



universität
wien

DISSERTATION

Titel der Dissertation

„Towards Truly Distributed Computing: Uniting Theory,
Algorithms and Practice“

Verfasser

Gerhard Niederbrucker

angestrebter akademischer Grad

Doktor der technischen Wissenschaften (Dr. techn.)

Wien, 2014

Studienkennzahl lt. Studienblatt: A 786 880

Dissertationsgebiet lt. Studienblatt: Informatik

Betreuerin / Betreuer: Assoz. Prof. Dr. Wilfried Gansterer

Gewidmet meiner Großmutter,
CHRISTINE SCHACHERMAYER († 2011),
welcher nicht der Lebensabend vergönnt war,
den sie sich stets wünschte.

Für alle „entartischen“ Gedanken,
alle „Schwarze“, „Bummerl“, „Steckerl“,
und viele andere Dinge.

Zusammenfassung

Wissenschaftliche Berechnungen profitieren heutzutage oft von der Leistungsfähigkeit (großer) Parallelrechner. Solch parallele Systeme beeindrucken durch schnelle Gleitpunktarithmetik, tiefe Speicherhierarchien und ausgeklügelte Verbindungsnetze. Darüber hinaus können durch entsprechende Programmiertechnologien die Eigenschaften solcher Systeme zur Gänze kontrolliert und ausgeschöpft werden. Als Gegenstück zu parallelen Systemen betrachten wir verteilte Systeme (z. B. Sensornetzwerke), in welchen keinerlei globale Kontrollmechanismen verfügbar sind und die Kommunikation auf benachbarte Knoten beschränkt ist. Darüber hinaus betrachten wir unzuverlässige verteilte Systeme, wo etwa die Kommunikation (unbemerkt) fehlschlagen kann. Daher führen selbst einfachste Fragestellungen zu herausfordernden Problemen. Unsere Untersuchungen beschränken sich dabei nicht bloß auf klassische Ausprägungen verteilter Systeme, sondern zielen insbesondere auch auf zukünftige Höchstleistungsrechner ab, welchen zunehmend Eigenschaften verteilter System prophezeit werden.

Unser Hauptaugenmerk liegt auf dem Bereitstellen beweisbar robuster, fundamentaler Berechnungsroutinen unter realistischen Annahmen bezüglich (der Zuverlässigkeit von) verteilten Umgebungen. Im Zuge dessen behalten wir stets sowohl theoretische als auch praktische Aspekte im Auge. Auf diese Weise decken wir Probleme hinsichtlich der praktischen Anwendbarkeit bestimmter Methoden und deren Analysen auf. Unsere Beiträge bestehen unter anderem aus einer Konvergenztheorie für eine große Klasse verteilter Aggregationsalgorithmen, welche, im Gegensatz zu existierenden Resultaten, einzig auf lokalen Annahmen beruht. Darauf aufbauend zeigen wir, wie man generisch beweisbar robuste Algorithmen ableiten kann, welche den Herausforderungen in realen verteilten Systemen gewachsen sind. Aufgrund der iterativen Natur der betrachteten Algorithmen ist es naheliegend zu fragen, wie ein Knoten lokal das Erreichen der Zielgenauigkeit feststellen kann. Wir untersuchen dieses gemeinhin vernachlässigte Problem und tragen Methoden zur rein lokalen Genauigkeitsbestimmung bei.

Zusätzlich zum Hauptteil dieser Arbeit, aber auf demselben mathematischen Hintergrund aufbauend, entwickeln wir schnelle und genaue iterative Löser für ein großes strukturiertes Eigenwertproblem aus der Biochemie. Konkret betrachten wir die Berechnung der Quasispezies im sogenannten Quasispezies Modell, welche gleichbedeutend mit dem Auffinden des dominierenden Eigenvektors einer speziell strukturierten Matrix ist. Unsere Resultate stellen dabei ein Tandem aus neuartigen, die Problemstruktur einbeziehenden Algorithmen und deren effizienter (paralleler) Implementierung dar. In Summe können wir dadurch die handhabbare Problemdimension substantiell erhöhen und gleichzeitig auch allgemeinere Modellannahmen behandeln.

Abstract

Today, scientific computations often benefit from the computational power of (large scale) parallel computers. Such parallel systems impress by fast floating-point arithmetic, deep memory hierarchies and sophisticated interconnection networks. Moreover, appropriate programming technologies allow to fully exploit and control the entire functionality of these platforms. As a counterpart to parallel systems, we consider distributed systems (e.g., sensor networks) which do not provide global control mechanisms and where the communication capabilities are limited to simple point-to-point communication between neighboring nodes. Furthermore, we consider unreliable distributed systems, e.g., the communication between nodes may (silently) fail. Therefore, even aiming at imitating elementary parallel computations on a distributed system raises many challenging problems. Our investigations do not only target classical instances of distributed systems, such as sensor networks, but also future high performance systems, due to the predicted distributed nature of these systems.

