b(j)$ in the lattice formed by the boxes (see Alg. A.1). If i and j are in the same box then $\gamma_{ij} = (i,j)$.

As a consequence, any path γ_{ij} has at most $2\sqrt{n/\alpha \log n}$ edges. For any box b_1 with n_1 nodes, and any box b_2 with n_2 nodes, there are $n_1 n_2$ paths to design (choose γ_{ji} to be the backwards path of γ_{ij}). Since one gets better bounds by sharing fairly the available edges among paths, one should refine carefully the box paths to the node level. By Theorem A.3.1, each box contains w.h.p. between $a \log n$ and $b \log n$ nodes. Therefore there are at least $a^2 \log^2 n$ edges between any two adjacent boxes. Since there are $n_1 n_2 \leqslant b^2 \log^2 n$ paths to be built, one can build all these paths by using each edge along the box-path at most $(b/a)^2$ times per box-path.

Now, combining Eq. (4.5) applied with $\mathbf{P} = \mathbb{E}[\mathbf{W}]$, Eq. (5.9), and the fact that any path γ_{ij} has at most $2\sqrt{n/\alpha \log n}$ edges, gives:

$$\begin{aligned} |\gamma_{ij}| &\leqslant 2\sqrt{\frac{n}{\alpha \log n}} \cdot n \cdot d_2 n \log n \\ &= \frac{2d_2}{\sqrt{\alpha}}\, n^2 \sqrt{n \log n}. \end{aligned}$$

For any edge e,

$$\begin{aligned} \sum_{\gamma_{ij} \ni e} |\gamma_{ij}| \boldsymbol{\pi}(i)\boldsymbol{\pi}(j) &\leqslant \frac{2d_2}{\sqrt{\alpha}}\, n^2 \sqrt{n \log n} \cdot \frac{1}{n^2} \cdot \mathcal{C}_e \\ &= \frac{2d_2}{\sqrt{\alpha}} \sqrt{n \log n}\, \mathcal{C}_e, \end{aligned}$$

where $\mathcal{C}_e$ is the number of paths γ_{ij} that contain edge $e = (e_1,e_2)$. To upper bound $\mathcal{C}_e$, it is enough to compute the number of box-paths containing the box-edge $\big(b(e_1), b_{e(2)}\big)$ and multiplying it by $(b/a)^2$. Similarly to proof on the lattice (Th. 5.1.2), and based on Fig. 5.2,

$$\mathcal{C}_e \leqslant \frac{1}{4}\left(\frac{n}{\alpha \log n}\right)^{1.5} \frac{b^2}{a^2}.$$

[1] When using the conductance bound, we did not look for the cut-set $(B,\bar{B})$ which exactly minimizes the ergodic flow from B to $\bar{B}$ divided by $\boldsymbol{\pi}(B)$,

Therefore, for any edge e,

$$\begin{aligned}\sum_{\gamma_{ij}\ni e} |\gamma_{ij}|\boldsymbol{\pi}(i)\boldsymbol{\pi}(j) &\leqslant \frac{2d_2}{\sqrt{\alpha}}\sqrt{n\log n}\,\frac{1}{4}\left(\frac{n}{\alpha\log n}\right)^{1.5}\frac{b^2}{a^2}\\ &= \frac{d_2 b^2}{2\alpha^2 a^2}\,\frac{n^2}{\log n}.\end{aligned}$$

Thus, by the weighted path bound theorem,

$$\frac{1}{1-\lambda_2(\mathbb{E}[\mathbf{W}])} \leqslant \frac{d_2 b^2}{2\alpha^2 a^2}\,\frac{n^2}{\log n}.$$

Therefore, by Eq. 5.2, $T_{ave} = O(n^2)$. ■

5.2 Geographic Gossip and Neighborhood Gossip

The previous section on pairwise gossip might have seemed long, but the proofs that were developed there, establish most of the techniques used to analyze the other gossip algorithms. This section of geographic gossip and neighborhood gossip, and the next section on path averaging are variations of the previous proofs. Therefore, a good understanding of previous sections can help to read Sections 5.2 and 5.4.

Geographic gossip was described in page 23, and neighborhood gossip was in page 22. In a few words, in geographic gossip, any pair of nodes can be averaged to the expense of routing. In neighborhood gossip, the whole neighborhood of the node that randomly wakes up averages its estimates.

The convergence of both algorithms is obvious in the i.i.d setting, which is assumed from now on:

- *Geographic gossip:* Assume that the random routes are i.i.d. and define the overlay graph $\mathcal{G}' = (V, \mathcal{E}')$, where $e = (i,j) \in \mathcal{E}'$ if and only if nodes i and j pairwise average their estimates with a positive probability. If the graph $\mathcal{G}'$ is connected, then $\mathbb{E}[\mathbf{W}]$ is irreducible, and by Perron-Frobenius theorem, $\lambda_2(\mathbb{E}[\mathbf{W}]) < 1$. If the network $\mathcal{G}$ is dense enough to be connected, then $\mathcal{G}'$ is connected as well, since $\mathcal{G} \subseteq \mathcal{G}'$. Therefore, if $\mathcal{G}$ is connected, then Perron-Frobenius applies, and by Theorem 3.1.4, *geographic gossip converges to average consensus with probability* 1.

- *Neighborhood gossip:* If nodes wake up i.i.d. and if the network is jointly connected (see Def. 5.1.1), then by the same Perron-Frobenius argument as for geographic gossip, *neighborhood gossip converges to average consensus with probability* 1.

5.2.1 Speed of Convergence of Geographic Gossip

Geographic gossip is an interesting algorithm, which speeds up the performance of pairwise gossip by routing the estimates in the network. Even though the routing is expensive, the gain in algorithmic iterations is such that, as we will see in this section, routing is worth the effort.

On the lattice, nodes are perfectly regularly spread out in the network, which implies that, when a node wakes up in geographic gossip, it chooses another node in the network uniformly at random. In random geometric graphs however, nodes are irregularly spread out, and the distribution of chosen nodes is not uniform anymore. Indeed, routes can encounter holes. In addition, nodes located in sparsely populated regions are called more often. For this reason, the analysis of geographic gossip is separated in two. Lattices are considered first, since their analysis is straightforward. Then the RGG case is discussed.

Performance of geographic gossip on lattices

We first present the performance of geographic gossip on lattices, where $(\leftrightarrow, \updownarrow)$-routing can be used to route estimates.

Theorem 5.2.1 (Performance of geographic gossip on a lattice). The averaging time $T_{ave}(n, \epsilon, \eta)$ of geographic gossip on a lattice with n nodes, for $\epsilon = \beta/n$ and $\eta = 1/n^{\alpha}$, scales w.h.p. as:

$$T_{ave}(n, \epsilon, \eta) = \Theta\left(n \log n\right).$$

The total expected cost in $T_{ave}(n, \epsilon, \eta)$ iterations scales as $\Theta\left(n^{1.5} \log n\right)$. In addition, the consensus cost $\mathcal{E}_c$ (see Section 3.2.3), i.e. the number of sent messages asymptotically needed to reduce the error by a factor e, scales as:

$$\mathcal{E}_c = O\left(n^{1.5}\right).$$

Proof: In the lattice, any pair of nodes can average their estimates at every iteration with the same probability. Therefore the algorithm works as pairwise gossip in a complete graph. Therefore,

$$\frac{1}{1 - \lambda_2(\mathbb{E}[\mathbf{W}])} = \Theta(n).$$

Since in gossip algorithms, for any iteration t, $\mathbf{W}^T(t)\mathbf{W}(t) = \mathbf{W}(t)$, then $\lambda_2(\mathbb{E}[\mathbf{W}^T\mathbf{W}]) = \lambda_2(\mathbb{E}[\mathbf{W}])$, and T_{ave} scales as predicted by the theorem. Based on the fact that the expected length of the random routes scales as $\Theta\left(\sqrt{n}\right)$,

$$\begin{aligned} \mathcal{E}_c &= \mathbb{E}[\mathcal{E}]T_{ave} && (5.14)\\ &= \Theta\left(\sqrt{n}\right)\Theta\left(\frac{1}{1 - \lambda_2(\mathbb{E}[\mathbf{W}])}\right) && (5.15)\\ &= \Theta\left(n^{1.5}\right). && (5.16) \end{aligned}$$

■

Comments: Although in geographic gossip every iteration costs more messages than in pairwise gossip, geographic gossip saves $\sqrt{n}$ messages compared to pairwise gossip on lattices. The idea is that routing is order $\sqrt{n}$ times more expensive than communicating with direct neighbors only, but it saves order n gossip iterations.

Performance of geographic gossip on random geometric graphs

Remember how greedy routing is defined. When a node needs to create a random route, it starts by choosing a target position uniformly at random in the unit square where the network is embedded. Then nodes successively and greedily forward the packet to their neighbor which is closest to the target. The route stops where there is no such closer node. Two lemmas show that greedy routing reaches the node that is the closest from the target w.h.p. if the connection radius $r(n)$ is large enough, first on the unit torus, then on the unit square (see Appendix).

Theorem 5.2.2 (Performance of geographic gossip on an RGG). Assume that Conjecture A.3.1 is true, i.e. assume that the median Voronoi area scales as $\Omega(1/n)$. Then, the averaging time $T_{ave}(n, \epsilon, \eta)$ of geographic gossip on an RGG with n nodes and connection radius $r(n) > \sqrt{4 \log n / n}$, for $\epsilon = \beta/n$ and $\eta = 1/n^{\alpha}$, scales w.h.p. as:

$$T_{ave}(n, \epsilon, \eta) = \Theta\left(n \log n\right).$$

The total expected cost in $T_{ave}(n, \epsilon, \eta)$ iterations scales as $\Theta\left(n^{1.5}\sqrt{\log n}\right)$. In addition, the consensus cost $\mathcal{E}_c$ (see Section 3.2.3), i.e. the number of sent messages asymptotically needed to reduce the error by a factor e, scales as:

$$\mathcal{E}_c = O\left(\frac{n^{1.5}}{\sqrt{\log n}}\right).$$

Proof: A direct consequence of Lemma A.2.1 is that at any iteration, any node i is called with probability $V(i)$, where $V(i)$ is the area of the Voronoi cell of $V(i)$ (for border node, consider the intersection of the infinite Voronoi cell and of the unit square). Therefore, for $i \neq j$

$$\mathbb{E}[\mathbf{W}_{ij}] = \frac{V_i + V_j}{n}. \tag{5.17}$$

Let $V_{\lfloor n/2 \rfloor}$ be the median Voronoi area in the network, and form two sets A and B of nodes:

$$A = \{\text{nodes } i : V_i > V_{\lfloor n/2 \rfloor}\} \tag{5.18}$$
$$B = \{\text{nodes } i : V_i \leqslant V_{\lfloor n/2 \rfloor}\}. \tag{5.19}$$

Then, use the weighted path bound (page 85, Theorem 4.1.4) to compute an upper bound on $1/(1 - \lambda_2(\mathbb{E}[\mathbf{W}]))$, which according to Corollary 3.2.1, rules the speed of convergence of gossip algorithms.

For every pair of distinct nodes i and j, one path γ_{ij} should be designed, using "edges" weighted by the entries of $\mathbb{E}[\mathbf{W}]$. Here edges do not necessarily correspond to physical network edges since $\mathbb{E}[\mathbf{W}]$ is a full matrix, whereas the network is not a complete graph. There are two cases: "at least i or j is in A" and "both i and j are in B".

- *Case 1: "at least i or j is in A"* This case is the simpler case. Take

$\gamma_{ij} = (i, j)$. Then,

$$\begin{aligned} |\gamma_{ij}| &= \frac{1}{\boldsymbol{\pi}_{ij}\mathbb{E}[\mathbf{W}_{ij}]} \\ &= \frac{n^2}{V_i + V_j} \\ &\leqslant \frac{n^2}{V_{\lfloor n/2 \rfloor}}, \end{aligned} \tag{5.20}$$

since V_i or/and V_j is/are larger than $V_{\lfloor n/2 \rfloor}$.

- *Case 2: "both i and j are in B"* To design an efficient path between nodes with small Voronoi cells, it is useful to have it stop in A in order to use only "edges" with large weights. In other words, $\gamma_{ij} = ((i, k), (k, j))$, where k is a node in set A. Since there is the same number of nodes in A and in B, one can build all the paths $\{\gamma_{ij}\}_{i,j \in A}$, such that every directed edge between A and B is used only once.

$$\begin{aligned} |\gamma_{ij}| &= \frac{1}{\boldsymbol{\pi}_{ik}\mathbb{E}[\mathbf{W}_{ik}]} + \frac{1}{\boldsymbol{\pi}_{kj}\mathbb{E}[\mathbf{W}_{kj}]} \\ &= \frac{n^2}{V_i + V_k} + \frac{n^2}{V_k + V_j} \\ &\leqslant \frac{2n^2}{V_{\lfloor n/2 \rfloor}}, \end{aligned} \tag{5.21}$$

Since "edges" between nodes of A and nodes of B were used is Case 1 once and in Case 2 once as well, each "edge" is used in total at most twice. Therefore, for any "edge" e,

$$\begin{aligned} \sum_{\gamma_{ij} \ni e} |\gamma_{ij}| \boldsymbol{\pi}(i)\boldsymbol{\pi}(j) &\leqslant \sum_{\gamma_{ij} \ni e} \frac{2}{V_{\lfloor n/2 \rfloor}} \\ &\leqslant \frac{4}{V_{\lfloor n/2 \rfloor}}. \end{aligned}$$

As a consequence, $\kappa \leqslant 4/V_{\lfloor n/2 \rfloor}$, and

$$\frac{1}{1 - \lambda_2(\mathbb{E}[\mathbf{W}])} \leqslant \frac{4}{V_{\lfloor n/2 \rfloor}}.$$

Therefore,

$$T_{ave}(n, \epsilon, \eta) = O\left(\frac{\log n}{V_{\lfloor n/2 \rfloor}}\right),$$

and since $O\left(\sqrt{n/\log n}\right)$ messages are used in average at each iteration,

$$\mathcal{E}_c = O\left(\frac{\sqrt{n}}{V_{\lfloor n/2 \rfloor}\sqrt{\log n}}\right).$$

Assuming that $V_{\lfloor n/2 \rfloor} = \Omega(1/n)$ yields the theorem. ■

Comments: In the original paper [27] on geographic gossip, the proposed algorithm is different from the one presented in this thesis. The difference comes from the way to deal with the heterogeneity of the Voronoi cell sizes. In the original work, nodes with large Voronoi cells should reject from time to time the requests to average, in order to give their chance to other nodes with smaller Voronoi cells. In order not to reject too often requests so that the total number of messages does not explode, the authors also need Conjecture A.3.1, which stipulates that there are not too many small Voronoi cells to "save", so that the rejecting threshold can be taken high enough.

In this thesis, the point of view is different since large Voronoi cells are seen as a *strength*, as they relay the diffusion load of smaller Voronoi cells. There is therefore no need to reject messages, which uselessly wastes routing energy. Mathematically speaking, accepting all requests does not hurt nodes with small Voronoi cells, since these nodes wake up with probability $1/n$ anyway, which is equal to the order of magnitude of the size of most Voronoi cells in the network.

If the network that is used in practice does not follow Conjecture A.3.1, we suggest that nodes with large Voronoi cells do not reject messages either, in order to avoid unnecessary routings, which cost energy. Instead, nodes with large Voronoi cells could wake up with a smaller rate, so that nodes with small Voronoi cells globally wake up with a higher probability. However, that trick might not be necessary to maintain the performance of geographic gossip. Indeed, note that if Conjecture A.3.1 does not hold, then there are nodes with very large Voronoi area, which become very strong diffusion relays.

It would be interesting to determine in future work the set of "calling" probability distributions, such that geographic gossip with uniform i.i.d. waking ups has linear relaxation time. This set is larger than one can think. Take the following extreme example, where $n-1$ nodes $(1, 2, \ldots, n-1)$ are called with probability $p = \epsilon(n)/(n-1)$ each, where $\epsilon(n)$ goes to 0 when n grows. Note that node n is then necessarily called with probability $1-\epsilon(n)$. Then, for any node $i = 1, 2, \ldots, n-1$,

$$\begin{aligned}\mathbb{E}[\mathbf{W}_{in}] &= \frac{1}{n}(1-\epsilon(n)) + \frac{1}{n}\frac{\epsilon(n)}{n-1} \\ &= \Theta\left(\frac{1}{n}\right).\end{aligned}$$

Now build the following paths γ_{ij} in order to use the weighted path bound. For any pair of nodes i and j that are not equal to node n, build $\gamma_{ij} = ((i,n),(n,j))$, $\gamma_{i,n} = (i,n)$ and $\gamma_{n,i} = (n,i)$, so that any $|\gamma_{ij}|$ is of order $\Theta(n^2)$. It is easy to see that every directed edge (i,n) and (n,i) is used in $n-1$ paths each. Therefore,

$$\sum_{\gamma_{ij} \ni e} |\gamma_{ij}| \boldsymbol{\pi}(i)\boldsymbol{\pi}(j) = \Theta(n),$$

which implies that the relaxation time $1/(1-\lambda_2(\mathbb{E}[\mathbf{W}]))$ has order $O(n)$. This example showed that if there are many nodes with small Voronoi cells, it is possible that the remaining nodes, being stronger, are able to ensure the diffusivity of the algorithm by themselves. This example can be generalized easily:

if there are k nodes with Voronoi area larger than a threshold θ, then

$$\frac{1}{1-\lambda_2(\mathbb{E}[\mathbf{W}])} \leqslant \frac{n}{k\theta}.$$

Therefore if k and θ can be found such that $k\theta = \Theta(1)$, then the relaxation time behaves linearly.

Note that the use of Conjecture A.3.1 in [27] is hidden. However the authors give an interesting hint to support it. For a *given* node i, they show with a simple geometric argument that:

$$\mathbb{P}\left[V(i) > \frac{c}{n}\right] \geqslant 1 - 4c. \tag{5.22}$$

If the Voronoi cell sizes $\{V_i\}_{i=1,\ldots,n}$ were independent, this would show that there is indeed a constant proportion of nodes with Voronoi cell size of order $\Omega(1/n)$. The analysis of the distribution of Voronoi cell areas is a difficult problem, which is often studied only heuristically, with the support of computer simulations [34].

5.2.2 Speed of Convergence of Neighborhood Gossip

Neighborhood gossip is a gossip algorithm that was designed to take advantage of the broadcast nature of wireless communications. In pairwise gossip, when a node i wakes up, all its neighbors hear its estimate, but only one gets to use that information. In neighborhood gossip, on the contrary, all the neighbors send their estimates back to node i. Node i computes the average of all the estimates in the neighborhood and broadcasts it, so that node i and its neighbors all update their estimates to the neighborhood average. At the end of one iteration, there were $m+1$ messages sent, where m is the size of the neighborhood, counting node i.

Theorem 5.2.3 (Performance of neighborhood gossip on lattices of radius r). The averaging time $T_{ave}(n,\epsilon,\eta)$ of neighborhood gossip on a lattice with n nodes and connection radius $r \in \mathbb{N}^*$, for $\epsilon = \beta/n$ and $\eta = 1/n^{\beta'}$, scales as:

$$T_{ave}(n,\epsilon,\eta) = \Theta\left(\frac{n^2}{m^2}\log n\right),$$

where m is the size of a typical neighborhood: $m = (2r+1)^2 = \Theta(r^2)$. The total expected cost in $T_{ave}(n,\epsilon,\eta)$ iterations scales as $\Theta\left(n^2 \log n/r^2\right)$. In addition, the consensus cost $\mathcal{E}_c$ (see Section 3.2.3), i.e. the number of sent messages asymptotically needed to reduce the error by a factor e, scales as:

$$\mathcal{E}_c = O\left(\frac{n^2}{r^2}\right).$$

Comments: Note that neighborhood gossip and pairwise gossip actually share the same order of consensus cost $\mathcal{E}_c$. Therefore taking advantage of the broadcast nature of wireless does not save any messages. However, neighborhood is interesting both for time and energy if "computation coding" is used, as explained in [58, 57], where the authors introduce and justify neighborhood gossip. The idea is that the neighbors can send their estimates back to node

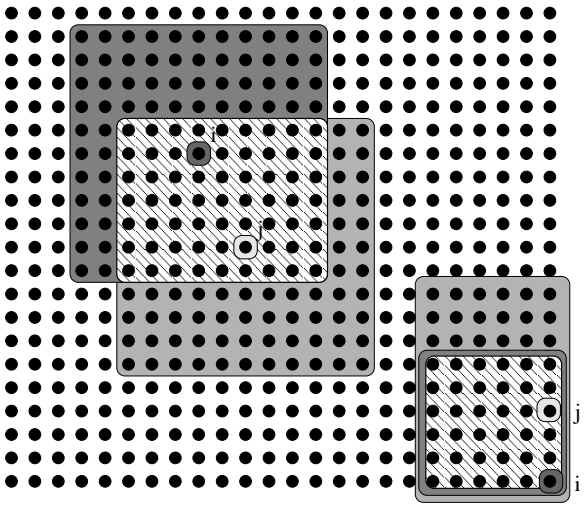


Figure 5.4: Counting the number of common neighbors of two nodes i and j, in the case where i and j are far from the edges, and in the case where i and j are close the edges.

i all at the same time. Using linear codes, the estimates add up "in the air", and node i directly recovers the sum of the estimates. In addition, since the emissions of all the neighbors add up in constructive interference, the received signal has a better signal over noise ratio, which helps to reduce the energy consumption of the network. The reader is encouraged to read [58, 57] for further details.

The proof of the performance theorem of neighborhood gossip in this thesis differs from the one given in [58], which is fairly short due to page constraints. The proof given here is not only more detailed, but also different, since it is based on the weighted path bound, whereas the authors in [58] chose the conductance bound.

Proof: The first step of this proof is to compute $\mathbb{E}[\mathbf{W}]$, which is slightly more technical than in previous Sections. If at iteration t, node i wakes up, then $\mathbf{W}(t) = \mathbf{W}^{(i)}$, where $\mathbf{W}^{(i)}_{jk} = 1/|\mathcal{N}_i|$, if and only if j and k are in the neighborhood $\mathcal{N}_i$ of node i, including node i itself. Otherwise, $\mathbf{W}^{(i)}_{kk} = 1$, and if $j \neq k$, $\mathbf{W}^{(i)}_{jk} = 0$. Since nodes wake up uniformly i.i.d. at random, if $i \neq j$, then

$$\mathbb{E}[\mathbf{W}_{ij}] = \frac{1}{n} \sum_{k \in \mathcal{N}_i \cap \mathcal{N}_j} \frac{1}{|\mathcal{N}_k|}. \tag{5.23}$$

If i and j are deep inside the lattice so that there is no border effects, then $|\mathcal{N}_k| = m$ for any k that is a neighbor of both i and j (i and j included). Thus in that case,

$$\mathbb{E}[\mathbf{W}_{ij}] = \frac{1}{nm} \mathcal{C}_{ij},$$

where $\mathcal{C}_{ij}$ is the number of nodes that have both i and j as neighbors. For any pair of nodes i and j, with integer lattice coordinates $(x(i), y(i))$ and

$(x(j), y(j))$, let $\delta_{ij} = \max(|x(i) - x(j)|, |y(i) - y(j)|) = \|i - j\|_\infty$ be the distance between i and j. As shown in the upper left part of Fig. 5.4, in the case where nodes i and j are far enough from the edges of the lattice, and if $\delta_{ij} \leqslant \sqrt{m} = 2r + 1$,

$$\begin{aligned} \mathcal{C}_{ij} &= \left(\sqrt{m} - |x(i) - x(j)|\right)\left(\sqrt{m} - |y(i) - y(j)|\right) \\ &\geqslant \left(\sqrt{m} - \delta_{ij}\right)^2. \end{aligned}$$

Therefore, if $\delta_{ij} < \sqrt{m}/2$, i.e. if $\delta \leqslant r$, then $\mathcal{C}_{ij} > m/4$ and

$$\mathbb{E}[\mathbf{W}_{ij}] = \frac{1}{nm}\mathcal{C}_{ij} > \frac{1}{4n}. \tag{5.24}$$

As illustrated by Fig. 5.4, Eq. (5.24) also holds for nodes that are subjected to edge effects. Indeed, note that neighborhoods contain at most m nodes, which implies that in Eq. (5.23), $1/|\mathcal{N}_k| > 1/m$. Furthermore, $\mathcal{C}_{ij} > m/4$ is still true as long as $\delta \leqslant r$.

The second part of the proof consists in applying the weighted path bound (page 85, Theorem 4.1.4) to the matrix $\mathbb{E}[\mathbf{W}]$, which is doubly stochastic. The proof here is analogous to the proof of pairwise gossip on a lattice of radius r. We repeat the proof with different orders of magnitude. In order to avoid redundancy, this proof is written in less details. Therefore, if needed, refer to the other proof in page 137.

A path γ_{ij} using "edges" of matrix $\mathbb{E}[\mathbf{W}]$ should be constructed between every pair of nodes i and j. To this end, divide the lattice in boxes of size $r/2 \times r/2$ such that any pair of nodes in adjacent boxes can be linked by an "edge" of weight larger than $1/4n$. Denote by $b(i)$ the box that contains node i. Then, for any pair of nodes i and j, construct the path $\gamma_{ij} = (i, i_1, \ldots, i_k, j)$ such that the sequence of boxes $(b(i), b(i_1), \ldots, b(i_k), b(j))$ is the horizontal-vertical box-path between $b(i)$ and $b(j)$. Paths can be designed such that for any given horizontal-vertical box-path, each edge is used in only one path γ_{ij} (see the proof of pairwise gossip in page 137).

The lengths of the paths that have just been designed are shorter than $2\sqrt{n}/(r/2) = 4\sqrt{n}/r$. Therefore,

$$\begin{aligned} |\gamma_{ij}| &\leqslant 4\frac{\sqrt{n}}{r} \cdot n \cdot 4n \\ &\leqslant \frac{16n^{2.5}}{r}. \end{aligned}$$

Therefore, for any edge e,

$$\sum_{\gamma_{ij} \ni e} |\gamma_{ij}| \boldsymbol{\pi}(i)\boldsymbol{\pi}(j) \leqslant \frac{16n^{2.5}}{rn^2}\mathcal{C}_e,$$

where $\mathcal{C}_e$ is the number of paths γ_{ij} that contain edge e. Using the same

argument as before, based on Fig. 5.2, for any edge e,

$$\begin{aligned} \mathcal{C}_e &\leqslant \frac{1}{4}\left(\frac{\sqrt{n}}{r/2}\right)^3 \\ &= 2\left(\frac{\sqrt{n}}{r}\right)^3. \end{aligned}$$

As a consequence,

$$\begin{aligned} \sum_{\gamma_{ij}\ni e} |\gamma_{ij}|\boldsymbol{\pi}(i)\boldsymbol{\pi}(j) &\leqslant 2\left(\frac{\sqrt{n}}{r}\right)^3 \frac{16\sqrt{n}}{r} \\ &= \frac{32n^2}{r^4}. \end{aligned}$$

Since $m^2 > 4r^2$,

$$\sum_{\gamma_{ij}\ni e} |\gamma_{ij}|\boldsymbol{\pi}(i)\boldsymbol{\pi}(j) \leqslant \frac{128n^2}{m^2}.$$

Thus, $\kappa = O(n^2/m^2)$, and

$$\frac{1}{1-\lambda_2(\mathbb{E}[\mathbf{W}])} = O\left(\frac{n^2}{m^2}\right).$$

Noting that each iteration demands order $\mathcal{E} = \Theta(m)$ messages,

$$\mathcal{E}_c = \mathbb{E}[\mathcal{E}]T_c = O\left(\frac{n^2}{m}\right).$$

■

Comment: An efficient way to mentally recover the result is to note that pairwise gossip on a lattice of radius r needs $O(n^2/m)$ iterations ($m = (2r+1)^2$), and has non-zeros coefficients $\mathbb{E}[\mathbf{W}_{ij}]$ m times smaller than neighborhood gossip. Therefore neighborhood gossip converges m times faster, and needs $O(n^2/m^2)$ iterations.

5.3 Broadcast Gossip

Just as neighborhood gossip, broadcast gossip is a distributed averaging algorithm designed to take advantage of the broadcast nature of wireless communications. Indeed, when a node wakes up and sends its estimate, the whole neighborhood hears it, but in pairwise gossip, only one node takes it into account. On the other hand, when an estimate is heard in neighborhood gossip, *all the neighbors* reply to the woken node. Disappointingly, the reply is too costly, and neighborhood gossip does not reduce the message cost compared to pairwise gossip (however, as explained in previous section, neighborhood gossip can reduce energy and time if using computation codes). Broadcast gossip is precisely an answer to this issue, since it does not require a reply from the neighborhood. Note that the costly neighborhood reply is necessary to maintain the global average equal to the initial average x_{ave}. Therefore, broadcast gossip

does not conserve the global average through iterations. Instead, it conserves average *in expectation*.

Broadcast gossip is originally described in two papers [31] and [7]. This section places their results in a slightly different network model, since a network reliability parameter p is added (as explained a first time in page 28) and since the network is modeled here as a time-varying *directed* graph $\{\mathcal{G}(t)\}_{t\in\mathbb{N}}$ ([7] worked on undirected graphs). At each iteration t, $\mathcal{G}(t) = (V, \mathcal{E}(t))$ is assumed here to be a subset of an undirected fixed graph $\mathcal{G} = (V, \mathcal{E})$, where every directed[2] edge $e \in \mathcal{E}$ of $\mathcal{G}$ is also an edge of $\mathcal{G}(t)$ with probability $p > 0$ independently of other edges and of time: for any directed edge of $\mathcal{G}$,

$$\mathbb{P}[e \in \mathcal{E}(t)] = p, \tag{5.25}$$

and $\{\mathcal{G}(t)\}_{t\in\mathbb{N}}$ is an i.i.d. sequence of graphs.

At every iteration t, a node i wakes up uniformly at random and broadcasts its estimate $x_i(t)$. Any node j, such that the directed edge (i, j) is in $\mathcal{G}(t)$, hears the estimate and updates to:

$$x_j(t+1) = \gamma x_j(t) + (1-\gamma)x_i(t),$$

for some parameter $\gamma \in [0, 1)$.

5.3.1 Convergence

Theorem 5.3.1. Assume that the sequence of directed networks can be modeled as in Eq. (5.25). If $\mathcal{G}$ is connected, then broadcast gossip converges to consensus with probability 1:

$$\mathbb{P}\left[\lim_{t\to\infty} \mathbf{x}(t) = c\mathbf{1}\right] = 1.$$

In addition, the consensus value is equal to x_{ave} in expectation:

$$\mathbb{E}[c] = x_{ave}.$$

Proof: First note that the network model assumes that the sequence of networks is i.i.d., so that the sequence of matrices $\{\mathbf{W}(t)\}_{t\in\mathbb{N}}$ is i.i.d. as well. Let $\boldsymbol{\eta}(t) = \mathbf{x}(t) - x_{ave}\mathbf{1}$, then, as shown in Section 3.1.4,

$$\mathbb{E}[\|\boldsymbol{\eta}(t)\|_2^2 \,|\boldsymbol{\eta}(t-1)] \leqslant \lambda_1\left(\mathbb{E}[\mathbf{W}(t)^T(\mathbf{I}-\mathbf{J})\mathbf{W}(t)]\right)\|\boldsymbol{\eta}(t-1)\|_2^2. \tag{5.26}$$

By Eq. (4.49) at page 126, $\lambda_1\left(\mathbb{E}[\mathbf{W}(t)^T(\mathbf{I}-\mathbf{J})\mathbf{W}(t)]\right) < 1$ if $\mathcal{G}$ is connected. Indeed, if $\mathcal{G}$ is connected, then $\lambda_1(\mathbf{L}_{\mathcal{G}}) > 0$. Eq. (5.26) implies the convergence of the algorithm with probability 1 by Theorem 3.1.4.

In addition, it can be proven that $\mathbb{E}[\mathbf{W}]$ is an irreducible doubly stochastic matrix. In particular, $\mathbb{E}[\mathbf{W}]\mathbf{1} = \mathbf{1}$ and $\mathbf{1}^T\mathbb{E}[\mathbf{W}] = \mathbf{1}^T$. $\mathbb{E}[\mathbf{W}]$ is aperiodic since for every node i, $\mathbb{E}[\mathbf{W}]_{ii} > 0$. The connectivity of $\mathcal{G}$ implies the irreducibility of $\mathbb{E}[\mathbf{W}]$, and by Perron-Frobenius theorem, $\lambda_2(\mathbb{E}[\mathbf{W}]) < 1$. By Theorem 3.1.6,

[2] an undirected edge is considered as two directed edges in opposite directions.

broadcast gossip thus converges in expectation:

$$\lim_{t\to\infty} \mathbb{E}[\boldsymbol{\epsilon}(t)] = 0,$$

which by Lebesgue theorem implies that $\mathbb{E}[c - x_{ave}] = \mathbb{E}[\lim_{t\to\infty} \boldsymbol{\epsilon}(t)] = 0$. ■

5.3.2 Speed of Convergence

Remember that time of convergence T'_{ave} relates here to $\|\boldsymbol{\eta}\|_2$, which measures the distance to consensus c, and not to $\|\epsilon\|_2$, which measures the distance to x_{ave}, since only $\|\boldsymbol{\eta}\|_2$ goes to zero with probability 1. As explained in page 54, for $\epsilon = \beta/n$ and $\eta = 1/n^{\beta'}$,

$$\begin{aligned} T'_{ave}(n,\epsilon,\eta) &= \Omega\left(\frac{\log n}{1-\lambda_2(\mathbb{E}[\mathbf{W}])}\right) \\ T'_{ave}(n,\epsilon,\eta) &= O\left(\frac{\log n}{1-\lambda_1(\mathbb{E}[\mathbf{W}^T(\mathbf{I}-\mathbf{J})\mathbf{W}])}\right), \end{aligned}$$

which allows to derive the following theorem:

Theorem 5.3.2 (Speed of convergence of broadcast gossip). The averaging time $T'_{ave}(n,\epsilon,\eta)$ of broadcast gossip on a time-varying graph $\mathcal{G}(t)$ with n nodes modeled as in Eq. (5.25), for $\epsilon = \beta/n$ and $\eta = 1/n^{\beta'}$, scales as:

$$\begin{aligned} T'_{ave}(n,\epsilon,\eta) &= \Omega\left(\frac{n\log n}{p(1-\gamma)\lambda_1^{\mathbf{L}_{\mathcal{G}}}}\right) \\ T'_{ave}(n,\epsilon,\eta) &= O\left(\frac{n\log n}{p\gamma(1-\gamma)\lambda_1^{\mathbf{L}_{\mathcal{G}}}}\right), \end{aligned}$$

where $\lambda_1^{\mathbf{L}_{\mathcal{G}}}$ is the second smallest eigenvalue of the Laplacian matrix of graph $\mathcal{G}$, which is the base graph of the sequence of graphs $\{\mathcal{G}(t)\}_{t\in\mathbb{N}}$ (see Section 4.4 for a summary on Laplacians).

Proof: Proving this theorem is purely computational, and it is not very enlightening. We state only the end results:

$$\begin{aligned} \mathbb{E}[\mathbf{W}] &= \mathbf{I} - \frac{p(1-\gamma)\mathbf{L}^{\mathcal{G}}}{n} \\ \mathbb{E}[\mathbf{W}^T(\mathbf{I}-\mathbf{J})\mathbf{W}]) &= \mathbf{I} - \mathbf{J} - 2\left(\frac{p\gamma(1-\gamma)}{n} - \frac{p(1-p)(1-\gamma)^2}{n^2}\right)\mathbf{L}_{\mathcal{G}} \\ &\quad - \frac{p^2(1-\gamma)^2}{n^2}\mathbf{L}_{\mathcal{G}}^2. \end{aligned}$$

■

Applications: For any node i, let $d(i)$ be the degree of node i. Based on some inequalities on laplacian eigenvalues stated in Section 4.4,

$$\begin{aligned} T'_{ave}(n,\epsilon,\eta) &= \Omega\left(\frac{n\ \mathrm{diam}(\mathcal{G})^2}{p(1-\gamma)\max\{d(i), i\in V\}\log n}\right) \\ T'_{ave}(n,\epsilon,\eta) &= O\left(\frac{n^2\ \mathrm{diam}(\mathcal{G})\log n}{p\gamma(1-\gamma)}\right). \end{aligned}$$

In a random geometric graph $\mathcal{G}$ with connection radius larger than $\sqrt{3\log n/n}$ so that $\mathcal{G}$ is log-balanced with high probability,

$$\begin{aligned} T'_{ave}(n,\epsilon,\eta) &= \Omega\left(\frac{n^2}{p(1-\gamma)\log^3 n}\right) \\ T'_{ave}(n,\epsilon,\eta) &= O\left(\frac{n^{2.5}\sqrt{\log n}}{p\gamma(1-\gamma)}\right). \end{aligned}$$

Finally, as derived previously in Section 4.4, the coefficient γ^*, which minimizes the largest eigenvalue of $\mathbb{E}[\mathbf{W}^T(\mathbf{I}-\mathbf{J})\mathbf{W}])$ is

$$\gamma* = \frac{n+2(1-p)-p\lambda_1^{\mathbf{L}_\mathcal{G}}}{2n+2(1-p)-p\lambda_1^{\mathbf{L}_\mathcal{G}}},$$

which is close to 1/2 on random geometric graphs.

[7] also works on the accuracy of the final estimation c of x_{ave} by studying the limiting behavior of $\|\epsilon(t)\|_2$. In particular, the authors show that the accuracy improves with increasing coefficient γ. But γ close to 1 implies slow convergence. There is thus a tradeoff between fast convergence and accuracy in broadcast gossip.

5.4 Path Averaging

In path averaging, every iteration generates a random route in the network so that all the nodes in the route update their estimates to their average. For example, if the random route has four nodes with estimates (x_1, x_2, x_3, x_4), then the four nodes update to $(x_1+x_2+x_3+x_4)/4$. Path averaging conserves the average through iterations, and it is a gossip algorithm.

5.4.1 Convergence

As for any gossip algorithm in the i.i.d. setting, by Eq. (3.48) and by Theorem 3.1.4 applied to $\|\epsilon(t)\|_2^2$, path averaging converges to x_{ave} with probability 1 if $\lambda_2(\mathbb{E}[\mathbf{W}]) < 1$. $\lambda_2(\mathbb{E}[\mathbf{W}]) < 1$ holds if $\mathbb{E}[\mathbf{W}]$ is irreducible and aperiodic (Perron-Frobenius theorem). Aperiodicity is guaranteed by the fact that for any node i, $\mathbb{E}[\mathbf{W}]_{ii} > 0$, and irreducibility holds if the overlay network defined by the positive entries of $\mathbb{E}[\mathbf{W}]$ is connected.

As in geographic gossip, on the torus, if $r(n) > \sqrt{4\log n/n}$, then *random* greedy routing reaches the node that is the closest to the target with high probability. On the unit square, take $r(n) > \sqrt{52\log n/9n}$. In that case, $\mathbb{E}[\mathbf{W}]$ is a full matrix, and it is irreducible. Note that having a full matrix $\mathbb{E}[\mathbf{W}]$ is not

necessary, we only require that its overlay graph is connected (not necessarily complete). For $\mathbb{E}[\mathbf{W}]$ to be irreducible, it is thus sufficient that the network $\mathcal{G}$ is connected, and it is not necessary that random greedy routing succeeds. Indeed, the probability that a node draws the target in the voronoi cell of one of its neighbors is positive. Hence if i and j are neighbors $\mathbb{E}[\mathbf{W}]_{ij} > 0$.

As a consequence, *path averaging converges to x_{ave} with probability* 1*, as soon as the network is connected.* However, as the Sections 5.4.3 and 5.4.4 show, path averaging converges faster if random greedy routing succeeds in reaching far away nodes. Put a figure that displays the mean route length as a function of c.

5.4.2 The Influence of the Routing Protocol.

The distribution of the random paths depends on the routing protocol that is used. It is therefore a key feature of the algorithm. This thesis discusses three different routing protocols:

1. *Greedy routing* is the routing used in geographic gossip. When a node wakes up it draws uniformly at random in the domain a target position. Then it greedily routes the message to its neighbor which is the *closest* to the target. This routing is convenient for geographic gossip, because it generates short and cheap routes. On the contrary, for path averaging, the greedy forwarding of message is too deterministic, which limits the diversity of routes and thus it reduces the mixing of the algorithm.

2. *Random greedy routing* (Fig. 5.5) is more adapted to path averaging since it adds randomness to the routing scheme. Indeed, in random greedy routing, the node greedily routes the message to a randomly chosen node among its neighbors that are *closer* to the target. This routing allows close nodes to communicate, whereas greedy routing always chooses the node that is closest to the target, i.e. one of the most distant neighbors. The increased randomness adds diversity to the routes, and helps the mixing of the information. See Fig. 5.7 to compare the mean route length of greedy routing and random greedy routing. As expected, random greedy routes are longer, which helps the convergence of path averaging. Surprisingly, the routes are already long before the graphs are connected. i.e. even before $c \simeq 1$ (we checked that fact on non connected graphs only, but we do not show it here). This illustrates how large the radius is required to be connect the "last" isolated node.

   The drawback of this efficient routing is that it is very hard to analyze in theory. Although simulations with random greedy routing are very promising, we had to define another routing: box routing.

3. $(\leftrightarrow, \updownarrow)$-*Box routing* is the routing which is studied in Sections 5.4.3 and 5.4.4. On the lattice we can define $(\leftrightarrow, \updownarrow)$-routing (Fig. 5.6(a), Alg. A.1), which allows to build on the RGG a modified version of $(\leftrightarrow, \updownarrow)$-routing, based on a box partition of the space. Divide the unit square in boxes of area $\alpha \log n/n$, where $\alpha > 1$, so that any box has at least one node with high probability (see Lemma A.1.1). Take $r(n) > \sqrt{5\alpha \log n/n}$ so that any node can communicate with any other node in any adjacent box (there are 4 adjacent boxes). Then when a node wakes up, it sends

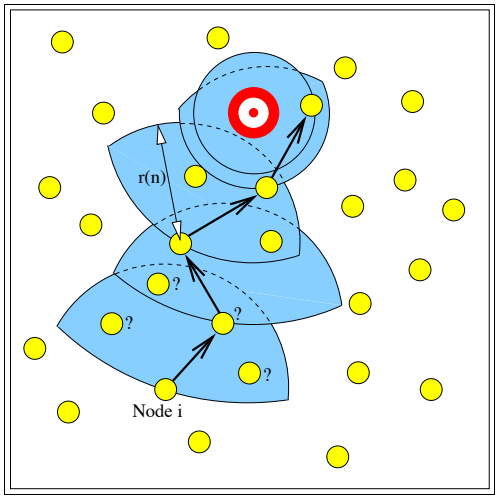


Figure 5.5: Random greedy routing. Node i has to choose the following node in the route among its neighbors that are closer to the target. In other words, it chooses a node uniformly at random in the intersection of the ball of radius $r(n)$ centered in node i and of the ball of radius d centered in the target, where d is the distance between node i and the target.