Our central objective is to provide provably reliable fundamental computing functionality, such as the distributed computation of aggregates, under realistic assumptions on (the reliability of) distributed environments. In the course of that, we consistently take theoretical as well as practical points of view into account. In doing so, we reveal several issues concerning the practical applicability of existing methods and their analyses. Our contributions consist among others of a convergence theory for a large class of distributed aggregation algorithms, which—in contrast to existing work—solely builds upon purely local assumptions. On top of that, we show how to generically derive provably reliable algorithms which can successfully cope with the challenges posed by real distributed systems. Due to the iterative nature of the algorithms we consider, the question how a node can locally reason whether or not it already reached a certain target accuracy, is apparent. We investigate this commonly neglected problem and contribute methods for purely local accuracy estimation.

In addition to the main thread of this thesis—but building upon the same mathematical background—we develop fast and accurate iterative solvers for a large scale structured eigenvalue problem arising in computational biology. More specifically, we consider the computation of the quasispecies in the so-called quasispecies model, which is tantamount to finding the dominating eigenvector of a certain data-sparse matrix. Our results constitute a tandem of novel structure-aware algorithms and their efficient (parallel) implementation on modern hardware. Altogether, we substantially raise the computationally tractable problem dimensions and generalize the applicable model assumptions.

Contents

Preface	xiii
1. Introduction	1
1.1. From Sensor Networks to Exascale Computing	2
1.2. Spreading and Aggregating Information	4
1.3. A Guide for Readers	6
1.3.1. Thesis Organization	6
1.3.2. Overview and Contributions	7
1.3.3. Prior Peer-Reviewed Publications	8
2. Parallel vs. Distributed Computing. What it is (not) about.	11
2.1. Definitions and Differentiation	11
2.1.1. Overview	11
2.1.2. Similarities and Differences	12
2.2. From Parallel Towards Distributed Computing	14
2.2.1. Parallel All-To-All Aggregation	15
2.2.2. Distributed All-To-All Aggregation	16
2.2.3. Challenges	17
2.3. Epidemic Algorithms: Merits and Challenges	18
2.3.1. Notes on Rumor Spreading	19
2.3.2. From Rumor Spreading to Aggregation	21
Lessons Learned: Parallel vs. Distributed Computing	24
3. Vector Iterations vs. Distributed Algorithms. Assumptions and Implications.	25
3.1. Background and Related Problems	26
3.1.1. Linear Iterations and Beyond	26
3.1.2. Related Problems	28
3.2. Two Faces of the Power Iteration	28
3.2.1. Power Iteration as Eigenvalue Solver	29
3.2.2. Power Iteration as Analytical Tool	32
3.3. (Un)Realistic Assumptions and Their Implications	33
3.3.1. Algorithm vs. Analysis	33
3.3.2. Communication	34
3.3.3. Asynchronicity	36

3.3.4. Topology	38
3.3.5. Convergence Behavior	39
3.4. Convergence Under Realistic Assumptions	40
3.5. Notes on Convergence Acceleration	44
3.6. Beyond Computing Simple Aggregates	46
3.6.1. Repeated Aggregations	46
3.6.2. Broadcasting-based Algorithms	47
Lessons Learned: Algorithm, Iteration, Analysis	48
4. Computing in Unreliable Environments. Theory, Algorithms & Practice.	49
4.1. Fault Tolerance in Theory and Practice	50
4.1.1. Fault Tolerance: Notions & Requirements	50
4.1.2. Robust Distributed Aggregation Algorithms	51
4.1.3. Fault Tolerance Methods in Parallel Computing	53
4.2. Fault Tolerant Algorithms: Derivation & Analysis	55
4.2.1. Derivation of Flow-based Algorithms	55
4.2.2. Analysis of Flow-based Algorithms	58
4.2.3. Derivation of History-based Algorithms	63
4.2.4. Analysis of History-based Algorithms	66
4.2.5. Recovery Overhead: Insights and Bounds	69
4.3. Coping with Asynchronicity	73
4.4. From Theory to Practice	76
4.4.1. Weaknesses of Existing Methods	76
4.4.2. Improving Fault Tolerance and Accuracy	79
4.4.3. An Excursion to Silent Data Corruption	84
Lessons Learned: Provable Robustness on Detours	86
5. while (not ε-accurate) do When to stop a computation (locally)?	87
5.1. Objectives and Prerequisites	88
5.2. Cooperative Parameter Estimation	91
5.2.1. Estimation of c	91
5.2.2. Estimation of r	91
5.2.3. A Minimalist Approach for Accuracy Estimation	92
5.3. Keeping Means: Bound and Float	92
5.3.1. Overview	92
5.3.2. Auxiliary Results	93
5.3.3. Bound-and-Float Algorithm	96
5.4. Experimental Results	97
5.4.1. Methodology	97
5.4.2. Experiments: Simple Parameter Estimation	100
5.4.3. Experiments: Bound-and-Float Algorithm	104
5.4.4. Interpretation of Results	108
Lessons Learned: (In)Sufficiency of Local Methods	109

6. The Power Iteration in HPC. A case study from biochemistry.	111
6.1. The Quasispecies Model	111
6.1.1. Overview	111
6.1.2. Model Definition and Biochemical Background	112
6.1.3. Existing Approaches vs. Contributions	115
6.2. Theoretical Insights and Fast Algorithms	116
6.2.1. Structured Implicit Sparsification	116
6.2.2. Fast Mutation Matrix Product	119
6.2.3. Computing the Quasispecies	121
6.2.4. Limitations of Power-Iteration Approaches	123
6.3. Parallel Quasispecies Computations	124
6.3.1. Implementation	124
6.3.2. Experimental Results	126
6.4. More General vs. More Structured Models	128
6.4.1. Hamming Distance-based Matrices	128
6.4.2. More General Mutation Processes	130
6.4.3. Utilizing Structured Fitness Landscapes	132
Lessons Learned: Quasispecies Model Meets HPC	137
7. Summary and Conclusions	139
7.1. Contributions	140
7.2. Open Problems and Future Work	142
A. Mathematical Background	143
A.1. Notation and Fundamentals	143
A.1.1. Linear Algebra	143
A.1.2. Miscellaneous	145
A.2. The Kronecker Product	145
A.2.1. Basic Properties	146
A.2.2. Kronecker Products and Fast Algorithms	150
A.3. Perron-Frobenius Theory	154
A.4. Graph Structures: Definitions and Properties	158
A.4.1. Connectivity Measures	159
A.4.2. Graph Structures	160
Listings	163
Bibliography	165
Index	179
Curriculum Vitae	181

Preface

Once upon a time in fall 2008, I wanted to attend a seminar on parallel computing at the Vienna University of Technology. Officially, I never succeeded. On the other hand—over the last couple of years—this unimposing seminar I have never gotten a certificate for radically changed my academic and personal life. I appreciate Christoph Überhuber for initiating more fruitful collaborations than he actually intended to.

This thesis and the research presented therein would not exist in its present form without the kind guidance of my advisor Wilfried Gansterer. I would like to thank him for giving me the chance to work within his research group, as well as for his support I received constantly over the last couple of years. Moreover, I am grateful for the funding I received during my time at the University of Vienna from the Austrian Science Fund (FWF) under project number S10608 (NFN SISE).

This thesis constitutes the final milestone of my long way through the educational system. Hence, I want to thank those who were responsible for the “ups” which let me overcome the “downs” I faced throughout the years. I would like to thank Arno Graspöck for being the first teacher whose classes I was really looking forward to. His absence in my last year of compulsory school—as well as the presence of his substitute—I will always regret. It is questionable whether I would have ever entered an university without having participated at the *Austrian/International Olympiad in Informatics*. My thanks go to the (former) contributors to the training camps and national competitions in Austria, Helmut Achleitner, Johann Fellner, Karl Josef Fuchs, and Gerald Futschek, for many inspiring moments and their dedication to the support of young people. Furthermore, I want to thank all *sheep* who contributed to five truly memorable years at my by this time second home *Guter Hirte*.

Last but not least, I want to wholeheartedly thank my parents and *moji malé rodině* for the invaluable support I received in the years prior to writing these lines.

Vienna, April 2014

Gerhard Niederbrucker

1 | Introduction

Complex scientific computations commonly break down to simple elementary operations. This holds especially for numerical linear algebra, which constitutes the core of most scientific computations. Nowadays, mainstream numerical linear algebra libraries build upon a common interface for computing the required elementary matrix/vector operations—the so-called *Basic Linear Algebra Subprograms* (BLAS) [Dongarra *et al.*, 2002a,b]. More specifically, the de-facto standard of BLAS allows to design generic numerical linear algebra libraries, which in the background utilize a highly tuned BLAS implementation in order to obtain high performance. Today, there are many sequential as well as parallel BLAS implementations available, including implementations from vendors specifically tuned for their respective hardware. The operations within BLAS are organized in three levels which build upon each other.

Level	Type of operations	Available operations (among others)
1	vector	$y = \alpha x + y$, $x^T y$, vector norms
2	matrix-vector	$y = \alpha Ax + \beta y$, $y = Tx$ for T triangular
3	matrix-matrix	$C = \alpha AB + \beta C$

Conceptually, it is easily verified that all BLAS operations break down to vector operations. In particular, a matrix-matrix multiplication can be seen as repeated matrix-vector multiplication, which itself can be seen as a repeated computation of dot products, as illustrated below.

$$C = AB \rightsquigarrow \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} b_{1,1} & \dots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \dots & b_{n,n} \end{pmatrix} \rightsquigarrow \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} c_{1,1} & \dots & c_{1,n} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \dots & c_{n,n} \end{pmatrix} \rightsquigarrow \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} c_{1,1} & \dots & c_{1,n} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \dots & c_{n,n} \end{pmatrix}$$

Therefore, comprehensive linear algebra functionality can be in principle provided whenever vector operations are available. However, in order to achieve maximal performance, dedicated higher level routines are required in general.

From a practical point of view, one key contribution of this thesis is the development of provably robust vector operations, which require substantially weaker assumptions on (the reliability of) the environment than existing methods do. Following the considerations above, this is the crucial step in order to enable numerical linear algebra functionality under conditions where existing approaches are not applicable or fail.