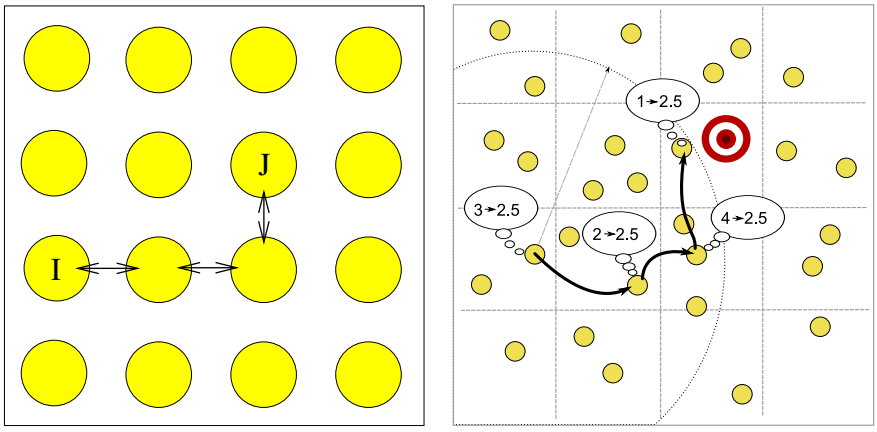
the message along the $(\leftrightarrow, \updownarrow)$ box-route towards the box that contains the target in the following way: nodes greedily choose the next node of the route uniformly at random among the nodes in the next box of the $(\leftrightarrow, \updownarrow)$-box route. The final node is thus a random node in the box which contains the target (Fig. 5.6(b), Alg. A.2).

Box-routing is easier to study because it canalizes the routes in tractable areas. However, as it is, the problem of computing the performance of path averaging is still difficult. We also get rid of edge effects by embedding the unit square on a torus. This allows to count routes that goes through a given node independently of the position of the node. Finally, if $\alpha > 3$ then each box has $\Theta(\log n)$ nodes w.h.p., which is needed to avoid unfair situations, and reach linear performance. This imposes $r(n) > \sqrt{15 \log n / n}$.

As a conclusion, box-routing is theoretically tractable, but in practice it is too constraining, and edge effects do exist. After the analysis, we will discuss on simulations the behavior of the algorithm when removing some of these constraints, and we will give suggestions to overcome the edge effects. The discussion will be based on the proofs of Sections 5.4.3 and 5.4.4, which are very enlightening although they hold in a rather artificial setting.

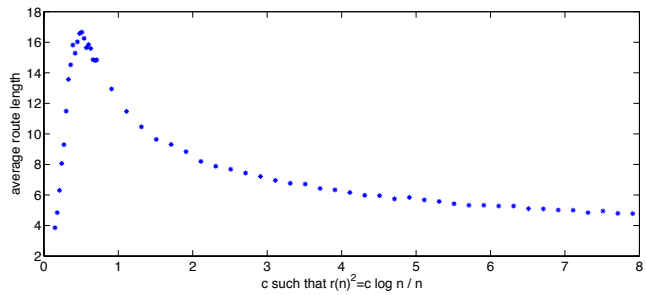
5.4.3 Speed of Convergence on Toroidal Lattices

The first step in the analysis is understanding the behavior of path averaging on toroidal lattices using $(\leftrightarrow, \updownarrow)$-routing. Throughout this manuscript, a toroidal

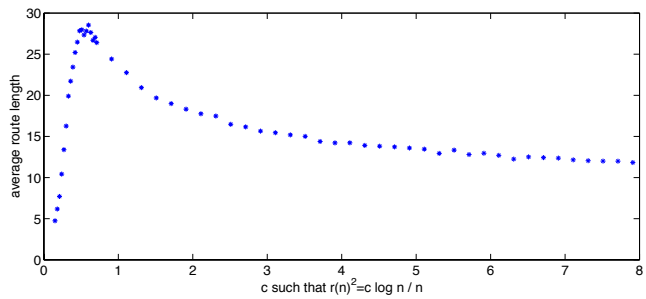


(a) $(\leftrightarrow, \updownarrow)$-route (b) box-path averaging

Figure 5.6: (a) Shortest $(\leftrightarrow, \updownarrow)$-route from I to J on the grid. (b) Example of box-path averaging on an RGG: The node with inital value 3 selects a random position and places a target. Using $(\leftrightarrow, \updownarrow)$-box routing towards that target, all the nodes on the path replace their values with the average of the four nodes.



(a) greedy routing



(b) random greedy routing

Figure 5.7: The average length of routes with greedy routing and random greedy routing on random geometric graphs of increasing connection radius. Each point is an average over 25 graphs and 50 routes each. Each graph has 2300 nodes.

lattice of n nodes is a 4-connected lattice on a torus of size $\sqrt{n} \times \sqrt{n}$. Assuming that each node knows its location in the lattice, $(\leftrightarrow, \updownarrow)$-path averaging performs as follows: At each iteration t, a randomly selected node I wakes up and selects a random destination node J so that the pair (I, J) is independently and uniformly distributed. The path between I and J is defined by the *shortest* $(\leftrightarrow, \updownarrow)$-route between I and J (see Fig. 5.6(a) and the description of $(\leftrightarrow, \updownarrow)$-routing in Alg. A.1 for non-toroidal lattices). On the torus, $(\leftrightarrow, \updownarrow)$-routing routes horizontally first, then vertically, always choosing the direction (left or right, up or down) which will generate the shortest route. The estimates of all the nodes on this path are aggregated and averaged by messages passed on this path, and at the end of the iteration the estimates of the nodes on this path are updated to their global average.

Theorem* 5.4.1 ($(\leftrightarrow, \updownarrow)$-path averaging on toroidal lattices). Let $\epsilon = \beta/n$ and $\eta = 1/n^{\beta'}$ On a $\sqrt{n} \times \sqrt{n}$ toroidal lattice, the averaging time $T_{ave}(n, \epsilon, \eta)$ of $(\leftrightarrow, \updownarrow)$-path averaging, described above, is $O(\sqrt{n} \log n)$. Let $C_{ave}(n, \epsilon, \eta)$ be the expected averaging cost in $T_{ave}(n, \epsilon, \eta)$ iterations: $C_{ave}(n, \epsilon, \eta) = \mathbb{E}[R] T_{ave}(n, \epsilon, \eta)$, where $\mathbb{E}[R]$ is the expected route length. Then $C_{ave}(n, \epsilon, \eta) = O(n \log n)$.

Proof: The proof of Theorem 5.4.1 is three-fold. First we define some notations, then we compute lower bounds on the entries of $\mathbb{E}[\mathbf{W}]$, and finally we upper bound the relaxation time of $\mathbb{E}[\mathbf{W}]$.

Notations We need to define the shortest distance on a torus. To this end, we introduce a torus absolute value $|.|_{\mathcal{T}}$ and a torus L_1 norm $\|.\|_1$. For any algebraic value x on a one dimensional torus (circle with $\sqrt{n}$ nodes) and any vector i on a $\sqrt{n} \times \sqrt{n}$ two dimensional torus,

$$\begin{aligned} |x|_{\mathcal{T}} &= \min(|x|, |x - \sqrt{n}|, |x + \sqrt{n}|) \\ \|i\|_1 &= |i_x|_{\mathcal{T}} + |i_y|_{\mathcal{T}}. \end{aligned}$$

We call $\ell_{ij} = \|j - i\|_1$ the L_1 distance between nodes i and j. The shortest routes between I and J have $\alpha = \ell_{IJ} + 1 = |J_x - I_x|_{\mathcal{T}} + |J_y - I_y|_{\mathcal{T}} + 1$ nodes to be averaged, thus the non-zero coefficients of their corresponding matrices $\mathbf{W}$ are all equal to $1/\alpha$.

To each route r, we assign a generalized gossip $n \times n$ matrix $\mathbf{W}^{(r)}$ that averages the current estimates of the nodes on the route. Consequently, at iteration t, $\mathbf{W}(t) = \mathbf{W}^{(r(t))}$, where $r(t)$ was randomly chosen. We call R the route random variable, $s(R)$ its starting node, $d(R)$ its destination node, and $\ell(R) = \ell_{s(R)d(R)} + 1$ its number of nodes. As we choose the shortest route, the maximum number of nodes a route can contain is $\sqrt{n}$ if $\sqrt{n}$ is odd, $\sqrt{n} + 1$ if $\sqrt{n}$ is even, which can be written as $2\lfloor \sqrt{n}/2 \rfloor + 1$ in short.

Evaluating $\mathbb{E}[\mathbf{W}]$ Far away nodes are less likely to be jointly averaged compared to neighboring ones (see Figure 4.5). The following lemma quantifies the behavior of $\mathbb{E}[\mathbf{W}_{i,j}]$ as a function of the distance between i and j. The main consequence of this lemma is that nodes that are separated by a distance smaller than $\sqrt{n}/2$ are strongly averaged together.

Lemma* 5.4.1. *(Expected $\mathbb{E}[\mathbf{W}]$ on the toroidal lattice)* For any pair of nodes (i, j), if their distance normalized to the maximum distance $\delta_{ij} = \|j - i\|_1/\sqrt{n}$

is smaller than a constant, then

$$\mathbb{E}[\mathbf{W}_{i,j}] = \Omega\left(\frac{1}{n^{1.5}}\right). \tag{5.27}$$

More precisely,

$$\mathbb{E}[\mathbf{W}_{i,j}] \geq \frac{2(1-\delta_{ij}+\delta_{ij}\log\delta_{ij})}{n\sqrt{n}}.$$

Proof:

For any given pair of nodes (i,j), we want to compute the (i,j)th entry of the matrix expectation $\mathbb{E}[\mathbf{W}]$. Only the $(\leftrightarrow,\updownarrow)$-routes which contain both these two nodes i and j will have a non-zero contribution in $\mathbb{E}[\mathbf{W}_{ij}]$. Pick such a route r, the (i,j)th entry of the corresponding averaging matrix is $\mathbf{W}_{i,j}^{(r)} = 1/\ell(r)$. We call $\mathcal{R}_{ij}^{\ell}$ the set of $(\leftrightarrow,\updownarrow)$-routes with ℓ nodes passing by node i and by node j, and denote $x^+ = \max(x,0)$. It is not hard to see that $(\ell-\ell_{ij})^+$ is the number of routes of length ℓ passing by i first and j next (see Fig. 5.8), so $|\mathcal{R}_{ij}^{\ell}| = 2(\ell-\ell_{ij})^+$. We thus have for any $i \neq j$:

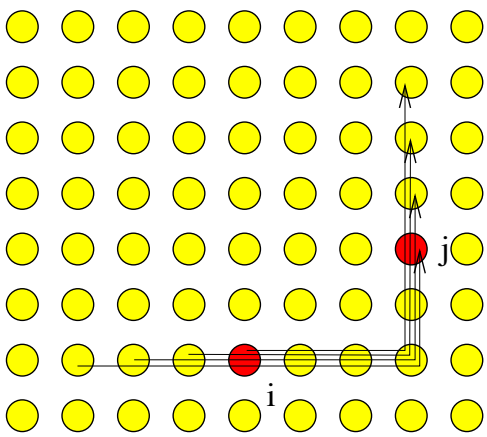


Figure 5.8: Counting the number of routes of length $\ell = 9$ nodes, in the case where $\ell_{ij} = 5$. There are $\ell - \ell_{ij} = 9 - 5 = 4$ possible routes with exactly ℓ nodes going through node i then through node j. We admit only routes going horizontally first then vertically.

$$\begin{aligned}
\mathbb{E}[\mathbf{W}_{i,j}] &= \sum_r \mathbf{W}_{i,j}^{(r)} \mathbb{P}[R=r] \\
&= \frac{1}{n^2} \sum_r \mathbf{W}_{i,j}^{(r)} \\
&= \frac{1}{n^2} \sum_{\ell=\ell_{ij}+1}^{2\lfloor \frac{\sqrt{n}}{2} \rfloor + 1} \frac{|\mathcal{R}_{ij}^{\ell}|}{\ell} \\
&= \frac{2}{n^2} \sum_{\ell=\ell_{ij}+1}^{2\lfloor \frac{\sqrt{n}}{2} \rfloor + 1} \frac{\ell - \ell_{ij}}{\ell},
\end{aligned}$$

from which we can deduce the following upper and lower bounds when $i \neq j$:

$$\begin{aligned}
\mathbb{E}[\mathbf{W}_{i,j}] &\leq \frac{2}{n^2} \int_{\ell_{ij}+1}^{\sqrt{n}+2} \frac{x - \ell_{ij}}{x} dx \\
&= \frac{2}{n^2} \left(\sqrt{n} - \ell_{ij} + 1 - \ell_{ij} \ln \frac{\sqrt{n}+2}{\ell_{ij}+1} \right). \\
\mathbb{E}[\mathbf{W}_{i,j}] &\geq \frac{2}{n^2} \int_{\ell_{ij}}^{\sqrt{n}} \frac{x - \ell_{ij}}{x} dx \\
&= \frac{2}{n^2} \left(\sqrt{n} - \ell_{ij} - \ell_{ij} \ln \frac{\sqrt{n}}{\ell_{ij}} \right).
\end{aligned}$$

$\mathbb{E}[\mathbf{W}_{i,j}]$ decreases from $\frac{2}{n\sqrt{n}}$ to $o(\frac{1}{n^2})$ as a function of ℓ_{ij}. To get a normalized expression with respect to $\sqrt{n}$, we use the coefficient δ_{ij} defined in the statement of Lemma 5.4.1.

$$\begin{aligned}
&\frac{2}{n\sqrt{n}} (1 - \delta_{ij} + \delta_{ij} \ln \delta_{ij}) \leq \mathbb{E}[\mathbf{W}_{i,j}] \leq \\
&\qquad \frac{2}{n\sqrt{n}} \left(1 - \delta_{ij} + \delta_{ij} \ln \delta_{ij} + \frac{1}{\sqrt{n}} - \delta_{ij} \ln \frac{\sqrt{n}+2}{\sqrt{n} + \frac{1}{\delta_{ij}}} \right).
\end{aligned}$$

This establishes the claim. In particular, if $\delta_{ij} = 1/2$, then $\mathbb{E}[\mathbf{W}_{i,j}] \sim \frac{1-\ln 2}{n\sqrt{n}}$. ■

Bounding $\lambda_2(\mathbb{E}[\mathbf{W}])$ We need now to upperbound the second largest eigenvalue in magnitude of $\mathbb{E}[\mathbf{W}]$, or equivalently, the relaxation time $1/(1-\lambda_2(\mathbb{E}[\mathbf{W}]))$.

Lemma* 5.4.2 (Relaxation time). The relaxation time of the matrix $\mathbb{E}[\mathbf{W}]$ in $(\leftrightarrow, \updownarrow)$-path averaging on toroidal lattices scales as:

$$\frac{1}{1 - \lambda_2(\mathbb{E}[\mathbf{W}])} = O(\sqrt{n}).$$

Proof: The Poincaré inequality (Theorem 4.1.4) bounds the second largest eigenvalue of a stochastic matrix and not necessarily its second largest eigenvalue *in magnitude*, which is the important quantity involved in Th. 3.2.1. It could happen that the smallest negative eigenvalue is larger in magnitude than

the second largest eigenvalue. Consequently, if we show that all the eigenvalues of $\mathbb{E}[\mathbf{W}]$ are positive, then the two eigenvalues coincide and we can use the Poincaré inequality to bound the second largest eigenvalue in magnitude. $\mathbb{E}[\mathbf{W}]$ is symmetric so all its eigenvalues are real. The sum of all the entries along the lines of $\mathbb{E}[\mathbf{W}]$ without counting the diagonal element is $O(1/\sqrt{n})$, whereas the diagonal elements are $\Theta(1)$, so by Gershgorin bound (Th. 4.1.5), all the eigenvalues of $\mathbb{E}[\mathbf{W}]$ are positive.

We can now use the bounds on $\mathbb{E}[\mathbf{W}]$ to bound its spectral gap with Poincaré inequality. $\mathbb{E}[\mathbf{W}]$ defines an overlay network $\mathcal{G}'$, where for any nodes i and j, (i,j) is an edge if $\mathbb{E}[\mathbf{W}_{ij}] > 0$. We say that edge (i,j) has a good capacity if $\mathbb{E}[\mathbf{W}_{ij}]$ is large, i.e. if $\delta_{ij} \leqslant 1/2$. From now on, an "edge" is an edge of the overlay network, and it is thus not necessarily an edge of the physical network the algorithm is running on.

We want to prove that path averaging performs $\sqrt{n}$ better than geographic gossip, where $\mathbb{E}[\mathbf{W}_{i,j}] = 1/n^2$. It is encouraging to note that for $\delta_{ij} \leqslant 1/2$, $\mathbb{E}[\mathbf{W}_{i,j}] \geqslant \frac{1-\ln 2}{n\sqrt{n}}$, which is precisely $\sqrt{n}$ better than $1/n^2$. It is thus possible to find good capacity edges with length equal to half of the whole graph. However very distant destinations remain problematic. Consider the extreme case of a distance $\sqrt{n}$ between two nodes i and j. There are only two routes that jointly average them: the route that goes from i to j, and the reverse one. These routes are selected with probability $1/n^2$ and $\mathbf{W}_{ij} = 1/\sqrt{n}$, implying that $\mathbb{E}[\mathbf{W}_{ij}] = 2/n^{2.5} \ll 1/n^{1.5}$.

Formally, for each ordered and distinct pair (i,j), we choose a 2-hop path γ_{ij} from i to j stopping by a node k chosen to be located approximatively half way between i and j. To be more precise, we define direction functions σ_x and σ_y, where $\sigma_x(i,j) = 1$ (respectively, $\sigma_y(i,j) = 1$) if the horizontal (resp., vertical) part of the route from i to j goes to the right (resp., up) and $\sigma_x(i,j) = -1$ (resp., $\sigma_y(i,j) = -1$) if it goes left (resp., down). The coordinates of k in the torus are:

$$\begin{aligned} k_x &= \left(i_x + \sigma_x(i,j)\lfloor\frac{|j_x - i_x|_{\mathcal{T}}}{2}\rfloor\right) \pmod{\sqrt{n}} \\ k_y &= \left(i_y + \sigma_y(i,j)\lfloor\frac{|j_y - i_y|_{\mathcal{T}}}{2}\rfloor\right) \pmod{\sqrt{n}}. \end{aligned}$$

In the road map γ we have just constructed, the longest edge is smaller than $\frac{\sqrt{n}}{2} + 1$ in L_1 distance. Therefore, for any edge e in γ, $\delta_e \leqslant 1/2$, and according to Lemma 5.4.1, $\mathbb{E}[\mathbf{W}_e] \geqslant \eta/n^{1.5}$, where η is a non negative constant slightly smaller than $1 - \ln 2$. Thus, for each path γ_{ij} we have:

$$\begin{aligned} |\gamma_{ij}| &= \frac{1}{\pi(i)\mathbb{E}[\mathbf{W}_{i,k}]} + \frac{1}{\pi(k)\mathbb{E}[\mathbf{W}_{k,j}]} \\ &= n\left(\frac{1}{\mathbb{E}[\mathbf{W}_{i,k}]} + \frac{1}{\mathbb{E}[\mathbf{W}_{k,j}]}\right) \\ &\leq \frac{2n^2\sqrt{n}}{\eta}. \end{aligned} \tag{5.28}$$

We can now compute the Poincaré coefficient:

$$\kappa = \max_e \sum_{\gamma_{ij} \ni e} |\gamma_{ij}| \pi_i \pi_j = \frac{1}{n^2} \max_e \sum_{\gamma_{ij} \ni e} |\gamma_{ij}|. \tag{5.29}$$

To compute this sum, we need to count the number of paths γ_{ij} that use a given edge e. In our construction, an edge e belongs to at most 8 paths. Indeed, if a path contains edge e, then e is either its first or second edge. In the first case, by construction, the second edge has to be approximatively as long as e. Moreover, because of quantized grid effects, there are actually only 4 different possible edges that can succeed e (see Fig. 5.9). Repeating this argument in the case where e is the second edge, we then obtain that an edge e appears in at most 8 paths. Combining (5.28) and (5.29), we get:

$$\kappa \leq \frac{16}{\eta} \sqrt{n}.$$

As a result,

$$\lambda_2(\mathbb{E}[\mathbf{W}]) \leq 1 - \frac{\eta}{16\sqrt{n}},$$

which yields Lemma 5.4.2. ■

The proof of Theorem 5.4.1 is completed by combining Lemma 5.4.2 and Th. 3.2.1. ■

In the next Section, we generalize this proof from lattices to log-balanced geometric graphs. The approach will be the same but the detailed computations will be different. Also, the construction of the paths will need some refinement.

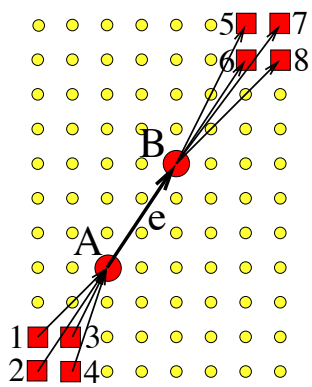


Figure 5.9: Number of paths including an edge $e = (A, B)$. Paths have two hops of equal length, where equality here is defined up to grid effects. Therefore, for a given edge e, there are at most 8 paths including e: $(1, A, B)$, $(2, A, B)$, $(3, A, B)$, $(4, A, B)$ and $(A, B, 5)$, $(A, B, 6)$, $(A, B, 7)$, $(A, B, 8)$.

5.4.4 Speed of Convergence on Toroidal Random Geometric Graphs

Box-path averaging is path averaging when using $(\leftrightarrow, \updownarrow)$-box routing instead of random greedy routing (see Item 3 in page 156) on a random geometric graph. The boxes are assumed to have size $\alpha \log n/n$ with $\alpha > 3$.

Theorem* 5.4.2 (Box-path averaging on toroidal RGG). Consider a random geometric graph $G(n, r)$ on the unit torus with $r(n) = \sqrt{\frac{c \log n}{n}}$, $c > 15$. Let $\epsilon = \beta/n$ and $\eta = 1/n^{\beta'}$. With high probability over graphs, the averaging time $T_{ave}(n, \epsilon, \eta)$ of box-path averaging is $O(\sqrt{n \log n} \log n)$. Furthermore, let $C_{ave}(n, \epsilon, \eta)$ be the expected averaging cost in $T_{ave}(n, \epsilon, \eta)$ iterations: $C_{ave}(n, \epsilon, \eta) = \mathbb{E}[R] T_{ave}(n, \epsilon, \eta)$, where $\mathbb{E}[R]$ is the expected route length. Then $C_{ave}(n, \epsilon, \eta) = O(n \log n)$.

Comments: A gossip algorithm which does not finish in finite time cannot converge with less than $n \log n$ messages up to a precision of order $\epsilon = \beta/n$ with high probability. Indeed, at least n messages are needed to average n values, and the $\log n$ factor is inherent to gossip. In centralized algorithms that can converge exactly in finite time, at least n messages are needed.

Therefore path averaging with box-routing on a torus balanced geometric graph is order optimal as far as gossip algorithms are concerned.

Proof: The proof of Theorem 5.4.2 follows the same organization as the proof of Theorem 5.4.1. First we define some notations, then we compute lower bounds on the entries of $\mathbb{E}[\mathbf{W}]$, and finally we upper bound the relaxation time of $\mathbb{E}[\mathbf{W}]$.

All the fundamental ideas coming from the proof on toroidal lattices in the previous section, appear here again, but sometimes in a more technical form.

Notations

In order to run $(\leftrightarrow, \updownarrow)$-box-routing, the unit torus is divided in boxes of size $\alpha \log n/n$. Hence, there are k boxes forming a torus grid as in the previous section and $k = \lceil \sqrt{n/(\alpha \log n)} \rceil^2 \simeq n/(\alpha \log n)$, for some $\alpha > 3$. Let ζ be the value close to α such that $k = n/(\zeta \log n)$.

Using the empirical uniformity theorem (Th. A.3.1), each box contains a number of nodes between $a \log n$ and $b \log n$. There are only a few modifications to make to the grid proof in order to obtain the balanced geometric graph proof. The idea is to notice that any route $r = (r_1, r_2, \cdots, r_\ell)$ can be attributed a box route $\widetilde{r}$ consisting of the boxes the nodes of r belong to. If $b(i)$ denotes the box node i belongs to, then $\widetilde{r} = (b(r_1), b(r_2), \cdots, b(r_\ell))$. Let n_i be the number of nodes in the box $b(i)$ node i belongs to. The sequence of n_i is fixed by the graph we are considering. The boxes form a grid and integer coordinates are assigned to each box. In this system of coordinates, for any pair of nodes i and j, ℓ_{ij} is the L_1 distance between boxes $b(i)$ and $b(j)$: $\ell_{ij} = \|b(j) - b(i)\|_1$. Denote by $\ell(r)$ the number of nodes in route r, $s(\widetilde{r})$ the starting box of route $\widetilde{r}$ and $d(\widetilde{r})$ its destination box. In our problem, the routes are random, and the random route variable is denoted by capital case letter: R, leading to other random variables $\widetilde{R}$, $\ell(R)$, $s(\widetilde{R})$, etc.

Evaluating $\mathbb{E}[\mathbf{W}]$

The following lemma lower bounds the entries of the matrix $\mathbb{E}[\mathbf{W}]$.

Lemma* 5.4.3 (Expected averaging matrix on a toroidal balanced geometric graph). Let $\mathbb{E}[\mathbf{W}]$ be the expected averaging matrix of box-path averaging on a toroidal balanced geometric graph. Let a and b be such that any box has between $a \log n$ and $b \log n$ nodes. For any pair of nodes (i, j) that do not belong to the same box, if their grid-distance normalized to the maximum grid-distance $\delta_{ij} = \ell_{ij}/\sqrt{k}$ is smaller than a constant, then

$$\mathbb{E}[\mathbf{W}_{ij}] = \Omega\left(\frac{1}{n\sqrt{n \log n}}\right). \tag{5.30}$$

More precisely,

$$\mathbb{E}[\mathbf{W}_{i,j}] \geq \frac{\zeta a}{b^2} \frac{2}{n^2} \sqrt{\frac{n}{\zeta \log n}} \left(1 - \delta_{ij} + \delta_{ij} \log \delta_{ij}\right). \tag{5.31}$$

Proof: For any node i and node j that do not belong to the same box, we want to compute the expectation of $\mathbf{W}_{ij}$. Counting the routes in this setting is complicated because each sender has at least $a \log n$ nodes to send its message to. In order to use our simple analysis of the grid, we condition the expectation on the box routes $\widetilde{R}$. Given a box route, $\mathbf{W}_{ij} = 0$ if i or j is not in the box route. On the contrary, if they both are in the box route, then $\mathbf{W}_{ij} = 1/\ell(\widetilde{R})$ with probability $1/(n_i n_j)$. Indeed, if i (or j) is in starting box, the probability that i is the starting node is $1/n_i$, because all the nodes wake up with the same rate. If i (or j) is in another box of the given box route, then the probability that i is chosen is $1/n_i$ as well, because the routing chooses next node uniformly among the nodes of the next box.

$$\begin{aligned}
\mathbb{E}[\mathbf{W}_{ij}] &= \mathbb{E}_{\widetilde{R}}\left[\mathbb{E}_R[\mathbf{W}_{ij}|\widetilde{R}]\right] \\
&= \mathbb{E}_{\widetilde{R}}\left[\frac{1}{n_i n_j} \frac{1}{\ell(\widetilde{R})} 1_{b(i)\in\widetilde{R}} 1_{b(j)\in\widetilde{R}}\right].
\end{aligned}$$

From now on, we are back to a problem with routes on a grid which has k "nodes". The difference with previous section is that routes are no longer uniform. Indeed, now, boxes wake up more frequently if they contain more nodes: the probability that box $b(i)$ wakes up is n_i/n. Destination boxes are still chosen uniformly at random with probability $1/k$ because there are k boxes in total. Note that $(\leftrightarrow, \updownarrow)$-box routes are entirely determined by their starting boxes and their destination boxes: $\mathbb{P}[\widetilde{R} = \widetilde{r}] = \mathbb{P}[s(\widetilde{R}) = s(\widetilde{r}), d(\widetilde{R}) = d(\widetilde{r})]$. We count box routes of different length separately as well. Let $\mathcal{R}_{ij}^{\ell}$ be the set

of box routes of size ℓ including $b(i)$ and $b(j)$.

$$\begin{aligned}
\mathbb{E}[\mathbf{W}_{ij}] &= \frac{1}{n_i n_j} \sum_{\widetilde{r}} \frac{1_{b(i)\in\widetilde{r}} 1_{b(j)\in\widetilde{r}}}{\ell(\widetilde{r})} \mathbb{P}[\widetilde{R} = \widetilde{r}] \\
&= \frac{1}{n_i n_j} \sum_{\ell=\ell_{ij}+1}^{2\lfloor\frac{\sqrt{k}}{2}\rfloor+1} \sum_{\widetilde{r}\in\mathcal{R}_{ij}^{\ell}} \frac{\mathbb{P}[\widetilde{R} = \widetilde{r}]}{\ell} \\
&= \frac{1}{n_i n_j} \sum_{\ell=\ell_{ij}+1}^{2\lfloor\frac{\sqrt{k}}{2}\rfloor+1} \sum_{\widetilde{r}\in\mathcal{R}_{ij}^{\ell}} \frac{\mathbb{P}[s(\widetilde{R}) = s(\widetilde{r}), d(\widetilde{R}) = d(\widetilde{r})]}{\ell} \\
&= \frac{1}{n_i n_j} \sum_{\ell=\ell_{ij}+1}^{2\lfloor\frac{\sqrt{k}}{2}\rfloor+1} \sum_{\widetilde{r}\in\mathcal{R}_{ij}^{\ell}} \frac{1}{\ell} \frac{n_{s(\widetilde{r})}}{n} \frac{1}{k}.
\end{aligned}$$

We now use the regularity of the graph : for any node m, $a \log n \leqslant n_m \leqslant b \log n$.

$$\begin{aligned}
\mathbb{E}[\mathbf{W}_{ij}] &\geqslant \frac{1}{(b\log n)^2} \sum_{\ell=\ell_{ij}+1}^{2\lfloor\frac{\sqrt{k}}{2}\rfloor+1} \frac{1}{\ell} \frac{a\log n}{n} \frac{\zeta \log n}{n} |\mathcal{R}_{ij}^{\ell}|. \\
&= \frac{\zeta a}{b^2} \frac{1}{n^2} \sum_{\ell=\ell_{ij}+1}^{2\lfloor\frac{\sqrt{k}}{2}\rfloor+1} \frac{|\mathcal{R}_{ij}^{\ell}|}{\ell} \\
&\geqslant \frac{\zeta a}{b^2} \frac{2}{n^2} \left(\sqrt{k} - \ell_{ij} - \ell_{ij} \ln \frac{\sqrt{k}}{\ell_{ij}} \right).
\end{aligned}$$

The last inequality comes from the same computation as for the lattice, and it can be reformulated as in Lemma 5.4.3 when using the normalized distance coefficient $\delta_{ij} = \ell_{ij}/\sqrt{k}$. ■

Bounding $\lambda_2(\mathbb{E}[\mathbf{W}])$

Lemma* 5.4.4 (Relaxation time of box-path averaging on a toroidal balanced geometric graph). The relaxation time of the matrix $\mathbb{E}[\mathbf{W}]$ in box-path averaging on toroidal balanced geometric graphs scales as:

$$\frac{1}{1 - \lambda_2(\mathbb{E}[\mathbf{W}])} = O(\sqrt{n \log n}). \tag{5.32}$$

Proof: As for the toroidal lattice, Poincaré's inequality is used to bound the relaxation time. The situation is very similar to the lattice case, except that boxes now contain $\Theta(\log n)$ nodes each.
Similarly to the toroidal lattice case, we will be using 2-hop paths for every pair of nodes, by adding one intermediate stop half-way. More precisely, this intermediate stop is chosen in the box whose coordinates on the underlying box-lattice are given by Eq. (5.28), where i and j are the lattice coordinates of the source and destination boxes. Once box-paths are fixed in this way, for each pair of nodes, we need to carefully and fairly assign the intermediate node

within the intermediate box that will become their relay node. It would not be wise to always choose the same intermediate node in the intermediate box for all the paths we need to design. Indeed an edge should not be used more than a constant number of times (it was 8 for the grid), otherwise it would create congestion. It is not hard to design such a set of paths because the number of nodes in each box varies at most by a constant multiplicative factor b/a.

To show this, assume that each box contains exactly the same number N of nodes. For any pair of boxes Box 1 and Box 3, let Box 2 be the half-way box that is chosen to be their relay box. There are N^2 paths to find between all the nodes in Box 1 and Box 3, but happily enough, there are N^2 edges between Box 1 and Box 2 and also between Box 2 and Box 3. Therefore, as we can see in Fig. 5.10, the box-path (Box 1, Box 2, Box 3) can correspond to N^2 node-paths all using different edges. This edge allocation technique can easily be extended to cases where the boxes do not have the same number of nodes, by using edges at most $\lceil b/a \rceil$ times each in every box-path. Indeed, in the worse congestion case, Box 1 and Box 3 have $b \log n$ nodes and Box 2 has $a \log n$ nodes only. First split Box 1 and Box 3 in groups of $a \log n$ nodes, which makes $\lceil b/a \rceil$ groups per box. Then design paths between each groups of Box 1 and Box 3 using intermediate nodes in Box 2 in exactly the same way as before, since the initial group, the intermediate box and the final group have the same number of nodes. For each pair of groups (group in Box 1, group in Box 3), all the edges between the group in Box 1 and Box 2, and all the edges between Box 2 and the group in Box 3 are used once and only once. Note that one group in each box has less than $a \log n$ nodes if b/a is not an integer; this is not an issue since it only implies that there are less paths to design and therefore less edges to use. Now, each group in Box 1 needs to connect to $\lceil b/a \rceil$ other groups in Box 3, and each of these $\lceil b/a \rceil$ group connections uses the same edges between the nodes of the group in Box 1 and the nodes in Box 2. The same reasoning holds for the edges between Box 2 and the groups of Box 3. Therefore, in the worse case, there is a path strategy that uses each edge at most $\lceil b/a \rceil$ times for each box-path.

There is a second refinement to the grid proof: solving the problem for

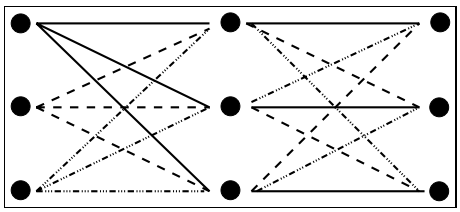


Figure 5.10: Path allocation when there are 3 nodes per box and thus 9 paths to design.

nodes that share a common box, which do not average jointly (Our bound on $\mathbb{E}[\mathbf{W}_{ij}]$ is zero). Note that there are strong edges to nodes in neighboring boxes. Formally, if node i and node j are in the same box, we design the path from i to j to be a two hop path stopping at a node located in the box above their box. By sharing fairly the available relay nodes within each box, the short

north-south edges might be used in $\lceil b/a \rceil$ extra paths.
We have constructed paths for any pair of nodes that uses at most $9\lceil b/a \rceil$ times each good capacity edge. First we have built the paths at the box level: each pair of boxes appears in at most 8 box paths as in previous theorem + 1 box-path to solve the lack of averaging inside a box, which makes 9 box-paths in total. Then we refined the box-paths at the node level. Each box-path between two boxes can be refined at the node level in such a way that edges in this path are used at most $\lceil b/a \rceil$ times. Therefore, each pair of nodes appears in at most $9\lceil b/a \rceil$ paths. The rest of the proof is identical to the grid proof.

For each path we have:

$$\begin{aligned} |\gamma_{ij}| &= \frac{1}{\pi(i)\mathbb{E}[\mathbf{W}_{i,k}]} + \frac{1}{\pi(k)\mathbb{E}[\mathbf{W}_{k,j}]} \\ &= n\left(\frac{1}{\mathbb{E}[\mathbf{W}_{i,k}]} + \frac{1}{\mathbb{E}[\mathbf{W}_{k,j}]}\right) \\ &\leq cn^2\sqrt{n\log n}, \end{aligned} \tag{5.33}$$

for some constant c. Inequality (5.33) was obtained with the same reasoning as on the lattice, using that $\delta_{i,k}$ and $\delta_{k,j}$ are smaller than the constant 1/2 and applying Lemma 5.4.3. We therefore conclude, using the Poincaré coefficient argument that

$$\kappa \leq 9\lceil \frac{b}{a} \rceil c\sqrt{n\log n}.$$

As a result, for n large enough, and some constant c',

$$\lambda_2 \leq 1 - \frac{1}{c'\sqrt{n\log n}},$$

which yields the lemma. ■
The proof of Theorem 5.4.2 is completed by combining Lemma 5.4.4 and Th. 3.2.1. ■

5.4.5 Simulations

Theorem 5.4.2 holds under constraints that might not be met in practice. In this section, we are going to discuss simulations where some on these constraints are relaxed.

Box-path averaging: the influence of the torus and of the box size.

Fig. 5.11 shows the behavior of the empirical consensus cost (see Def. 3.2.4) when running box-path averaging with parameters $\alpha = 2.5$ and $\alpha = 10$, on planar random geometric graphs, and on these graphs embedded in a torus. Here the cost is the number of sent messages, so that $\mathbb{E}[C]$ is proportional to the expected route length. As explained in page 74, it is necessary to check the the simulation samples have a low variance in order to make sure that they are meaningful. Measurements concentrate better on the torus than on a planar graph, because the latter induces edge effects and isolates the nodes on the edge of the graph domain. Similarly, the measurements concentrate better with large α, because the virtual boxes contain similar number of nodes for large

α, whereas with small α, some boxes are over-crowded compared to others. A node in a crowed box has less chance to be chosen when a route goes through its box, and it is paradoxically "isolated" in the algorithm. To summarize, since isolated nodes induce more variations in the convergence, small boxes, as well planar graphs present less concentrated measurements than large boxes and toruses.

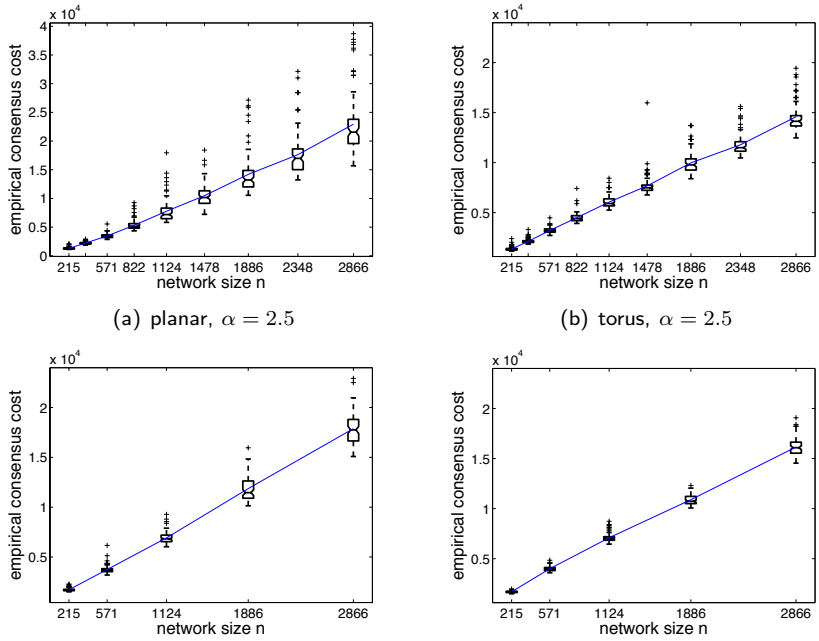
The parameter α does not influence much the empirical performance of box-path averaging on the torus; the algorithm performs better for $\alpha = 2.5$ than for $\alpha = 10$, probably because small α reduces the number of nodes within a virtual box (nodes in the same box never belong to the same route, and thus never directly average their estimates together, which is harmful for the performance of the algorithm), and maybe also because small α implies longer routes. On the contrary, on planar graphs, large $\alpha = 10$ reduces the isolation of nodes on the edge of the network domain, and it is beneficial to the performance of box-path averaging compared to small $\alpha = 2.5$.

Note that $\alpha = 2.5$ and 10 are both smaller than 15 which is needed in the theoretical analysis. On the simulations however, the behavior of consensus cost on the torus still looks linear.

Path averaging with random greedy routing.

Standard gossip, geographic gossip and path averaging were run on random geometric graphs with a growing number n of nodes in the unit square. First, Fig. 5.12(a) checks that the mean route length in path averaging is indeed almost of order $O(\sqrt{n})$ (it is $O(\sqrt{n/\log n})$ in theory, fact that is not proven in this thesis). Then, in order to evaluate and compare the performances of the algorithms, $T_{t_1}^{emp}(t)$ and $C_{t_1}^{emp}(t)$ were measured for $t_1 = 700$ and values of t increasing linearly with n, starting at $t = 1750$. The algorithms were run several times for every n by sampling different RGG's. We observed that the sets of measurements concentrate, not only with respect to the initial signal, but also with respect to the sampled graphs, which justifies their relevance. Fig 5.12(b), which displays average measurements of $C_{t_1}^{emp}(t)$, shows that path averaging with random greedy routing behaves strikingly better than standard gossip and geographic gossip, when, for example, $r(n) = \sqrt{c \log n/n}$ with $c = 4.5$. For other values of c, the performance of our algorithm also greatly improves previous gossip schemes.