In parallel computing one usually distinguishes *shared memory* and *distributed memory* systems. In a shared memory system the (possibly many) processing units can all access the entire memory. Opposed to that, a distributed memory system consists of (many) nodes which all host one or more processing units and a local memory. Hence, in order to perform a (non-trivial) parallel computation on a distributed memory system, (physical) communication between the nodes is required.

In the remainder of this thesis we mostly consider *distributed systems*, which we strictly distinguish from distributed memory (parallel) systems (see [Section 2.1](#)). For short, in case of parallel systems we assume that all system properties can be accessed and modified via appropriate programming techniques. In particular, we assume the availability of global control mechanisms, e.g., for (global) node synchronization. In contrast to that, we speak about a distributed system, whenever the nodes within the system act autonomously without having access to global information and control mechanisms. Moreover, we additionally consider distributed systems to be unreliable, in contrast to parallel systems (e.g., the communication between nodes may fail). Altogether, our assumptions rule out standard (parallel) approaches towards providing numerical linear algebra functionality on a distributed system, due to the lack of (global) control and the unreliability of certain operations.

We sketched above how numerical linear algebra breaks down to elementary vector operations (by sacrificing peak performance). In particular, these considerations extend to the case, where the data is distributed among the nodes. Consequently, the crucial operation for providing comprehensive numerical linear algebra functionality on a distributed system is the (distributed) computation of a dot product. With respect to our assumptions on a distributed system, this kind of computations involve several non-trivial problems which are not solved so far. When building complex functionality based on very simple operations, one of course hopes that advances (e.g., fault tolerance properties) of the low level operations translate up to higher levels. The fact that the (beneficial) properties of the robust low level operations we develop in this thesis indeed can be extended to higher level (matrix) operations has been extensively investigated before (see [Straková \[2013\]](#) and the references therein).

Besides our actual aim to enable numerical linear algebra functionality under conditions where existing approaches cannot be applied or fail, we also investigate in this thesis how the developed methods can be employed in other contexts, such as load balancing. In particular, our results on fault tolerance can be partially carried over to related problems for which fault tolerance aspects of the kind presented in this thesis have not been considered yet.

1.1. From Sensor Networks to Exascale Computing

No matter how a computer system looks, its primary purpose is to solve suitable computational problems. Depending on the particular capabilities and dimensions of a system, the tractable problems differ widely. In our investigations we target

distributed systems which do not provide global control mechanisms and in which the reliability of, for instance, the communication cannot be guaranteed. In order to successfully cope with the unreliability and decentralized nature of such systems, we develop throughout this thesis novel algorithmic approaches for fundamental problems such as the robust distributed computation of aggregates.

Among others, *sensor networks* are an important representative of the low-end of the computing landscape. In sensor networks it is often assumed that the individual sensors are very limited in their capabilities and that they can only exchange information within their immediate *neighborhood*, e.g., the nodes within their radio transmission range. Commonly, the overall objective consists of computing a network-wide quantity based on *local* measurements, *local* computations and *local* information exchanges (within the neighborhood). Such an objective can be as simple as tracking the average temperature within a large building. On the other hand, also more subtle tasks such as reconstructing a (physical) field based on local measurements are of interest. While the quest for computing functionality on sensor networks received a lot of attention, we depict later the insufficiency of existing approaches (given our assumptions and requirements). Subsequently, we also develop novel methods which meet the challenging conditions we pose.

Opposed to the low-end, we see today's fastest high performance computers at the high-end of the computing landscape. Currently, the—with respect to the LINPACK benchmark [Dongarra, Luszczek, *et al.*, 2003]—fastest computers in the world¹ deliver a sustained performance of several to tens of PFlop/s, where one PFlop/s is tantamount to executing 10^{15} (peta) floating-point operations per second. High-end machines of this scale are complex systems which require sophisticated methods throughout the whole software stack in order to achieve a significant percentage of the theoretical peak performance. Much efforts of current fundamental high performance computing research are devoted to the objective of building a so-called *exascale computer*, which reaches a sustained performance of 10^{18} (exa) Flop/s.

At first glance, there is seemingly not much what would relate sensor networks and a future exascale computer which can perform 10^{18} floating operations per second. Yet, a closer look reveals that the opposite is true. In contrast to earlier milestones in high performance computing, the transition to exascale is expected to be disruptive [Dongarra, Beckman, *et al.*, 2011] in the sense that a major redesign of the hardware as well as the corresponding software stack is believed to be required. This development is mostly due to the target power consumption of about 20 megawatt² and (physical) limitations in circuit design. Especially the target power consumption is crucial, as it is witnessed by the Top 500 list of supercomputers. In particular, the currently fastest computer in the world, *Tianhe-2*, requires already now about 18 megawatt.

¹See <http://www.top500.org/list/2013/06/>, Top 500 list of supercomputers, edition November 2013.

²This challenging power consumption target is primarily pushed by the US Department of Energy and the institutions it funds. Clearly, exascale efforts in countries such as China or India may not adhere to this restriction in their quest for building the first exascale system in the world. See Tolentino and Cameron [2012] for a discussion of this issue.

Mostly due to the harsh energy restrictions, a slight shift from traditional tightly coupled high performance systems towards decentralized distributed systems may be required in the future. Among others, the sheer scale and anticipated technology used to build an exascale computer will most likely lead to a substantially increased failure frequency compared to nowadays systems [Dongarra, Beckman, *et al.*, 2011]. Hence, the robust distributed methods we develop in this thesis (primarily for low-end distributed systems) are potentially also of interest in the context of future high performance systems. Fault tolerance, in particular also at the algorithmic level, has been a central pillar of exascale computing research from the very beginning [Cappello *et al.*, 2009]. The necessity of substantially improved fault tolerance techniques throughout all levels was also manifested in widely considered exascale roadmaps [Dongarra, Beckman, *et al.*, 2011]. Recent parallel computing research already reflects this trend towards the commonly expected higher robustness requirements of future high performance systems. A detailed overview on this subject follows in [Section 4.1.3](#).

We have previously seen that, at least conceptually, providing comprehensive numerical linear algebra functionality for an unreliable decentralized system solely relies on providing an appropriate distributed dot product, which is nothing else than a distributed weighted summation. For various reasons (discussed later), existing methods which address this problem cannot (to the best of our knowledge) cope with the challenges raised by the setup considered throughout this thesis.

As a building block for our developments we consider (existing) epidemic-style aggregation algorithms because of the high degree of flexibility they provide. Our contributions consist of exploiting this flexibility algorithmically in order to eventually obtain robust methods which meet all our requirements. In the course of that, we also cope with several practical subtleties, e. g., how to determine locally at a node, whether or not a prescribed target accuracy has been reached.

1.2. Spreading and Aggregating Information

In a distributed computation usually every node provides a piece of initial information which contributes to the overall result. Many non-trivial computations, e. g., a distributed summation, require that all nodes exchange their initial information in order to obtain the correct result (at every node). The challenge with respect to this requirement is that in distributed systems of the kind we consider, the nodes can only communicate with their immediate neighbors and that they do not have any information about the nodes outside of their neighborhood.

A popular problem in (theoretical) computer science is that a single node aims at spreading its local information to all other nodes in the network. This problem, commonly known as *rumor spreading*, attracted a lot of interest over the last two decades and has been analyzed into great detail (see [Section 2.3.1](#) for an overview). Very simple epidemic-style processes turned out to be very successful in providing robust solutions which—for particular network topologies—can even attain theoretical

lower bounds with respect to time and communication complexity. The natural “dual” of spreading information is the *aggregation* of information. Suppose that the nodes in a rumor spreading process save the neighboring node who informed them first about the rumor to be spread over the network. This additional relationship leads to a tree rooted at the node who initially knows about the rumor, because all other nodes keep exactly one node who informed them first about the rumor. Furthermore, the communication pattern this tree describes can be used to aggregate local data of all nodes at the root (i.e., the node who initially kept the rumor). Figure 1.1 depicts this relationship between rumor spreading and aggregation visually.

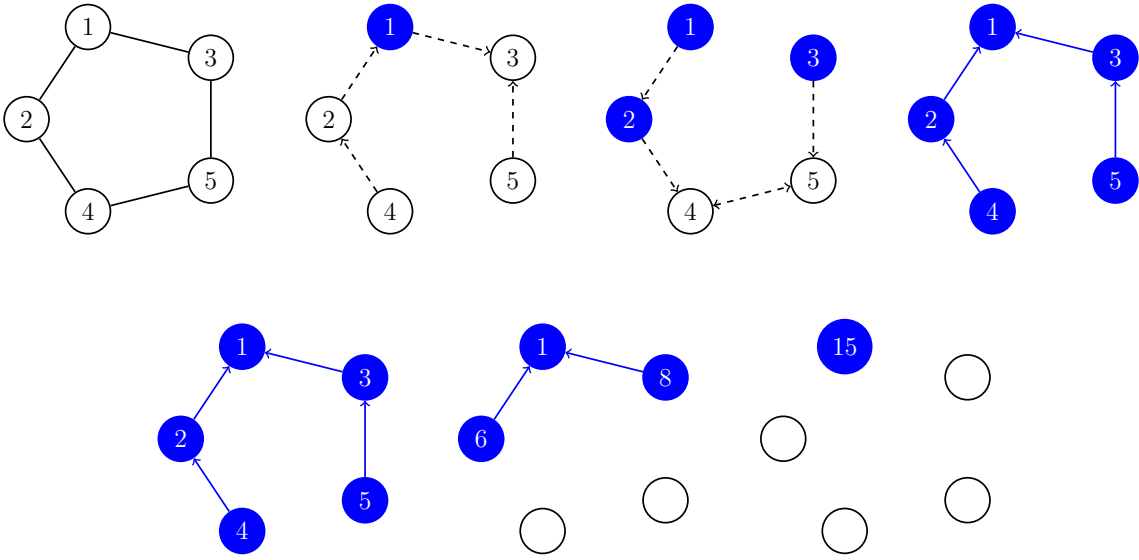


Figure 1.1.: We consider a (ring-)network of five nodes (top left). The remainder of the first row illustrates how the rumor (filled nodes) is spread from node one to all other nodes. Dashed arrows indicate which node is called and the direction of an arrow indicates if the rumor is pushed or pulled (cf. Section 2.3.1). Solid arrows point to the node which informed a particular node first about the rumor. In the second row we depict that by reverting the order of communications in the rumor spreading process, we obtain an aggregation—here a summation of the node identifiers.