Most importantly, for small connection radius $r(n)$ (small c), the number of messages $C_{t_1}^{emp}(t)$ behaves slightly super-linearly in n, and as c increases, the behavior improves. This is illustrated in Fig. 5.12(c) and 5.12(d), where the measurements are shown with boxplots. Boxplots display the lower and upper quartiles as well as the median of a set of measurements. In the figures, each boxplot represents 60 measurements, since for each n, 15 graphs were sampled and path averaging was run starting from 4 different signals on each of them. Measurements in Fig. 5.12(d) are better concentrated than measurements in Fig. 5.12(c), mainly because large radii $r(n)$ reduces edge effects (see Fig. 5.13). Note that a node which stays inactive for a long period of time, destroys the smoothness of convergence (Fig. 5.12(e)), and corrupts the concentration of the measurements. Nodes on the edges of the network domain are not likely to appear on routes; they are naturally isolated. These routing edge effects do not exist in standard and geographic gossip, which usually present very con-



(a) planar, $\alpha = 2.5$ (b) torus, $\alpha = 2.5$

(c) planar, $\alpha = 10$ (d) torus, $\alpha = 10$

Figure 5.11: Box-path averaging on RGG and the behavior of its empirical consensus costs. For a given α, box-path averaging is run on networks of size n, where n is chosen such that $\sqrt{n/(\alpha \log n)}$ is very close to an integer k. The unit area is split into virtual boxes of length $1/k$, so that the k^2 boxes have the same size, each of them containing in average approximatively $\alpha \log n$ nodes. For each figure, box-path averaging was run on 15 random geometric graphs per n, with 6 different random initial signals per graph: there are 90 measurements per boxplot. Each figure also displays the average empirical consensus cost with a solid line. Top figures are run with virtual boxes of parameter $\alpha = 2.5$, and bottom figures with $\alpha = 10$. Left figures show measurements on planar graphs (they include edge effects), right figures on toruses (without edge effects).

centrated measurements. The slight super-linearity of the empirical consensus cost for small c is probably due to these edge effects and to the connectivity unbalance (variance in the degree of the nodes), which is accentuated for small connection radii.

An efficient way to reduce edge effects for small c, is to systematically extend the routes until the edges of the network, by keeping the route direction (angle) unchanged. Even though path averaging with random greedy routing was not analyzed theoretically, it seems empirically very efficient.

5.5 One-way Path Averaging

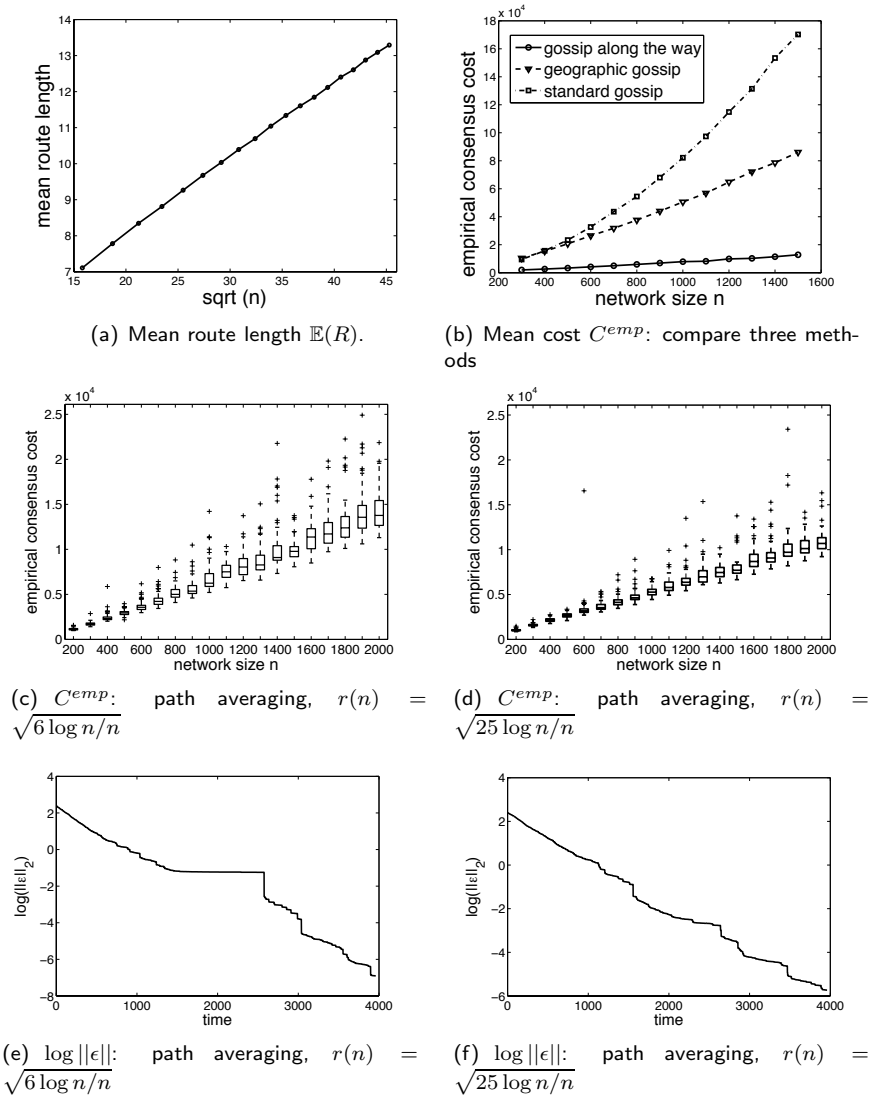
In this section, we will focus on the evolved version of one-way path averaging only (Alg. 2.12). Remember that one-way path averaging is an algorithm which updates estimates along routes, with no need of routing information back to the source of the route. This feature is the key strength of our algorithm, since it makes it robust against time-varying communication links. In order to make it possible, nodes keep in memory two variables instead of one. Each node i keeps a sum s_i and a weight ω_i, such that the estimate of the average at node i is $\mathbf{x}_i = s_i/\omega_i$. The sums $s_i(0)$ are initialized with the measurements $\mathbf{x}_i(0)$, and the weights are initialized with $\omega_i(0) = 1$. When a node i wakes up, it splits its sum s_i and its weight ω_i into a random number H of equal "chunks" $(s_i/H, \omega_i/H)$, and it distributes them along a random path of H nodes. Node i updates to $(s_i/H, \omega_i/H)$, and the $H-1$ other nodes in the route each take a chunk and add it to their own sum and weight. If the random route cannot be extended on H nodes because it reaches the edges of the network, the last border node collects the remaining chunks. If we were studying the naive version of one-way path averaging, the description of the algorithm would stop here. In the evolved version of one-way path averaging, when the second node i_2 of the route receives the chunks of node i, it divides its sum s_{i_2} and its weight ω_{i_2} into $H-1$ equal chunks before it adds the chunk it received from node i. Then, node i_2 sends the $H-2$ aggregate chunks $(s_i/H + s_{i_2}/(H-1), \omega_i/H + \omega_{i_2}/(H-1))$ to the remaining $H-2$ nodes of the route. Each node in the route aggregates in the same way its own contribution. The last node of the route thus receives contributions of all previous nodes in the route.

One-way path averaging is not a gossip algorithm, which makes its analysis based on tools on Chapter 3 impossible. However, we manage to prove the convergence of one-way path averaging, and we present simulations to illustrate its promising performance.

5.5.1 An Unbiased Estimator

As seen in Equations (2.15) and (2.16) in Section 2.2.4, the estimates in one-way path averaging are updated linearly, as in any distributed average consensus algorithm. In other words, at each averaging round k, there is an averaging matrix $\mathbf{W}(k)$, such that $\mathbf{x}(k) = \mathbf{W}(k)\mathbf{x}(k-1)$.

Property 5.5.1 (Averaging matrix W in one-way path averaging). At any averaging round k, the averaging matrix $\mathbf{W}(k)$ in one-way path averaging



(a) Mean route length $\mathbb{E}(R)$.

(b) Mean cost C^{emp}: compare three methods

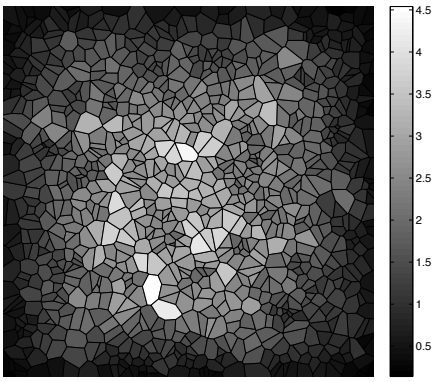
(c) C^{emp}: path averaging, $r(n) = \sqrt{6 \log n/n}$

(d) C^{emp}: path averaging, $r(n) = \sqrt{25 \log n/n}$

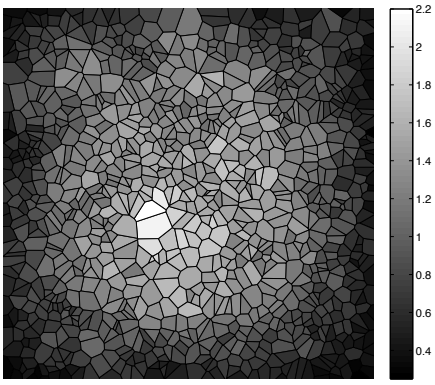
(e) $\log ||\epsilon||$: path averaging, $r(n) = \sqrt{6 \log n/n}$

(f) $\log ||\epsilon||$: path averaging, $r(n) = \sqrt{25 \log n/n}$

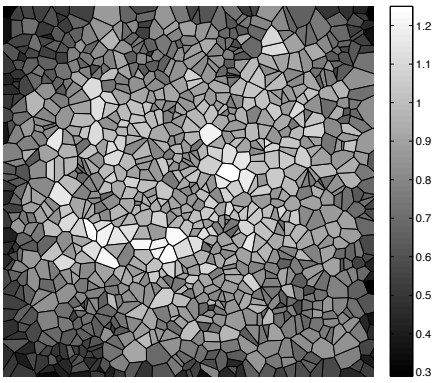
Figure 5.12: Performance of path averaging. The simulations were performed over 15 graphs per n and 4 runs per graph starting from different random initial signals. Empirical consensus costs $C^{emp}_{t_1}(t)$ were measured with $t_1 = 700$ and values of t increasing linearly with n starting from 1750. (a) The mean route length in random greedy routing behaves in $\sqrt{n/\log n}$. (b) Comparison between the average empirical consensus costs of standard gossip, geographic gossip (without rejection sampling) and path averaging with $r(n) = \sqrt{4.5 \log n/n}$. (c), (d) Empirical consensus costs C^{emp} for radii $r(n) = \sqrt{6 \log n/n}$ and $r(n) = \sqrt{25 \log n/n}$. (e), (f) Examples of error decay in $\log$ scale, for $n = 1500$ nodes and for radii $r(n) = \sqrt{6 \log n/n}$ and $r(n) = \sqrt{25 \log n/n}$. Larger radii imply smoother convergence, and more concentrated measurements.



(a) $c = 1$



(b) $c = 6$



(c) $c = 25$

Figure 5.13: Random greedy routing. For each node, we represent the empirical percentage of routes it participates in, by coloring its Voronoi cell according to a linear color scale. As c, and thus $r(n)$, increases, edge effects are attenuated. Note that isolated nodes in the network (large Voronoi cells) appear in more routes than nodes that are in densely populated zones (small Voronoi cells). Indeed, isolated nodes are "inevitable" hops when routes should cross sparsely populated zones, mostly when $r(n)$ is small.

is such that:

$$\mathbf{W}(k)\mathbf{1} = \mathbf{1}, \tag{5.34}$$
$$\mathbf{1}^T\mathbf{W}(k) \neq \mathbf{1}^T. \tag{5.35}$$

Eq. (5.34) in the above property guarantees that if consensus is reached, then consensus is kept. The second equation (Eq. (5.35)) shows that the average is not conserved through iterations, which contrasts with most previous algorithms. Only broadcast gossip had averaging matrices which did not admit $\mathbf{1}$ as a left eigenvector, but its expected averaging matrix $\mathbb{E}[\mathbf{W}]$ did: $\mathbf{1}^T\mathbb{E}[\mathbf{W}] = \mathbf{1}^T$. On the contrary, in one-way path averaging, even the expected averaging matrix does not admit $\mathbf{1}$ as a left eigenvector: $\mathbf{1}^T\mathbb{E}[\mathbf{W}] \neq \mathbf{1}^T$.

However this does not prevent one-way path averaging from computing averages. One-way path averaging conserves information through iterations in a different way than previous algorithms:

Property 5.5.2 (Global sum and weight conservation). Let $\{\mathbf{x}_i(0)\}_{1\leqslant i\leqslant n}$ be the n measurements of the network, and let x_{ave} be their average. In one-way path averaging, at the end of any averaging round k,

$$\sum_{i=1}^{n} s_i(k) = nx_{ave},$$
$$\sum_{i=1}^{n} \omega_i(k) = n.$$

Property 5.5.2 is sufficient for one-way path averaging to be an unbiased estimator of the average.

Remark* 5.5.1 (Unbiased estimator). If the estimates in one-way path averaging converge to a consensus c, then $c = x_{ave}$, where x_{ave} is the average of the measurements $\{\mathbf{x}_i(0)\}_{1\leqslant i\leqslant n}$.

Proof: If the estimates converge to a value c, then, for any $1 \leqslant i \leqslant n$,

$$\lim_{t\to\infty} s_i(k)/x_i(k) = c,$$

where k indexes averaging rounds. Since there is a finite number n of nodes, this can be translated as follows: For any $\varepsilon > 0$, there is an iteration k_0, such that for any iteration $k \geqslant k_0$, and any $1 \leqslant i \leqslant n$,

$$\frac{s_i(k)}{\omega_i(k)} \in [c-\varepsilon, c+\varepsilon]. \tag{5.36}$$

Eq. (5.36) and Property 5.5.2 imply that

$$\begin{aligned}
s_i(k) &\in [\omega_i(k)(c-\varepsilon), \omega_i(k)(c+\varepsilon)], \quad \text{for any } 1 \leqslant i \leqslant n,\\
\sum_{i=1}^{n} s_i(k) &\in \left[\sum_{i=1}^{n} \omega_i(k)(c-\varepsilon), \sum_{i=1}^{n} \omega_i(k)(c+\varepsilon)\right],\\
nx_{ave} &\in [n(c-\varepsilon), n(c+\varepsilon)],\\
x_{ave} &\in [c-\varepsilon, c+\varepsilon]. \qquad (5.37)
\end{aligned}$$

Eq. 5.37 holds for any $\varepsilon > 0$, therefore, $c = x_{ave}$. ■

5.5.2 Convergence to Consensus

Preliminary Remarks

Figure 5.14(a) displays in logarithmic scale the error $\boldsymbol{\epsilon}(k) = \|\mathbf{x}(k) - x_{ave}\mathbf{1}\|$ in a simulation of one-way path averaging on a RGG of 1500 nodes and radius $r(n) = \sqrt{2\log n/n}$. The error globally decays exponentially fast, as in the previous algorithms. However, contrarily to gossip algorithms, a close look at the curve shows that the error is not a monotone function, because it sometimes very slightly increases. This non monotonicity shows already that the decay mechanism of the error in one-way path averaging is different.

Based on the stochastic nature of $\mathbf{W}(k)$, for any t, the following remark is straightforward. Let $M(k) = \max_i \mathbf{x}_i(k)$, and let $m(k) = \min_i \mathbf{x}_i(k)$. Since for every averaging round k, $\mathbf{W}(k)\mathbf{1} = \mathbf{1}$, and $\mathbf{W}(k)$ is a nonnegative matrix,

$$\begin{aligned} M(k) &\leqslant M(k-1), \\ m(k) &\geqslant m(k-1), \\ M(k) - m(k) &\leqslant M(k-1) - m(k-1). \end{aligned}$$

As a consequence, there are values M_∞ and m_∞, such that

$$\begin{aligned} \lim_{k\to\infty} M(k) &= M_\infty, \\ \lim_{k\to\infty} m(k) &= m_\infty. \end{aligned}$$

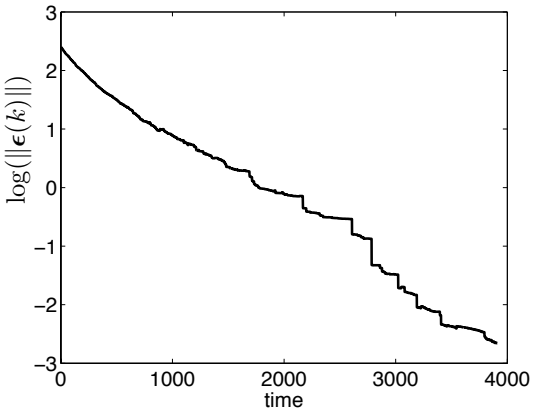
In previous sections, we would study in detail the matrices $\{\mathbf{W}(k)\}_{k>0}$ and the vectors $\{\mathbf{x}(k)\}_{k>0}$ to prove that the algorithm converges ($M_\infty = m_\infty$). However, here, it is better to adopt a different approach. Indeed, although the routes are i.i.d., $\{\mathbf{W}(k)\}_{k>0}$ is not i.i.d., which makes it difficult to study. Instead, we will first focuse on the behavior of the sums and weights separately. Then, we will combine their behaviors to conclude that one-way path averaging converges.

Assumptions and Notations

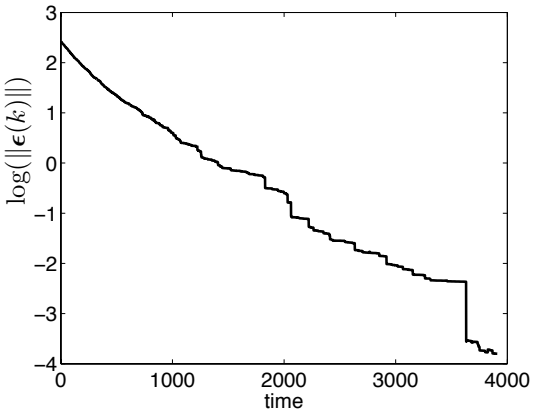
We assume that the network $\mathcal{G}$ is a connected graph in which i.i.d. routes can be generated. For example, $\mathcal{G}$ can be a random geometric graph with n nodes and connection radius $r(n)$ large enough to connect the network, on which we greedily generate routes. Another possibility is a lattice network on which horizontal or vertical routes are generated. We can also consider $(\leftrightarrow, \updownarrow)$-routing.

In order to generate i.i.d. routes, nodes wake up at random thanks to independent local exponential clocks, and initiate routes independently of previous routes according to a distribution which is identical in time. In our simulations, when a node wakes up, it draws a target T outside of the unit square and a number H of hops in order to generate a geographic random route of length H towards T. If needed (holes, border of network), a route can be interrupted before the Hth hop.

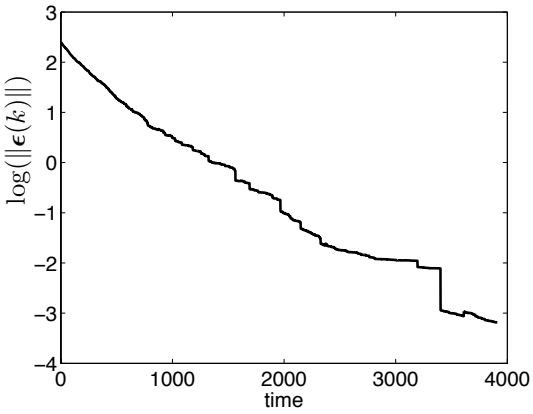
In order to study convergence, a number of notations need to be introduced.



(a) $\alpha = 0$



(b) $\alpha = 1$



(c) $\alpha = 2$

Figure 5.14: Evolution of the error in a logarithmic scale in simulations of one-way path averaging, for different values of α. Contrary to gossip algorithms, the error is not always decreasing. $n = 1500$ nodes, $r(n) = \sqrt{2 \log n / n}$.

Diffusion Matrix Let $\mathbf{s}$ be the vector gathering the sums s_i, $1 \leq i \leq n$, and let $\boldsymbol{\omega}$ be the vector gathering the weights ω_i, $1 \leq i \leq n$.

Definition* 5.5.1 (Diffusion Matrix). For any iteration k, the diffusion matrix $\mathbf{D}(k)$ is such that

$$\begin{aligned} \boldsymbol{s}(k) &= \boldsymbol{s}(k-1)\mathbf{D}(k), \\ \boldsymbol{\omega}(k) &= \boldsymbol{\omega}(k-1)\mathbf{D}(k). \end{aligned}$$

Let i be the node which generates a route at time k. When running *naive* one-way path averaging, the diffusion matrix $\mathbf{D}(k)$ is such that:

- $\mathbf{D}_{jj}(k) = 1$ if $j \neq i$,
- $\mathbf{D}_{ij}(k) = 1/H$ if j is in the route generated by i,
- $\mathbf{D}_{jl}(k) = 0$ otherwise.

In particular $\mathbf{D}_{ii}(k) = 1/H$. Note that $\mathbf{D}$ is a stochastic matrix, since it is non-negative and $\mathbf{D}\mathbf{1} = \mathbf{1}$.

An example: Re-indexing nodes starting with the nodes in the route and finishing with the remaining nodes, the diffusion matrix $\mathbf{D}(k)$ can be written as:

$$\mathbf{D}(k) = \begin{pmatrix} 1/H & 1/H & \dots & 1/H & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

In the *evolved* path averaging algorithm, the diffusion matrix writes differently. Let $r(1), r(2), \dots, r(\ell)$ be the nodes in the route r at iteration k ordered from the source node to the destination node. Then,

- $\mathbf{D}_{jj}(k) = 1$ if $j \notin r$,
- $\mathbf{D}_{r(s)r(t)}(k) = 1/(H - s + 1)$ if $t \geq s$,
- $\mathbf{D}_{ij}(k) = 0$ otherwise.

Example: Re-indexing nodes starting with $r(1), r(2), \dots, r(\ell)$, the diffusion matrix $\mathbf{D}(k)$ can be written as:

$$\mathbf{D}(k) = \begin{pmatrix} 1/H & 1/H & \dots & 1/H & 1/H & 0 & \dots & 0 \\ 0 & 1/(H-1) & \dots & 1/(H-1) & 1/(H-1) & 0 & \dots & 0 \\ 0 & 0 & \ddots & \vdots & \vdots & 0 & \dots & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

In the evolved one-way path averaging case, $\mathbf{D}(k)$ is a stochastic matrix as well.

Note that in both algorithms, the diffusion matrix $\mathbf{D}(k)$ at iteration k is entirely determined by the route that has been generated at k. Therefore, if routes are i.i.d., then $\{\mathbf{D}(k)\}_{k>0}$ is i.i.d. as well.

Overlay Graph $\mathcal{G}'$

Definition 5.5.1 (Overlay Graph). For a given network $\mathcal{G}$, a given route distribution and a given one-way path averaging algorithm, the overlay graph $\mathcal{G}'$ is a directed graph which is based on the expectation $\mathbb{E}[\mathbf{D}]$ of the i.i.d. diffusion matrices $\{\mathbf{D}(k)\}_{k>0}$:

- The vertices of $\mathcal{G}'$ are the vertices of $\mathcal{G}$.
- $e = (i, j)$ is a directed edge of $\mathcal{G}'$ if and only if $\mathbb{E}[\mathbf{D}_{ij}] > 0$.

For example, in naive path averaging, $e = (i, j)$ is an overlay edge if and only if node j is on a route initiated by node i with positive probability.

The overlay graph $\mathcal{G}'$ is an important concept because our main convergence theorem will only assume that $\mathcal{G}'$ is connected. Note that $\mathcal{G}'$ cannot be connected if $\mathcal{G}$ is not connected.

Contribution Matrix

Definition* 5.5.2 (Contribution Matrix P). The contribution matrix $\mathbf{P}(k)$ at iteration k is such that

- $\mathbf{s}(k)^T = \mathbf{s}(0)^T\mathbf{P}(k) = \mathbf{x}(0)^T\mathbf{P}(k)$,
- $\boldsymbol{\omega}(k)^T = \boldsymbol{\omega}(0)^T\mathbf{P}(k) = \mathbf{1}^T\mathbf{P}(k)$.

As a consequence, at any time t, the estimate $x_i(t)$ of any node i can be written:

$$x_i(t) = \frac{\sum_{j=1}^{n} p_{ji}(t)x_j(0)}{\sum_{j=1}^{n} p_{ji}(t)}, \tag{5.38}$$

where $p_{ij} = \mathbf{P}_{ij}$.

In particular, $\mathbf{P}(0) = \mathbf{I}$.

It is easy to see that

$$\mathbf{P}(k) = \mathbf{P}(k-1)\mathbf{D}(k).$$

Therefore,

$$\mathbf{P}(k) = \mathbf{D}(1)\mathbf{D}(2)\ldots\mathbf{D}(k).$$

The contribution matrices $\{\mathbf{P}(k)\}_{k\geq 0}$ are products of i.i.d. stochastic matrices.

The previous sections in the thesis were also about products of i.i.d. stochastic matrices. The difference here is that the products are on the right, whereas before they were on the left (we would study $\mathbf{D}(k)\mathbf{D}(k-1)\ldots\mathbf{D}(1)$).

Markov chain interpretation For notational issues, we denote now time/iteration as t, in order to save k for other purposes. Thus t is an integer variable here.

The matrices $\{\mathbf{D}(t)\}_{t\geq 1}$ can be seen as transition probability matrices of an nonhomogeneous Markov chain in an i.i.d. environment. $p_{ji}(t)$ is then the probability of being in state i at time t starting from state j.

In some cases, an nonhomogeneous Markov chain looses the memory of the past, i.e. the current state distribution does not depend so much on the initial state: $p_{ji}(t) \approx p_{1i}(t)$. Suppose that $p_{ji}(t) = p_{1i}(t)$ for all j, then Eq (5.38) becomes:

$$\begin{aligned} x_i(t) &= \frac{\sum_{j=1}^n p_{ji}(t)x_i(0)}{\sum_{j=1}^n p_{ji}(t)} \\ &= \frac{p_{1i}(t)\sum_{j=1}^n x_i(0)}{\sum_{j=1}^n p_{1i}(t)} \\ &= \frac{\sum_{j=1}^n x_i(0)}{n} \\ &= x_{ave}. \end{aligned}$$

An important part of the proof of convergence will be to show that our Markov chain indeed looses memory, or in more appropriate words, that it is weakly ergodic.

Definition 5.5.2 (Weak ergodicity). Let $\mathbf{P}(m,k) = \mathbf{D}(m)\mathbf{D}(m+1)\ldots\mathbf{D}(k)$. An nonhomogeneous Markov chain with transition matrices $\{\mathbf{D}(t)\}_{t\geq 0}$ is called weakly ergodic if for all $m \geq 0$,

$$\lim_{k\to\infty} \sup_{\mu,\nu} d_V(\mu^T\mathbf{P}(m,k), \nu^T\mathbf{P}(m,k)) = 0,$$

where the supremum is taken over all probability distributions μ, ν on $1, 2, \ldots, n$, and where

$$d_V(\mu,\nu) = \frac{1}{2}\sum_{i=1}^n |\mu(i) - \nu(i)|.$$

When taking $\mu = \boldsymbol{e}_i$ and $\nu = \boldsymbol{e}_j$, weak ergodicity implies that

$$\lim_{t\to\infty} \max_{i,j} \sum_{k=1}^n |p_{ik}(t) - p_{jk}(t)| = 0.$$

Convergence Theorem

Theorem* 5.5.1 (Convergence of one-way path averaging). In both naive and evolved one-way path averaging, if the overlay network $\mathcal{G}'$ is connected, then the estimates $\mathbf{x}(t)$ converge to x_{ave} with probability 1.

To prove this theorem, we will first derive a number of lemmas.

Lemma 5.5.1. If the overlay network $\mathcal{G}'$ is connected, then there is a deterministic time T and a constant c such that

$$\mathbb{P}[\mathbf{D}(1)\mathbf{D}(2)\ldots\mathbf{D}(T) > c] > 0.$$

The proof of this lemma appears in a paper from Tahbaz-Salehi et al. [73], where matrices are multiplied on the left. Multiplying matrices on the right does not change the proof.

Proof: For any time t, $\mathbf{D}(t)$ is stochastic thus $\mathbb{E}[\mathbf{D}]$ is stochastic as well. $\mathbb{E}[\mathbf{D}_{ij}] > 0$ if and only if (i,j) is an edge in the overlay network $\mathcal{G}'$. Therefore if $\mathcal{G}'$ is connected then $\mathbb{E}[\mathbf{D}]$ is irreducible. Whenever node i is not on a route, $\mathbf{D}_{ii}(t) = 1$. As a consequence $\mathbb{E}[\mathbf{D}]_{ii} > 0$ and $\mathbb{E}[\mathbf{D}]$ is aperiodic. Since $\mathbb{E}[\mathbf{D}]$ is irreducible and aperiodic, it is primitive and there is an m such that $\mathbb{E}[\mathbf{D}]^m > 0$.

The sequence of matrices $\{\mathbf{D}(t)\}_{t\geq 1}$ is i.i.d., hence $\mathbb{E}[\mathbf{D}(1)\mathbf{D}(2)\ldots\mathbf{D}(m)] = \mathbb{E}[\mathbf{D}]^m > 0$, and $\mathbb{P}\left[(\mathbf{D}(1)\mathbf{D}(2)\ldots\mathbf{D}(m))_{ij} > 0\right] > 0$ for any entry (i,j). For any time t, the diagonal coefficients of $\mathbf{D}(t)$ are non-zero, thus if the $(i,j)^{th}$ entry of $\mathbf{P}(k,k+m-1) = \mathbf{D}(k)\mathbf{D}(k+1)\ldots\mathbf{D}(k+m-1)$ is positive, then $\mathbf{P}_{ij}(t) > 0$ for all $t \geq k+m-1$. Now take $T = n(n-1)m$. The probability that $\mathbf{P}(T) > 0$ is larger than the probability that $\mathbf{P}_{12}(1,m) > 0$ and that $\mathbf{P}_{13}(m+1,2m) > 0$, ..., and that $\mathbf{P}_{n,n-1}(T-m+1,T) > 0$. By independence of $\{\mathbf{D}(t)\}_{t\geq 1}$,

$$\begin{aligned}\mathbb{P}[\mathbf{P}(T) > 0] &\geq \mathbb{P}[\mathbf{P}_{1,2}(1,m) > 0]\mathbb{P}[\mathbf{P}_{1,3}(m+1,2m) > 0]\ldots \\ &\quad \ldots\mathbb{P}[\mathbf{P}(T-m+1,T) > 0] \\ &> 0.\end{aligned}$$

Therefore, there is a $c > 0$ such that $\mathbb{P}[\mathbf{D}(1)\mathbf{D}(2)\ldots\mathbf{D}(T) > c] > 0$. ■

Lemma 5.5.2 (Weak ergodicity). If the overlay network $\mathcal{G}'$ is connected, then $\{\mathbf{D}(t)\}_{t\geq 1}$ is weakly ergodic.

Proof: To prove Lemma 5.5.2, some knowledge on weak ergodicity is useful. There are two famous criteria to prove weak ergodicity: a basic one, which introduces Dobrushin's coefficient δ, and a block version of the basic criterion, which we will use in this proof.

Let $\mathbf{Q}$ be a stochastic matrix. The Dobrushin's coefficient $\delta(\mathbf{Q})$ of matrix $\mathbf{Q}$ is defined as:

$$\delta(\mathbf{Q}) = \frac{1}{2}\max_{ij}\sum_{k=1}^{n}|\mathbf{Q}_{ik} - \mathbf{Q}_{jk}|.$$

One can show that $0 \leq \delta(\mathbf{Q}) \leq 1$, and that for any stochastic matrices $\mathbf{Q}_1$ and $\mathbf{Q}_2$,

$$\delta(\mathbf{Q}_1\mathbf{Q}_2) \leq \delta(\mathbf{Q}_1)\delta(\mathbf{Q}_2). \tag{5.39}$$

Another useful fact is that for any stochastic matrix $\mathbf{Q}$

$$1 - \delta(\mathbf{Q}) \geq \max_j \min_i \mathbf{Q}_{ij} \geq \min_{i,j}\mathbf{Q}_{ij}. \tag{5.40}$$

The basic weak ergodicity criterion states that $\{\mathbf{D}(t)\}_{t\geq 1}$ is weakly ergodic if and only if for all $m > 0$,

$$\lim_{k\to\infty}\delta(\mathbf{P}(m,k)) = 0.$$

The block criterion for weak ergodicity uses Eq. (5.39): $\{\mathbf{D}(t)\}_{t\geq 1}$ is weakly ergodic if and only if there is a strictly increasing sequence of integers $\{k_s\}_{s\geq 1}$

such that

$$\sum_{s=1}^{\infty}(1-\delta(\mathbf{P}(k_s+1,k_{s+1})))=\infty. \tag{5.41}$$

We use this criterion with $k_s = sT$, where T was defined in Lemma 5.5.1.

A joint consequence of Lemma 5.5.1 and of the strong law of large numbers is that the event $\{\mathbf{D}(k_s+1)\mathbf{D}(k_s+2)\ldots\mathbf{D}(k_{s+1})>c\}$ happens infinitely often with probability 1. Hence, using Eq. 5.40, the event $\{1-\delta\left(\mathbf{P}(k_s+1,k_{s+1})\right)>c\}$ happens infinitely often with probability 1. We can thus conclude that the block criterion (5.41) holds with probability 1 and that $\{\mathbf{D}(t)\}_{t\geq 1}$ is weakly ergodic. ■

To prove the convergence theorem (Th. 5.5.1), we will upperbound the error $\|\boldsymbol{\epsilon}(t)\|_\infty = \|\mathbf{x}(t)-x_{ave}\mathbf{1}\|_\infty$ with a non-increasing function $f(t)$. Let $\eta_{ji}(t) = p_{ji}(t) - \sum_{j=1}^n p_{ji}(t)/n = p_{ji}(t) - \omega_i(t)/n$, and define the following function f_i:

$$f_i(t) = \frac{\sum_{j=1}^n |\eta_{ji}(t)|}{\omega_i(t)}.$$

Lemma* 5.5.1. The function

$$f(t) = \max_{1\leq i\leq n} f_i(t)$$

is non-increasing when running the naive one-way path averaging algorithm. Furthermore,

$$\|\boldsymbol{\epsilon}(t)\|_\infty \leq \|\mathbf{x}(0)\|_\infty f(t). \tag{5.42}$$

Finally, the lemma holds in the evolved path averaging case.

Proof: We start by proving Eq. (5.42) for both the naive and the evolved versions of one-way path averaging. For any node i,

$$\begin{aligned}
|x_i(t)-x_{ave}| &= \left|\frac{\sum_{j=1}^n p_{ji}(t)x_j(0)}{\sum_{j=1}^n p_{ji}(t)} - x_{ave}\right| \\
&= \left|\frac{\sum_{j=1}^n (\omega_i(t)/n+\eta_{ji}(t))x_j(0)}{\omega_i(t)} - x_{ave}\right| \\
&= \left|\frac{\sum_{j=1}^n \eta_{ji}(t)x_j(0)}{\omega_i(t)}\right| \\
&\leq \|\mathbf{x}(0)\|_\infty \frac{\sum_{j=1}^n |\eta_{ji}(t)|}{\omega_i(t)} \\
&= \|\mathbf{x}(0)\|_\infty f_i(t).
\end{aligned}$$

As a consequence,

$$\begin{aligned}
\|\boldsymbol{\epsilon}(t)\|_\infty &= \max_{1\leq i\leq n} |x_i(t)-x_{ave}| \\
&\leq \max_{1\leq i\leq n} \|\mathbf{x}(0)\|_\infty f_i(t) \\
&= \|\mathbf{x}(0)\|_\infty f(t).
\end{aligned}$$

Next, we need to prove that $f(t)$ is a non-increasing function. We start with

the naive algorithm. Suppose that at iteration t, node k wakes up, divides its sum and weight by H and creates a route on which node i is. We study separately all the functions $f_j(t)$, for $1 \leq j \leq n$. The case of $f_k(t)$ is easy: $f_k(t) = f_k(t-1)$ since $\omega_k(t) = \omega_k(t-1)/H$ and $\eta_{jk}(t) = \eta_{jk}(t-1)/H$ for all j (the factors $1/H$ cancel out). The case of $f_i(t)$ is trickier. We need to check that $f_i(t) \leq f(t-1)$.

$$\begin{aligned}
f_i(t) &= \frac{\sum_{j=1}^n |\eta_{ji}(t-1) + \eta_{jk}(t-1)/H|}{\omega_i(t-1) + \omega_k(t-1)/H} \\
&\leq \frac{\sum_{j=1}^n |\eta_{ji}(t-1)| + \sum_{j=1}^n |\eta_{jk}(t-1)/H|}{\omega_i(t-1) + \omega_k(t-1)/H} \\
&\leq \max\left(\frac{\sum_{j=1}^n |\eta_{ji}(t-1)|}{\omega_i(t-1)}, \frac{\sum_{j=1}^n |\eta_{jk}(t-1)/H|}{\omega_k(t-1)/H}\right) \qquad (5.43) \\
&= \max\left(\frac{\sum_{j=1}^n |\eta_{ji}(t-1)|}{\omega_i(t-1)}, \frac{\sum_{j=1}^n |\eta_{jk}(t-1)|}{\omega_k(t-1)}\right) \\
&= \max(f_i(t-1), f_k(t-1)) \\
&\leq f(t-1).
\end{aligned}$$

The second inequality (5.43) comes from the following equality: for any $a, b, c, d > 0$,

$$\frac{a+c}{b+d} = \frac{b}{b+d} \cdot \frac{a}{b} + \frac{d}{b+d} \cdot \frac{c}{d} \leq \max\left(\frac{a}{b}, \frac{c}{d}\right).$$

If a node j is not on the route, then its sum and weight are not changed and f_j does not change. We have thus proved that for any node j, whether it initiates a route, is contained on a route or is not, $f_j(t) \leq f(t-1)$. Therefore $f(t) = \max_j f_j(t) \leq f(t-1)$ and f is a non-increasing function.

Consider now the evolved one-way path averaging algorithm. Suppose that at iteration t, route $r = \{r(1), r(2), \ldots, r(H)\}$ is generated. We need to prove that $f(t) \leq f(t-1)$. To do this, we decompose the operations of the algorithm in several steps and we prove that each step does not increase the function f. One evolved one-way path averaging round is equivalent to:

- First, perform a naive one-way path averaging round with route $\{r(H-1), r(H)\}$.
- Then, perform a naive one-way path averaging round with route $\{r(H-2), r(H-1), r(H)\}$.
- ...
- Perform a naive one-way path averaging round with route $\{r(2), \ldots, r(H-2), r(H-1), r(H)\}$.
- Finally, perform a naive one-way path averaging round with route $\{r(1), r(2), \ldots, r(H-1), r(H)\}$.

Note that in reality the algorithm does not proceed in this order, but that in the end, the final global operation is the same. Since a naive one-way path averaging round does not increase f and because an evolved path averaging

round can be seen as a combination of naive one-way path averaging rounds, an evolved path averaging round does not increase f either: $f(t) \leq f(t-1)$. ■

Lemma* 5.5.2. Assume that the overlay network $\mathcal{G}'$ is connected. Then there is a constant α such that, for any time t, with probability 1, there is a time $t_1 \geq t$ at which $\min_i \omega_i(t_1) \geq \alpha$.

Proof: Lemma 5.5.2 is a consequence of Lemma 5.5.1, which states that there is a deterministic time T such that

$$\mathbb{P}[\mathbf{D}(1)\mathbf{D}(2)\ldots\mathbf{D}(T) > c] > 0.$$

Taking $k_s = t + sT$, by the independence of $\{\mathbf{D}(t)\}_{t>0}$ and by the strong law of large numbers, the event $\{\mathbf{D}(k_s+1)\mathbf{D}(k_s+2)\ \ldots\mathbf{D}(k_{s+1}) > c\}$ eventually happens with probability 1. Let t_1 be the first time larger than t such that $\mathbf{D}(t_1-T+1)\mathbf{D}(t_1-T+2)\ldots\mathbf{D}(t_1) > c$. Then the weights at time t_1 satisfy

$$\begin{aligned}\boldsymbol{\omega}(t_1)^T &= \boldsymbol{\omega}(t_1-T)^T\mathbf{D}(t_1-T+1)\mathbf{D}(t_1-T+2)\ldots\mathbf{D}(t_1)\\ &> c\boldsymbol{\omega}(t_1-T)^T\mathbf{1}\mathbf{1}^T,\end{aligned}$$

because weights are always positive. Now, since the sum of weights is equal to n, $\boldsymbol{\omega}(t_1-T)^T\mathbf{1} = n$ and

$$\boldsymbol{\omega}(t_1)^T > cn\mathbf{1}^T.$$

Taking $\alpha = cn$ concludes the proof. ■

To prove Theorem 5.5.1, it remains to show that $f(t)$ converges to 0.

Proof:[Theorem 5.5.1] For any $\varepsilon > 0$, according to Lemma 5.5.2, there is a time t_0 such that for any $t \geq t_0$,

$$\max_{i,j}\sum_{k=1}^{n}|p_{ik}(t) - p_{jk}(t)| < \varepsilon.$$

As a consequence $|p_{ik}(t) - p_{jk}(t)| < \varepsilon$ for any i, j, k. Hence $|\eta_{jk}(t)| < \varepsilon$ as well. Indeed,

$$\begin{aligned}|\eta_{jk}(t)| &= \left|p_{jk}(t) - \sum_{i=1}^{n}\frac{p_{ik}(t)}{n}\right|\\ &= \left|\sum_{i=1}^{n}\frac{p_{jk}(t) - p_{ik}(t)}{n}\right|\\ &\leq \sum_{i=1}^{n}\frac{|p_{jk}(t) - p_{ik}(t)|}{n}\\ &< \sum_{i=1}^{n}\frac{\varepsilon}{n}\\ &= \varepsilon.\end{aligned}$$

Therefore, for any $t \geq t_0$ and any $1 \leq i \leq n$,

$$f_i(t) < \frac{n\varepsilon}{\omega_i(t)},$$

and therefore

$$f(t) < \frac{n\varepsilon}{\min_i \omega_i(t)}.$$

Using Lemma 5.5.2, there is a constant α such that, with probability 1, there is a time $t_1 \geq t_0$ at which $\min_i \omega_i(t_1) \geq \alpha$. Then, for any ε', it suffices to take $\varepsilon = \alpha\varepsilon'/n$ to conclude that there is a time t_1 with probability 1 such that $f(t_1) < \varepsilon'$. Since f is non increasing, for all time $t \geq t_1$, $f(t) < \varepsilon'$; in other words $f(t)$ converges to 0. ■

5.5.3 Controlling Weights.

The proof of Theorem 5.5.1 uses the fact that weights are infinitely often larger than a constant $c > 0$, thus avoiding small weights might favor the performance of one-way path averaging. In this section, we explore the idea of controlling the weights, in order to help them be large enough more often, and also to limit the apparition of very large weights. Very large weights create robustness concerns. If a node with large weight crashes, then a substantial amount of information may be lost. Also, it might be good for performance reasons not to have large weights uselessly accumulating information without sharing it. Finally, small weights are problematic in practice for quantization reasons. What is the estimate of a node with a weight that has been quantized to 0?