Even though rumor spreading and aggregation are closely related problems, they are—at least from a theoretical point of view—not equally difficult. In particular, aggregation is provably more difficult than rumor spreading [Chen and Pandurangan, 2012]. While this distinction is by itself not crucial for our results, we elaborate in Section 2.3.2 on this topic in order to work out the subtleties behind the problem of aggregation, especially in the presence of failures.

In a distributed setting, aggregation in the spirit as we depicted it so far in this section fails, due to the lack of global control. Moreover, we are usually interested in so-called all-to-all aggregations where not only one, but all of the nodes obtain the target aggregate. At first glance, problems of this kind seem to be substantially more difficult than simple rumor spreading. In contrast to that, it turns out that efficient all-to-all aggregation algorithms can be built upon the skeleton of certain epidemic-style rumor spreading processes enriched by some local logic for the aggregate computation. The simplicity and flexibility in the design of such algorithms allow us to develop a rich fault tolerance methodology on top of methods which neglect the possibility of failures in the system. Moreover, the close connection between spreading and aggregating information also motivates the rule of thumb that networks where rumors spread quickly also allow for efficient computation of aggregates.

1.3. A Guide for Readers

In the following we provide a concise overview of the structure and key contributions of this thesis. Concretely, we depict the organization of this thesis, outline the individual chapters together with the key contributions developed therein, and give an overview of the (previously appeared) peer-reviewed publications this thesis builds upon.

1.3.1. Thesis Organization

This thesis is partitioned into the main text (Chapters 1–5), containing novel contributions regarding distributed computing problems, a “bonus chapter” (Chapter 6), which depicts parallel computing research complementing the main thread of this thesis, a concise summary of the entire text (Chapter 7), and a comprehensive exposition of the mathematical background (Appendix A) required for this thesis.

While the upcoming Chapter 2 does not yet contain novel technical contributions, its value lies in setting up the thesis in terms of motivation, focus, and objectives. Moreover, in Chapter 2 we shed light on various aspects which are commonly neglected in the literature. Our technical contributions together with a detailed discussion and distinction from existing work follow in Chapters 3–5 as well as in Chapter 6. Additionally to the actual content, all but the current chapter also end with a concise summary of the main developments and insights gained within the respective chapter. At the end, Chapter 7 concludes the entire thesis and outlines open questions for future research. The mathematical background in Appendix A is added as a source of reference, in order to allow for a self-contained exposition. Familiarity with these standard notions and basic results is assumed, which can—as needed—be obtained by reading (parts of) Appendix A.

For the sake of convenience, complete lists of figures, tables, and algorithms, as well as a comprehensive index of technical terms and symbols can be found at the end of

this thesis. Moreover, the appended bibliography contains additionally to standard bibliographic data also back references to the actual citations within the text.

1.3.2. Overview and Contributions

In [Chapter 2](#) we first discuss how we strictly distinguish between the terms “parallel” and “distributed” computing throughout this text. This is of high importance, since especially the usage of the term “distributed computing” in different scientific communities causes severe ambiguities. Based on the aforementioned differentiation between parallel and distributed computing, we show how the step from parallel to distributed computing can conceptually be accomplished, which possibilities this step offers, and which challenges it causes. Hence, we investigate how basic parallel computing functionality can be provided on truly distributed systems. In order to be able to provide basic vector operations in a distributed way, we consider utilizing epidemic-style algorithms which require substantially fewer assumptions on the (reliability of the) environment. In the course of that, we identify two central questions we subsequently investigate in this thesis:

1. Up to which extent can the flexibility of epidemic algorithms be exploited, e. g., in order to obtain high fault tolerance at the algorithmic level?
2. How can we decide locally when to terminate an epidemic-style iterative algorithm, i. e., how can a node autonomously deduce when its local result meets a predefined target accuracy?

Accordingly, the key contributions of this thesis are centered around these two questions. In particular, [Chapter 3](#) and [Chapter 4](#) provide novel contributions regarding the first question, whereas [Chapter 5](#) is devoted to approaches coping with the second question.

In [Chapter 3](#) we focus on the (convergence) analyses of certain classes of distributed aggregation algorithms. Our contributions in this context are twofold. First, we review existing algorithms and their analyses in order to point out the practical limitations which commonly made assumptions cause. In particular, we depict that even in case of a failure-free environment, most existing methods suffer from too strong assumptions on the environment. Consequently, as a second step, we present a convergence theory for a class of distributed aggregation algorithms assuming only properties which we can also guarantee in practical distributed environments. Moreover, we discuss the generic applicability of convergence acceleration techniques and how to systematically build higher level operations based on simple aggregations.

While the aforementioned convergence theory is of high importance by its own, it is also crucial for the fault tolerance techniques we develop and analyze in [Chapter 4](#). Succeeding an exposition of current fault tolerance techniques, we generically show how to derive fault tolerant variants of all algorithms contained in the class of distributed algorithms considered in [Chapter 3](#). In the course of that, we employ two different fault

tolerance approaches with slightly different properties and analyze them. Especially our analysis of the derived fault tolerant algorithms in an unreliable environment substantially improves prior related results. We further elaborate on aspects regarding the transition from theory to computational practice. In this context we discover several subtleties, e.g., caused by floating-point arithmetic, and show how handle (most of) them.

All (iterative) distributed algorithms which are considered in this thesis produce a (converging) sequence of estimates of the target value locally at each node. Naturally, we assume that neither of the nodes knows the correct result and therefore, the nodes also cannot directly judge the quality of their local estimates. In [Chapter 5](#) we develop methods which allow a node to locally predict the accuracy it achieved so far. Despite the widespread application of iterative distributed algorithms this problem received surprisingly little attention in the existing literature. We present two conceptually different approaches and evaluate their strengths and weaknesses.

Complementary to the main thread of this thesis, we depict in [Chapter 6](#) recent high performance computing research with the sole objective to increase the performance and tractable problem dimensions in case of an application problem from biochemistry. It is particularly interesting to observe that we employ in this context several of the formal results and tools which we also utilize before in a more theoretical context.

1.3.3. Prior Peer-Reviewed Publications

This thesis comprises a conglomerate of prior peer-reviewed publications enriched by plenty of supplementary material and novel insights into existing work. Concretely, this dissertation project has already led to the following peer-reviewed publications.

Gansterer, W.N., Niederbrucker, G., Straková, H., and Schulze Grotthoff, S. (2011). “Robust Distributed Orthogonalization Based on Randomized Aggregation”. In: *Proceedings of the Second Workshop on Scalable Algorithms for Large-Scale Systems*. ScalA 2011, pp. 7–10 (see pp. [52](#), [55](#), [57](#)).

DOI: [10.1145/2133173.2133177](#)

Gansterer, W.N., Niederbrucker, G., Straková, H., and Schulze Grotthoff, S. (2013). “Scalable and Fault Tolerant Orthogonalization Based on Randomized Distributed Data Aggregation”. In: *Journal of Computational Science* **4**(6), pp. 480–488 (see pp. [46](#), [51](#), [52](#), [55](#), [57](#)).

DOI: [10.1016/j.jocs.2013.01.006](#)

Niederbrucker, G. and Gansterer, W.N. (2014). *Robust Distributed Aggregation*. (under revision, IEEE Transactions on Signal Processing) (see pp. [40](#), [51](#), [55](#), [58](#)).

Niederbrucker, G. and Gansterer, W.N. (2011a). “A Fast Solver for Modeling the Evolution of Virus Populations”. In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. SC 2011, 74:1–74:11 (see pp. [111](#), [124](#), [132](#), [137](#)).

DOI: [10.1145/2063384.2063483](#)

Niederbrucker, G. and Gansterer, W.N. (2011). “Efficient Solution of Evolution Models for Virus Populations”. In: *Procedia Computer Science* **4**. Proceedings of the International Conference on Computational Science, ICCS 2011, pp. 126–135 (see pp. [116](#), [126](#), [137](#)).

DOI: [10.1016/j.procs.2011.04.014](https://doi.org/10.1016/j.procs.2011.04.014)

Niederbrucker, G. and Gansterer, W.N. (2013). “Robust Gossip-Based Aggregation: A Practical Point of View”. In: *Proceedings of the 2013 Meeting on Algorithm Engineering & Experiments*. ALENEX 2013, pp. 133–147 (see pp. [50](#), [76](#), [84](#)).

URL: <http://knowledgecenter.siam.org/0238-000028/>

Niederbrucker, G., Straková, H., and Gansterer, W.N. (2012). “Improving Fault Tolerance and Accuracy of Distributed Reduction Algorithms”. In: *Proceedings of the Third Workshop on Scalable algorithms for Large-Scale Systems*. ScalA 2012, pp. 7–10 (see pp. [53](#), [76](#), [79](#), [81](#)).

DOI: [10.1109/SC.Companion.2012.89](https://doi.org/10.1109/SC.Companion.2012.89)

Straková, H., Niederbrucker, G., and Gansterer, W.N. (2013). “Fault Tolerance Properties of Gossip-Based Distributed Orthogonal Iteration Methods”. In: *Procedia Computer Science* **18**, pp. 189–198 (see p. [46](#)).

DOI: [10.1016/j.procs.2013.05.182](https://doi.org/10.1016/j.procs.2013.05.182)

Moreover, throughout this thesis we also utilize several results and especially supplementary material from the thesis

Niederbrucker, G. (2013). “Structured Preconditioners for Matrix-free Iterative Methods”. Master’s Thesis. Vienna University of Technology, Faculty of Mathematics and Geoinformation, Institute of Analysis and Scientific Computing (see pp. [111](#), [123](#), [128](#), [130](#), [131](#), [133](#), [137](#), [145](#), [146](#), [154](#)).