Weights can become arbitrarily small, and without weight control, $\min_i \omega_i(k)$ indeed becomes very small ($\approx 10^{-4}, 10^{-5}$ in Fig. 5.15(a)). Fig. 5.15(b) shows that on the same simulation, $\max_i \omega_i(k)$ varies between 20 and 50 approximatively, which corresponds up to 3.3% of the total network weight $n = 1500$. Fig. 5.18(a) displays the spatial distribution of weights at iteration 3910 and Fig. 5.18(b) displays the spatial distribution of weights averaged over time. We observe that weights are larger in the center, on the edges, and especially in the corners of the unit square. Central nodes happen to be more often on routes; thus, their weights get often increased. As far as border nodes are concerned, they happen to be on routes more rarely, but when they do, they usually are destination nodes, and they collect all the remaining "chunks", so that they aggregate larger weights (steps 20 and 17 in Alg. 2.12). Thus, optimizing the target location probability distribution could help to attenuate the overweight of border and corner nodes.

We also propose another way to control the weights by setting the clock intensity $\lambda_i(k)$ at any node i at time t equal to

$$\lambda_i(k) = \lambda_0 \omega_i^{\alpha}(k), \tag{5.44}$$

where λ_0 and α are parameters to be defined. The idea is the following: when a node has a large weight, then it wakes up faster than if it had a small weight. Since when a node wakes up, its weight get divided by a number $H > 1$, large weights get divided faster, and thus they do not survive long in the network. Inversely, a node with a small weight will not wake up easily, and will stop dividing its weight further down (unless it happens to be on a route, in which case it divides its weight first, and then adds the weights it received from the previous nodes in the route). We are going to consider in this section three different parameters α: $0, 1$ and 2. See Fig. 5.15 to see the evolution of the weights, and observe that increasing α does control better the fluctuations of

the weights.

In such a setting, given a set of weights $\{\omega_i\}_{1\leqslant i\leqslant n}$, the probability that node i wakes up in the next averaging round is equal to:

$$\mathbb{P}[\text{node } i \text{ wakes up next}] = \frac{\omega_i^\alpha}{\sum_{i=1}^n \omega_i^\alpha}. \tag{5.45}$$

The global clock ticks exponentially with intensity $\lambda_0 \sum_{i=1}^n \omega_i^\alpha$. Note that if $\alpha = 0$ or $\alpha = 1$, then $\lambda_0 \sum_{i=1}^n \omega_i^\alpha = \lambda_0 n$, which is constant over time. This might be an advantage for congestion control reasons. When $\alpha = 2$, Fig. 5.16 shows that the global clock intensity increases first, then it reaches a stationary regime below $2\lambda_0 n$. Fig. 5.14 shows that the algorithm seems to converge with both $\alpha = 1$ and $\alpha = 2$ as well. Proving this is part of future work. As far as performance is concerned, Fig. 5.17 shows that one-way path averaging seems to converge just a little faster with $\alpha = 1$ and $\alpha = 2$ than with $\alpha = 0$. The main motivation for weight control is thus robustness.

Finally, we propose a third way to control weights. To ensure that all the weights are always larger than a value $\omega_{\min}$, there is a simple solution, which can be combined with the previous one. It consists in limiting the number of hops of the route generated by any node i at time t to $\lfloor \omega_i(k)/\omega_{\min} \rfloor$. A node with weight smaller than $2\omega_{\min}$ freezes its clock, until its weights becomes large enough again.

5.5.4 Robustness, Quantization and Performance.

We are now going to discuss several issues: robustness and congestion control, quantization, and performance in one-way path averaging.

Robustness and congestion control:

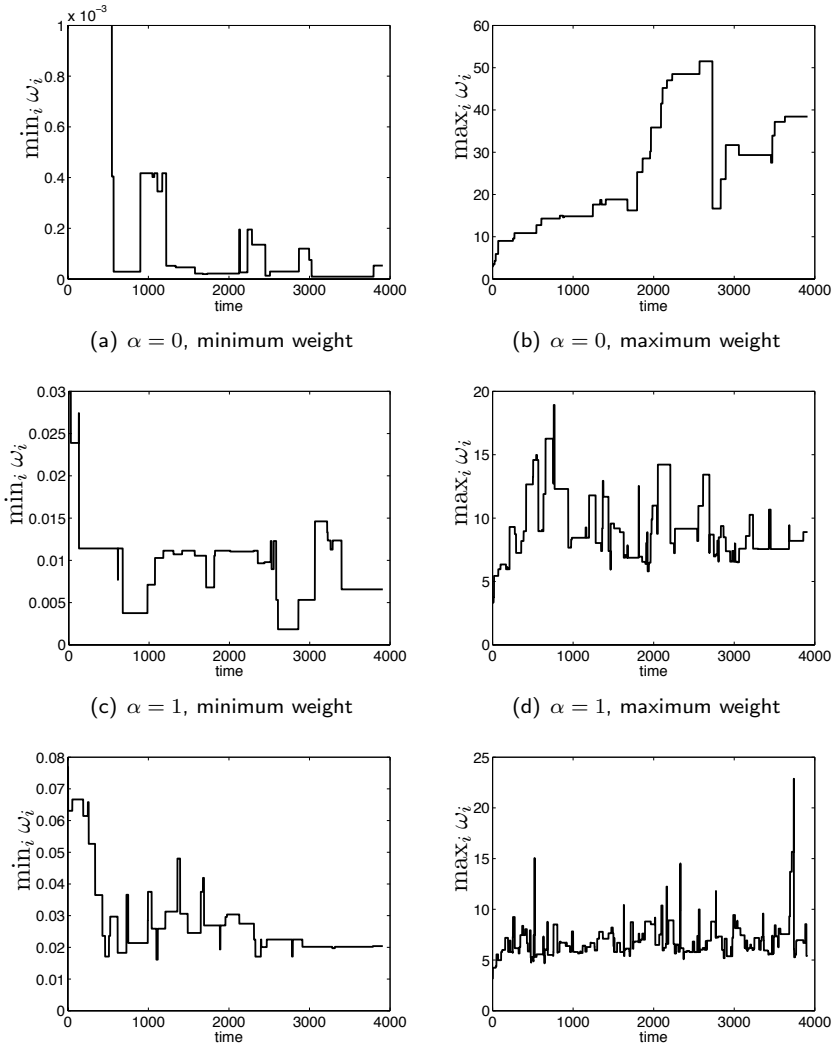
There is a tradeoff over the exponent α used in Eq. (5.44). If α is too small, then some weights can become very large, which, as we explained above, rises robustness issues if a node with large weight happens to die. On the contrary, as α increases, the global clock ticks faster, which might overload the network. It is then possible to impose a small enough λ_0 to compensate for this effect. But this will impose a slow start for the algorithm.

Large α can also reduce the fairness of the algorithm. As we can see on Fig.5.18, a large α implies that central nodes nodes wake up more often, and empty their battery faster. In addition, since there is already naturally more traffic load in the center of the network than on the edges, large α could make the situation worse and rise congestion issues.

Quantization:

We should not forget that in practice signals are quantized. One-way path averaging is delicate to quantize, and we propose the following scheme. When a node i with state (s_i, ω_i) wakes up, it draws a number H according to a probability distribution f, possibly with rejection sampling above a threshold $H_{\max}$. Node i creates and sends on a path $H-1$ "chunks" (s', ω') such that

$$\frac{s_i}{s'} = \frac{\omega_i}{\omega'} \geqslant H - 1. \tag{5.46}$$



(a) $\alpha = 0$, minimum weight

(b) $\alpha = 0$, maximum weight

(c) $\alpha = 1$, minimum weight

(d) $\alpha = 1$, maximum weight

(e) $\alpha = 2$, minimum weight

(f) $\alpha = 2$, maximum weight

Figure 5.15: Evolution of the minimum weight and of the maximum weight for $\alpha = 0, 1$ and 2. $n = 1500$ nodes, $r(n) = \sqrt{2 \log n / n}$.

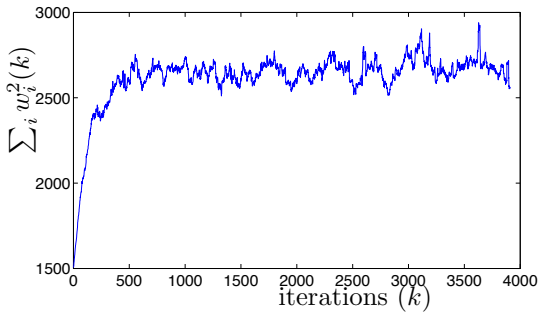


Figure 5.16: Global clock intensity $\sum_i w_i^2(k)$ ($\lambda_0 = 1$).

Then, node i updates to $(s_i-(H-1)s', \omega_i-(H-1)\omega')$, so that $\sum_{i=1}^n s_i = nx_{ave}$ and $\sum_{i=1}^n \omega_i = n$. Therefore, if consensus is reached, then the consensus value is equal to x_{ave}.

When consensus is reached, consensus is kept thanks to the crucial Property (5.46) on equal ratios. Indeed, Property (5.46) guarantees that $\mathbf{W1} = \mathbf{1}$, but this property is not easy to follow in a quantized setting. Indeed, it is not obvious that there are quantization levels q_s and q_ω, such that

$$\frac{s_i}{q_s} = \frac{\omega_i}{q_\omega} \geqslant H - 1. \tag{5.47}$$

In order to facilitate additions and substractions, we suggest that both sums and weights are uniformly quantized, but not necessarily with the same quantization step. In a memory constraint setting, weights can have a coarse quantization step, so that sums can get quantized as finely as possible. The idea is that the weights will decide on the ratio, and that the sums will adapt to the desired ratios as precisely as possible.

Suppose that the weights are uniformly quantized with a coarse quantization step Δ, such that $\Delta^{-1} \in \mathbb{N}$. Then define ω' as

$$\omega' = \Delta \left\lfloor \frac{\omega_i}{\Delta H} \right\rfloor, \tag{5.48}$$

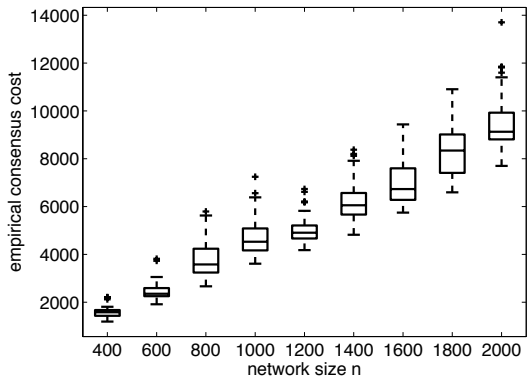
where H is drawn randomly according to a distribution f *truncated* from values of H larger than $H_{\max} = \omega_i/\Delta$. If $\omega_i = \Delta$, then node i freezes its clock. If $\omega_i = 2\Delta$, then node i can only generate routes with $H = 2$ nodes.

Now, let δ be the uniform quantization step for the sums $\{s_i\}_{1\leqslant i\leqslant n}$, then

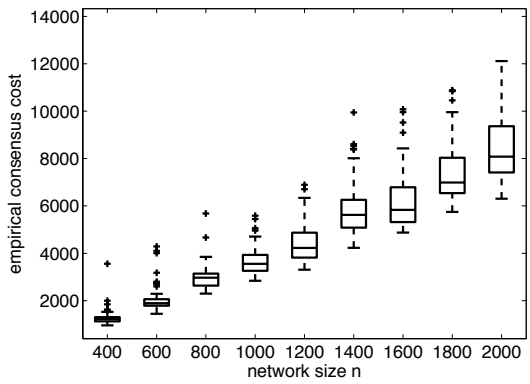
$$s' = \delta \text{ round}\left(\frac{s_i}{\delta}\frac{\lfloor \omega_i/(\Delta H)\rfloor}{\omega_i/\Delta}\right).$$

Note that intermediate nodes in the route can also send their own contribution to the rest of the route in the same way. In the quantized setting, intermediate nodes will not necessarily be able to create enough chunks for all the remaining nodes on the route, but only to a few of them because of the rejection sampling of H above the threshold $H_{\max} = \omega_i/\Delta$ in Eq. 5.48.

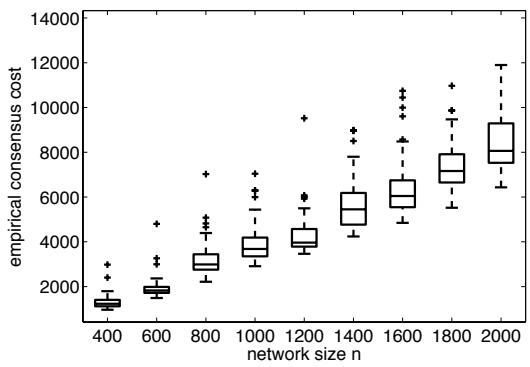
Figure 5.19 shows the behavior of quantized one-way path averaging for



(a) $\alpha = 0$

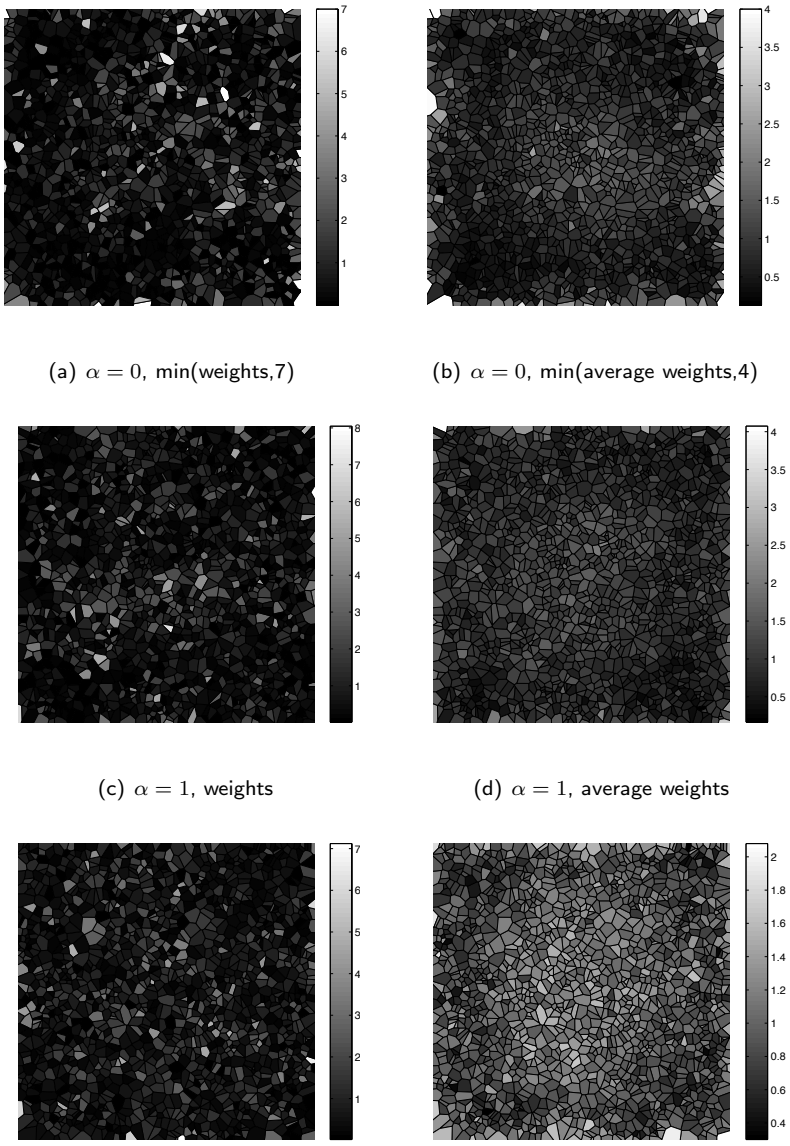


(b) $\alpha = 1$



(c) $\alpha = 2$

Figure 5.17: Empirical consensus costs for one-way path averaging (evolved version) for growing network and $r(n) = \sqrt{6 \log n / n}$. The cost of each iteration here is equal to the route length, which is not necessarily equal to H if the route hits the edges of the network.



(a) $\alpha = 0$, min(weights,7)

(b) $\alpha = 0$, min(average weights,4)

(c) $\alpha = 1$, weights

(d) $\alpha = 1$, average weights

(e) $\alpha = 2$, weights

(f) $\alpha = 2$, average weight

Figure 5.18: Map of weights ω for $\alpha = 0, 1$ and 2. We ran one-way path averaging on a random geometric graph of $n = 1500$ nodes, and radius $r(n) = \sqrt{2 \log n / n}$. According to a linear color scale, for each node i, we color its Voronoi cell with color $\omega_i(k)$ at time $t = 3910$ in the left column, and with color $\left(\sum_{t=1}^{3910} \omega_i(k)\right) / 3910$ in the right column. For $\alpha = 0$, we had to truncate the color scale in order to avoid almost all-black figures due to a maximum weight larger than 40 and a maximum average weight larger than 20.

different values of Δ and δ. In both Figures 5.19(a) and 5.19(b), $\Delta = 0.1$, which is quite large. $\delta = 0.001$ in Fig. 5.19(a), whereas the quantization step for sums is 100 times finer ($\delta = 0.00001$) in Fig. 5.19(b). We can see that the estimations are more accurate with the finer δ. Keeping $\delta = 0.00001$, and decreasing the quantization step of weights to $\Delta = 0.001$ (Fig. 5.19(c)) helps the convergence to go faster, probably because smaller Δ increases the rejection threshold $H_{\max} = \omega_i/\Delta$, and makes routes longer.

Performance

Figure 5.20 is the source of our motivation for studying one-way path averaging. They show the empirical consensus cost $C_{t_1}^{emp}(k)$ of one-way path averaging and path averaging on random geometric graphs of growing size with radius $r(n) = \sqrt{6 \log n/n}$. It uses the same parameters $r(n)$, t_1 and t, and also the same routes in both simulations. The cost of an iteration is taken to be equal to the length of the generated route in one-way path averaging, and equal to two times the route length in path averaging. We can see that one-way path averaging is not less efficient than path averaging by not routing information back.

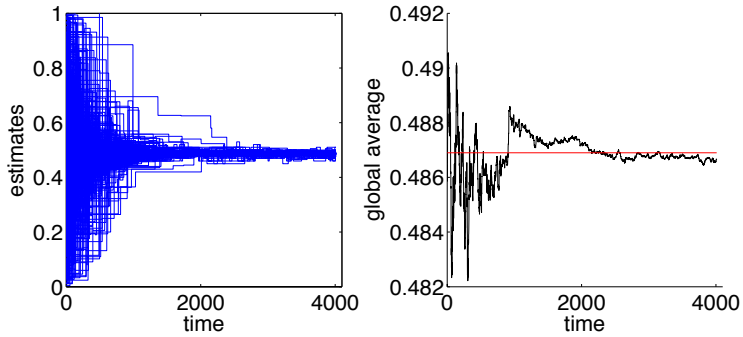
There are many ways to generate routes, and the performance of path averaging and one-way path averaging depends on the routing scheme. For example, Fig. 5.12(c) shows the consensus cost of path averaging in the same conditions except that the target is drawn uniformly at random inside the unit square. Path averaging performs worse with this target distribution, than with the one described in Alg. 2.10, which was used the simulations of Fig. 5.20.

There are many parameters which influence the respective performance of path averaging and one-way path averaging, and thus it is not possible to say if one algorithm performs better than the other in absolute terms. The examples of simulations we give here show that one-way path averaging is a very interesting algorithm to consider, because it is adapted to time-varying wireless sensor network, it is probably very efficient, and it is quantizable.

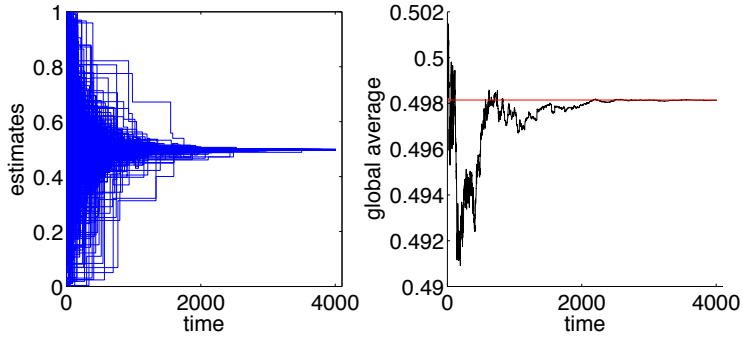
Conclusion

In this chapter we investigated the performance of pairwise gossip, neighborhood gossip, broadcast gossip, geographic gossip, and path averaging on several topologies (see Table 5.5.4). In the first three algorithms, only direct neighbors exchange information, which prevents the algorithms from being very efficient. By routing information and allowing any pair of nodes to average their estimates, geographic gossip saves a factor $\sqrt{n/\log n}$ of messages over pairwise gossip. Under some conditions, path averaging, which averages entire random routes, saves an additional factor $\sqrt{n/\log n}$ compared to geographic gossip, and requires only $O(n \log n)$ messages to converge at a precision of $1/n$.

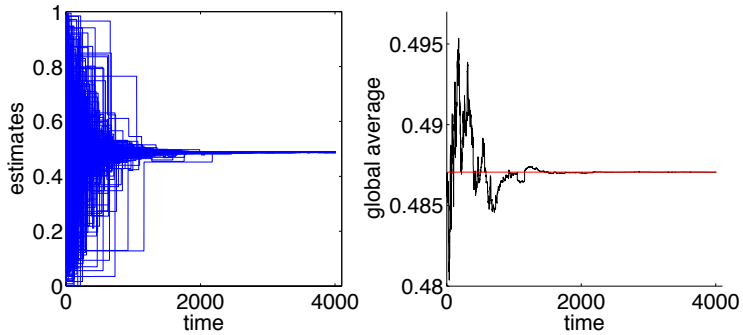
We observed on simulations that relaxing sufficient conditions for convergence of path averaging in $O(n \log n)$ messages does not impact significantly its performance. Finally, we studied one-way path averaging, which requires the routing of information only in a single direction along random routes (unlike path averaging where information is transfered in one direction and then in the opposite direction). This algorithm is particularly interesting from a prac-



(a) $\Delta = 0.1$, $\delta = 0.001$

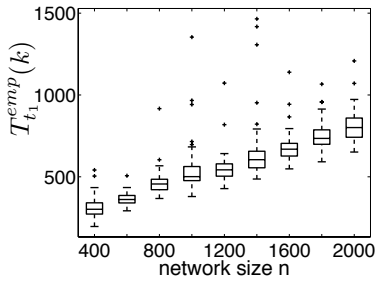


(b) $\Delta = 0.1$, $\delta = 0.00001$

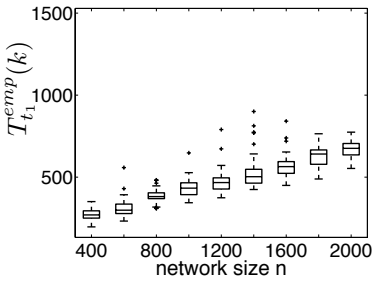


(c) $\Delta = 0.001$, $\delta = 0.00001$

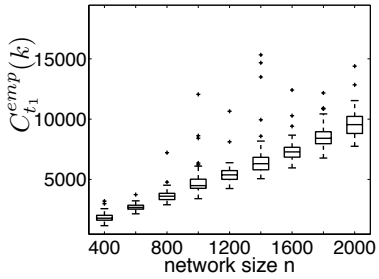
Figure 5.19: Quantized one-way path averaging on random geometric graphs of $n = 500$ nodes and radius $r(n) = \sqrt{2 \log n / n}$ with decreasing quantization step. The left figures simultaneously display the estimates $s_i(k)/\omega_i(k)$ for all nodes i. The right figure shows the evolution of the global average of the estimates of the nodes. The horizontal line shows x_{ave}. Here the distribution f was taken uniform over $[2, \sqrt{2}/r(n)]$.



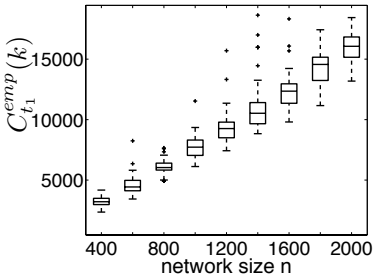
(a) $T_{t_1}^{emp}(k)$, one-way path averaging, $\alpha = 1$



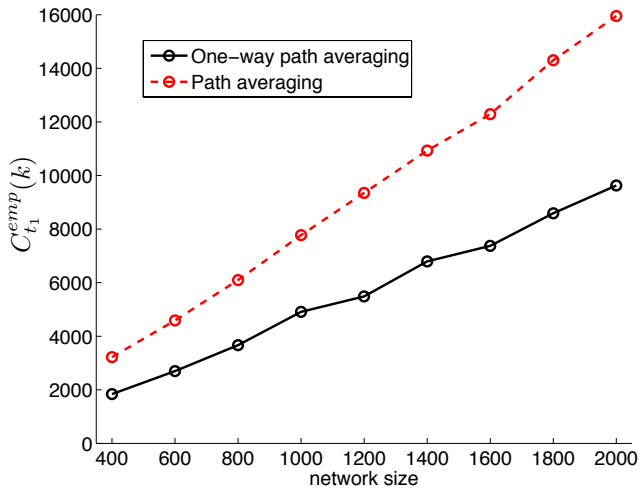
(b) $T_{t_1}^{emp}(k)$, path averaging



(c) $C_{t_1}^{emp}(k)$, one-way path averaging, $\alpha = 1$



(d) $C_{t_1}^{emp}(k)$, path averaging



(e) Average $C_{t_1(k)}^{emp}$

Figure 5.20: Comparing the empirical consensus cost for one-way path averaging and path averaging, with equal parameters $r(n) = \sqrt{6 \log n / n}$, t_1 and t.The cost of each iteration here is equal to the route length for one-way path averaging and to two times the route length for path averaging.

tical point of view and it seems to perform better than all the algorithms we described previously. We believe that one-way path averaging is an exciting direction for future research.

Algorithm	Graph topology	Number of averaged nodes per round	Expected cost per round: $\mathbb{E}[R]$	Mixing time $\frac{1}{1-\lambda_2(\mathbb{E}[\mathbf{W}])}$	Averaging time T_{ave}	Total expected cost: $\mathbb{E}[R]T_{ave}$
PAIRWISE	complete graph	2	$\Theta(1)$	$n-1$	$\Theta(n\log n)$	$\Theta(n\log n)$
	lattice	2	$\Theta(1)$	$\Theta(n^2)$	$\Theta(n^2\log n)$	$\Theta(n^2\log n)$
GOSSIP	lattice, radius r	2	$\Theta(1)$	$\Theta\left(n^2/r^2\right)$	$\Theta\left(n^2\log n/r^2\right)$	$\Theta\left(n^2\log n/r^2\right)$
	RGG	2	$\Theta(1)$	$\Theta\left(n^2/\log n\right)$	$\Theta\left(n^2\right)$	$\Theta\left(n^2\right)$
NEIGHBORHOOD	complete graph	n	$\Theta(n)$	1	1	$\Theta(n)$
GOSSIP	lattice, radius r	$(2r+1)^2$	$\Theta\left(r^2\right)$	$O\left(n^2/r^4\right)$	$O\left(n^2\log n/r^4\right)$	$O\left(n^2\log n/r^2\right)$
GEOGRAPHIC	lattice	2	$\Theta\left(\sqrt{n}\right)$	$n-1$	$\Theta(n\log n)$	$\Theta\left(n^{1.5}\log n\right)$
GOSSIP	RGG	2	$O\left(\sqrt{n/\log n}\right)$	$O\left(n\right)$	$O\left(n\log n\right)$	$O\left(n^{1.5}\sqrt{\log n}\right)$
PATH	toroidal lattice	$O\left(\sqrt{n}\right)$	$\Theta\left(\sqrt{n}\right)$	$O(\sqrt{n})$	$O(\sqrt{n}\log n)$	$O(n\log n)$
AVERAGING	toroidal RGG	$O\left(\sqrt{n/\log n}\right)$	$O\left(\sqrt{n/\log n}\right)$	$O(\sqrt{n\log n})$	$O(\sqrt{n\log n}\log n)$	$O(n\log n)$

Table 5.1: Performance of gossip algorithms. The cost is the number R of messages that needs to be sent at each gossip round. In the last row, path averaging on toroidal random geometric graphs is run using $(\leftrightarrow, \updownarrow)$-box routing

Chapter 6

Interval Consensus and Voting

As explained in the introduction, the voting problem is an old problem, which is solved in this chapter. All the nodes in a network vote for 0 or 1 and want to know in a distributed fashion whether there is a majority of 0's or 1's in the network. The nodes are allowed to communicate a limited amount of information to their neighbors. Based on the received information and on an updating rule, which is identical from one node to another, nodes iteratively update their states. Eventually, by considering its local state, every node should know the majority vote.

Previous work has focused on 1 bit states (two states only: 0 and 1), and has failed to make the states converge to the majority vote. In this thesis, we propose to allow more than 1 bit memory at each node. In particular, we show that 2 bits states are sufficient to solve the voting problem with probability 1 in any connected network. Furthermore, we extend our solution to the multiple voting problem, where nodes can choose between more than 2 candidates. We give exacts solutions for the ternary and the quaternary voting problems.

The voting problem is actually linked to the average consensus problem. The next section explains how quantized versions of gossip lead to voting automata. More precisely, the voting problem is a sub-category of the *interval consensus* problem, which comes from the average consensus problem.

6.1 From Quantized Average Consensus to Voting

Distributed average consensus algorithms have been designed to solve distributed coordination problems in networks. Most of them, however, rely on the exchange of analog values and on infinitely precise memory at each node. Kashyap, Başar and Srikant suggested in [50] an average consensus algorithm over integers, which is a quantized version of pairwise gossip [14]. In their setting, nodes initially measure some integer values, and the two integers framing the average of these values are denoted by L and $L+1$. When convergence is reached, some nodes have state L, the remaining nodes have state $L+1$, and the overall average is preserved. This quasi-consensus was called "quantized consensus".

Kar and Moura [48] and Aysal, Coates and Rabbat [6] designed probabilistic algorithms that are able to reach a true consensus. In [6], it is shown that if nodes use probabilistic quantization at each iteration, then all the states converge to a common but random quantization level. These probabilistic algorithms compute an unbiased estimate of the average of the initial data, but with a non zero variance, hence the precision of the results is not guaranteed.

On the contrary, in the first algorithm we mentioned [50] , if a node reaches state L, it concludes that the average lies in $[L-1, L+1]$. A node with state $L+1$ outputs the interval $[L, L+2]$. Both conclusions are correct, but the nodes did not reach an *interval consensus*. This absence of consensus may lead the nodes to take two different and uncoordinated decisions.

It would be interesting to construct quantized distributed averaging algorithms that reach interval consensus. One particular instance of this interval consensus problem is the *voting problem*. Indeed, if one is able to compute the average of the initial votes (0's and 1's), then comparing the average to 0.5 is sufficient to deduce the majority vote. Assume that the number n of nodes is chosen odd in order to always have a strict majority. Then note that the algorithm of Kashyap et. al [50]. solves the voting problem if the quantization step is smaller than $1/(2n+1)$, which requires to code states on order $O(\log n)$ bits. The interesting question now is whether the voting problem has a solution with states of size $O(1)$, indeed larger than 1 bit, yet smaller than $O(\log n)$ bits. As a direct consequence of our work, we show that 2 bits are sufficient to solve the voting problem, without taking into account the termination phase of the algorithm. Remember that termination cannot be done with less than $O(\log n)$ bits. However, if x voting processes are run in parallel, then each process requires 2 bits in the voting algorithm phase, and the verification phase needs $O(\log n)$ bits in *total*. Hence, per process, the whole algorithm requires $2 + O(\log n/x)$ bits, which is small if the number x of processes exceeds $\log n$.

6.1.1 The interval consensus problem

At time $t = 0$, n nodes measure some quantized values $(x_1[0], x_2[0], \ldots, x_n[0])$, where $\mathbb{R}$ has been uniformly quantized with step δ. We denote by x_{ave} the average of the n measurements:

$$x_{ave} = \frac{1}{n}\sum_{i=1}^{n} x_i[0].$$

The nodes can communicate through a connected network $\mathcal{G}$ and we are given an ordered subset of quantization levels $\Theta_1 < \Theta_2 < \ldots < \Theta_r$ called thresholds. The goal is to design a quantized distributed algorithm such that nodes can tell whether the average x_{ave} is smaller than Θ_1, or between Θ_1 and Θ_2, or between Θ_2 and Θ_3, or ..., or larger than Θ_r. At each step of the algorithm, nodes can store a limited number of bits as their current state, and neighboring nodes can exchange their states. Based on their final state, all the nodes should reach a consensus on the interval $[\Theta_i, \Theta_{i+1}]$ which contains x_{ave}. To simplify, we assume that x_{ave} can not be threshold level ($x_{ave} \neq \Theta$).

Throughout this section, three examples are discussed where nodes initially vote for 0 or 1. For each of these problems, the parameters should be chosen in the following way:

1. *The binary voting problem.* Are there more 0's or 1's? Parameters: $\delta = 0.5$, $\Theta = 0.5$, 3 quantization levels: $\{0, 0.5, 1\}$, n odd.

2. *The large majority voting problem.* Is there a large majority winner? A large majority winner gets more than 2/3 of the votes. Parameters: $\delta = 1/3$, $\Theta_1 = 1/3$, $\Theta_2 = 2/3$, 4 quantization levels: $\{0, 1/3, 2/3, 1\}$, $n/3 \notin \mathbb{N}$.

3. *The quorum checking problem.* Have at least 2/3 of the nodes voted for 1? Parameters: $\delta = 1/3$, $\Theta = 2/3$, 4 quantization levels $\{0, 1/3, 2/3, 1\}$, $n/3 \notin \mathbb{N}$.

Note that the only difference between the parameters of the last two problems is the presence of the threshold level $\Theta_1 = 1/3$ in the large majority voting problem. The quantization level 1/3 appears in the quorum checking problem even though 1/3 is not a threshold level because we assume that the quantization is *uniform*.

6.1.2 Interval consensus algorithm

Gossip based algorithm

Just as in [50], our algorithm is a quantized version of the pairwise gossip algorithm. Similarly to gossip, at the beginning of every round of our algorithm, an undirected edge of the communication network is randomly selected and its two end-nodes exchange their states. We denote by p_e the probability that edge e is chosen. The most common way of selecting edges is to assign a random exponential clock to each node. When their clock activates, nodes wake up and choose a neighbor uniformly at random among their neighbors. In that setting, if i and j are neighbors, edge $e = (i, j)$ is chosen with positive probability $p_e = (1/nd_i) + (1/nd_j)$, where d_i and d_j are the degrees of nodes i and j.

In pairwise gossip, nodes update their states to the average of the two states. By iterating this update rule over the successively chosen edges, all the states progressively converge to the average of the initial states. Our goal is to modify this simple averaging rule so that the states are quantized and so that the nodes reach an interval consensus.

Threshold values are split into two states

In order to achieve our goal, we assign *two states* to each threshold level Θ while all the other quantization levels are represented by one state only. An ordinary state will be denoted by its quantization value. The two states with threshold value Θ are distinguished by Θ^- and Θ^+. We order the set of states: if the quantization level of state x is smaller than the quantization level of state y, we write $x \prec y$. Also, for any threshold level Θ, we adopt the following convention: $\Theta^- \prec \Theta^+$. As a result, for example, the voting problem functions with 4 states, coded by the two bits we announced. The four states are ordered: $0 \prec 0.5^- \prec 0.5^+ \prec 1$. We adopt all the natural ordering vocabulary, which we adapt to $\prec$: min, max, $=$, $\preceq$, $\succ$, $\succeq$. In particular the notion of *consecutive states* is crucial. In previous example, 0 and 0.5^- are consecutive states. So are 0.5^- and 0.5^+. But 0 and 0.5^+ are *not* consecutive states.

Definition* 6.1.1 (Convergence). A quantized gossip algorithm has converged iff all the nodes have either equal or consecutive states.

In other words, an algorithm has converged when there are two consecutive states x and y such that every state in the network is equal to x or y. Suppose that we have run a converging algorithm that *preserves average*, and that the average x_{ave} is in $[\Theta_1, \Theta_2]$, then necessarily, the two converging states are Θ_1^+, Θ_2^- or quantized levels between these two. Individually, each node knows in which interval x_{ave} is. Even a node with state Θ_1^+ makes the correct decision, because the + sign tells it that $x_{ave} \geq \Theta_1$. Thanks to the threshold state splitting, we are able to locally decide common intervals. To simplify the wording, we say that state x has color C, if a node that has state s decides that interval C contains x_{ave}. For example in the voting problem, states 0 and 0.5^- have color 0 (they lead to the conclusion that 0 wins), and states 1 and 0.5^+ have color 1 (they lead to the conclusion that 1 wins)

Properties of the algorithm

In this section, we list the desirable properties of our algorithm. They are sufficient for the algorithm to converge, but not necessary. If nodes i and j are activated at time t:

- **Conservation property** The average should be preserved: $x_i[t+1] + x_j[t+1] = x_i[t] + x_j[t]$.
- **Contraction property** $x_i[t+1]$ and $x_j[t+1]$ should be either equal or consecutive states. Furthermore, if $x_i[t] = x_j[t]$ then $x_i[t+1] = x_j[t+1] = x_i[t] = x_j[t]$.
- **Mixing property** If $x_i[t] \preceq x_j[t]$, then $x_i[t+1] \succeq x_j[t+1]$. In particular, if $x_i[t]$ and $x_j[t]$ are consecutive states, then states are swapped: $x_i[t+1] = x_j[t]$ and $x_j[t+1] = x_i[t]$.

The algorithm in [50] has similar properties, which we have adapted to our setting. Our three properties imply two other properties we will use in Section 6.1.3:

Consequence 1: The conservation property together with the contraction property imply that:

$$\min(x_i[t], x_j[t]) \preceq x_i[t+1] \preceq \max(x_i[t], x_j[t]). \tag{6.1}$$

Consequence 2: The mixing property admits a converse, which we call the **swapping property**: if two states are swapped, then they are equal or consecutive. Indeed, otherwise they would contract.

Formulation of our algorithm

To simplify, we consider that quantization levels are centered at 0, i.e. they can be written $k\delta$, with k integer. At time t, an edge is randomly chosen. We

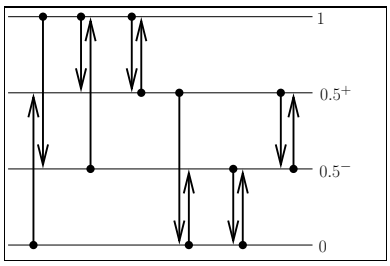


Figure 6.1: Update rules for 2 bits voter. The figure shows all the different pairwise updates possible and reads as follows: $(0, 1)$ is updated to $(0.5^+, 0.5^-)$, $(1, 0.5^-)$ is updated to $(0.5^+, 1)$, etc.

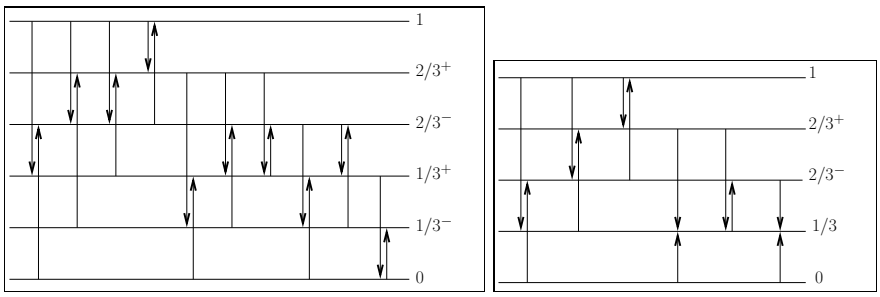
denote by i the activated node with smaller state and by j the other activated node: $x_i[t] \preceq x_j[t]$. Nodes i and j update their states according to the following rules:

$$\begin{aligned} x_i[t+1] &= \left\lceil \frac{x_i[t] + x_j[t]}{2\delta} \right\rceil \delta. \\ x_j[t+1] &= \left\lfloor \frac{x_i[t] + x_j[t]}{2\delta} \right\rfloor \delta. \end{aligned}$$

When $x_i[t+1]$ or $x_j[t+1]$ is equal to a threshold value Θ, we need to specify whether they are equal to Θ^+ or Θ^-. There are four cases:

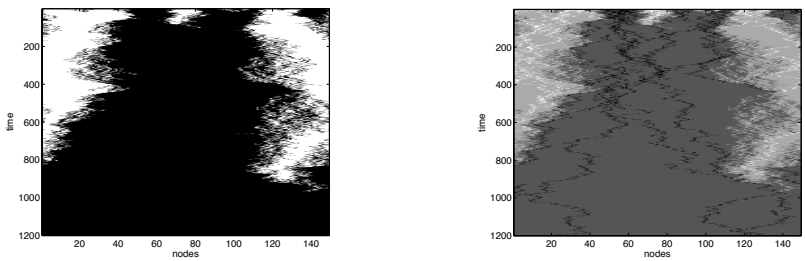
- If $x_i[t] = x_j[t]$, then $x_i[t+1] = x_j[t+1] = x_i[t]$.
- If $x_i[t] \neq x_j[t]$ and $x_i[t+1] = x_j[t+1] = \Theta$, then $x_i[t+1] = \Theta^+$ and $x_j[t+1] = \Theta^-$.
- If only $x_i[t+1] = \Theta$, then $x_i[t+1] = \Theta^-$.
- If only $x_j[t+1] = \Theta$, then $x_j[t+1] = \Theta^+$.

It is easy to check that the three properties hold for this algorithm. In Fig. 6.1 and 6.2, we show the update rules for the example problems of Section 6.1.1. By the conservation property, if the algorithm converges, then the network reaches consensus on the interval containing $\mathbf{x}_{ave}$ (as in Fig. 6.3, where a 2 bits voter simulation converges). In next section, we prove that our algorithm converges in finite time with probability 1.



(a) $2/3$ large majority voter (b) $2/3$ quorum checker

Figure 6.2: Update rules for the $2/3$ large majority voter and the $2/3$ quorum checker. The figures show all the different pairwise updates possible except for the easy consecutive states swaps.



(a) One bit displayed (b) Two bits displayed

Figure 6.3: The 2 bits binary voting algorithm. The algorithm was run on a circular network of 149 nodes, displayed here on a line. The successive state configurations are shown with time increasing down the page (149 iterations per line). In (a), black pixels represent states 0 and 0.5^-, white pixels represent states 0.5^+ and 1. In (b), the four states are distinguished with white (0), light gray (0.5^-), dark gray (0.5^+) and black (1). There were initially 3 more black states than white states.

6.1.3 Convergence Theorem

Theorem* 6.1.1. Let T be the first time the algorithm has converged. If the updating rules follow all the properties of Section 6.1.2, then $\mathbb{P}[T < \infty] = 1$.

Proof: Let $\overline{x}[t] = \max_i x_i[t]$ and $\underline{x}[t] = \min_i x_i[t]$. Equation (6.1) implies that $\{\overline{x}[t]\}_{t\geq 0}$ is an integer non increasing sequence and $\{\underline{x}[t]\}_{t\geq 0}$ is an integer non decreasing sequence. The first sequence being larger than the second, both sequences are bounded so that they admit limits in finite time, which we denote by $\overline{x}_\infty$ and $\underline{x}_\infty$. Let t_0 be the smallest iteration at which all the states in the network are larger than or equal to $\underline{x}_\infty$ and smaller than or equal to $\overline{x}_\infty$. For any $t \geq t_0$, let $\overline{\mathcal{M}}[t]$ (respectively, $\underline{\mathcal{M}}[t]$) be the set of nodes with state $\overline{x}_\infty$ (respectively, $\underline{x}_\infty$) at time t. Both sets have a non increasing number of elements. Therefore there is a time t_1 after which their cardinalities remain constant.

Let $\overline{m}[t_1]$ be a node in $\overline{\mathcal{M}}[t_1]$, and $\underline{m}[t_1]$ be another node in $\underline{\mathcal{M}}[t_1]$. We are now going to construct recursively two sequences of nodes $\{\overline{m}[t]\}_{t\geq t_1}$ and $\{\underline{m}[t]\}_{t\geq t_1}$. For any $t \geq t_1$,

$$\overline{m}[t+1] = \begin{cases} i[t] & \text{if } \overline{m}[t] \text{ and } i[t] \text{ communicate at time } t. \\ \overline{m}[t] & \text{if } \overline{m}[t] \text{ does not communicate at time } t. \end{cases}$$

$$\underline{m}[t+1] = \begin{cases} i[t] & \text{if } \underline{m}[t] \text{ and } i[t] \text{ communicate at time } t. \\ \underline{m}[t] & \text{if } \underline{m}[t] \text{ does not communicate at time } t. \end{cases}$$

We prove recursively that node $\overline{m}[t]$ has state $\overline{x}_\infty$ at time t for every $t \geq t_1$. It is true at time t_1 by construction of node $\overline{m}[t_1]$. Suppose that it is true until time t. Then, if $\overline{m}[t]$ communicates with a node $i[t]$ at time t, the two nodes will update their states. In order to keep the number of nodes in $\overline{\mathcal{M}}[t]$ constant, one of the two nodes should update its state to $\overline{x}_\infty$. By the Mixing property, we know that $i[t] = \overline{m}[t+1]$ has a state larger than the state of $\overline{m}[t]$ at time $t+1$. Therefore $\overline{m}[t+1]$ has state $\overline{x}_\infty$ at time $t+1$. If $\overline{m}[t]$ does not communicate at time t, then its state does not change and $\overline{m}[t+1]$ has state $\overline{x}_\infty$ at time $t+1$. For the same reasons, for any $t \geq t_1$, node $\underline{m}[t]$ has state $\underline{x}_\infty$ at time t.

Let T_{meet} be the first time at which $\overline{m}$ and $\underline{m}$ are the two end-nodes of the randomly chosen edge, or in other words the first time when $\overline{m}$ and $\underline{m}$ communicate with each other:

$$T_{meet} = \min\{t : \overline{m}[t] \text{ and } \underline{m}[t] \text{ communicate at time } t.\}$$

By definition of the sequence $\{\overline{m}[t]\}_{t\geq t_1}$ and $\{\underline{m}[t]\}_{t\geq t_1}$, $\overline{m}[T_{meet}]$ and $\underline{m}[T_{meet}]$ swap states at time T_{meet}. By the swapping property, we conclude that, at time T_{meet}, $\underline{x}_\infty$ and $\overline{x}_\infty$ are either equal or consecutive states, which implies that the algorithm has converged. In other words, the convergence time T is smaller than T_{meet}: $T \leq T_{meet}$. The last step of the proof is to show that $\mathbb{P}[T_{meet} < \infty] = 1$, which implies that $\mathbb{P}[T < \infty] = 1$.

We consider the joint process $\{\overline{m}[t], \underline{m}[t]\}_{t\geq t_1}$. Note that for any t, $\overline{m}[t] \neq \underline{m}[t]$. This process is a Markov chain over the state space E (here the states are pair of nodes in the network):

$$E = [1,n]^2 \setminus \{(1,1),(2,2),\ldots,(n,n)\}.$$

The communication network being connected, $\{\overline{m}[t], \underline{m}[t]\}_{t\geq t_1}$ is irreducible. Noting that E is a finite set, we can conclude that the Markov chain is positive recurrent. For any pair $e = (i,j)$ of neighboring nodes in the communication network, the Markov chain $\{\overline{m}[t], \underline{m}[t]\}_{t\geq t_1}$ admits a positive transition probability $p = p_e$ from state (i,j) to state (j,i). The first time such a transition occurs is equal to T_{meet}. To finish the proof, it is sufficient to notice that any transition with positive probability of an irreducible Markov chain over a finite state space is used in finite time with probability 1. This particular point of Markov chain theory is proven in following lemma.

Lemma 6.1.1. Let E be a finite set. Any transition with positive probability of an irreducible Markov chain over E is used in finite time with probability 1.

Proof: Let $\{X_n\}_{n\geq 0}$ be an irreducible Markov chain over E. We consider

the Markov chain $\{Y_n\}_{n\geq 0} = \{(X_n, X_{n+1})\}_{n\geq 0}$ over the state space of positive transition edges of chain $\{X_n\}_{n\geq 0}$. $\{Y_n\}_{n\geq 0}$ is irreducible as well, and the number of positive transition edges of chain $\{X_n\}_{n\geq 0}$ is finite. Therefore $\{Y_n\}_{n\geq 0}$ is positive recurrent and it reaches any of its states in finite time with probability 1. □

6.1.4 Back to the voting problem.

Denote by $\mathcal{A}_4$ the four states automaton described in Fig. 6.1. It has been proven that $\mathcal{A}_4$ solves the voting problem, in the sense that every agent is able to eventually tell the majority bit based on its own state. The automaton $\mathcal{A}_4$ is advantageously implementable on any connected network with guaranteed success. Last but not least, $\mathcal{A}_4$ can be easily generalized to many other automata. For example, we can build an automaton $\mathcal{A}_{2k}$ with $2k$ states ($k \geq 2$), k of color 0, and k of color 1. The states should form a uniform quantization of the interval $[0, 1]$. If $k = 3$, the six states are the usual $0, 0.5^-, 0.5^+$ and 1, with extra states 0.25 and 0.75. The six states automaton is shown in Fig. 6.4. As explained later, the number of states influences the speed of convergence of the automaton.

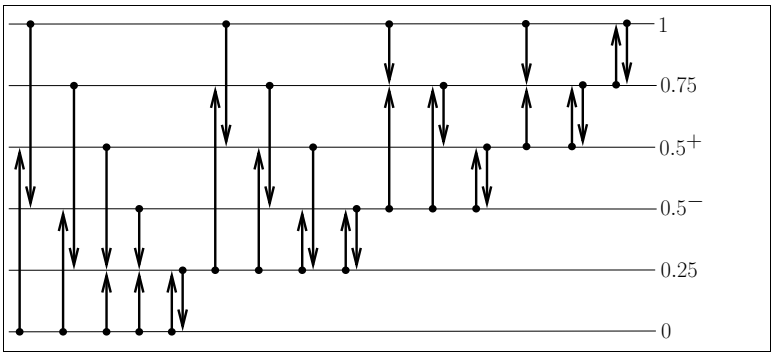


Figure 6.4: The 6 states voting automaton.

As we have seen, another possible generalization consists in classifying the density with a threshold value which is different than 0.5. For example, we can build a 2/3 density classifier with 5 states (0, 1/3, $2/3^-$, $2/3^+$ and 1), which determines if there is initially at least twice as many 1's as 0's. It is also possible to design automata with multiple thresholds. In Fig. 6.2 (a), the automaton learns if there is an initial bit twice as numerous as the other bit, and, if this is the case, which bit is the largely majority one. Many more automata can be built to solve extended density classification tasks.

Designing successful automata is simple as much as studying their speed of convergence is arduous. There are two different converging times: the earliest time at which all agents have states of the majority bit color, and the earliest time at which all states are consecutive or equal. The two converging times are not necessarily equal, the latter being larger than or equal to the former. Both of them depend on multiple parameters such as the number of agents, the topology of the network, the distribution of the wake up times, the number

of automaton states, and the initial density. An analogy can be constructed between the voting algorithms analysis and chemical kinetics. Each state update rule can be seen as a chemical reaction between two reactants yielding two products (e.g. $0 + 1 \rightarrow 0.5^- + 0.5^+$). The system starts with the two reactants 0 and 1, and after a number of parallel and concurrent reactions, it stabilizes with two products which indicate the denser initial reactant. The speed of one reaction depends on the rate at which the reactants collide (pairwise communicate in our case), and this rate depends on the densities of the reactants in the network; in a complete graph the updates behave precisely as second order chemical reactions. Figure 6.5 displays the average evolution of the reactants in a complete graph for initial configurations of density 0.6, using $\mathcal{A}_4$ first, $\mathcal{A}_6$ next. Although $\mathcal{A}_6$ creates more intermediary states than $\mathcal{A}_4$, the experiments of Fig. 6.5 show that it converges faster.

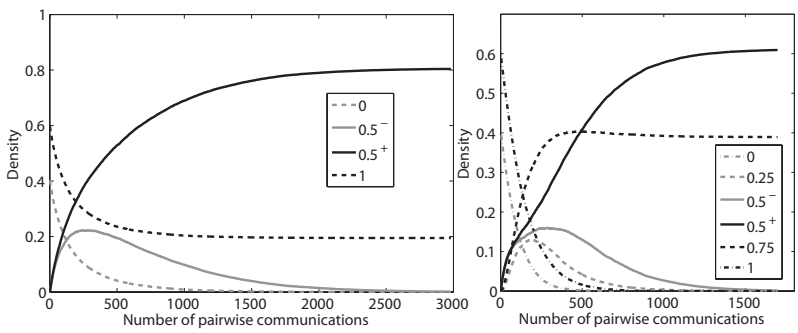


Figure 6.5: Average evolution of the states densities as the number of pairwise communications increases for $\mathcal{A}_4$ and $\mathcal{A}_6$. In both cases, 200 experiments were run on a complete graph with 149 agents, initial density 0.6

The final bottleneck reaction in $\mathcal{A}_4$ is $1 + 0.5^- \rightarrow 1 + 0.5^+$, whereas in $\mathcal{A}_6$, it is $0.75 + 0.5^- \rightarrow 0.75 + 0.5^+$. Due to average conservation, the density of 0.75 in the second case rapidly becomes twice as large as the density of 1 in the first case, which considerably speeds up the convergence. The densities of these bottleneck reactants decrease when the density of 1's in the initial configuration tends to 0.5. Hence, the algorithm is slower when the initial numbers of 0's and 1's are balanced than when there is a clear majority.

The influence of the network size and topology is illustrated in Fig. 6.6 with experiments on circular lattices of dimensions 1, 2 and 3 and on complete graphs of increasing sizes.

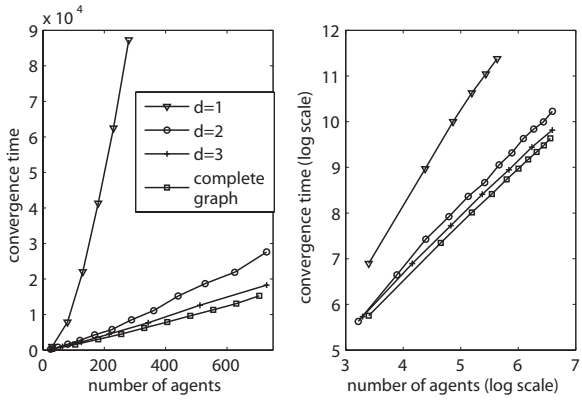


Figure 6.6: Network topology and size vs average speed of convergence. 4 topologies were tested: circular lattices of dimensions 1, 2 and 3, and the complete graph. Every sample point results from 100 experiments, using the 4 states automaton and initial configurations of density 0.6 ($\delta = 0.2n$).

As expected, $\mathcal{A}_4$ runs faster on small and well connected networks. The converging time of $\mathcal{A}_4$ in a complete graph with n nodes can be easily estimated. Let $\delta = n_1 - n_0$, where n_i is the number of nodes with initial state i and let 1 be the majority vote ($\delta > 0$). Let Y be the number of agents with state 0.5^- when the last 0 disappears. Then, in the worst case where $Y = O(n)$, $\mathcal{A}_4$ runs in $O(n^2 \log n)$ steps if δ is a constant, and in $O(n \log n)$ steps if δ is proportional to n ($\delta = \alpha n$).

Computing the speed of convergence of the voting automata in non complete graphs is not straightforward because states densities no longer suffice to describe the system. Instead, their precise location in the network should be considered, which considerably increases the number of variables to study. In [28], Draief and Vojnovic found upper bounds on the expected convergence time using mean-field approximation techniques for the 4 state automaton.

States perform weighted random walks in the network before they can meet, and depending on the network connectivity, the meeting process can be slow. The circle is particularly penalized by this phenomenon, which explains its slow convergence shown in Fig. 6.6. To overcome this information flow issue in the circle, one can imitate the GKL rule (see Introduction, page 7) in the following way. A woken up agent with state 0 calls its right neighbor, and agent with state 1 calls its left neighbor and an agent with a 0.5 state calls its left or right neighbors with equal probabilities 1/2. Thanks to this calling policy, states 0 and 1 drift in opposite directions and meet quickly. The efficiency of this strategy is illustrated in Fig. 6.7.

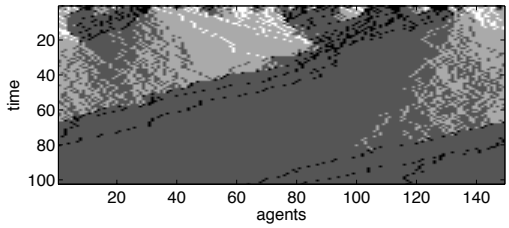


Figure 6.7: $\mathcal{A}_4$ with drift policy on a circle of 149 agents, displayed here on a line. The successive state configurations are shown with time increasing down the page (149 iterations per line). The 4 states are white (0), light gray (0.5^-), dark gray (0.5^+) and black (1). $\delta = 3$.

6.2 Multiple Voting

The approach of this section is different since the interval consensus problem is not considered here. We construct a general and theoretical framework, which is specifically adapted to the Voting problem. This leads to a theory which is very geometrical and rather abstract, and which includes the 4 states automaton as well as all the automata $\mathcal{A}_{2k}$, for $k > 1$, that were stated in the previous section. We recommend a good understanding of the previous section before reading the present one.

A finite set $\mathcal{V}$ of agents is connected in a graph $G = (\mathcal{V}, \mathcal{E})$, which is possibly time-varying. Each agent initially votes for a "color" among a finite set $\mathscr{C}$ of colors. By means of local communications only, and using constant, identical and simple updating rules, the agents want to distributively reach a state of consensus indicating the initial majority color. So far, this problem has appeared in its binary version ($|\mathscr{C}| = 2$), under two names such as density classification task or voting problem. In the previous section, an asynchronous graph automaton using 4 states was constructed in Section 6.1.2 for arbitrary connected networks, solving the binary voting problem in finite time with probability 1 for any vote configuration presenting a strict majority. Our goal is to generalize the previous graph automaton to automata that solve the *multiple* voting problem ($|\mathscr{C}| \geqslant 2$).

The general structure of our approach is the following. First we define pairwise asynchronous graph automata (PAGA), which is the class of automata the 4 state automaton in Section 6.1.2 belongs to. Then, in Section 6.2.1, we show that the voting problem has a geometrical interpretation, which guides our solution. In Section 6.2.3, we state a number of sufficient conditions for a PAGA to compute the majority color almost surely (a.s) in finite time in any connected network. The conditions on the updating rules depend only on the number $|\mathscr{C}|$ of possible votes: they are advantageously independent of the network topology. A number of abstract objects are defined to write the conditions on the PAGA and to prove convergence based on these conditions (the proof is in Section 6.2.4). To illustrate these objects with concrete examples, we describe beforehand, in Section 6.2.2, the 4 state automaton, which solves the binary voting problem, as well as a simple 15 state automaton, which solves the ternary voting problem. The theory we develop in this paper is able to handle more complicated voting automata, which we describe in Section 6.3.

In particular we build a 100 state automaton, which fulfills all conditions for $|\mathscr{C}| = 4$, the quaternary voting problem .

Pairwise Asynchronous Graph Automata (PAGA)

Assume that the agents $\mathcal{V}$ are fixed, but that the edges $\mathcal{E}$ connecting them can be time-varying. We denote by $\mathcal{G}$ the (finite) set of graphs representing the network over time, and by $\mathcal{D}$ the set of discrete probability distributions.

Definition* 6.2.1 (Pairwise asynchronous graph automata (PAGA)). A pairwise asynchronous graph automaton $\mathcal{A}$ is an ordered triple (Q, I, Δ, f), where

- Q is a finite set of states;
- $I \subseteq Q$ is a set of initial states;
- Δ is a local transition function of the form: $Q \times Q \to Q$;
- f is a function of the form $\mathcal{G} \to \mathcal{D}$. For a given graph $G = (\mathcal{V}, \mathcal{E})$, $f(G)$ is a probability distribution over $\mathcal{E}$.

A PAGA is run asynchronously. At each iteration t, an edge of the current graph $G(t) \in \mathcal{G}$ is drawn at random with the probability distribution $f(G(t))$. The two end agents i and j of the selected edge exchange their states q_i and q_j. Agent i updates its state q_i to $\Delta(q_i, q_j)$, and agent j updates its state q_j to $\Delta(q_j, q_i)$.

There is a classical example of function f to implement a PAGA in a distributed way. Agents are given random exponential clocks. When its clock ticks, an agent calls a neighboring agent in the current graph $G(t)$ at random. These two agents form the selected edge. If the neighbor choice is done uniformly then, for an edge $e = (i, j)$ where i has degree d_i and where j has degree d_j, $f(G)(e) = (1/d_i + 1/d_j)/|V|$. We have not defined final states, a PAGA theoretically never terminates. Therefore, a termination algorithm should be run in parallel. It is well known that an algorithm which detects consensus requires at least $O(\log n)$ bits at each node [5]. If there are x processes of votes, which majority needs to be computed, then each node needs $O(x \log(|Q|) + \log n)$ memory bits. In that case, the termination algorithm detects consensus, when the x processes have converged to consensus.

6.2.1 Voting PAGA

The possible votes are also called colors for simplicity. Let C_{maj} be the set of the initial majority color(s) (there might be a tie between several colors).

Definition* 6.2.2 (Voting PAGA for set $\mathscr{C}$, and graphs $\mathcal{G}$). Let $\mathscr{C}$ be the set of colors. A voting PAGA is a pairwise asynchronous graph automaton (Q, I, Δ, f) which admits the following properties:

- There is a set $\mathcal{P}$ such that $Q \subseteq \mathcal{P} \times \mathscr{C}$. Any state q can be written as $q = (x, c)$; x is called the coordinate (or point) of q, and c its color. By extension, an agent with state q is also said to have coordinate x and color c.

- For every color $c \in \mathscr{C}$, there is a single state $(o_c, c) \in I$, which is adopted initially by the agents voting for c; $I = \{(o_c, c) : c \in \mathscr{C}\}$.
- When running with this initial state configuration, there is a finite time T after which all the agents in $\mathcal{G}$ almost surely have color in C_{maj}.

According to this definition, a voting PAGA solves the voting problem for colors $\mathscr{C}$ and network $\mathcal{G}$. Indeed, after iteration T, all the agents know the majority vote based on their states. The goal of this paper is to state sufficient conditions on Q, I, Δ, f and $\{G(t)\}_{t \geqslant 0} \in \mathcal{G}^{\mathbb{N}}$ for a PAGA to be a voting PAGA.

The Distributed Center of Mass Problem

To achieve our goal, we first state our problem as a geometry problem. Let $S_{|\mathscr{C}|}$ be the regular simplex with $|\mathscr{C}|$ vertices (denoted by $O_1, \ldots, O_{|\mathscr{C}|}$), and G be their center of mass (or barycenter). For every $c \in \mathscr{C}$, vertex O_c is called the vertex of color c. S_1 is a point, S_2 is a line, S_3 is an equilateral triangle, S_4 is a regular tetrahedron, etc. We denote by $\vec{e_c}$ the vector $\vec{e_c} = \vec{GO_c}$. By definition $\sum_{c=1}^{|\mathscr{C}|} \vec{e_c} = 0$. For any color $c \in \mathscr{C}$, let n_c be the number of agents that voted for c. We claim that the position of the barycenter of the system $\{(O_1, n_1), \ldots, (O_{|\mathscr{C}|}, n_{|\mathscr{C}|})\}$ contains the majority vote information. Indeed, the following lemma shows that there is a bijective mapping between the barycenter points in $S_{|\mathscr{C}|}$ and the vote ratio configurations $(n_1/|\mathcal{V}|, \ldots, n_{|\mathscr{C}|}/|\mathcal{V}|)$, where $\sum_{c=1}^{|\mathscr{C}|} n_c = |\mathcal{V}|$.

Lemma 6.2.1 (Center of mass mapping or Karatheodory theorem). For any point $B \in S_{|\mathscr{C}|}$, there is a unique sequence $(\lambda_1, \lambda_2, \ldots, \lambda_{|\mathscr{C}|}) \in [0,1]^{|\mathscr{C}|}$, such that $\sum_{c=1}^{|\mathscr{C}|} \lambda_c = 1$ and

$$\vec{GB} = \sum_{c=1}^{|\mathscr{C}|} \lambda_c \vec{e_c}.$$

Proof: Suppose that there were two such sequences $(\lambda_1, \lambda_2, \ldots, \lambda_{|\mathscr{C}|})$ and $(\mu_1, \mu_2,$
$\ldots, \mu_{|\mathscr{C}|})$. Then,

$$0 = \sum_{c=1}^{|\mathscr{C}|} (\lambda_c - \mu_c)\vec{e_c} = \sum_{c=1}^{|\mathscr{C}|} (\lambda_c - \mu_c)(\vec{e_c} - \vec{e_1}) = \sum_{c=2}^{|\mathscr{C}|} (\lambda_c - \mu_c)(\vec{e_c} - \vec{e_1}).$$

By definition of volume, the determinant $det((\vec{e_2} - \vec{e_1}), (\vec{e_3} - \vec{e_1}), \ldots, (\vec{e_{|\mathscr{C}|}} - \vec{e_1}))$ is not equal to zero which means that the vectors $(\vec{e_2} - \vec{e_1}), (\vec{e_3} - \vec{e_1}), \ldots, (\vec{e_{|\mathscr{C}|}} - \vec{e_1})$ are linearly independent. Therefore, for any $c \geq 2$, $\lambda_c = \mu_c$. But $\sum_{c=1}^{|\mathscr{C}|} \lambda_c = \sum_{c=1}^{|\mathscr{C}|} \mu_c = 1$, hence $\lambda_c = \mu_c$ for every $c = 1, \ldots, |\mathscr{C}|$. ■

The regularity of $S_{|\mathscr{C}|}$ is not used in the proof of Lemma 6.2.1, but it gives a very simple criterion to find the majority color $\arg\max_i n_i$: it is the color corresponding to the vertex of $S_{|\mathscr{C}|}$ which is the closest to the barycenter B of the system $\{(O_1, n_1), \ldots, (O_{|\mathscr{C}|}, n_{|\mathscr{C}|})\}$.

From these observations comes an idea for a voting algorithm: assign to each agent the vertex corresponding to its voting color, have the agents compute distributively the center of mass of the assigned vertices, and make them

compute locally its closest vertex. Pairwise gossip for average consensus [14] is a distributed algorithm which computes averages. It acts like a PAGA except that $Q = \mathbb{R}$; when two agents i and j exchange their states x_i and x_j, they both update their state to $(x_i + x_j)/2$. This procedure conserves the global average, and with time, the states in the network approximate the true average up to any desired level of precision, if the network is jointly connected in time. This algorithm is obviously generalizable to multiple dimensions. In this paper, we derive a quantized version of pairwise gossip with $|Q| < \infty$, such that based on its state, an agent can eventually tell which is the closest vertex to the center of mass.

In this work, $\mathcal{P}$ is a set of points in $S_{|\mathscr{C}|}$ that is carefully chosen, because the set of states Q is determined by $\mathcal{P}$. Let $A \in \mathcal{P}$, and x be the coordinates of A, then we use state notations (A, c) and (x, c) interchangeably.

Definition* 6.2.3 (Valid states with respect to $\mathcal{P}$). Let $\mathcal{P}$ be a set of points in $S_{|\mathscr{C}|}$. For any point $A \in S_{|\mathscr{C}|}$, $d_{min}(A) = \min_{\gamma \in \mathscr{C}} d(A, O_\gamma)$, where d is the Euclidean distance. A valid state with respect to $\mathcal{P}$ is defined as follows:

$$(A, c) \text{ is a valid state} \iff \begin{cases} A \in \mathcal{P}, \\ d(A, O_c) = d_{min}(A). \end{cases}$$

In our work, the state set Q is taken as the set of valid states with respect to a set $\mathcal{P}$: $Q = \{(A, c) : A \in \mathcal{P} \text{ and } d(A, O_c) = d_{min}(A)\}$. By extension, we say that a point $A \in S_{|\mathscr{C}|}$ has color c if $d(A, O_c) = d_{min}(A)$. Note that point $A \in S_{|\mathscr{C}|}$ may have several colors, whereas a state $(x, c) \in Q$ has a unique color c.

In the next section, we give two simple examples of PAGA so that the theory developed in Sections 6.2.3 and 6.2.4 can already be supported by concrete examples.

6.2.2 Simple Examples of Voting PAGA

The two following PAGA are voting PAGA if the graphs $\{G(t)\}_{t \geqslant 0}$ are i.i.d. in $\mathcal{G}$, and if $\widetilde{G} = (\mathcal{V}, \bigcup_{\mathcal{G}} \mathcal{E})$ is a connected graph, i.e. graphs in $\mathcal{G}$ are *jointly connected*.

The 4 State Binary Voter

We recall here the 4 state automaton derived in Section 6.1.2, in order to get used to the geometrical framework in a simple example. In the binary voter problem, $\mathscr{C} = \{1, 2\}$ and $S_{|\mathscr{C}|} = S_2$ is a line segment O_1O_2. The barycenter G of the segment is is thus its center. To construct Q, we first define $\mathcal{P} = \{O_1, G, O_2\}$. Then, according to Definition 6.2.3, $Q = \{(O_1, 1), (G, 1), (G, 2), (O_2, 2)\}$. $I = \{(O_1, 1), (O_2, 2)\}$. We choose for example the classical function f described

in Section 6.2. For every pair of states (q_1, q_2), we define $\Delta(q_1, q_2)$ as follows:

q_1 \ q_2	$(O_1, 1)$	$(G, 1)$	$(G, 2)$	$(O_2, 2)$
$(O_1, 1)$	$(O_1, 1)$	$(G, 1)$	$(G, 1)$	$(G, 2)$
$(G, 1)$	$(O_1, 1)$	$(G, 1)$	$(G, 2)$	$(O_2, 2)$
$(G, 2)$	$(O_1, 1)$	$(G, 1)$	$(G, 2)$	$(O_2, 2)$
$(O_2, 2)$	$(G, 1)$	$(G, 2)$	$(G, 2)$	$(O_2, 2)$

. (6.2)

As updates are pairwise asynchronous, we recommend to read the table by pairs: $\Delta(q_1, q_2)$ and $\Delta(q_2, q_1)$, and to check that, as in pairwise gossip, the center of mass is conserved by the automaton updates.

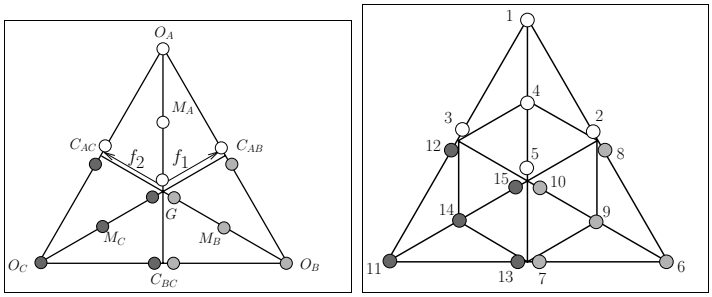
The 15 State Ternary Voter

In the ternary voter problem, $\mathscr{C} = \{1, 2, 3\}$ and $S_{|\mathscr{C}|} = S_3$ is an equilateral triangle $O_1O_2O_3$. The barycenter G of the simplex is located at the intersection of the medians $O_1C_{2,3}$, $O_2C_{1,3}$ and $O_3C_{1,2}$, where $C_{a,b}$ is the middle of O_aO_b. For $i = 1, 2, 3$, let M_i be the middle of GO_i. To construct Q, we first define $\mathcal{P} = \{O_1, O_2, O_3, C_{1,2}, C_{1,3}, C_{2,3}, M_1, M_2, M_3, G\}$. Then, according to Definition 6.2.3, $Q = \{(O_1, 1), (O_2, 2), (O_3, 3), (C_{1,2}, 1), (C_{1,2}, 2), (C_{1,3}, 1), (C_{1,3}, 3), (C_{2,3}, 2), (C_{2,3}, 3), (M_1, 1), (M_2, 2), (M_3, 3)(G, 1), (G, 2), (G, 3)\}$. To simplify notations, the states are numbered from 1 to 15 as in Fig. 6.8(b). $I = \{(O_1, 1), (O_2, 2), (O_3, 3)\}$. We choose the same function f as in previous automaton. For every pair of states (q_1, q_2), we define $\Delta(q_1, q_2)$ as follows:

q_1 \ q_2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	4	4	8	5	2	2	4	12	3	5	3	4
2	1	2	4	4	5	6	10	8	9	10	14	4	10	15	10
3	1	5	3	4	5	9	14	5	10	15	11	12	14	14	15
4	1	2	3	4	5	9	10	2	10	5	14	3	15	12	5
5	4	2	3	4	5	9	7	8	9	10	14	12	13	14	15
6	2	8	10	8	9	6	7	8	9	9	13	10	7	7	9
7	4	9	15	5	10	6	7	9	9	10	11	15	13	14	15
8	1	2	4	4	5	6	10	8	9	10	14	4	10	15	5
9	4	8	5	2	10	6	7	8	9	10	14	15	7	15	10
10	4	2	3	4	5	9	7	8	9	10	14	12	13	14	15
11	3	15	12	12	14	7	13	15	13	14	11	12	13	14	14
12	1	5	3	4	5	9	14	5	10	5	11	12	14	14	15
13	4	9	15	5	10	6	7	9	9	10	11	15	13	14	15
14	4	5	12	5	15	9	13	10	7	15	11	12	13	14	15
15	4	2	3	4	5	9	7	8	9	10	14	12	13	14	15

(6.3)

To get familiar with the automaton, we suggest that the reader checks some transitions (together with their symmetric transitions), and locates them on Fig. 6.8(b): $\Delta(1, 6)$, $\Delta(1, 5)$, $\Delta(1, 7)$, $\Delta(1, 9)$....



(a) State names. (b) State numbering.

Figure 6.8: The ternary voting problem. (a) The 15 states of the automaton described in Section 6.2.2 and their associated tiling. (b) Same points $\mathcal{P}$ as in Fig. 6.8(b), but associated with a tiling which does not follow conditions $Q2$ and $Q3$.

6.2.3 Sufficient Conditions to Construct Voting PAGA

Our goal is to build a theoretical framework in which we can show that all the automata we formulate in this paper do converge and solve the voting problem in finite time with probability 1. In this section, we fix a set of colors $\mathscr{C}$, and we state sufficient conditions on Q, I, Δ, f and $\{G(t)\}_{t\geqslant 0}$ for a PAGA to be a voting PAGA on $\mathscr{C}$. We design the conditions such that they are fulfilled by all the automata we present in this paper, so that we can conclude that our automata are indeed voting PAGA. Before we state the conditions, we first need to set up a few definitions. $S_{|\mathscr{C}|}$ is a regular simplex with $|\mathscr{C}|$ vertices embedded in the Euclidean space of dimension $|\mathscr{C}|-1$.

Definition* 6.2.4 (Tiling associated to $\mathcal{P}$). A tiling associated to $\mathcal{P} \subset S_{|\mathscr{C}|}$ is a set of tiles, which are convex polytopes with positive volume, such that: (i) the tile interiors are disjoint, (ii) the union of all tiles is equal to $S_{|\mathscr{C}|}$, and (iii) any tile vertex is in $\mathcal{P}$.

Example 6.2.1. $\{O_1G, GO_2\}$ is a tiling associated to $\mathcal{P} = \{O_1, G, O_2\}$ from Section 6.2.2. Fig. 6.8 shows two tilings associated with points $\mathcal{P}$ of Section 6.2.2. □

Comments: We design voting automata with a finite number of states only. Therefore the set of state coordinates $\mathcal{P}$ is finite, and if the barycenter B of votes does not coincide with a point of $\mathcal{P}$, the automaton cannot reach a consensus on the coordinates of B. Instead, the idea is to have the agents coordinates converge to several points in $\mathcal{P} \bigcap \mathcal{T}$, where $\mathcal{T}$ is the tile which contains barycenter B.

Definition* 6.2.5 (Tile-neighbor). Two points in $\mathcal{P}$ are tile-neighbors if and only if they belong to the same tile.

Definition* 6.2.6 (i-face, colors of an i-face.). For any $i = 0, 1, \ldots, |\mathscr{C}|-1$, the i-faces of a tile $\mathcal{T}$ are recursively defined as follows:

- If $i = |\mathscr{C}| - 1$, then the only $(|\mathscr{C}| - 1)$-face is $\mathcal{T}$ itself,

- If $i < |\mathscr{C}| - 1$, then the i-faces are the intersections of the tangent hyperplanes of dimension i with the $(i+1)$-face(s) of $\mathcal{T}$.

In addition, the **colors of an i-face** $\mathcal{F}$ are the colors of its barycenter $G_{\mathcal{F}}$ (center of mass of its vertices); their set is $\{c : d(G_F, O_c) = d_{min}(G_F)\}$.

Comments: The vertices of an i-face of a tile $\mathcal{T}$ are also vertices of $\mathcal{T}$. Therefore, they belong to $\mathcal{P}$.

The 0-faces are the vertices of the tiles, the 1-faces are their edges, etc.

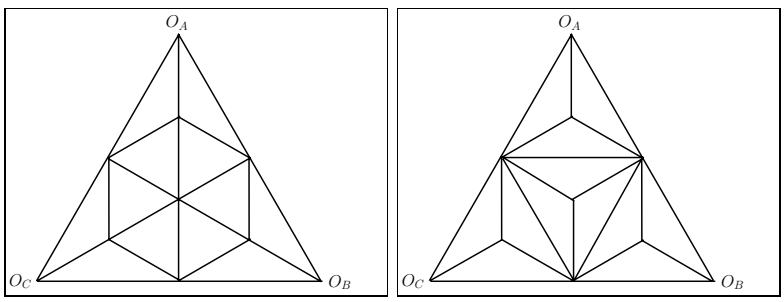
The study of i-faces is important because it is possible that the barycenter B of votes lies on a face and not in the interior of a tile. In this case, B belongs to more than one tile. To avoid any ambiguity, we will always consider the i-face it belongs to, and not only the tile(s) which contain(s) B.

We are now ready to state the conditions on Q.

Condition 6.2.1 (on Q). Q is constructed as in Definition 6.2.3, with a set of points $\mathcal{P}$, which admits an associated tiling such that:

- **Q1**: No tile interior contains a point with several colors: for any tile $\mathcal{T}$, for any A in the interior $\mathcal{T}^\circ$ of tile $\mathcal{T}$, $|\{c : d(A, O_c) = d_{min}(A)\}| = 1$.
- **Q2**: Any i-face $\mathcal{F}_i$ of any tile has at most one $(i-1)$-face $\mathcal{F}_{i-1}$ that does not have exactly the same colors as $\mathcal{F}$. If there is such an $(i-1)$-face $\mathcal{F}_{i-1}$, then, for any point $A \in \mathcal{F}_i \backslash \mathcal{F}_{i-1}$, A has exactly the same colors as $\mathcal{F}$.
- **Q3**: The tiling is consistent: if a set of points are pairwise tile-neighbors, then they all belong to the same tile.

Remark 6.2.1. $Q1$ implies that all the interior points of a tile share the same unique color. Indeed, if there were two points of different colors, then, by continuity of the distance function, there would be a two-color point somewhere between them; this is forbidden by $Q1$ since the tiles are convex. The set of points in $S_{|\mathscr{C}|}$ that have more than one color form *boundaries* between the strict majority zones, and they lie on tile faces of dimension smaller than $|\mathscr{C}| - 1$. Fig. 6.8 shows two tilings associated with the set $\mathcal{P}$ of the automaton of Sec. 6.2.2, one that follows Condition 6.2.1 (Fig. 6.8(b)) and one that does not (Fig. 6.9(b)).



(a) Correct tiling. (b) $Q2$ and $Q3$ do not hold.

Figure 6.9: Tilings for the triangle.

Condition 6.2.2 (on I). The simplex vertices $\{O_c\}_{c\in\mathscr{C}} \subset \mathcal{P}$ and I is the set $\{(O_c, c)\}_{c\in\mathscr{C}}$, where the initial state of an agent with vote c is (O_c, c).

Before we can state conditions on Δ, the notion of tiling needs to be refined. While coordinates converge to some tile points $\mathcal{T} \cap \mathcal{P}$, the states (coordinates + colors) converge to a cell, which is a set of states with tile-neighbor coordinates.

Definition* 6.2.7 (Cell $\mathcal{C}$ of a face $\mathcal{F}$). For any $i = 0, 1, \ldots, |\mathscr{C}| - 1$, and for any i-face $\mathcal{F}$ of any tile, a cell $\mathcal{C}$ is constructed. Let $c_1, \ldots, c_\ell$ be the ℓ colors of $\mathcal{F}$ and $(x_1, \ldots, x_{n_\mathcal{F}})$ the coordinates of the $n_\mathcal{F}$ points of $\mathcal{F} \cap \mathcal{P}$. Then $\mathcal{C} = \{(x_1, c_1), \ldots, (x_1, c_\ell), \ldots\ldots, (x_{n_\mathcal{F}}, c_1), \ldots, (x_{n_\mathcal{F}}, c_\ell)\}$ is the subset of $\mathcal{P} \times \mathscr{C}$ with coordinates $x_1, \ldots, x_{n_\mathcal{F}}$ and colors $c_1, \ldots, c_\ell$. $\mathcal{C}$ is said to have dimension i.

Example 6.2.2. The binary voting PAGA (Section 6.2.2) with tiling $\{O_1G, GO_2\}$ has cells $\{(O_1, 1), (G, 1)\}, \{(O_2, 2), (G, 2)\}, \{(O_1, 1)\}, \{(O_2, 2)\}$ and $\{(G, 1), (G, 2)\}$. See Fig. 6.2.3 for some examples of cells of the ternary voting PAGA (Section 6.2.2) □

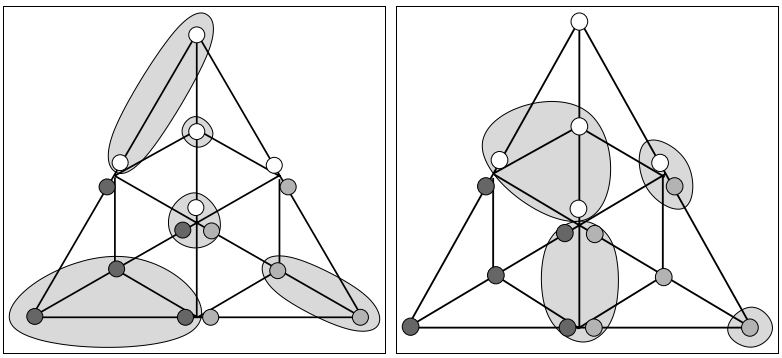


Figure 6.10: Example of cells of the ternary voting PAGA of Section 6.2.2.

Example 6.2.3. For any tile $\mathcal{T}$, we construct a cell of dimension $|\mathscr{C}| - 1$ with points $\mathcal{T} \bigcap \mathcal{P}$, and *one* color: the color of the interior of $\mathcal{T}$ (Remark 6.2.1). □

Example 6.2.4. Let G be the barycenter of $S_{|\mathscr{C}|}$. It is easy to show that $G \in \mathcal{P}$. The cell of dimension 0 constructed with G is $\{(G, 1), \ldots, (G, |\mathscr{C}|)\}$. □

Lemma* 6.2.1. Condition $Q1$ implies that all the elements of a cell are valid states with respect to $\mathcal{P}$: $\mathcal{C} \subset Q$.

Proof: Let $\mathcal{T}$ be a tile from which $\mathcal{F}$ was constructed, and let c_1 be the unique color of the tile interior $\mathcal{T}^\circ$ (see Remark 6.2.1). By continuity of the distance function, any point in $\mathcal{F} \subseteq \mathcal{T}$ has color c_1. Therefore, if $\ell = 1$, $\mathcal{C} \subset Q$. Suppose now that $\ell > 1$ (which excludes $i = |\mathscr{C}| - 1$), then by Definition 6.2.6, the barycenter $G_\mathcal{F}$ of $\mathcal{F}$ has colors $c_1, \ldots, c_\ell$. In particular $G_\mathcal{F}$ lies on the mediator of $O_{c_1}O_{c_2}$, which is a hyperplane of dimension $|\mathscr{C}| - 2$. If $\mathcal{F}$ is not entirely included in the mediator, then the mediator crosses $\mathcal{T}^\circ$, which is excluded by $Q1$. Indeed, if this was the case, there would be points in $\mathcal{T}^\circ$ that are

equidistant from O_{c_1} and O_{c_2}: c_2 would also be a color of $\mathcal{T}^\circ$. Therefore $\mathcal{F}$ is included in all the mediators $O_{c_1}O_{c_j}$ for $2 \leqslant j \leqslant \ell$, and any point of $\mathcal{F}$ is equidistant from $O_{c_1}, \ldots, O_{c_\ell}$. The points already have color c_1, therefore they also have colors $c_2, \ldots, c_\ell$. □ ■

Remark 6.2.2. The previous proof also shows that all the points in the i-interior of an i-face have exactly the same colors, which are the colors of the i-face. Indeed, we can replace $G_{\mathcal{F}}$ by any other interior point in the proof.

Definition* 6.2.8 (Adjacent states). Two states are adjacent if they belong to the same cell.

Remark 6.2.3. A state is adjacent with itself. The points of two adjacent states are tile-neighbors, but the reverse statement is not true. On the other hand, the following holds: *Two states (x_1, c_1) and (x_2, c_2) are adjacent if and only if points of coordinates x_1 and x_2 are tile-neighbors and if $(x_1, c_2) \in Q$ and $(x_2, c_1) \in Q$.*

Proof: ($\Rightarrow$) obvious. ($\Leftarrow$) Suppose that x_1 and x_2 are tile-neighbors and that they both have colors c_1 and c_2. Let $\mathcal{F}$ be the face of smallest dimension containing x_1 and x_2 and let i be its dimension. Points between x_1 and x_2 also have colors c_1 and c_2 (Voronoi cells are convex). $\mathcal{F}$ being the face of smallest dimension, then such points are in the i-interior of $\mathcal{F}$. By Remark 6.2.2, $\mathcal{F}$ has colors c_1 and c_2, and the cell constructed with $\mathcal{F}$ has these colors as well. □ ■

Definition* 6.2.9 (Smallest cell). Let $\mathcal{Z}$ be a subset of a tile. The smallest cell of $\mathcal{Z}$ is the cell of smallest dimension whose points convex hull contains $\mathcal{Z}$.

Comments: In the proofs of Theorems 6.2.1 and 6.2.2, we will show that the states of the agents eventually belong to the smallest cell of the barycenter B of the votes. This will guarantee that the agents eventually adopt the colors of B, which are the majority colors.

Condition 6.2.3 (on Δ). Condition $Q1$ holds so that Q is organized in cells as in Definition 6.2.7. For any states $q_1 = (x_1, c_1)$ and $q_2 = (x_2, c_2)$, let $q_1' = (x_1', c_1') = \Delta(q_1, q_2)$ and $q_2' = (x_2', c_2') = \Delta(q_2, q_1)$. Δ enjoys the following properties:

- **Δ1**: Conservation of the center of mass: $x_1' + x_2' = x_1 + x_2$.
- **Δ2**: Cell contraction: q_1' and q_2' are adjacent.
- **Δ3**: Mixing: if x_1 and x_2 are tile-neighbors, then $x_1' = x_2$ and $x_2' = x_1$.[1]
- **Δ4**: Color swap: Let $\mathcal{C}$ be the smallest cell of $\{x_1', x_2'\}$. If c_2 is a color of $\mathcal{C}$, then $c_1' = c_2$. Similarly, if c_1 is a color of $\mathcal{C}$, then $c_2' = c_1$.
- **Δ5**: Spatial contraction: There is an energy function $V : \mathcal{P} \to \mathbb{R}^+$ such that $V(x_1') + V(x_2') \leqslant V(x_1) + V(x_2)$, where the inequality is an equality if and only if $x_1' = x_2$ and $x_2' = x_1$.

[1] In practice, it is worth refining Δ3 to "Δ swaps tile-neighbor coordinates unless it is possible to further contract them within their tile T". If there are coordinates y and z in $T \cap \mathcal{P}$ such that $x_1 + x_2 = y + z$ and such that $d(y, z) < d(x_1, x_2)$, then it is interesting to set x_1' and x_2' equal to y and z in either order.

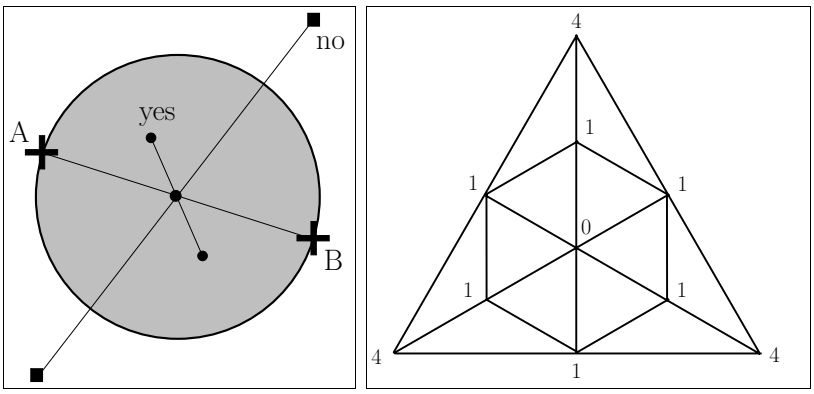
Remark 6.2.4. Let G (barycenter of $S_{|\mathscr{C}|}$) be the origin of the Euclidean space where $S_{|\mathscr{C}|}$ lies. $\Delta 5$ is called *spatial contraction* condition because $V(x) = x^T x$, which is a valid energy function for the automata of Section 6.2.2, has a simple geometrical contraction interpretation. For any point of coordinates x, $x^T x$ is the square distance to the origin G of the Euclidian space. Assume that $\Delta 1$ holds and let $\epsilon = x_1' - (x_1 + x_2)/2 = (x_1 + x_2)/2 - x_2'$. Then V satisfies $\Delta 5$ if and only if $\|\epsilon\|_2 < \|(x_1 - x_2)/2\|_2$ (except when $x_1' = x_2$ and $x_2' = x_1$): x_1' and x_2' are inside the circle centered at $(x_1 + x_2)/2$ which goes through x_1 and x_2. Indeed,

$$\begin{aligned}
V(x_1) + V(x_2) - (V(x_1') + V(x_2')) &= x_1^T x_1 + x_2^T x_2 \\
&\quad -(\frac{x_1^T + x_2^T}{2} + \epsilon^T)(\frac{x_1 + x_2}{2} + \epsilon) \\
&\quad -(\frac{x_1^T + x_2^T}{2} - \epsilon^T)(\frac{x_1 + x_2}{2} - \epsilon) \\
&= 2\left[\left(\frac{x_1 - x_2}{2}\right)^T \left(\frac{x_1 - x_2}{2}\right) - \epsilon^T \epsilon\right] \\
&= 2\left[\|\frac{x_1 - x_2}{2}\|_2^2 - \|\epsilon\|_2^2\right].
\end{aligned}$$

V satisfies the spatial contraction property if and only if

$$\|\epsilon\|_2 \leq \|\frac{x_1 - x_2}{2}\|_2.$$

The inequality should be strict as soon as states do not swap coordinates. For the ternary automaton of Section 6.2.2, V can be visualized in Fig. 6.11(b)



(a) geometrical interpretation (b) V for ternary automaton

Figure 6.11: Spatial contraction property (a) The pair of state updates for states in A and B should lie in the gray circle for the function $V(x) = x^T x$ to be a spatially contracting energy function. (b) $V(x) = x^T x$ is a valid energy function of the ternary automaton of Section 6.2.2

Condition 6.2.4 (on f and $\{G(t)\}_{t \geqslant 0}$). $\{G(t)\}_{t \geqslant 0}$ is i.i.d. Let $\mathcal{G}$ be the

finite set defined by $\mathcal{G} = \{G : \mathbb{P}[G(t) = G] > 0\}$. Let $\mathcal{E}' \subseteq \bigcup_{\mathcal{G}} \mathcal{E}$ be the set of edges e that are chosen with positive probability: $\mathcal{E}' = \{e \in \bigcup_{\mathcal{G}} \mathcal{E} : \exists G \in \mathcal{G}, f(G)(e) > 0\}$. $G' = (\mathcal{V}, \mathcal{E}')$ is a connected graph, i.e. graphs $\{G(t)\}_{t \geqslant 0}$ are jointly connected.

6.2.4 Almost Sure Convergence in Finite Time

Theorem* 6.2.1. Assume that Conditions $1, 2$ and 3 hold for a given PAGA. If there is a finite time T after which all agents states belong to the same cell, then, after T, every agent color is in C_{maj}.

Proof: Let $\mathcal{C}$ be the cell of smallest dimension the agents states belong to after time T, let $\mathcal{F}$ be the face $\mathcal{C}$ is built from and let R be the set of agents coordinates. Let B be the barycenter of the initial votes; B has color(s) C_{maj}. In this proof, we show that $\mathcal{C}$ is the smallest cell of B, that it implies that $\mathcal{C}$ has colors C_{maj}, and thus that agents have colors C_{maj}. By Conditions 6.2.2 and $\Delta 1$ (conservation of center of mass), B lies in the convex hull of the points of $\mathcal{C}$, i.e. $B \in \mathcal{F}$. Let $\mathcal{C}_s$ be the smallest cell of B, and $\mathcal{F}_s$ be the face it is built from. We need to show that $\mathcal{F} = \mathcal{F}_s$. First $\mathcal{F}_s \subseteq \mathcal{F}$ ($\mathcal{F}_s$ is a face of $\mathcal{F}$) because B belongs to both faces, and $\mathcal{F}_s$ is the one of smallest dimension. Suppose that $\mathcal{F} \neq \mathcal{F}_s$, then there is a point of R in $\mathcal{F} \backslash \mathcal{F}_s$. Tiles are convex, thus their faces are convex too. Hence, any point in $\mathcal{F} \backslash \mathcal{F}_s$ is not in the tangent hyperplane $\mathcal{H}$ to $\mathcal{F}$ containing $\mathcal{F}_s$, and $\mathcal{F} \backslash \mathcal{F}_s$ actually lies on one side only of $\mathcal{H}$. But B is a weighted barycenter of the points in R, thus B cannot be in $\mathcal{H}$, hence $B \notin \mathcal{F}_s$. This contradicts $B \in \mathcal{F}_s$. Therefore $\mathcal{F} = \mathcal{F}_s$, and $\mathcal{C}$ is the smallest cell of B. Two cases appear: (i) Either B constitutes a 0-face and then $\mathcal{C}$ is the cell built from the 0-face $\{B\}$ and color(s) C_{maj}. (ii) Or B is not a 0-face. Then $B \in \mathcal{F}^\circ$, and $\mathcal{C}$ has exactly the colors C_{maj} of B (Remark 6.2.2). The agents states are in $\mathcal{C}$, thus they have colors in C_{maj}. □ ■

Theorem* 6.2.2 (Main theorem). If $\{G(t)\}_{t \geqslant 0}$ and $\mathcal{A} = (Q, I, \Delta, f)$ satisfy Conditions 1,2,3 and 4, then $\mathcal{A}$ is a **voting PAGA**, i.e. there is a finite time T after which all the agents colors are a.s. in C_{maj}.

Proof: Let T be the first time when all agents states are in the same cell. By $\Delta 3$ and $\Delta 4$, if q_1 and q_2 are adjacent states, then Δ swaps states: $\Delta(q_1, q_2) = q_2$ and $\Delta(q_2, q_1) = q_1$. Therefore, states remain in the same cell forever after T [2]. Thus, according to Theorem 6.2.1, it is sufficient to show that there is a time $T < \infty$ when all agents are a.s. in the same cell to prove Theorem 6.2.2.

Sketch of the proof: This proof has two parts: "Coordinates" and "Colors". In the "coordinates" part, we use the spatial contraction property so set up a time T_1 after which states coordinates are systematically swapped whenever two agents communicate. We show that any pair of coordinates is eventually swapped with probability 1 because the network is jointly connected. By $\Delta 2$, updated states are adjacent, hence swapped coordinates are tile-neighbors. As a consequence, at time T_1, every pair of agents a.s. has tile-neighbor coordinates. The tiling being consistent, all the coordinates are a.s. in the same tile. Then we need to show that states eventually adopt the "right colors", i.e. the colors of a cell containing the coordinates of the agents. This is proven in the "color

[2] Note that states remain in the same cell even when Footnote 1 is implemented.

part" of the proof. Condition $Q2$ guarantees that there is an agent whose coordinates x_a admit only right colors. Each time an agent with coordinates x_a communicates with a neighbor, it transforms its neighbor's color into a right color. Since this coordinate x_a travels in the network thanks to $\Delta 3$, it eventually attracts all the colors into C_{maj}. This is how the algorithm proceeds, and the proof translates this phenomenon in the following way. Formally, we show that there is a time T_2 after which the number of right colors is maximum. Then we show that with probability 1, coordinate x_a eventually meets all the colors. Therefore, with probability 1, at time T_2, all the colors of the agents were already right colors.

Coordinates: For $i = 1, \ldots, |\mathcal{V}|$, let $q_i(t)$ be the state of agent i at time t. The total energy $\{E(t)\}_{t\geqslant 0} = \{\sum_{i=1}^{|\mathcal{V}|} V(q_i(t))\}_{t\geqslant 0}$ is a non increasing sequence ($\Delta 5$). Thus, it reaches a minimum in finite time T_1. We first show that, after T_1, all agents states are a.s. pairwise tile-neighbors, i.e. they all belong to a tile $\mathcal{T}$ ($Q3$). Take a pair of agents a and b (with coordinates x_a and x_b at time T_1) and build the following sequence: $a(T_1) = a$, $b(T_1) = b$ and for any $t \geqslant T_1$,

$$a(t+1) = \begin{cases} i(t) & \text{if } a(t) \text{ and } i(t) \text{ communicate at time } t. \\ a(t) & \text{if } a(t) \text{ does not communicate at time } t. \end{cases} \tag{6.4}$$

$$b(t+1) = \begin{cases} i(t) & \text{if } b(t) \text{ and } i(t) \text{ communicate at time } t. \\ b(t) & \text{if } b(t) \text{ does not communicate at time } t. \end{cases} \tag{6.5}$$

After T_1, by $\Delta 5$, pairs of communicating agents swap coordinates (otherwise E increases), therefore, for $t \geqslant T_1$, agent $a(t)$ (respectively, $b(t)$) has coordinates x_a (resp., x_b). We consider the joint process $\{a(t), b(t)\}_{t\geq T_1}$, which is a *homogeneous* Markov chain over the state space $H = [1, |\mathcal{V}|]^2 \setminus \{(1,1), (2,2), \ldots, (|\mathcal{V}|, |\mathcal{V}|)\}$ (for any t, $a(t) \neq b(t)$). Indeed, at any time t, $(a(t+1), b(t+1))$ depends on $(a(t), b(t))$ and on the edge $e(t)$ that is selected at random in an i.i.d. fashion at time t: $\mathbb{P}[e(t) = e] = \sum_{G\in\mathcal{G}} f(G)(e)\mathbb{P}[G(t) = G]$ is independent of t ($\{G(t)\}_{t\geqslant 0}$ is i.i.d). By Condition 4, the set $\mathcal{E}'$ of edges e such that $\mathbb{P}[e(t) = e] > 0$ forms a connected graph $G' = (\mathcal{V}, \mathcal{E}')$. Therefore $\{a(t), b(t)\}_{t\geq T_1}$ is irreducible. Moreover, for any edge $e = (i, j) \in \mathcal{E}'$, the Markov chain $\{a(t), b(t)\}_{t\geq T_1}$ admits a positive transition probability $p = \mathbb{P}[e(t) = e]$ from state (i, j) to state (j, i). It was shown in [10] that any transition with positive probability of an irreducible Markov chain over a finite state set is used in finite time with probability 1. Therefore, with probability 1, there is a finite time T_{ab} such that agents $a(T_{ab})$ and $b(T_{ab})$ communicate and swap their coordinates x_a and x_b at time T_{ab}. $\Delta 2$ imposes that updated states are adjacent. Thus x_a and x_b are tile-neighbors with probability 1. Therefore, at time T_1, states coordinates are a.s. pairwise tile-neighbors.

Colors: Let $\mathcal{C}$ be the smallest cell that the agents coordinates occupy at time T_1, and let us show that all the agents states end up in $\mathcal{C}$. If $\mathcal{C}$ has dimension 0, then the agents states are necessarily in $\mathcal{C}$ at time T_1 (such a cell gathers all the colors of a given tile vertex). Suppose now that $\mathcal{C}$ has at least two states of different coordinates. Let $i > 0$ be its dimension and let $\mathcal{N}(t)$ be the set of agents whose states are in $\mathcal{C}$, i.e. whose colors are colors of $\mathcal{C}$. $|\mathcal{N}(t)|$ is a non decreasing integer sequence, thus it reaches a maximum in finite time $T_2 \geqslant T_1$. Indeed, any cell of smaller dimension than $\mathcal{C}$, and which points are in $\mathcal{C}$, has the colors of $\mathcal{C}$ and maybe some more; thus $\Delta 4$ implies that any time

an agent with a state in $\mathcal{N}(t)$ is activated, then its activated neighbor in $G(t)$ adopts its color (and thus its state). $\mathcal{C}$ being the smallest cell of the agents coordinates, and considering condition $Q2$, $(*)$ there is at least one agent a in the network whose coordinates x_a at time T_2 admits *only* colors of $\mathcal{C}$. Let c_a be its color. Now, take any other agent b with state (x_b, c_b) at time T_2, and build the same sequences $\{a(t)\}_{t \geqslant T_2}$ and $\{b(t)\}_{t \geqslant T_2}$ as in Eq. 6.4, 6.5, with $a(T_2) = a$, $b(T_2) = b$. Then, for $t \geqslant T_2$, agent $a(t)$ has state (x_a, c_a). Furthermore, if c_b is not a color of $\mathcal{C}$, then $(**)$ the color of agent $b(t)$ at time t is *not* in $\mathcal{C}$ either (otherwise $|\mathcal{N}(t)|$ would increase). At any time t, the coordinates of $b(t)$ are x_b. From the "coordinates" part of the proof, we know that $a(t)$ and $b(t)$ communicate in finite time with probability 1. Therefore, there is a.s. a finite time T'_{ab} when agents $a(T'_{ab})$ and $b(T'_{ab})$ communicate. Agent $b(T'_{ab})$ updates its state to (x_a, c_a) and agent $a(T'_{ab})$ updates to (x_b, c) for some color c, such that states (x_a, c_a) and (x_b, c) are adjacent ($\Delta 2$). But x_a admits only colors in $\mathcal{C}$, and Remark 6.2.3 shows that the states (x_a, c_a) and (x_b, c) are adjacent only if x_a has color c, i.e., by statement $(*)$, only if c is a color of $\mathcal{C}$. Therefore, with probability 1, c is a color of $\mathcal{C}$, which is possible only if c_b is a color of $\mathcal{C}$ as well (by statement $(**)$). In other words, for any agent b, at time $T_2 < \infty$ (and after time T_2), its state (x_b, c_b) is a.s. in $\mathcal{C}$; at time T_2, agents states a.s. belong to a single cell $\mathcal{C}$. ■

6.3 Evolved Examples of Voting PAGA

6.3.1 Binary and ternary voters

For the binary and the ternary voting problems at least, voting PAGA are not unique. Sets $\mathcal{P}$ from Section 6.2.2 can be made larger by adding for example a point in the middle of each edge of each tile. This leads to the automata $\mathcal{A}_{2k}$ described in Section 6.1.4 for the binary voting problem. For the ternary voting problem, refer to Fig. 6.12 to see the new set $\mathcal{P}$, as well as its tiling and its associated energy V.

A large set $\mathcal{P}$ with a radically different tiling can also be designed for the ternary voting problem, as shown on Fig. 6.13. The resulting automaton is interesting because it is not spatially contracting in the euclidian sense. However, an energy $V(x) \neq x^T x$ can be found to show that the automaton does classify majority among three votes. Unfortunately, the automaton has $33 = 2^5 + 1$ states, which still requires 6 bits. Remember that the ternary PAGA of Section 6.2.2 has 15 states, that can be coded on 4 bits.

Larger sets $\mathcal{P}$ lead to voting PAGA which tend to reach consensus on C_{maj} faster, as explained in Section 6.1.4. Their transition functions Δ are omitted because they do not fit on a page, but the reader can easily infer the transitions. Indeed, if a proper tiling and an energy function V are given, it is easy to build a correct Δ function by following the "instructions" of Condition 6.2.3. In a few words, for any pair of states $q_1 = (x_1, c_1)$ and $q_2 = (x_2, c_2)$,

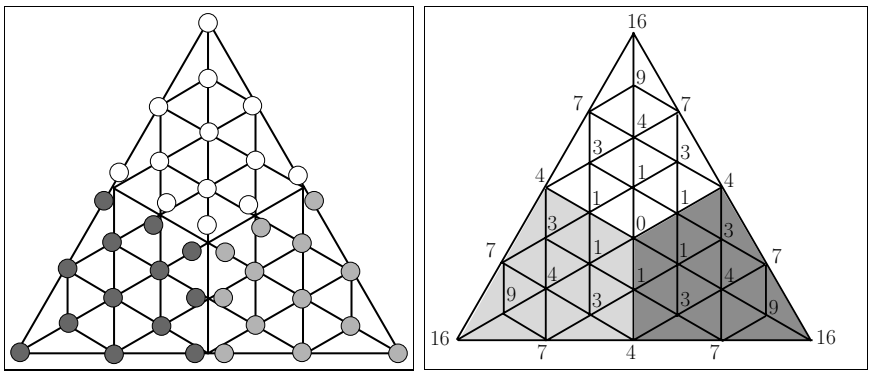
1. Compute the center of mass $m = (x_1 + x_2)/2$.
2. If m is inside a tile $\mathcal{T}$,
   - Look for two points y and z in $\mathcal{T}$, that have center of mass m.

- If there are several such pairs, take the pair that minimizes $V(y) + V(z)$.
- Assign (y, c_m) and (z, c_m) as updates $q'_1 = (x'_1, c'_1)$ and $q'_2 = (x'_2, c'_2)$. It is heuristically interesting to assign them in an order that maximizes
$$d(x_1, x'_1) + d(x_2, x'_2). \tag{6.6}$$

3. If m is on a face $\mathcal{F}$,

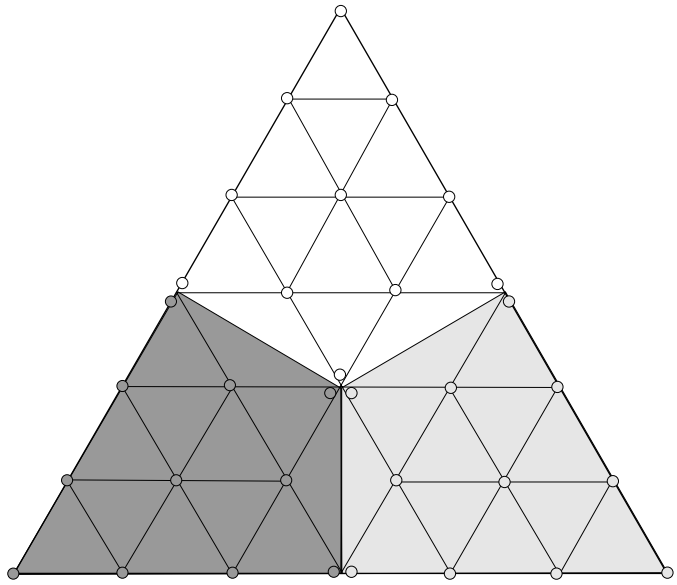
- Look for two points y and z in $\mathcal{F}$, that have center of mass m, that minimize the sum of energies V.
- Assign the update coordinates in order to maximize the sum of distances as in Eq. (6.6).
- The update colors should be colors of m. Follow $\Delta 4$ when possible. In the case where $\Delta 4$ does not impose any color, then choose arbitrarily any color c of m. We recommend that if m has several colors, then the two updates should have different colors. Also it is more elegant to respect the symmetry of the triangle, and to impose the same global number of updates of color 1, color 2 and color 3. The goal is not to give the advantage to any color in the automaton in case of a tie between two or three colors. Note that the triangle has a rotational symmetry, which makes perfectly balanced automata possible, whereas the tetrahedron is not rotationally symmetrical. It is thus difficult to construct well-balanced automata on the tetrahedron.

To summarize, the tiling and the energy function contains all the necessary information to code a correct automaton. From a design point of view, constructing a new voting PAGA boils down to finding a new tiling on which Conditions 6.2.1 and 6.2.3 can be applied.

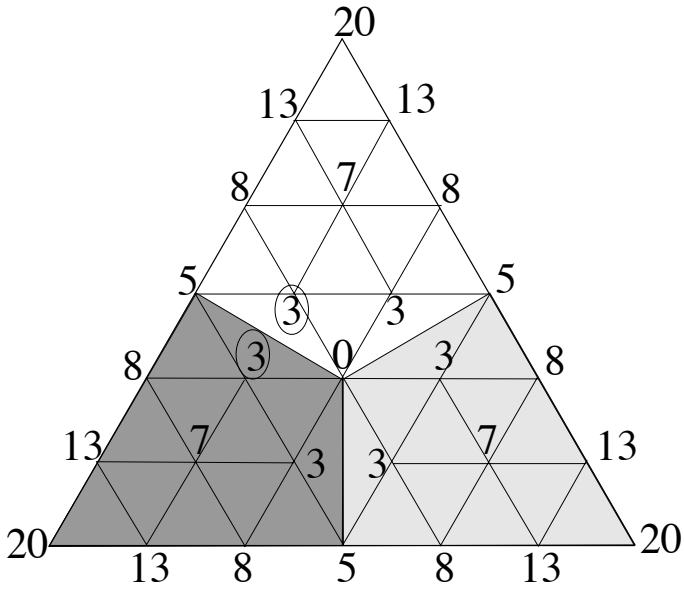


(a) 39 states (b) energy V

Figure 6.12: Ternary voter automaton with 39 states. The energy function in (b) is $V(x) = xx^T$



(a) 33 states



(b) energy V

Figure 6.13: Ternary voter automaton with 33 states. The energy function in (b) is *not* $V(x) = xx^T$. When the two circled states with $V = 3$ situated on both sides of GC_{12} communicate, they update to G ($V = 0$) and C_{12} ($V = 5$), one with color 1 (white), the other with color 2 (gray). $\Delta 5$ holds, but this updating is not spatially contracting in the euclidian sense.

color A				color B				color C				color D			
1	3	3	3	26	-3	-3	3	51	-3	3	-3	76	3	-3	-3
2	2	2	2	27	-2	-2	2	52	-2	2	-2	77	2	-2	-2
3	1	2	2	28	-1	-2	2	53	-1	2	-2	78	1	-2	-2
4	2	2	1	29	-2	-2	1	54	-2	2	-1	79	2	-2	-1
5	2	1	2	30	-2	-1	2	55	-2	1	-2	80	2	-1	-2
6	1	1	2	31	-1	-1	2	56	-1	1	-2	81	1	-1	-2
7	1	2	1	32	-1	-2	1	57	-1	2	-1	82	1	-2	-1
8	2	1	1	33	-2	-1	1	58	-2	1	-1	83	2	-1	-1
9	1	1	1	34	-1	-1	1	59	-1	1	-1	84	1	-1	-1
10	1	1	0	35	-1	-1	0	60	-1	1	0	85	1	-1	0
11	1	0	1	36	-1	0	1	61	-1	0	-1	86	1	0	-1
12	0	1	1	37	0	-1	1	62	0	1	-1	87	0	-1	-1
13	0	0	3	38	0	0	3	63	0	0	-3	88	0	0	-3
14	0	3	0	39	0	-3	0	64	0	3	0	89	0	-3	0
15	3	0	0	40	-3	0	0	65	-3	0	0	90	3	0	0
16	0	0	2	41	0	0	2	66	0	0	-2	91	0	0	-2
17	0	2	0	42	0	-2	0	67	0	2	0	92	0	-2	0
18	2	0	0	43	-2	0	0	68	-2	0	0	93	2	0	0
19	0	0	1	44	0	0	1	69	0	0	-1	94	0	0	-1
20	0	1	0	45	0	-1	0	70	0	1	0	95	0	-1	0
21	1	0	0	46	-1	0	0	71	-1	0	0	96	1	0	0
22	-1	1	1	47	1	-1	1	72	1	1	-1	97	-1	-1	-1
23	1	1	-1	48	-1	-1	-1	73	-1	1	1	98	1	-1	1
24	1	-1	1	49	-1	1	1	74	-1	-1	-1	99	1	1	-1
25	0	0	0	50	0	0	0	75	0	0	0	100	0	0	0

Table 6.1: State indexing for the 4-voter automaton with 100 states.

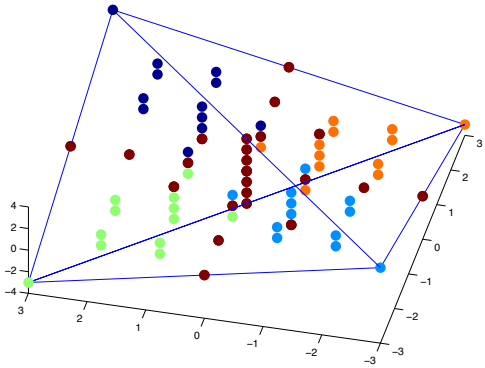
6.3.2 Quaternary voter

We have successfully built a quaternary voting PAGA.

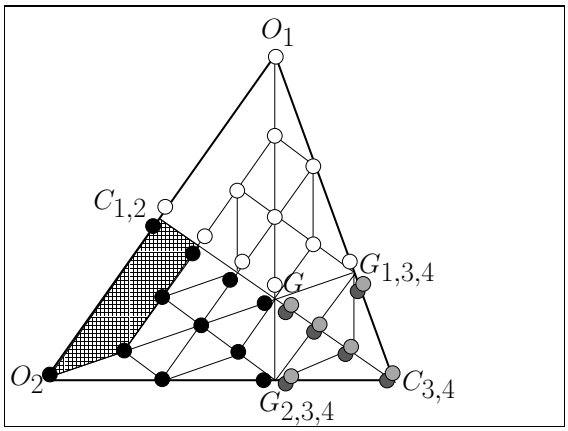
Construction of set $\mathcal{P}$

$\mathcal{S}_4$ is a regular tetrahedron with vertices O_1, O_2, O_3 and O_4 and barycenter G. Denote by C_{ij} the middle of edge O_iO_j and form the orthogonal basis $(\vec{GC}_{12}, \vec{GC}_{13}, \vec{GC}_{14})/3$ with origin at G. Consider the 3D-lattice $\mathcal{L}$ generated by the $\mathbb{Z}$-span of the basis, and let $\mathcal{P}$ be the intersection of $\mathcal{L}$ with $\mathcal{S}_4$ minus the points situated on the edges of $\mathcal{S}_4$ different from $\{O_i\}_{i\in[1,4]}$ and from $\{C_{ij}\}_{i\neq j\in[1,4]^2}$[3]. $\mathcal{P}$ generates 100 states, 25 of each color. See Fig. 6.14(b) and 6.14(a) to visualize the 100 states, which are listed in Table 6.1.

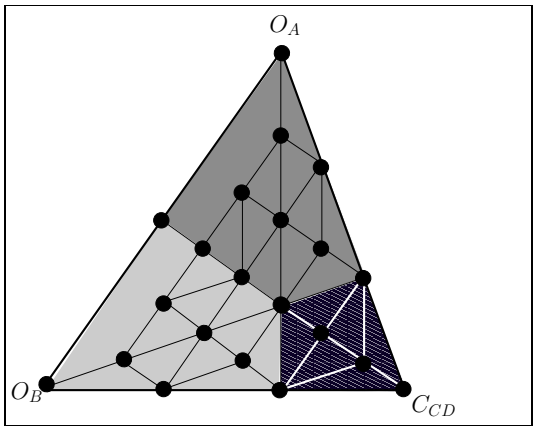
[3]This is useful to limit the number of states. Also, note that the quaternary PAGA, restricted to each of the 4 faces of the tetrahedron should constitute a ternary PAGA, which appears to be impossible if these points are not removed. This simplex face constraint is the main difficulty in the design of voting PAGA for arbitrary $\mathscr{C}$.



(a) Set $\mathcal{P}$



(b) States on a cut



(c) A cut of the tiling

Figure 6.14: (a) Set $\mathcal{P}$ in the tetrahedron. (b) States and tiling for the quaternary PAGA on the cut of $\mathcal{S}_4$ along the hyperplane $O_1O_2C_{3,4}$. A state (x, c) is represented by a circle of color c at position x. Color 1 is white, 2: black, 3: light gray, 4: dark gray. $G_{i,j,k}$ is the barycenter of face $O_iO_jO_k$. Note that Footnote 1 can be applied on the shaded 2-face. (c) The tiling, without the states.

Tiling

The 3D-lattice $\mathcal{L}$ cuts the tetrahedron in unit cubes. Each unit cube is divided in 6 tiles, shaped as tetrahedra. The 6 tetrahedra in each cube share a common edge, which is one of the 4 big diagonals[4]. Therefore, for each unit cube, one should choose one big diagonal. There are two cases:

1. The cube does not cross a frontier zone. Then, the big diagonal goes through the cube vertex closest to O_c, where is c is the color of the cube.

2. The cube crosses a frontier zone. Then the big diagonal is included in the frontier. Moreover no tetrahedron should cross the frontier zone.

Finally, for cubes that are close to the edges of the tetrahedron, we had to adapt the shape of the tiles to make the automaton work. See Fig. 6.14(c).

Energy V

Building the energy function V is slightly more complicated than for the triangle, since the quaternary PAGA is not strictly spatially contracting in the euclidian sense. Indeed, if the middle m of two states is equal to the center of mass of a cube, then the updated states should be the vertices of the big diagonal of the cube chosen in the tiling construction. Now, if the two states are originally another big diagonal of the same cube, then the updates do not spatially contract, i.e. they lie on the same sphere centered in m. Therefore, $V = x^T x$ is not appropriate. However, $V(x) = x^T x + \sqrt{x^T x}$ works, as it adds a perturbation factor that favors states aligned with G. The resulting energy table in given in Table 6.2.

Transition function Δ

It is important to understand that only the function Δ is needed to implement a voting algorithm. As we have already computed Δ, there is no need to understand how it was built or why it works to use it. Δ is a simple $|Q| \times |Q|$ table that indicates how to update the states for any pair of states. For the tetrahedron, 10000 entries have thus been computed, using all the possible symmetries of the tetrahedron. The detailed construction method and a few entries are given in Appendix B.

Simulations

Fig. 6.15 shows a simulation, where the automaton runs on a circle network. The algorithms induced by the automata can be slow and there are heuristic to make them converge faster.

[4] In more technical terms, the big diagonal is called the circumradius.

states				energy
1	26	51	76	16.0981
2	27	52	77	7.7321
3	28	53	78	6
4	29	54	79	6
5	30	55	80	6
6	31	56	81	4.2247
7	32	57	82	4.2247
8	33	58	83	4.2247
9	34	59	84	2.3660
10	35	60	85	1.7071
11	36	61	86	1.7071
12	37	62	87	1.7071
13	38	63	88	6
14	39	64	89	6
15	40	65	90	6
16	41	66	91	3
17	42	67	92	3
18	43	68	93	3
19	44	69	94	1
20	45	70	95	1
21	46	71	96	1
22	47	72	97	2.3660
23	48	73	98	2.3660
24	49	74	99	2.3660
25	50	75	100	0

Table 6.2: An energy function for the 100 states for the 4-voter automaton. $V(x) = 0.5(x^T x + \sqrt{x^T x})$.

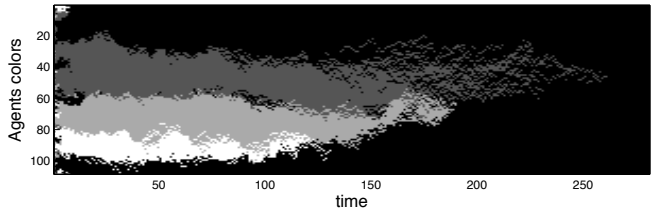


Figure 6.15: The quaternary PAGA running on a circle of 108 agents with votes: 33 blacks, 30 dark grays, 25 light grays and 20 whites. The successive state color configurations are shown with time increasing towards the right ($2 * 108 = 216$ iterations per column).

- *Nodes with state coordinate G do not wake up.* It is easy to see that the algorithm then still converges. The algorithm is especially slow when the majority is not clear. In that case, many nodes have a state at G, and it is thus worth making them silent. By reducing interference, the remaining nodes can decide to wake up more often.

- *In an RGG or a grid, add directional bits.* Each node adds two directional bits to its memory. The two bits indicate "up", "down", "left" or "right". Whenever a node wakes up, it chooses a neighbor in the direction of its directional bit. The two nodes update their states and *swap* their direction bits as well. Then, the two nodes check if they do have a neighbor in the direction proposed by their new directional bit. If they do not, they update their directional bit at random, proportionally to the number of neighbors they have in each direction.

  This idea works very well on simulation, but there is no mathematical evidence for it. The main idea is to give directionality to the states, in order to make the information flow faster in the network. If node 1 (with state q) wakes up and calls node 2, then node 2 updates to a state closer q. States might have even swapped, and node 2 might have updated to state q. Suppose that this is the case and that node 2 now wakes up. It would be a waste of energy to call node 1 and give state q back! This event is prevented by directional bits, since node 2 would then call a node in the opposite direction of node 1. With directional bits, state q can travel in straight lines, bouncing on the edges of the network, and not in a symmetrical random walk style anymore.

- *directional bits in presence of network bottleneck.* If the network has a known bottleneck, then modify the directional bits of the nodes in front of the bottleneck towards the nodes in the bottleneck. This increases the information flow through the bottleneck and the algorithm converges faster.

Conclusion

In this chapter we considered the interval consensus problem, which we obtained by relaxing the average consensus problem so that the nodes only need

to know the interval that contains the average, and not the exact value of the average. This simplification allowed us to build reliable quantized algorithms, which converge almost surely in finite time in any connected network. In addition, we solved the voting problem, as it is a specific instantiation of the interval consensus problem. In particular, we showed that the voting problem can be solved with 2 bits of memory at each node.

In the second part of this chapter, we focused on the multiple voting problem, where every node votes for one candidate among $|\mathscr{C}| \geqslant 2$ candidates and needs to learn the majority vote of the network. We stated a number of sufficient conditions for an automaton to solve the multiple voting problem. We then designed automata fulfilling these conditions for the cases of $|\mathscr{C}| = 2, 3$ and 4 possible votes. These automata do not have final states, so simple parallel procedures need to be implemented to detect consensus and terminate the algorithm. Many related questions remain open. Do voting automata exist for any $|\mathscr{C}|$? What is the speed of convergence of a voting automaton? How can the distribution f of activated edges be optimized to speed up convergence? Most importantly, what is the minimum number of bits that agents need to store/exchange in order to solve the multiple voting problem for a given $|\mathscr{C}|$?

INTERVAL CONSENSUS AND VOTING

Problems

- **Interval consensus problem**: Nodes in a network hold values, which average is x_{ave}. Given a set of intervals, the nodes need to reach a distributed consensus on the interval which contains the average x_{ave}, using a finite number of states only. The **binary voting problem** is a particular instantiation of the interval consensus problem, where the values are either 0 or 1, and where the interval set is $\{[0, 0.5], (0.5, 1]\}$. If the average of the values are in $[0, 0.5]$, then there are more 0's than 1's. Otherwise there are more 1's than 0's.

- **Multiple voting problem** : Nodes in a network hold elements (votes) chosen in a finite set $\mathscr{C}$. Using a finite number of states, nodes need to reach a distributed consensus on the majority vote in the network. The **binary voting problem** is the multiple voting problem when $|\mathscr{C}| = 2$.

  The multiple voting problem can be seen as an extension of the interval consensus problem to higher dimensions. Each element of $\mathscr{C}$ can be mapped to a vertex of the regular simplex of dimension $|\mathscr{C}| - 1$. The simplex is split in $|\mathscr{C}|$ zones: for each vertex O of the simplex, define a zone by the set of points that are closer to O. The nodes need to reach an agreement on the zone that contains the center of mass of the votes.

Solutions

- **Pairwise Asynchronous Graph Automata** (PAGA): Let Q be a set of states. Q is partitioned in as many subsets as intervals for the interval consensus problem and in $|\mathscr{C}|$ subsets for the multiple voting problem. At every iteration, a random edge (i, j) of the network is chosen. Node i and j exchange their states q_i and q_j and update them according to an automaton Δ:

$$q_i(t) = \Delta\left(q_i(t-1), q_j(t-1)\right), \tag{6.7}$$
$$q_j(t) = \Delta\left(q_j(t-1), q_i(t-1)\right). \tag{6.8}$$

- We design states Q and automata Δ that solve the **interval consensus** problem almost surely (a.s.) in finite time in any connected network, when the boundaries of the intervals are in $\delta\mathbb{Z}$ for some value δ.

- We give sufficient conditions on Q and Δ for solving the **multiple voting** problem a.s. in finite time in any connected network.

- **Binary voter automata** $|\mathscr{C}| = 2$: We design a set of automata using $2k$ states, for $k \in \mathbb{N}/\{0, 1\}$. As a consequence, 4 states are sufficient to solve the binary voting problem.

- **Ternary voter automata** $|\mathscr{C}| = 3$: We design 3 automata using, respectively, 15, 33 and 39 states.

- **Quaternary voter automaton** $|\mathscr{C}| = 4$: We design one automaton using 100 states.

Chapter 7

Conclusion

Throughout this dissertation, we investigated distributed algorithms that allow nodes in a network to converge to a consensus. Each node initially suggests some value, and by exchanging local messages, nodes manage to reach a compromise on the average or the majority of the suggested values.

We especially focused on algorithms that are efficient on random geometric graphs, because they model well wireless sensor networks. All the algorithms that we studied were described in detail in Chapter 2. We started by describing the classical pairwise gossip algorithm, and we further mentioned neighborhood gossip and broadcast gossip. These three algorithms perform updates with local information only, hence they converge slowly on topologies that do not have long-distance connections, which is the case of random geometric graphs. We then described geographic gossip, path averaging and one-way path averaging, which create artificial long-distance connections by routing information through the network.

In Chapter 5, we showed that algorithms involving routing converge with smaller communication cost than algorithms that are constrained to use local information only. In order to prove the convergence and to compute the performances of the mentioned algorithms, we used the results stated in preliminary Chapters 3 and 4. Chapter 3 gave general tools to prove the convergence and to estimate the performances of distributed average consensus algorithms. Chapter 4 studied in detail the second largest eigenvalue in magnitude of doubly stochastic matrices, which is needed to derive the results of Chapter 5. In particular we wrote an in-depth analysis of the cases where the Markov chains evolve on lattices and balanced trees. Lattices are useful to study random geometric graphs, and balanced trees are a good toy-example to understand the impact of bottlenecks on the convergence of our algorithms.

Then, in Chapter 5, we proved the convergence and computed the performance of every mentioned algorithm, except for one-way path averaging, whose analysis is still at an early stage. All the results were conveniently gathered in Table 5.5.4. It is particularly interesting that, on random geometric graphs, the total message cost gains a $\sqrt{n/\log n}$ factor from pairwise gossip to geographic gossip, and again a $\sqrt{n/\log n}$ factor from geographic gossip to path averaging. We proved that path averaging requires $O(n \log n)$ messages to converge to the average at a precision of $1/n$ with high probability. The $\log n$ factor is due to the exponential approach to consensus of our algorithm. The n factor can-

not be further reduced because at least n messages need to be sent to average the values held by n nodes. Path averaging is thus a gossip algorithm that is order-optimal in message cost.

Finally, in Chapter 6, we solved the binary voting problem, by noticing that it is closely related to the averaging problem. Inspired by the classic averaging problem, we constructed the interval consensus problem. Instead of computing the exact average, an interval consensus algorithm distributively finds out which interval contains the average among a set of predefined intervals. The advantage of this rephrased problem is that the set of intervals is finite, whereas the range of potential averages increases with the network size n. For example, if the initial signal has only 0 and 1, there are $n+1$ potential averages: $\{k/n\}_{0\leqslant k\leqslant n}$. Hence the classic averaging problem requires increasing memory at each node, whereas we were able to solve the interval consensus problem with a finite number of states only.

As a by-product, we showed that the binary voting problem can be solved with 2 bits of memory at every node. Furthermore, we proposed algorithms that distributively compute majority among 2, 3 or 4 votes. We showed that the ternary voting problem can be solved with 4 bits per node, and that the quaternary voting problem can be solved with 7 bits per node.

7.1 Future Research

There are still many interesting problems related to distributed computing in wireless sensor networks. In this section, we give three directions that could be natural extensions of this work.

7.1.1 Large Deviations

Throughout this work, we ran our simulations based on the observation that the error $\|\boldsymbol{\epsilon}(t)\|$ has a very low variance. In [31], Fagnani showed this particular point for small times when $\{\mathbf{W}(t)\}_{t\geqslant 1}$ are i.i.d. stochastic matrices. It would be interesting to know precisely how long the concentration result holds on graphs like random geometric graphs. It would also be interesting to extend the result to stationary and ergodic matrices. This would validate the performance observed on simulations, in cases where theoretical analysis is arduous. Finally it would complete the Master Thesis work on the Lyapunov exponent by Patrick Denantes, who sadly passed away this summer.

7.1.2 Voting Algorithms

We built interval consensus algorithms and voting algorithms that converge correctly a.s. in finite time in any connected network. The next step is to compute their speed of convergence. Draief and Vojnović [28] recently computed the speed of convergence of the 4 state automaton for binary voting using mean field techniques. It would be interesting to extend their analysis to binary voter automata using $2k$ states with $k \geqslant 2$, and quantify how the speed of convergence depends on k.

Developing techniques to compute the speed of convergence of the ternary automata and of the quaternary automaton is also an interesting direction for

future work. Finally, a fundamental question remains: What is the minimum number of states needed to solve the multiple voting problem for a given number of votes $|\mathscr{C}|$?

7.1.3 Mobility and Virtual Mobility

Our results on path averaging showed that routing information accelerates distributed average consensus algorithms. Routing allows nodes that are far from each other to exchange their states, which helps the diffusion of information. There are at least two possible ways to extend this idea: first by having the nodes themselves move (mobility), or by introducing directional bits as explained on page 224 (virtual mobility).

Mobility

Studying realistic mobile networks is difficult because the random processes are time correlated and not i.i.d. anymore. An ideal mobility model for random geometric graphs allowing to recover i.i.d. features has been studied in [72]. In this model, every node decides with equal probability whether it moves horizontally or vertically for the whole duration of the algorithm. Then, at each iteration, the network performs a pairwise gossip round and each node draws a random i.i.d. horizontal (or vertical) coordinate, which will be its position in the following iteration. The authors show that pairwise gossip runs in $O(n \log n)$ in such a mobile network, which thus offers the same diffusivity as the complete graph.

Intuitively, mobility helps the diffusion of information since nodes that are initially far away (and would stay far away in a fixed network) may eventually become neighbors and be able to exchange their measurements via a direct link. In order to benefit from the mobility of the nodes, the algorithm should have infrequent averaging rounds. This way the nodes have enough time to change their position between rounds and the information is diffused with no energy being spent for communication. The result is a highly inexpensive algorithm with a possibly large delay. This could be applicable, for example, in environmental monitoring applications where wireless sensors are attached to humans or animals. On the other hand, if the agents are robots, typically more energy is spent for moving than for communicating. In such a setting, where communication energy consumption is not an issue, having fast convergence can be attractive. Thus, studying the effect of slow velocity on pairwise gossip can have some interesting applications.

Such a study on slow velocity could also have some profound implications in physics. It is well known that turbulence helps heat transfer. There is a heuristic model which states that increasing the particle velocity helps the heat transfer as long as the velocity is smaller than a value, which depends on the size of the particles' heat interaction range and on the size of the system. Increasing particle velocity beyond that point does not improve heat diffusion. In other words, when stirring a cup of coffee, there is a limit to the speed of the spoon after which the coffee temperature is not going to homogenize any faster.

Interestingly, there is a also speed after which pairwise gossip does not converge any faster, as shown on Fig. 7.2(a). We have been collaborating with

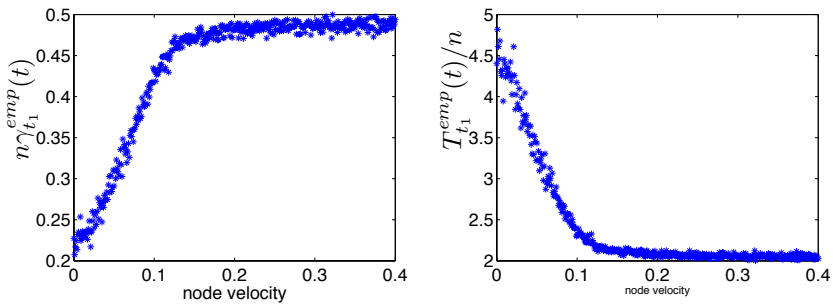
Figure 7.1: Leonard Da Vinci was fascinated by turbulence. We heard that he had already noticed the velocity threshold phenomenon for heat diffusion.

the heat and mass transfer laboratory at EPFL on the connection between heat diffusion in the presence of convection and pairwise gossip in mobile networks. We have learned that they have no theoretical explanation of the heuristic model that we just described. They are therefore very interested in our pairwise gossip "model", which can be seen as an asynchronous discretization of the heat diffusion equation, and which could perhaps solve an open question (see Fig. 7.1).

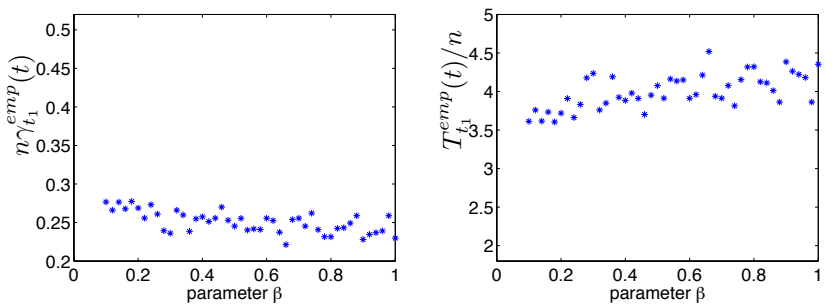
We also learned that, according to another heuristic model, heat diffuses similarly in a system of n particles moving at velocity v and in a system of $(1-\beta)n$ motionless particles and βn particles moving at velocity v/β, for values of β in $(0,1]$. This statement has probably been simplified for our understanding, and we experimented it on simulations. Fig. 7.2(b) shows that similar results hold in our mobility setting, where nodes moves in random independent directions on the torus at a constant speeds. More precisely, on our simultion, pairwise gossip converges approximatively as fast if n nodes move at velocity v as if $(1-\beta)n$ nodes are motionless and βn nodes move at velocity v/β, for values of β in $(0,1]$. If this were true, it could have interesting applications in the management of mobility in wireless sensor networks. We thus believe that this partial motion problem is an interesting direction for future research.

Virtual Mobility

Directional bits are designed to accelerate algorithms based on pairwise interactions, as pairwise gossip or voting automata. Each node not only memorizes an estimate (or state), but also 2 directional bits, which indicate in which direction they should call a neighbor whenever they wake up. When two nodes communicate and exchange their estimates, they also exchange their directional bits. Then, any node that received a new directional bit checks if it has neighbor in the corresponding direction. If it does not have any such neighbor, then it draws random new directional bits, proportionally to the number of neighbors



(a) Velocity threshold



(b) Partial motion and varying velocity

Figure 7.2: 7.2(a): Empirical decay rate and empirical consensus time of pairwise gossip as a function of the node velocity v. Each node moves on the torus in a random direction chosen independently from the directions of the other nodes. At every iteration, each node moves by a distance v/n. The simulation was run with 500 nodes and connection radius $\sqrt{2\log n/n}$. 7.2(b): Empirical decay rate and empirical consensus time of pairwise gossip as a function of β. βn mobile nodes move at velocity v/β and $(1-\beta)$ nodes are motionless.

it has in each direction.

If we were to give a color to each node, and if nodes were to pairwise exchange their colors according to their directional bits, then a given color would travel in the network with directionality. Without the directional bits, colors would move as symmetric random walkers. As directionality is given to information, and not to the agents themselves, we say that direction bits give virtual mobility to the network.

We observed that using directional bits accelerates pairwise algorithms. The effect on pairwise gossip is modest, but it becomes significant when nodes are updated with coefficients $\alpha > 0.5$ (as in Eq. (3.79) and (3.80) in Section 3.2.3). Directional bits can be an inexpensive solution to increase the performance of pairwise gossip, in cases where geographic routing is not likely to succeed (e.g. in the presence of holes and/or unreliable links). As far as voting algorithms are concerned, directional bits improves drastically their performance on simulations, and it would be interesting to prove this analytically.

Appendix A

Random geometric graphs

This Appendix lists and proves all the necessary properties concerning random geometric graphs. Recall the definition of a random geometric graph:

Definition A.0.1 (Random geometric graph $G(n, r)$). A random geometric graph $G(n, r)$ is a random graph, where n nodes are drawn independently and uniformly in a convex unit area. Two nodes are connected by an edge if and only if their distance is smaller than radius r.

Unless stated otherwise, the default unit area in this thesis is the unit square. See Fig. A.1 for an example of random geometric graph.

A.1 Connectivity of Random Geometric Graphs

Remember that we say that a graph is connected if and only if one can find a path of edges between each pair of nodes. Obviously, for random geometric graphs, the larger the connection radius r is, the larger the probability of connecting the nodes is. The goal of this section is to determine some minimum connection radii which guarantee connectivity with high probability (w.h.p.) on the unit disk and on the unit square.

It is important to note that the minimum connection radius $r(n)$ to connect an RGG depends on the shape of the area the graph is embedded in. Indeed nodes that are located close to the edges have truncated connectivity regions, and they are thus more difficult to connect to the other nodes. Since the shape of the area determines the edges of the network, connectivity needs to be treated differently on the unit disk and on the unit square. Note that the easiest area to connect is the torus, since the torus does not have any edge, and every node has a connectivity area equal to $\pi r(n)^2$.

Figure A.2 shows the connectivity of random geometric graphs on the unit disk and on the unit square, when $r(n) = \sqrt{c \log n / n}$ for varying parameters c. We can observe that random geometric graphs on the unit disk connect with smaller radii than on the unit square. This was predictable since the unit disk has "more edges " than the unit square: the perimeter of the unit disk is π whereas the perimeter of the unit square is $4 > \pi$. Furthermore, a node at the corner of the square has a connectivity region truncated by $1/4$, which is considerable. Corner nodes are therefore difficult to connect to the rest of the

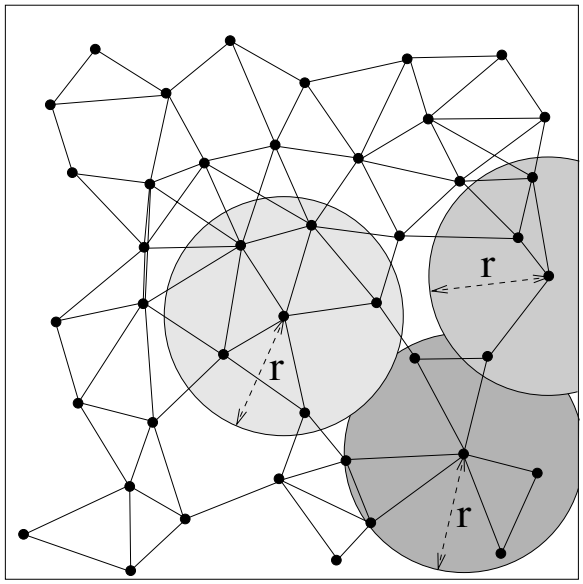


Figure A.1: A realization of random geometric graph.

network. Such a drastic truncation of the connectivity regions does not occur on the disk.

A.1.1 Connectivity of an RGG on the unit disk

The connectivity of a random geometric graph on the unit disk is derived in [42], a famous paper by Gupta and Kumar. The main theorem in [42] gives a necessary and sufficient condition for connectivity, which is based on percolation theory:

Theorem A.1.1. If n nodes are uniformly i.i.d. placed in a disc of unit area in $\mathbb{R}^2$ and if the connection radius is equal to:

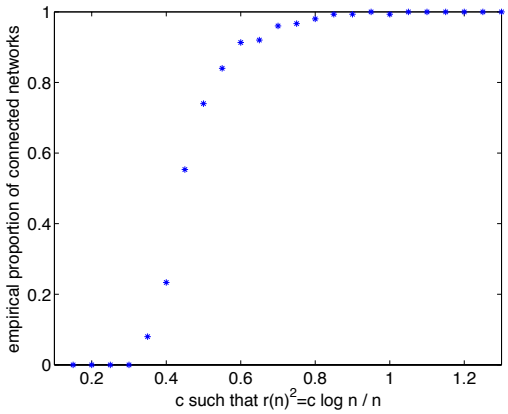
$$r(n) = \sqrt{\frac{\log n + g(n)}{\pi n}},$$

then the resulting network is asymptotically connected with probability one if and only if $g(n) \to +\infty$.

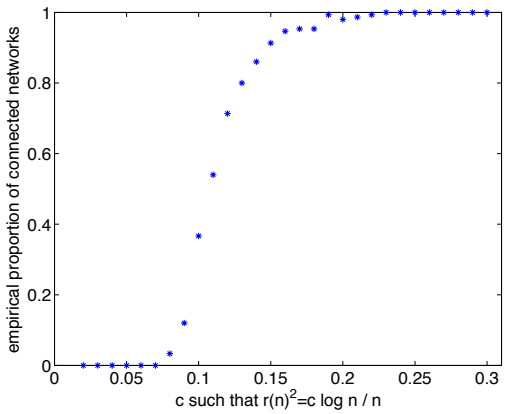
As a consequence, taking $g(n) = \epsilon \log n$ shows that on the unit *disk*, an RGG is connected with high probability if $r(n) = \sqrt{c \log n / n}$ and $c > 1/\pi \simeq 0.32$.

A.1.2 Connectivity of an RGG on the unit square

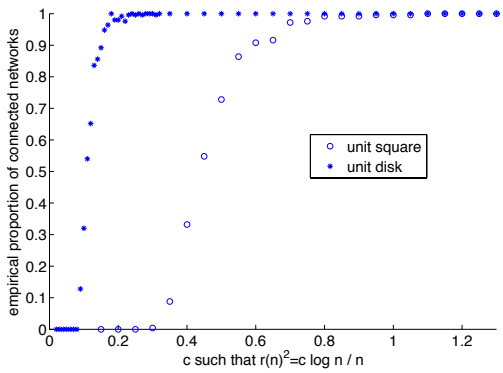
On the unit square, my experience on simulations taught me that $c \geqslant 1$ is large enough to easily connect an RGG with 100 nodes and more (see Fig. A.2), but that coefficients c around 0.3 do not connect an RGG. The following theorem ensures the connectivity of random geometric graphs on the unit square.



(a) unit square



(b) unit disk



(c) unit square and unit disk

Figure A.2: Empirical distribution of the connectivity of an RGG on the unit square and on the unit disk as a function of the coefficient c. The presence of corners in the unit square makes the connection more difficult and demands larger connection radius to connect the network than in the unit disk. In each simulation, 1500 nodes were drawn uniformly at random. Each point is an average over 150 independent random networks in (a) and (b), and over 250 independent random networks in (c).

n \ c	$1/\pi$	1	4	6	25	40
50	0,158	0,280	0,559	0,685	1,399	1,769
100	0,121	0,215	0,429	0,526	1,073	1,357
200	0,092	0,163	0,326	0,399	0,814	1,029
500	0,063	0,111	0,223	0,273	0,557	0,705
1000	0,047	0,083	0,166	0,204	0,416	0,526
2000	0,035	0,062	0,123	0,151	0,308	0,390

Table A.1: $r(n) = \sqrt{c \log n / n}$ for some values of c and of n

Theorem A.1.2 (Connectivity of an RGG on the unit square). A random geometric graph $G(n, r)$, where radius r scales as

$$r(n) = \sqrt{\frac{c \log n}{n}},$$

is connected with high probability if $c > 5$, and with very high probability if $c > 10$

Comments: According to simulations, the true limiting parameter c should be much smaller than 5. However we will content ourselves of this crude bound, since random graph theory is not the main concern of our work. Getting the best bound would require a much finer analysis. It is already quite satisfactory to know that $r(n)$ should scale as $\Omega(\log n/n)$ to connect an RGG, which is the same scaling law as for the unit disk.

Nevertheless, as shown in Table A.1, for networks that have less than 100 nodes, i.e. for the networks that we are able to build in practice for the moment, taking c larger than 6 does not have much meaning, since this would impose connection radii larger than 0.5, which is already half the network size! In order to check that our algorithms make some practical sense, we have tested them with coefficients c smaller than their theoretical bounds in our proofs.

In order to show Theorem A.1.2, we will first state the empty-box lemma. This simple but important lemma will be useful in the whole appendix. It introduces the division of the unit square in square boxes, which is a classical and easy trick to write proofs on random geometric graphs.

Lemma A.1.1 (Empty box). Draw n points (nodes) independently and uniformly at random in a unit square, and for a given $\alpha > 0$, divide the unit square in boxes of size:

$$\sqrt{\frac{\alpha \log n}{n}} \times \sqrt{\frac{\alpha \log n}{n}}.$$

Then, the probability that one specific box b is empty scales as:

$$\mathbb{P}[\text{Box } b \text{ is empty}] = O\left(n^{-\alpha}\right),$$

and the probability that there is at least one empty square scales as:

$$\mathbb{P}[\exists \text{ an empty box}] = O\left(n^{1-\alpha}\right).$$

In other words, if $\alpha > 1$, then there is at least one node in each square with high probability. If $\alpha > 2$, then there is at least one node in each square with *very* high probability.

Proof: Since nodes are drawn uniformly i.i.d., the probability that one specific box b is empty (event $E(b)$) is equal to:

$$\begin{aligned}\mathbb{P}[E(b)] &= \left(1 - \frac{\alpha \log n}{n}\right)^n \\ &\leqslant \exp(-\alpha \log n) \\ &= n^{-\alpha}.\end{aligned}$$

By the union bound, the probability that there is at least one empty box (event $\bigcup_{\text{boxes } b} E(b)$) scales as:

$$\begin{aligned}\mathbb{P}[\exists \text{ an empty box}] &= \mathbb{P}\left[\bigcup_{\text{boxes } b} E(b)\right] \\ &\leqslant \sum_{\text{boxes } b} \mathbb{P}[E(b)] \\ &\leqslant \sum_{\text{boxes } b} n^{-\alpha} \\ &= O\left(\frac{n}{\alpha \log n} n^{-\alpha}\right) \\ &= O\left(n^{1-\alpha}\right).\end{aligned}$$

■

Proof:[Theorem A.1.2] Divide the unit square in boxes of size $\alpha \log n/n$, just as in the "empty box" Lemma. If $r(n)$ is larger than $\sqrt{5}$ times the width of the boxes ($\sqrt{\alpha \log n/n}$), then any node in any box can communicate with any other node in any of its 4 adjacent boxes (simple geometric argument). According to the "empty box" Lemma, boxes all have at least one node with high probability if $\alpha > 1$. Therefore $c > 5$ implies that all the boxes can be connected w.h.p., and that the RGG is thus connected w.h.p. ($\alpha > 2$ imposes $c > 10$). ■

A.2 Routing in Random Geometric Graphs

A.2.1 The Routing Algorithms

There are many routing algorithms that were developed for random geometric graphs. In this thesis we use only three of them: greedy routing, random greedy routing and $(\leftrightarrow, \updownarrow)$-box routing. Greedy routing and random greedy routing have the ambition of being implementable on a sensor network, where nodes know their locations either by GPS or by any localization algorithm. $(\leftrightarrow, \updownarrow)$-box routing is a routing protocol which is not really adapted to a practical implementation, since it further requires the knowledge of virtual box positions as in the empty box lemma (Lemma A.1.1). In addition, it requires a square network area, which is rarely the case in practice. However it is much easier to analyze than both greedy routing and random greedy routing. We choose

to study $(\leftrightarrow, \updownarrow)$-box routing because it resembles random greedy routing, and we believe that our algorithms should not behave very differently when using $(\leftrightarrow, \updownarrow)$-box routing or random greedy routing.

The three routing algorithms have the same inputs: a sender node s and a target position T, which is chosen uniformly at random in the network area. There is no need that a node is located at position T.

1. **Greedy routing**: the message is greedily forwarded to the neighbor that is closest to the target T. The route stops when a node does not have a neighbor which is closer to the target T. See Alg 2.4 for a precise description of greedy routing.

2. **Random greedy routing**: the message is greedily forwarded to a random neighbor that is closer to the target T. The route stops when a node does not have a neighbor which is closer to the target T. See Alg 2.6 for a precise description of random greedy routing.

3. $(\leftrightarrow, \updownarrow)$-**box routing** (Alg. A.2): it is a variation of $(\leftrightarrow, \updownarrow)$-routing, which is a simple routing algorithm on the lattice, described in Alg. A.1, in which a message is routed to its destination horizontally first, then vertically. To run $(\leftrightarrow, \updownarrow)$-box routing, the RGG should be divided in boxes. The goal of $(\leftrightarrow, \updownarrow)$-box routing is to send the message m to any node in the box B which contains the target T. $(\leftrightarrow, \updownarrow)$-box routing proceeds by greedily forwarding the message to a randomly chosen node in a neighboring box, chosen such that the resulting sequence of boxes forms the $(\leftrightarrow, \updownarrow)$-route from box $b(s)$ to the destination box B in the overlay box-lattice.

   It is important to notice that here, $(\leftrightarrow, \updownarrow)$-box routing reaches a *random* node in the destination box.

Algorithm A.1: $(\leftrightarrow, \updownarrow)$-routing on a lattice.

Assume that a node s has a message m to transmit to a destination node d. For any node i, let (x_i, y_i) be the coordinates of node i and let $i_{\leftarrow}$, $i_{\rightarrow}$, $i_{\downarrow}$, and $i_{\uparrow}$ be its left, right, top and bottom neighbors in the lattice.

```
Initialize i ← s.
while |x_i − x_d| > 0 do
   if x_i < x_d then
      Node i sends message m to node i_→.
   else
      Node i sends message m to node i_←.
   end if
end while
while |y_i − y_d| > 0 do
   if y_i < y_d then
      Node i sends message m to node i_↑.
   else
      Node i sends message m to node i_↓.
   end if
end while
```

Algorithm A.2: $(\leftrightarrow, \updownarrow)$-box routing on a RGG.

Assume that a node s has a message m to transmit to a destination *box* B. The boxes form a lattice, and each box has integer coordinates. In particular, box B has coordinates (x_B, y_B). For any node i, let (x_i, y_i) be the coordinates of the box $b(i)$ which contains node i, and let $b(i)_{\leftarrow}$, $b(i)_{\rightarrow}$, $b(i)_{\downarrow}$, and $b(i)_{\uparrow}$ be its left, right, top and bottom neighboring boxes in the overlay box-lattice.

```
Initialize i ← s.
while |x_i − x_B| > 0 do
   if x_i < x_B then
      Node i sends m to a node chosen uniformly at random in box b(i)_→.
   else
      Node i sends m to node chosen uniformly at random in box b(i)_←.
   end if
end while
while |y_i − y_B| > 0 do
   if y_i < y_B then
      Node i sends m to node chosen uniformly at random in box b(i)_↑.
   else
      Node i sends m to node chosen uniformly at random in box b(i)_↓.
   end if
end while
```

A.2.2 Performance

We say that a routing is successful is the message reaches its destination. In box-routing, this means that the message should be able to reach any node in the destination box (the box where the target T is). In greedy routing and in random greedy routing, this means that the message should reach the closest node to the target T.

Performance of $(\leftrightarrow, \updownarrow)$-box routing

Theorem A.2.1 (Performance of $(\leftrightarrow, \updownarrow)$-box routing). Assume that the boxes have size $\sqrt{\alpha \log n/n} \times \sqrt{\alpha \log n/n}$. Then $(\leftrightarrow, \updownarrow)$-box routing is successful if

$$r(n) \geqslant \sqrt{\frac{5\alpha \log n}{n}}.$$

Proof: A simple geometric argument explains this theorem: the largest distance between two neighboring unit boxes is $\sqrt{5}$ by the Pythagorean theorem. The theorem is obtained by scaling this observation to boxes of size $\sqrt{\alpha \log n/n} \times \sqrt{\alpha \log n/n}$, and by noticing that any node can communicate with any other node in any of its neighboring boxes if the radius is larger than the largest distance between two neighboring boxes. ■

Corollary A.2.1. If $r(n) > \sqrt{5 \log n/n}$, then there is a box size such that $(\leftrightarrow, \updownarrow)$-box routing is successful with high probability.

Proof: Divide the unit square in boxes of size $r(n)/\sqrt{5} \times r(n)/\sqrt{5}$. Then by the empty box lemma, every box contains at least one node with high probability,

and every node can communicate with any other node in any of its neighboring boxes. This implies that $(\leftrightarrow, \updownarrow)$-box routing is successful with high probability. ■

Performance of greedy routing and random greedy routing on the torus

It has often been claimed in the literature that the success $(\leftrightarrow, \updownarrow)$-box-routing "clearly" implies the success of greedy routing. This claim is perhaps true, but it is obscure to us, and we have decided to find a bound on the minimum radius which guarantees the success of greedy routing and random greedy routing.

Theorem* A.2.1 (Performance of (random) greedy routing on the torus). On the torus, greedy routing and random greedy routing reach the node that is the closest to the target w.h.p. if the connection radius $r(n)$ is larger than:

$$r(n) > \sqrt{\frac{4 \log n}{n}}.$$

In order to prove Theorem A.2.1, we need the following lemma:

Lemma* A.2.1 (Non-empty disk). Draw n nodes uniformly i.i.d. at random in the unit square embedded on the torus. Then, all the disks of radius γ in the toroidal unit square such that

$$\gamma > \sqrt{\frac{\log n}{n}},$$

contain at least one node with high probability.
In addition, if radius $\gamma > \sqrt{2 \log n / n}$, then all the disks contain at least one node with *very* high probability.

Proof: The proof of this theorem is based on the "empty box " Lemma and on some geometric considerations. Divide the unit square in boxes of width $\sqrt{\alpha \log n / n}$ with $\alpha > 1$ for the "w.h.p." statement, and with $\alpha > 2$ for the "w.v.h.p." statement. This defines one lattice partition L_1 on the unit square. It is convenient to resize the unit square such that each box has a unit area, i.e. to multiply the square area by $n/(\alpha \log n)$. Define now lattice partitions $L_2, L_3, \ldots, L_9$ as shifts of L_1: for any $k = 2, \ldots, 9$,

$$L_k = L_1 + \left\lfloor \frac{k-1}{3} \right\rfloor \frac{\mathbf{e_1}}{3} + \mathrm{mod}\,(k-1, 3)\, \frac{\mathbf{e_2}}{3},$$

where $\mathbf{e_1}$ and $\mathbf{e_2}$ are the horizontal and vertical unit vectors (see Fig. A.3 to picture the origins of all 9 lattices). In other words, the lattice L_1 is shifted by thirds in both directions to generate 8 other lattices. Note that the union of the 9 lattices divides each box of each lattice in 9 sub-boxes, one of these 9 sub-boxes, called the central sub-box, being surrounded by the 8 other sub-boxes.

For any lattice L_k, the probability $p(n)$ that L_k has an empty box is asymptotically equal to 0 (see the "empty box" Lemma A.1.1). Therefore, by a union bound on the 9 lattices, the probability that any of the 9 lattices has an empty box is smaller than $9p(n)$, which also vanishes to 0 when n goes to infinity. In

other words,

$$\mathbb{P}\left[\forall \text{ box } B \in \{L_k\}_{k=1,\ldots,9}, B \neq \emptyset\right] \underset{n\to\infty}{\longrightarrow} 1, \tag{A.1}$$

where $A = \emptyset$ here means that there is no point that fell in A.

Now take a disk $\mathcal{C}$ centered at a point x with radius $\gamma > 1$, and consider the sub-box b that contains point x (if there are several such sub-boxes, choose any). In Fig. A.3, this sub-box is painted in dark gray. The key point of this proof is to show that the box B, whose central sub-box is b, is included in $\mathcal{C}$ (B is painted in light gray in Fig. A.3).

Fig. A.3 shows two examples of disks. Disk 1 shows a classical case, where x is inside a sub-box. Disk 2 on the contrary shows the worst case, which is when x is at the corner of a sub-box. The largest distance from x to the edges of B is then equal to 2 sub-box diagonals, which is equal to $2 * \sqrt{2}/3 < 0.95 < 1$. Therefore, even in the worst case, box B is inside the disk $\mathcal{C}$ centered at x of radius $\gamma > 1$.

If all boxes of all 9 lattices contain at least one point, then all disks of radius larger than 1 contain at least a point as well. Therefore,

$$\mathbb{P}\left[\forall \text{ circle } \mathcal{C} : \gamma > 1, \mathcal{C} \neq \emptyset\right] \geqslant \mathbb{P}\left[\forall \text{ box } B \in \{L_k\}_{k=1,\ldots,9}, B \neq \emptyset\right],$$

where the right hand side converges to 1 when n goes to infinity according to Eq. (A.1). Therefore,

$$\mathbb{P}\left[\forall \text{ circle } \mathcal{C} : \gamma > 1, \mathcal{C} \neq \emptyset\right] \underset{n\to\infty}{\longrightarrow} 1,$$

which concludes the proof. ■

Proof:[Theorem A.2.1] The same proof works for greedy routing and for random greedy routing. The idea of this proof is to show that if a route fails to reach the closest node to the target, then there is a "hole" in the network. Based on the fact that the existence probability of this "hole" vanishes to 0 as n grows, we can show that the route failure probability goes to 0 as well. There are no edge effects to consider in this proof, since we assume that the RGG is embedded on the torus.

If (random) greedy routing fails, then, as shown on Fig. A.4 and as explained in this paragraph, there is a region without any node in the network. Indeed, assume that the route fails to reach the closest node to target T. Then the last node i of the route does not have any neighbor closer to the target than he is. In other words, the light gray region defined by the intersection between the disk centered at i of radius $r(n)$ and the disk centered at the target of radius $\|T - i\|$, does not contain any node. As the target gets closer to i, the light gray region shrinks (see Fig. A.4). If the target is at distance $r(n)/2$ from i, then the route cannot fail anymore, since any closer node to the target would be inside the connection disk of node i. Therefore, for any position of the target such that the route can fail to reach the closest node to the target, if the route fails, then the empty light gray region necessarily contains a disk of radius $r(n)/2$, which is shaded in dark gray in Fig. A.4.

Since $r(n) > \sqrt{4 \log n/n}$, the dark gray disk has a radius larger than $\sqrt{\log n/n}$. By the non-empty disk lemma (Th. A.2.1), all such disks are non-empty with high probability. Therefore, any light gray region is non-empty with high probability as well. By consequence, with high probability, no route

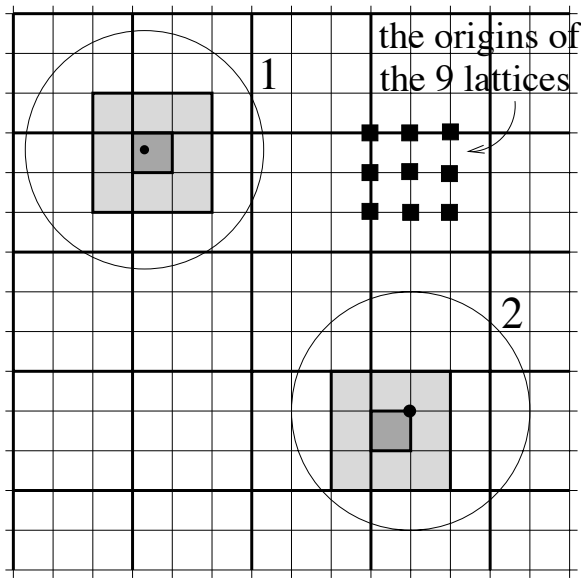


Figure A.3: Figure to support the proof of Lemma A.2.1. The thicker lines show the boxes of lattice L_1. The thiner lines delimitate the sub-boxes. The 9 small black squares show the lattice shifts. Finally, there are two examples of circles of radius 1 (= width of 3 sub-boxes), and of the lattice box they contain. It is easy to see that for any disk of radius 1, there is at least one gray box *inside* the disk.

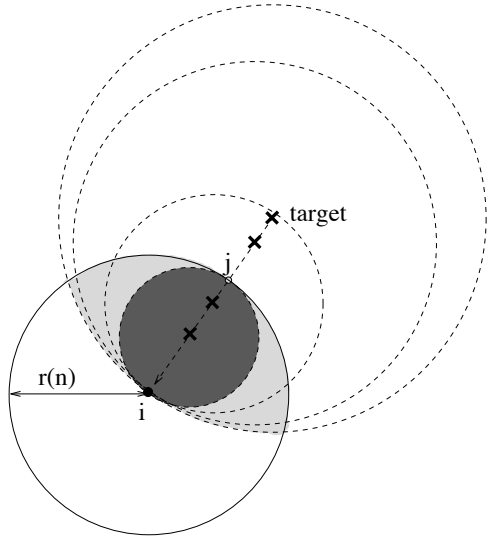


Figure A.4: Node i needs to route a message towards the target. Suppose that i is not the closest node to the target, then the routing fails to reach the closest node if the light/dark gray region is empty. This region is larger than the dark gray disk, which has diameter $r(n)$ and which goes through i and j. j is the point which is on the circle centered at i of radius $r(n)$ and which is aligned with i and the target. If the network domain is convex, then j and the center of the dark gray disk are inside the network domain. This remark is useful for the proof of Lemma A.2.2 on the unit square.

can fail.

Note that if $r(n) > \sqrt{8 \log n / n}$, then the lemma holds with very high probability. ■

Performance of greedy routing and random greedy routing on the unit square

The following theorem is a refinement of Theorem A.2.1 from the torus to the unit square, i.e. it takes into account the edge effects of the unit square.

Theorem* A.2.2 (Performance of (random) greedy routing on the unit square). On the unit square, greedy routing reaches the node that is the closest from the target w.h.p. if the connection radius $r(n)$ is larger than:

$$r(n) > \sqrt{\frac{52 \log n}{9\ n}}.$$

Comments: $52/9 \simeq 5.8$ is larger than 5, the constant which was found for the success of $(\leftrightarrow, \updownarrow)$-box-routing, and for the connectivity of the random geometric graph.

Proof: The proof is essentially the same as the previous one. The only difference is that the non-empty disk theorem has to be extended to the case

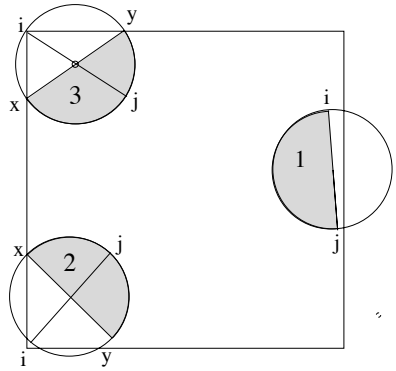


Figure A.5: Disks of radius $r(n)/2$ that link diametrically opposed nodes i and j in the unit square. Any such disk has at least a semi-disk inside the unit square.

where the disks overlap the edges of the unit square. For the purpose of Theorem A.2.2, note that the disks of interest (the dark gray disks of radius $r(n)/2$ defined in the previous proof) are not arbitrarily close to the edges. We will show that this guarantees a substantial overlap of the dark gray disks with the unit square and implies the existence of a node in these disks with high probability.

More precisely, the centers of the disks are inside the unit square (see the caption of Fig. A.4), and they cannot be closer than $r(n)/2$ to the corners of the unit square. Indeed each disk is at least half-contained in the unit square, which implies that the center of each disk cannot be closer than $r(n)/2$ to a corner. This half-disk constraint comes from the fact that in the previous proof, the dark-gray disk goes through two points i and j inside the unit square that are diametrically opposed (see Fig. A.4). Suppose that the disk does intersect the edges of the square. Then, either the disk intersects only 1 edge, in which case the semi-disk defined by i and j is inside the square (case 1 in Fig. A.5), or it intersects two different edges in points x and y. In that case, the semi-disk passing by x is inside the unit square (case 2 in Fig. A.5). In the extreme case where i is a corner of the square (case 3 in Fig. A.5), x and y are diametrically opposed, and they define a semi-disk inside the square. In this extreme case, the center of the disk is exactly at distance $r(n)/2$ from a corner of the square.

Note that the fact that the center of the dark-gray disks are not closer than $r(n)/2$ to the corners of the square could have been proved in a much simpler way: take any two points i and j in the square, separated by distance $r(n)$, then, by Thales theorem[1], their middle is necessarily further than $r(n)/2$ from the corners. However, it is interesting to see that there is always half a disk inside the square to understand why the disks of interest are non empty with high probability.

Please refer to the proof of Lemma. A.2.1 for the definitions and notations used in this proof. In particular, rescale the unit square to an area $n/\alpha \log n$, with $\alpha > 1$, such that all the unit area boxes of lattices $\{L_k\}_{k=1,\ldots,9}$ contain at least one node with high probability. Remember that each box is divided in 9

[1] An inscribed angle in a semicircle is a right angle.

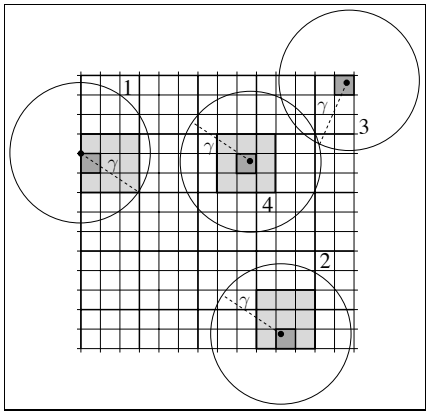


Figure A.6: Construction of a box included in a disk, when the disk center is close to the edge of the square. This figure can be compared to Fig. A.3, where the disk center is in a sub-box that is not on the edges of the square. In this case, the light gray box is constructed by putting the dark-gray sub-box at its *center* (Case 4 of this figure). When the dark-gray sub-box is on the edge of the square, this construction is not possible anymore, and γ has to be taken larger in order to build the light gray box by putting the dark-gray sub-box on its *side*.

sub-boxes.

The goal is to find a radius γ such that any disk of this radius, which center is not in a corner sub-box, contains at least one box of $\{L_k\}_{k=1,\ldots,9}$. In the proof of Lemma A.2.1, where the disks are entirely included in the unit square, $\gamma = 1$ is sufficient. Here, we show that $\gamma = \sqrt{13}/3 \simeq 1.2$ includes the case of disks that are closer to the edges (see Fig. A.6), although it excludes the case where the center x is in one of the 4 corner sub-boxes.

Take a disk $\mathcal{C}$ centered in x with radius $\gamma = \sqrt{13}/3$, where x is not in one of the 4 corner sub-boxes (Case 3 in Fig. A.6 is not allowed). $\sqrt{13}/3$ is the length of the hypothenuse of the triangle of sides $2/3$ and 1, as shown in Case 1 of Fig. A.6. Since $\gamma > 1$, then any disk, which center is in a sub-box strictly inside the square, contains at least one node with high probability (see proof of Lemma A.2.1, and Case 4 of Fig. A.6). Consider now the case where x is in a sub-box situated on the edge of the square, but not at a corner. Case 1 of Fig. A.6 is the extreme case where x is on the edge of the square, and at the intersection of two sub-boxes. As shown on the figure, $\mathcal{C}$ contains one box (painted in light gray), which is the box that is surrounding the sub-box which contains x (dark-gray sub-box). As illustrated by Case 2, for any center x in any edge sub-box, the disk centered at x of radius γ contains at least one light gray box.

By rescaling back to the unit square, it has just been shown that any disk of radius $\gamma(n) > \sqrt{13\alpha \log n/9n}$ centered anywhere in the unit square except in the 4 corner sub-boxes contains at least one node with high probability. Take now $r(n) = 2\gamma(n) > \sqrt{52\alpha \log n/9n}$. The corner sub-box width is $\sqrt{\alpha \log n/9n}$, and its diagonal ($= \sqrt{2\alpha \log n/9n}$) is then smaller than $r(n)/2 = \gamma = \sqrt{13\alpha \log n/9n}$. Therefore, any dark gray disk of radius $r(n)/2$, which

center is further than $r(n)/2$ to the corners of the unit square (and thus not in a corner sub-box), contains at least one box, and thus at least one node with high probability. Therefore, greedy routing reaches the closest node to the target with high probability as well. As α can be taken arbitrarily close to 1, the Lemma holds. ■

A.3 Balanced Random Geometric Graphs

In random geometric graphs, the nodes are drawn uniformly at random. This section shows various results stating that the resulting graphs empirically present a uniform structure with high probability: nodes are regularly spread out (empirical uniformity theorem), every node has degree $\Theta(\log n)$ (empirical log-balance theorem), and nodes do not clutter too much so that there are not too many very small Voronoi cells (Median Voronoi area conjecture).

A.3.1 Balanced boxes

Theorem* A.3.1 (Empirical uniformity). Draw independently and uniformly n nodes in a unit square, and divide the unit square in boxes of size $\sqrt{\alpha \log n/n} \times \sqrt{\alpha \log n/n}$. If $\alpha > 3$, then there is a pair (a, b) of reals such that $a \leqslant \alpha \leqslant b$ and such that every box contains between $a \log n$ and $b \log n$ with high probability. If $\alpha > 4$, then the result holds with very high probability.

Proof: The space is divided in boxes of surface $\alpha \log n/n$ with $\alpha > 3$. The goal of this proof is to show that the number of nodes in every box is jointly lower bounded by $a \log n$ and upper bounded by $b \log n$, for some constants $a < \alpha < b$ with high probability. Denote by E the set of Bernouilli distributions which parameter p is not included in $[a \log n/n, b \log n/n]$.

Start by fixing a box B. For any node i, $\mathbb{P}[i \in B] = \alpha \log n/n$. Let $X_i^B = 1_{i \in B}$. The X_i^B are i.i.d. random variables with distribution Q, where Q is Bernouilli of parameter $\alpha \log n/n$. $Q_B^n(E)$ is the probability that a realization of $X_1^B, \cdots, X_n^B$ has an empirical distribution in E, i.e. $Q_B^n(E)$ is the probability that the empirical proportion of nodes that fall in B is not in $[a \log n/n, b \log n/n]$. Then, according to Sanov's theorem [20],

$$Q_B^n(E) \leqslant (n+1)^2 2^{-nD(P^* \| Q)},$$

where

$$P^* = \arg\min_{P \in E} D(P \| Q)$$

is the distribution in E that is closest to Q in relative entropy. If we can find a and b such that $Q_B^n(E) = O(n^{-\zeta})$ with $\zeta \geqslant 1$, then we conclude that the event A "There is a box with less than $a \log n$ nodes or more than $b \log n$ nodes"

happens with low probability:

$$\begin{aligned}\mathbb{P}[A] &\leqslant \sum_B Q_B^n(E) && \text{(A.2)}\\ &\leqslant \sum_B n^{-\zeta} \\ &= \frac{n^{1-\zeta}}{\alpha \log n}. && \text{(A.3)}\end{aligned}$$

Therefore, if $\zeta \geqslant 1$, we can conclude that the probability of having all the boxes with a number N of nodes such that $a \log n \leqslant N \leqslant b \log n$, goes to 1 when n goes to infinity.

Now it remains to prove that, for any $\alpha > 3$, such a and b can be found. Take a bernouilli distribution P_q of parameter $q \log n/n$.

$$\begin{aligned}D(P_q \| Q) &= (1 - \frac{q \log n}{n}) \log \left(\frac{1 - \frac{q \log n}{n}}{1 - \frac{\alpha \log n}{n}} \right) + \frac{q \log n}{n} \log(\frac{q}{\alpha}) \\ &= \frac{\log n}{n} \left[\alpha - q(1 - \log(\frac{q}{\alpha})) \right] + o(\frac{\log n}{n}).\end{aligned}$$

Denote by f the function $f(x) = \alpha - x(1 - \log(x/\alpha))$. The derivative of f is $df/dx = \log(x/\alpha)$. Therefore f decreases on $[0, \alpha]$ and increases on $[\alpha, \infty]$. Note that $f(0) = \alpha > 3$, $f(\alpha) = 0$ and $\lim_{x \to \infty} f(x) = \infty$. Thus, for a small enough and b large enough, if $q \notin [a, b]$, then there is a f_0 such that $f(q) \geqslant f_0 > 3$ and $D(P_q \| Q) \geqslant r \log n/n$.

$$\begin{aligned}Q_B^n(E) &\leqslant (n+1)^2 2^{-f_0 \log n} \\ &= \frac{(n+1)^2}{n^{f_0}} \\ &= O(n^{-(f_0 - 2)}).\end{aligned}$$

Assuming that $\alpha > 3$, we have just proved that we can find an f_0 larger than 3, such that $\zeta = f_0 - 2 > 1$ (by definition of ζ, $Q_B^n(E) = O(n^{-\zeta})$). Therefore, if $\alpha > 3$, by Eq. A.3, the probability of event A goes to zero, which concludes the proof.

Note that if $\alpha > 4$, then f_0 can be taken larger than 4, and $\zeta = f_0 - 2 > 2$. Therefore, if $\alpha > 4$, the theorem holds with very high probability. ■

A.3.2 Balanced degrees

Definition* A.3.1 (log-balanced graphs). A graph is log-balanced with respect to d_1 and d_2 if every node has between $d_1 \log n$ and $d_2 \log n$ neighbors.

Theorem* A.3.2 (Empirical log-balance). Let $G(n, r(n))$, where $r(n) = \sqrt{c \log n/n}$, be a random geometric graph on the torus. If $c > 3$, then there are some d_1 and d_2 such that a random geometric graph is log-balanced with respect to d_1 and d_2 with high probability. If $c > 4$, then the result holds with very high probability.

Proof: For any node i, let $\mathcal{N}_i$ be the number of neighbors of node i. Divide

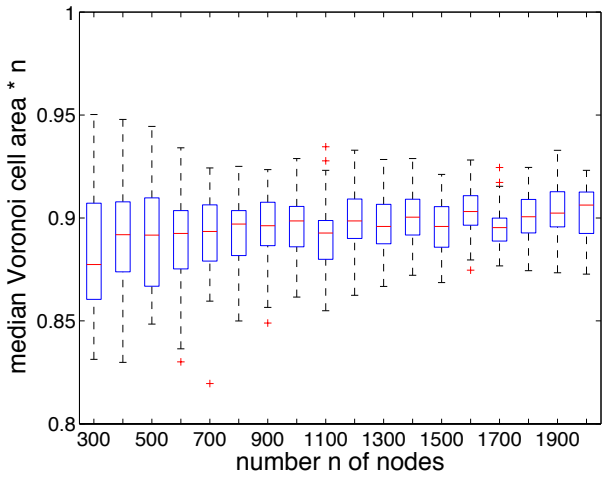


Figure A.7: Median Voronoi cell area. Boxplots with 50 samples for each n. Voronoi cells on the edges of the square domain were not taken into account.

the unit square in boxes of size $c \log n/n$. This creates a lattice partition L_1 of the unit square, on which Theorem A.3.1 applies, since $c > 3$.

($|\mathcal{N}_i| \geqslant d_1 \log n$): Consider the same shift lattice trick as in the proof of the "non-empty disk" theorem (Lemma A.2.1). Then, by applying the union bound on the 9 lattices in Eq. (A.2), one can show that all the boxes in all the 9 lattices contain between $a \log n$ and $b \log n$ nodes with high probability. Then, by the same geometrical argument as in Lemma A.2.1, any disk of radius larger than or equal to $\sqrt{c \log n/n}$ contains at least one box of one lattice partition. Therefore, any such disk contains at least $a \log n$ w.h.p. As a consequence any node has at least $a \log n$ neighbors w.h.p.

($|\mathcal{N}_i| \leqslant d_2 \log n$): Any disk of radius $\sqrt{c \log n/n}$ intersects at most 9 boxes of the partition lattice L_1. Therefore any node has at most $9b \log n$ neighbors.

Taking $d_1 = a$ and $d_2 = 9b$ yields the theorem.

Note that if $c > 4$, then, according to Theorem. A.3.1, the result holds with very high probability. ■

A.3.3 Balanced Voronoi cells

Not much is known on the distribution of the sizes of the Voronoi cells of a random geometric graph. Voronoi cells have been studied in random Poisson point processes (infinite areas), and yet most results there are empirical [34].

Conjecture* A.3.1 (Median Voronoi area). Draw independently and uniformly n nodes in a unit square. Compute the areas of the Voronoi cells and order them in increasing order: $V_1 \leqslant V_2 \leqslant \ldots \leqslant V_n$. Then the median Voronoi area $V_{\lfloor n/2 \rfloor} \geqslant 0.8/n$ with high probability.

In order to illustrate the conjecture, see Fig. A.7.

Appendix B

The quaternary automaton construction

We generate automatically the automaton Δ. We do not want to favour one color more than others. Thus, each time we generate a transition $\Delta(s_1, s_2)$, we also generate rotated versions of it. Let R be one of the appropriate rotation we define later, then the R rotated version is computed as follows:

$$\Delta(R(s_1), R(s_2)) = R(\Delta(s_1, s_2)).$$

To define a state rotation, one needs to define a classical 3D coordinates rotation R_{rot}, and a color permutation R_{col}. For any state s of coordinates x_s and color c_s, $R(x_s, c_s) = (R_{rot}(x_s), R_{col}(c_s))$ should be a state. Furthermore R should be a state permutation, i.e. it should be bijective. In our basis, the vertices of the simplex have coordinates:

$$\begin{array}{rcl} O_A & : & (3,3,3), \\ O_B & : & (-3,-3,3) \\ O_C & : & (-3,3,-3) \\ O_D & : & (3,-3,-3) \end{array}$$

We will use six different rotations:

- Rotation R^1: axial symmetry with respect to (C^1_{AB}, C^1_{CD}).

$$\begin{array}{rcl} R^1_{rot} & = & \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\ R^1_{col}(A) & = & B, \\ R^1_{col}(B) & = & A, \\ R^1_{col}(C) & = & D, \\ R^1_{col}(D) & = & C. \end{array}$$

- Rotation R^2: axial symmetry with respect to (C^1_{AC}, C^1_{BD}):

$$\begin{aligned} R^2_{rot} &= \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \\ R^2_{col}(A) &= C, \\ R^2_{col}(C) &= A, \\ R^2_{col}(B) &= D, \\ R^2_{col}(D) &= B. \end{aligned}$$

- Rotation R^3: axial symmetry with respect to (C^1_{AD}, C^1_{BC}):

$$\begin{aligned} R^3_{rot} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \\ R^3_{col}(A) &= D, \\ R^3_{col}(D) &= A, \\ R^3_{col}(B) &= C, \\ R^3_{col}(C) &= B. \end{aligned}$$

- Rotation R^4: rotation of axis (C^2_{ABC}, D):

$$\begin{aligned} R^4_{rot} &= \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \\ R^4_{col}(A) &= B, \\ R^4_{col}(B) &= C, \\ R^4_{col}(C) &= A, \\ R^4_{col}(D) &= D. \end{aligned}$$

- Rotation R^5: rotation of axis (C^2_{ABC}, D):

$$\begin{aligned} R^5_{rot} &= \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix}, \\ R^5_{col}(A) &= C, \\ R^5_{col}(B) &= A, \\ R^5_{col}(C) &= B, \\ R^5_{col}(D) &= D. \end{aligned}$$

- Rotation R^6: reflexion symmetry with respect to (C^1_{AB}, C, D):

$$\begin{aligned} R^6_{rot} &= \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\ R^6_{col}(A) &= B, \\ R^6_{col}(B) &= A, \\ R^6_{col}(C) &= C, \\ R^6_{col}(D) &= D. \end{aligned}$$

State coding: States of color A are indexed from 1 to 25, B from 26 to 50, C from 51 to 75 and D from 76 to 100. First we sort states of color A by increasing distance to O_A. Then we generate states of color B in the following way: state number $25+x$ is $R^1(x)$. Similarly, states of color C are generated by rotating states of color A with rotation R^2 and states of color D are generated by rotating states of color A with rotation R^3. The 100 states are listed in Table 6.1, and some are shown in Fig. B.1.

Coding the automaton.
Let i and j be two state indices. The function `computeTransition` has inputs i and j and outputs $\Delta(i,j)$ and $\Delta(j,i)$. First it computes the coordinates of the middle m of state i and state j, as well as its colors c.

- If m belongs to $\mathcal{P}$ then $\Delta(i,j)$ is a state with coordinates m. If m is monocolor of color c then `computeTransition` outputs the index of the state of coordinates m and color c. Otherwise m has more than one color. If m has the color c_j of j, then $\Delta(i,j)$ is the state (m, c_j). Similarly, if m has the color c_i of i, then $\Delta(j,i)$ is the state (m, c_i). If a color swap is not possible, `computeTransition` outputs a state of coordinates m with any of its colors. For fairness reasons, in that case where colors cannot be swapped, it is preferable to design `computeTransition` such that $\Delta(i,j)$ and $\Delta(j,i)$ have different colors.

- If m does not belong to $\mathcal{P}$, then `computeTransition` should list all the ordered pairs of states whose middle is m and whose colors are colors of m. Then select the pairs of minimum length (the most contracted). If possible, keep pairs that swap colors; if this is not possible keep pairs that swap one color only (if m is multicolor, it is more fair to choose two distinct colors); if this is not possible either, then keep all the current choices of pairs (if m is not monocolor, select only pairs of different colors). If m is monocolor choose the pairs that contain the smallest index among pair indices (the closest to the simplex vertex of the color of m). Finally select pairs (y,z) that maximize $\|y - x_i\| + \|z - x_j\|$ (we believe this choice accelerates convergence). If there is still the choice between several pairs, then `computeTransition` should choose randomly any of them. Finally, `computeTransition` outputs the first state of the pair as $\Delta(i,j)$, and the second one as $\Delta(j,i)$.

We describe now the function `buildAutomaton` that builds the automaton table which is a 100×100 matrix. Initialize A and $flags$ with two 100×100 matrices full of zeros. A is the automaton and $flags$ keeps track of the values of A that have already been computed. Each time we write a line of pseudo code of the form:

$A(i,j) \leftarrow x$,

we imply that the operation is done only if $flag(i,j) = 0$; we then implicitely update the flag to 1: $flag(i,j) \leftarrow 1$. We do not write it in the pseudo code to make it shorter.

1: **for** $i = 1 : 25$ **do**
2: **for** $j = 1 : 25$ **do**
3: $(y, z) =$`computeTransition`(i, j)
4: $A(i, j) \leftarrow y$
5: $A(j, i) \leftarrow z$
6: $A(R^1(i), R^1(j)) \leftarrow R^1(y)$
7: $A(R^1(j), R^1(i)) \leftarrow R^1(z)$
8: $A(R^2(i), R^2(j)) \leftarrow R^2(y)$
9: $A(R^2(j), R^2(i)) \leftarrow R^2(z)$
10: $A(R^3(i), R^3(j)) \leftarrow R^3(y)$
11: $A(R^3(j), R^3(i)) \leftarrow R^3(z)$
12: **end for**
13: **end for**
14: **for** $i = 1 : 25$ **do**
15: **for** $j = 26 : 50$ **do**
16: $(y, z) \leftarrow$`computeTransition`(i, j)
17: $A(i, j) \leftarrow y$
18: $A(j, i) \leftarrow z$
19: $A(R^2(i), R^2(j)) \leftarrow R^2(y)$
20: $A(R^2(j), R^2(i)) \leftarrow R^2(z)$
21: $i' \leftarrow R^4(i)$
22: $j' \leftarrow R^4(j)$
23: $y' \leftarrow R^4(y)$
24: $z' \leftarrow R^4(z)$
25: $A(i', j') \leftarrow y'$
26: $A(j', i') \leftarrow z'$
27: $A(R^1(i'), R^1(j')) \leftarrow R^1(y')$
28: $A(R^1(j'), R^1(i')) \leftarrow R^1(z')$
29: $i' \leftarrow R^5(i)$
30: $j' \leftarrow R^5(j)$
31: $y' \leftarrow R^5(y)$
32: $z' \leftarrow R^5(z)$
33: $A(i', j') \leftarrow y'$
34: $A(j', i') \leftarrow z'$
35: $A(R^3(i'), R^3(j')) \leftarrow R^3(y')$
36: $A(R^3(j'), R^3(i')) \leftarrow R^3(z')$

37: $i \leftarrow R^6(i)$
38: $j \leftarrow R^6(j)$
39: $y \leftarrow R^6(y)$
40: $z \leftarrow R^6(z)$
41: Run lines 17 to 36 with the new i, j, y and z.
42: **end for**
43: **end for**

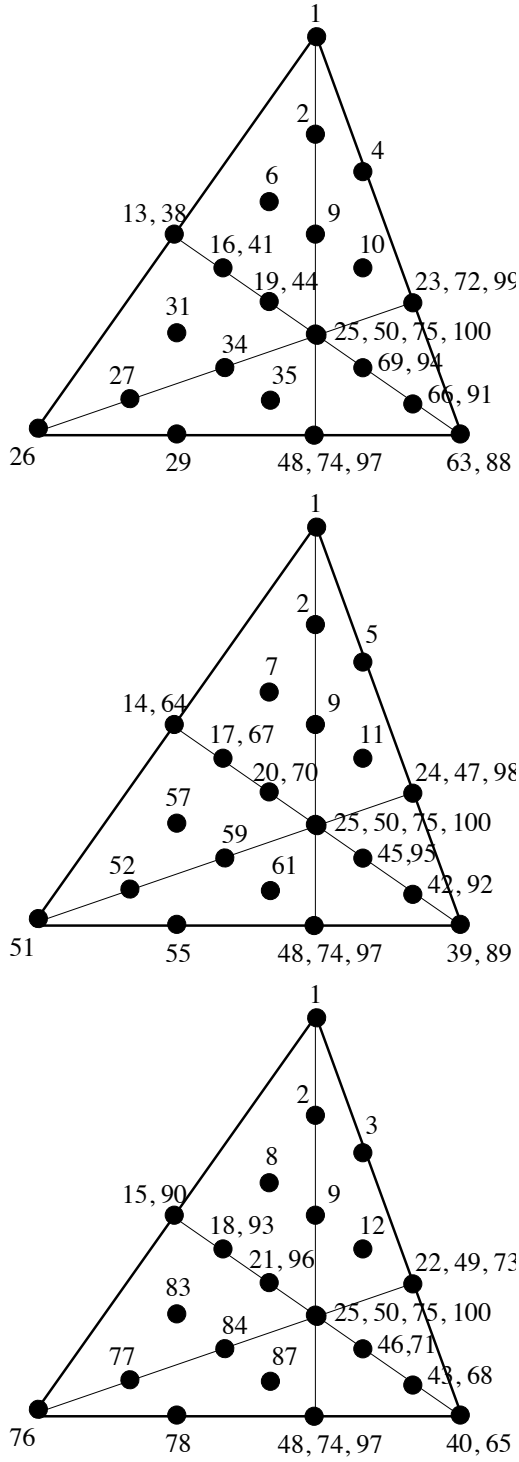


Figure B.1: State indices on 3 cuts of the tetrahedron all containing axis (O_A, C^2_{BCD}). We suggest that the reader tries out some of the transitions of the automaton on these figures.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	1	2	3	4	5	6	7	8	2	4	5	3	13	14	15	16	17	18	6	7	8	3	4	5	9
2	1	2	3	4	5	6	7	8	9	9	9	9	13	14	15	6	7	8	9	9	9	12	10	11	9
3	1	2	3	2	6	6	7	9	9	9	9	12	13	14	8	16	17	9	12	12	9	22	10	11	12
4	1	2	7	4	2	9	7	8	9	10	9	9	6	14	15	9	17	18	9	10	10	12	23	11	10
5	1	2	2	8	5	6	9	8	9	9	11	9	13	7	15	16	9	18	11	9	11	12	10	24	11
6	1	2	3	2	5	6	9	5	9	9	11	12	13	17	18	16	12	11	19	12	11	22	10	24	19
7	1	2	3	4	2	3	7	9	9	10	9	12	16	14	18	12	17	10	12	20	10	22	23	11	20
8	1	2	2	4	5	9	4	8	9	10	11	9	16	17	15	11	10	18	11	10	21	12	23	24	21
9	2	2	3	4	5	6	7	8	9	10	11	12	16	17	18	19	20	21	19	20	21	12	10	11	25
10	2	4	7	4	8	9	7	8	9	10	9	20	19	14	15	19	17	18	25	20	21	20	23	21	25
11	2	5	6	8	5	6	9	8	9	21	11	9	13	20	15	16	20	18	19	25	21	19	21	24	25
12	2	3	3	7	6	6	7	9	9	9	19	12	13	14	21	16	17	21	19	20	25	22	20	19	25
13	1	2	3	6	5	6	3	5	6	6	11	12	13	22	5	16	22	24	16	16	16	22	19	24	19
14	1	2	3	4	7	3	7	4	7	10	7	12	3	14	23	22	17	23	17	17	17	22	23	20	20
15	1	2	8	4	5	5	4	8	8	10	11	8	24	4	15	24	23	18	18	18	18	21	23	24	21
16	1	6	3	6	5	6	6	6	6	9	11	12	13	7	8	16	12	11	19	19	19	22	25	24	19
17	1	7	3	4	7	7	7	7	7	10	9	12	6	14	8	12	17	10	20	20	20	22	23	25	20
18	1	8	8	4	5	8	8	8	8	10	11	9	6	7	15	11	10	18	21	21	21	25	23	24	21
19	2	6	6	9	6	6	9	9	9	9	11	12	16	12	11	16	12	11	19	12	25	22	25	24	25
20	2	7	7	7	9	9	7	9	9	10	9	12	12	17	10	12	17	10	25	20	10	22	23	25	25
21	2	8	9	8	8	9	9	8	9	10	11	9	11	10	18	11	10	18	11	25	21	25	23	24	25
22	3	3	3	7	6	6	7	9	12	12	12	12	13	14	9	16	17	9	19	20	12	22	20	19	25
23	4	4	7	4	8	9	7	8	10	10	10	10	9	14	15	9	17	18	10	20	21	20	23	21	25
24	5	5	6	8	5	6	9	8	11	11	11	11	13	9	15	16	9	18	19	11	21	19	21	24	25
25	2	9	9	9	9	9	9	9	9	10	11	12	16	17	18	19	20	21	19	20	21	22	23	24	25

Table B.1: Partial automaton: interaction between states of color A.

	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
1	38	13	13	16	13	13	16	16	6	9	6	6	13	24	22	16	11	12	6	9	9	5	9	3	9
2	31	41	13	44	13	16	16	16	19	19	12	11	13	24	22	6	11	12	9	11	12	11	25	12	9
3	30	36	41	36	49	13	44	49	16	19	22	19	13	47	49	16	19	22	12	19	12	11	25	22	12
4	31	41	16	44	16	19	19	19	19	25	12	11	6	98	73	9	21	20	9	21	20	11	25	12	10
5	28	37	47	37	41	13	47	44	16	19	19	24	13	47	49	16	24	19	11	11	19	24	25	12	11
6	27	31	37	31	36	41	37	36	44	44	16	16	13	37	36	16	47	49	19	19	19	24	50	22	19
7	30	36	41	36	41	16	44	49	19	25	12	19	16	47	60	12	25	73	12	25	20	11	75	22	20
8	28	37	41	37	41	16	47	44	19	25	19	11	16	85	49	11	98	25	11	21	25	24	100	12	21
9	31	31	37	34	36	41	37	36	44	50	19	19	16	45	46	19	47	49	19	25	25	11	50	12	25
10	31	34	37	34	36	44	37	36	44	50	25	25	19	85	60	19	85	60	25	100	75	21	94	20	25
11	28	31	31	34	31	37	37	34	37	37	44	47	13	42	36	16	45	36	19	50	50	24	45	19	25
12	30	31	31	34	31	36	34	36	36	36	49	44	13	37	43	16	37	46	19	50	50	19	46	22	25
13	26	27	28	31	30	31	28	30	31	31	36	37	38	47	30	41	47	49	41	41	41	47	44	49	44
14	30	36	44	46	49	22	50	73	73	70	73	20	22	50	54	22	75	73	17	70	67	20	70	73	70
15	28	37	47	45	44	24	98	50	98	96	21	98	5	98	50	24	98	100	18	93	96	98	96	21	96
16	26	31	28	31	30	31	31	31	31	34	36	37	38	32	33	41	37	36	44	44	44	47	50	49	44
17	30	36	36	36	36	44	36	46	50	75	73	25	6	45	58	12	50	60	20	75	70	25	75	73	70
18	28	37	37	37	37	44	45	37	50	100	25	98	6	82	46	11	85	50	21	96	100	98	100	25	96
19	27	31	31	34	31	31	34	34	34	34	36	37	41	37	36	41	37	36	44	37	50	47	50	49	50
20	30	34	34	34	36	36	34	36	36	46	49	44	12	45	60	12	45	60	12	50	75	25	75	73	75
21	28	34	37	34	34	37	37	34	37	45	44	47	11	85	46	11	85	46	25	85	50	98	100	25	100
22	30	30	31	33	30	31	34	33	36	36	36	36	38	34	40	41	34	43	44	36	46	44	46	49	50
23	34	34	37	35	36	44	45	46	50	75	75	100	9	84	59	9	84	59	10	86	62	96	94	70	75
24	28	28	28	32	31	31	32	34	37	37	37	37	38	39	34	41	42	34	44	45	37	47	45	44	50
25	27	34	34	34	34	34	34	34	34	35	36	37	41	42	43	44	45	46	44	45	46	47	48	49	50

Table B.2: Partial automaton: interaction between states of color A and states of color B.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
26	13	38	38	41	38	38	41	41	31	34	31	31	38	49	47	41	36	37	31	34	34	30	34	28	34
27	6	16	38	19	38	41	41	41	44	44	37	36	38	49	47	31	36	37	34	36	37	36	50	37	34
28	5	11	16	11	13	38	19	24	41	44	47	44	38	22	24	41	44	47	37	44	37	36	50	47	37
29	6	16	41	19	41	44	44	44	44	50	37	36	31	73	98	34	46	45	34	46	45	36	50	37	35
30	3	12	13	12	16	38	22	19	41	44	44	49	38	22	24	41	49	44	36	36	44	49	50	37	36
31	2	6	12	6	11	16	12	11	19	19	41	41	38	12	11	41	22	24	44	44	44	49	25	47	44
32	5	11	16	11	16	41	19	19	44	50	37	44	41	22	85	37	50	98	37	50	45	36	100	47	45
33	3	12	16	12	16	41	19	19	44	50	44	36	41	60	24	36	73	50	36	46	50	49	75	37	46
34	6	6	12	9	11	16	12	11	19	25	44	44	41	20	21	44	22	24	44	50	50	36	25	37	50
35	6	9	12	9	11	19	12	11	19	25	50	50	44	60	85	44	60	85	50	75	100	46	94	45	50
36	3	6	6	9	6	12	12	9	12	12	19	19	38	17	11	41	20	11	44	25	25	49	20	44	50
37	5	6	6	9	6	11	9	11	11	11	19	19	38	12	18	41	12	21	44	25	25	44	21	47	50
38	1	2	3	6	5	6	3	5	6	6	11	12	13	3	24	16	22	24	16	16	16	22	19	24	19
39	5	11	19	21	24	47	25	98	98	95	98	45	28	25	79	47	100	98	42	95	92	45	95	98	95
40	3	12	22	20	19	49	73	25	73	71	46	73	49	73	25	49	73	75	43	68	71	73	71	46	71
41	1	6	3	6	5	6	6	6	6	9	11	12	13	7	8	16	12	11	19	19	19	22	25	24	19
42	5	11	11	11	11	19	11	21	25	100	98	50	31	20	83	37	25	85	45	100	95	50	100	98	95
43	3	12	12	12	12	19	20	12	25	75	50	73	31	57	21	36	60	25	46	71	75	73	75	50	71
44	2	6	6	9	6	6	9	9	9	9	11	12	16	12	11	16	12	11	19	25	11	22	25	24	25
45	5	9	9	9	11	11	9	11	11	21	24	19	37	20	85	37	20	85	50	25	100	50	100	98	100
46	3	9	12	9	9	12	12	9	12	20	19	22	36	60	21	36	60	21	36	60	25	73	75	50	75
47	5	5	6	8	5	6	9	8	11	11	11	11	13	9	15	16	9	18	19	11	21	19	21	24	25
48	9	9	12	10	11	19	20	21	25	75	100	75	34	59	84	34	59	84	35	61	87	71	69	95	75
49	3	3	3	7	6	6	7	9	12	12	12	12	13	14	9	16	17	9	19	20	12	22	20	19	25
50	2	9	9	9	9	9	9	9	9	10	11	12	16	17	18	19	20	21	19	20	21	22	23	24	25

Table B.3: Partial automaton: interaction between states of color B and states of color A.

Bibliography

[1] N. Ahmed, S. S. Kanhere, and S. Jha, "The holes problem in wireless sensor networks: a survey," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 2, pp. 4–18, 2005.

[2] D. Aldous and J. Fill, "Reversible markov chains and random walks on graphs, book in preparation," *URL for draft at http://www. stat. Berkeley. edu/users/aldous.*

[3] N. Alon and V. Milman, "λ_1, isoperimetric inequalities for graphs, and superconcentrators." *J. COMBINAT. THEORY SER. B.*, vol. 38, no. 1, pp. 73–88, 1985.

[4] W. Anderson and T. Morley, "Eigenvalues of the laplacian of a graph," *Linear and Multilinear Algebra*, vol. 18, no. 2, pp. 141–145, 1985.

[5] D. Angluin, J. Aspnes, Z. Diamadi, M. Fischer, and R. Peralta, "Computation in networks of passively mobile finite-state sensors," *Distributed Computing*, vol. 18, no. 4, pp. 235–253, 2006.

[6] T. C. Aysal, M. Coates, and M. Rabbat, "Distributed average consensus using probabilistic quantization," *Statistical Signal Processing, 2007. SSP '07. IEEE/SP 14th Workshop on*, pp. 640–644, Aug. 2007.

[7] T. Aysal, M. Yildiz, and A. Scaglione, "Broadcast gossip algorithms," in *IEEE Information Theory Workshop, 2008. ITW'08*, 2008, pp. 343–347.

[8] F. Benezit, A. Dimakis, P. Thiran, and M. Vetterli, "Geographic gossip with path averaging is order optimal," *Allerton*, 2007.

[9] ——, "Order-optimal consensus through randomized path averaging," *submitted for publication.*

[10] F. Benezit, P. Thiran, and M. Vetterli, "Interval consensus: From quantized gossip to voting," *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2009.

[11] D. Bertsekas, M. Athans, and J. Tsitsiklis, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," 1984.

[12] D. Bertsekas and J. Tsitsiklis, *Parallel and distributed computation.* Old Tappan, NJ (USA); Prentice Hall Inc., 1989.

[13] V. Blondel, J. Hendrickx, A. Olshevsky, and J. Tsitsiklis, "Convergence in multiagent coordination, consensus, and flocking," in *44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05*, 2005, pp. 2996–3000.

[14] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip algorithms : Design, analysis and applications," in *Proceedings of the 24th Conference of the IEEE Communications Society (INFOCOM 2005)*, 2005.

[15] ——, "Randomized gossip algorithms," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[16] P. Brémaud, *Markov Chains. Gibbs Fields, Monte Carlo Simulation, and Queues.* Springer, 1999.

[17] M. S. Capcarrère and M. Sipper, "Necessary conditions for density classification by cellular automata," *Phys. Rev. E*, vol. 64, no. 3, p. 036113, Aug 2001.

[18] M. S. Capcarrere, M. Sipper, and M. Tomassini, "Two-state, $r = 1$ cellular automaton that classifies density," *Phys. Rev. Lett.*, vol. 77, no. 24, pp. 4969–4971, Dec 1996.

[19] F. Chung, *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92) (Cbms Regional Conference Series in Mathematics).* American Mathematical Society, 1997.

[20] T. M. Cover and J. A. Thomas, *Elements of Information Theory.* New York: Wiley, 1991.

[21] G. Cybenko, "Dynamic load balancing for distributed memory multiprocessors," *Journal of parallel and distributed computing*, vol. 7, no. 2, pp. 279–301, 1989.

[22] P. G. de S and C. Maes, "The Gacs-Kurdyumov-Levin automaton revisited," *Journal of Statistical Physics*, vol. 67, pp. 507–522, May 1992.

[23] A. Dembo and O. Zeitouni, *Large deviations techniques and applications.* Springer Verlag, 1998.

[24] P. Denantes, F. Bénézit, P. Thiran, and M. Vetterli, "Which Distributed Averaging Algorithm Should I Choose for my Sensor Network?" in *IEEE INFOCOM 2008.* [Online]. Available: http://www.ieee-infocom.org/

[25] P. Diaconis and D. Stroock, "Geometric bounds for eigenvalues of markov chains," in *Annals of Applied Probability*, vol. 1, 1991.

[26] K. Diks and A. Pelc, "System diagnosis with smallest risk of error," in *WG '96: Proceedings of the 22nd International Workshop on Graph-Theoretic Concepts in Computer Science.* London, UK: Springer-Verlag, 1997, pp. 141–150.

[27] A. G. Dimakis, A. D. Sarwate, and M. J. Wainwright, "Geographic gossip: efficient aggregation for sensor networks," in *ACM/IEEE Symposium on Information Processing in Sensor Networks*, 2006.

[28] M. Draief and M. Vojnovic, "Convergence speed of binary interval consensus," Microsoft Research, Tech. Rep. MSR-TR-2009-86, August 2009.

[29] J. Elson and D. Estrin, "Time synchronization in wireless sensor networks," 2003.

[30] F. Fagnani and S. Zampieri, "Average consensus with packet drop communication," in *Decision and Control, 2006 45th IEEE Conference on*, 2006, pp. 1007–1012.

[31] ——, "Randomized consensus algorithms over large scale networks," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 4, 2008.

[32] Q. Fang, J. Gao, and L. Guibas, "Locating and bypassing holes in sensor networks," *Mobile Networks and Applications*, vol. 11, no. 2, pp. 187–200, 2006.

[33] J. Fax, R. Murray, N. Syst, and C. Woodland Hills, "Information flow and cooperative control of vehicle formations," *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1465–1476, 2004.

[34] J. Ferenc and Z. Nda, "On the size distribution of poisson voronoi cells," *Physica A: Statistical Mechanics and its Applications*, vol. 385, no. 2, pp. 518–526, 2007.

[35] M. Fiedler, "Algebraic connectivity of graphs," *Czechoslovak Mathematical Journal*, vol. 23, no. 98, pp. 298–305, 1973.

[36] M. J. Fischer, "The consensus problem in unreliable distributed systems (a brief survey)," in *Proceedings of the 1983 International FCT-Conference on Fundamentals of Computation Theory.* London, UK: Springer-Verlag, 1983, pp. 127–140.

[37] M. Fischer, N. Lynch, and M. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[38] H. Fukś, "Nondeterministic density classification with diffusive probabilistic cellular automata," *Phys. Rev. E*, vol. 66, no. 6, p. 066106, Dec 2002.

[39] H. Fuks-acute, "Solution of the density classification problem with two cellular automata rules," *Phys. Rev. E*, vol. 55, no. 3, pp. R2081–R2084, Mar 1997.

[40] H. Furstenberg and H. Kesten, "Products of random matrices," *The Annals of Mathematical Statistics*, pp. 457–469, 1960.

[41] A. Giridhar and P. Kumar, "Distributed clock synchronization over wireless networks: Algorithms and analysis," in *Decision and Control, 2006 45th IEEE Conference on*, 2006, pp. 4915–4920.

[42] P. Gupta and P. Kumar, "Critical power for asymptotic connectivity in wireless networks. stochastic analysis, control," *Optimization and Applications*, pp. 547–566, 1998.

[43] B. Hardekopf, K. Kwiat, and S. Upadhyaya, "Secure and fault-tolerant voting in distributed," 2001. [Online]. Available: http://citeseer.ist.psu.edu/425069.html

[44] Y. Hassin and D. Peleg, "Distributed probabilistic polling and applications to proportionate agreement," in *ICALP*, 1999, pp. 402–411.

[45] R. Horn and C. Johnson, *Matrix Analysis.* Cambridge: Cambridge University Press, 1987.

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

[47] M. Jerrum and A. Sinclair, "Approximating the permanent," *SIAM journal on computing*, vol. 18, p. 1149, 1989.

[48] S. Kar and J. M. F. Moura, "Distributed consensus algorithms in sensor networks: Quantized data," *CoRR*, vol. abs/0712.1609, 2007.

[49] B. Karp and H. Kung, "Gpsr: greedy perimeter stateless routing for wireless networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking.* ACM New York, NY, USA, 2000, pp. 243–254.

[50] A. Kashyap, T. Başar, and R. Srikant, "Quantized consensus," *Automatica*, vol. 43, no. 7, pp. 1192–1203, 2007.

[51] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE*, vol. 44. IEEE COMPUTER SOCIETY PRESS, 2003, pp. 482–491.

[52] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982.

[53] M. Land and R. K. Belew, "No perfect two-state cellular automata for density classification exists," *Phys. Rev. Lett.*, vol. 74, no. 25, pp. 5148–5150, Jun 1995.

[54] J. Meyer, Carl D. and R. J. Plemmons, "Convergent powers of a matrix with applications to iterative methods for singular linear systems," *SIAM Journal on Numerical Analysis*, vol. 14, no. 4, pp. 699–705, 1977. [Online]. Available: http://www.jstor.org/stable/2156488

[55] B. Mohar, "The laplacian spectrum of graphs," *Graph theory, combinatorics, and applications*, vol. 2, pp. 871–898, 1991.

[56] N. H. Mustafa and A. Pekec, "Majority consensus and the local majority rule," in *ICALP '01: Proceedings of the 28th International Colloquium on Automata, Languages and Programming,*. London, UK: Springer-Verlag, 2001, pp. 530–542.

[57] B. Nazer, A. Dimakis, and M. Gastpar, "Local interference can accelerate gossip algorithms," *networks*, vol. 3, p. 5.

[58] ——, "Neighborhood gossip: Concurrent averaging through local interference."

[59] A. Oppenheim and R. Schafer, *Discrete-time signal processing.* Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1989.

[60] V. I. Oseledec, "A multiplicative ergodic theorem. Characteristic Ljapunov, exponents of dynamical systems," *Trudy Moskov. Mat. Obšč.*, vol. 19, pp. 179–210, 1968.

[61] D. Peleg, "Local majorities, coalitions and monopolies in graphs: a review," *Theor. Comput. Sci.*, vol. 282, no. 2, pp. 231–257, 2002.

[62] D. Peleg and A. Wool, "The availability of quorum systems," *Inf. Comput.*, vol. 123, no. 2, pp. 210–223, 1995.

[63] Y. Rabani, A. Sinclair, and R. Wanka, "Local divergence of markov chains and the analysis of iterative load-balancing schemes," in *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS98)*, 1998, pp. 694–705.

[64] M. Rabbat, R. Nowak, and J. Bucklew, "Robust decentralized source localization via averaging," in *IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05)*, vol. 5, 2005.

[65] M. Rabbat and R. Nowak, "Decentralized source localization and tracking [wireless sensor networks]," in *IEEE International Conference on Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04)*, vol. 3, 2004.

[66] M. S. Raghunathan, "A proof of Oseledec's multiplicative ergodic theorem," *Israel J. Math.*, vol. 32, no. 4, pp. 356–362, 1979.

[67] W. Ren and E. Atkins, "Distributed multi-vehicle coordinated control via local information exchange," *International Journal of Robust and Nonlinear Control*, vol. 17, no. 10-11, pp. 1002–1033, 2007.

[68] S. Roch, "Bounding fastest mixing," *eprint arXiv: math/0409429*, 2004.

[69] D. Ruelle, "Ergodic theory of differentiable dynamical systems," *Inst. Hautes Études Sci. Publ. Math.*, no. 50, pp. 27–58, 1979.

[70] P. Sarkar, "A brief history of cellular automata," *ACM Comput. Surv.*, vol. 32, no. 1, pp. 80–107, 2000.

[71] A. Sarlette, S. Tuna, V. Blondel, and R. Sepulchre, "Global synchronization on the circle," in *Proc. 17th IFAC World Congress*, 2008.

[72] A. Sarwate and A. Dimakis, "The impact of mobility on gossip algorithms," in *28th Conference on Computer Communications (INFOCOM)*, 2009.

[73] A. Tahbaz-Salehi and A. Jadbabaie, "Necessary and sufficient conditions for consensus over random independent and identically distributed switching graphs," in *Decision and Control, 2007 46th IEEE Conference on*, 2007, pp. 4209–4214.

[74] M. Tomassini and M. Venzi, "Evolution of asynchronous cellular automata for the density task," in *PPSN VII: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*. London, UK: Springer-Verlag, 2002, pp. 934–944.

[75] J. Tsitsiklis, "Problems in decentralized decision making and computation," Ph.D. dissertation, M. I. T., Dept. of Electrical Engineering and Computer Science, 1984.

[76] S. Vanka, V. Gupta, and M. Haenggi, "On consensus over stochastically switching directed topologies."

[77] P. Walters, *An introduction to ergodic theory*, ser. Graduate Texts in Mathematics. New York: Springer-Verlag, 1982, vol. 79.

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

[79] 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*, 2005, pp. 63–70.

[80] J. Zhao, R. Govindan, and D. Estrin, "Computing aggregates for monitoring wireless sensor networks," in *2003 IEEE International Workshop on Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE*, 2003, pp. 139–148.

Curriculum Vitae

Florence Bénézit

Chemin de Rionza 5
1020 Renens
Switzerland

Audiovisual Communications Laboratory (LCAV)
School of Computer and Communication Sciences (I&C)
Ecole Polytechnique Fédérale de Lausanne (EPFL)
BC322 - Station 14
1015 Lausanne, Switzerland

Email: `florence.benezit@epfl.ch`
Web: `http://lcavwww.epfl.ch/~benezit`

Personal

Date of birth:	April 24, 1981.
Nationality:	French.

Education

2005 - 2009 Ph.D. student in Computer and Communication Sciences, Audiovisual Communications Laboratory (LCAV), School of Computer and Communication Sciences, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland.

2001 - 2005 Ecole Polytechnique, France.

1998 - 2001 Preparation classes in mathematics and physics at Lycée privé Sainte Geneviève de Versailles, France.

Professional Experience

2005 - present **Research and Teaching Assistant**, Audiovisual Communications Laboratory (LCAV), I&C, EPFL, Switzerland.
Research in distributed average consensus algorithms

Assisted students in undergraduate and graduate courses
Supervised students for Master thesis projects

2007 **Internship at Wireless Foundations**, University of California, Berkeley, California, USA
Research on information dissemination in wireless sensor networks, supervised by Prof. Kannan Ramchandran.

2004 **Internship in the GRASP laboratory**, University of Pennsylvania, Philadelphia, USA
Research on spectral methods for image segmentation, supervised by Prof. Jianbo Shi.

Publications

Journal papers

1. F. Bénézit, P. Thiran and M. Vetterli, "Four states are sufficient to classify density in any connected network," in preparation, 2009.

2. F. Bénézit, A. Dimakis, P. Thiran and M. Vetterli, "Order-Optimal Consensus through Randomized Path Averaging," submitted to *IEEE Transaction on Information Theory*, 2008.

Conference papers

1. F. Bénézit, P. Thiran and M. Vetterli, "Interval consensus: From quantized gossip to voting," *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2009.

2. P. Denantes, F. Bénézit, P. Thiran and M. Vetterli, "Which Distributed Averaging Algorithms Should I Choose for my Sensor Network?," *Proc. IEEE Infocom*, 2008.

3. F. Bénézit, P. Denantes, A. Dimakis, P. Thiran and M. Vetterli, "Reaching consensus about gossip, convergence times and costs," *Workshop ITA*, 2008.

4. F. Bénézit, A. Dimakis, P. Thiran and M. Vetterli, "Geographic gossip with path averaging is order optimal," *Allerton*, 2007.

5. T. Cour, F. Bénézit, J. Shi, "Spectral segmentation with multiscale graph decomposition," *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.