URL: <http://permalink.obvsg.at/AC11105284>

Self-citation policy. Henceforth, whenever an entire (sub-)section originates in—and thus strongly overlaps with—a previously published text, its title is accompanied by an appropriate reference to the respective published text (cf. [Section 3.4](#) for the first section of this kind).

2 | Parallel vs. Distributed Computing

What it is (not) about

In this chapter, we first elaborate on a clear differentiation between the notions of parallel and distributed systems/algorithms. Thereafter, we look at the fundamental problem of data aggregation from a parallel computing as well as from a distributed computing point of view. We depict (i) the conceptual similarities between parallel and distributed approaches, (ii) which challenges the transition from the parallel to the distributed regime causes, and (iii) which opportunities this transition enables in terms of algorithm design. We complement these insights by a discussion of the (beneficial) properties of epidemic processes and argue about the commonly undervalued difficulty of distributed aggregation (in practice). Together, all these considerations motivate our technical contributions we develop in the upcoming Chapters 3–5.

2.1. Definitions and Differentiation

Nowadays various kinds of parallel computer architectures are employed which differ in the way of supporting parallelism at the hardware level. In any case, from a high level point of view a parallel computer is nothing else than a computer which can execute several instructions at the same time. In contrast to parallel systems, the notion of distributed systems is very fuzzy due to its use in many scientific communities and contexts. Therefore, in our differentiation between these two notions, we cannot and do not aim at “ultimate correctness”, but rather at a definition what we mean by a distributed system throughout this thesis.

2.1.1. Overview

In comparisons between parallel and distributed systems, we basically think of (large-scale) parallel systems without a globally shared memory. Such so-called *distributed memory parallel systems* comprise of (potentially many) independent processing units

which are interconnected with each other. In programming this kind of parallel systems, MPI (*Message Passing Interface*, see [Snir et al. \[1998\]](#))¹ constitutes the de-facto standard for the communication between the processing units. The utilization of MPI allows to develop scalable codes where the number of concurrently executed processes can be specified at runtime, e. g., based on the available resources. No matter whether two processes are executed on the same physical unit, they can exclusively communicate by MPI primitives. In particular, each process can communicate with each other process, independently of the topology of the underlying physical network. Besides basic point-to-point communication functionality, MPI also provides so-called *collective operations*, which can involve many or even all processes. In particular, MPI provides functionality for the global dissemination and aggregation of data.

For the sake of an easy comparison to distributed systems, we look at parallel systems from an MPI perspective. This means that whenever we talk about a parallel system we identify individual MPI processes as separate units of the system (which communicate with each other via MPI)². This means we neglect the ability of real systems to host several MPI processes on a single physical unit. In such cases MPI implementations internally switch between different routines depending if physical communication is required or not. We also neglect in our simplified treatment of parallel systems any form of parallelism below the level of MPI.

In contrast to parallel systems, we assume distributed systems to be much less tightly coupled. Frequently considered examples of distributed systems are (wireless) sensor networks, peer-to-peer networks or (mobile) ad-hoc networks. The individual units in such distributed systems act for the most part autonomously and do not have access to global control mechanisms. Another crucial limitation is that we expect that only physically connected nodes can communicate with each other, i. e., we do not rely on routing capabilities or other kinds of multi-hop communication. For instance, in a wireless sensor network the communication capabilities can be limited by the range of the radio communication. Abstractly, we model the topology of the underlying communication network as an undirected or directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the nodes $i \in \mathcal{V}$ represent the individual units in the system and the existence of an edge $(i, j) \in \mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ indicates that node i can communicate with node j . We call the set $\mathcal{N}_i := \{j \in \mathcal{V} \mid (i, j) \in \mathcal{E}\}$ the *neighborhood* of node i and a node $j \in \mathcal{N}_i$ its *neighbor*. Of course, we can also model a parallel system in this way, which always results in a complete graph by our consideration above. Hence, we also use the term node instead of process in the case of parallel systems.

2.1.2. Similarities and Differences

Below, we distinguish in detail between the notions of parallel and distributed systems. [Table 2.1](#) concisely summarizes the substantial differences in our differentiation.

¹An up-to-date version of the MPI standard is available at <http://www.mpi-forum.org>.

²From this point of view, we can also compare shared memory parallel systems with distributed systems.

	Parallel systems	Distributed systems
Topology	fixed, regular and optimized for high performance	arbitrary and potentially dynamically changing
Local knowledge (at each node)	every node can query information about the entire system	every node solely knows about its neighboring nodes
Communication	reliable communication between all (pairs of) nodes	unreliable communication between neighboring nodes only
Global control	available for all nodes at every time	none of the nodes can access global control mechanisms

Table 2.1.: Summary of our assumptions on parallel and distributed systems.

Topology. In general, the nodes in modern parallel systems are statically interconnected in a very regular way, most commonly via some kind of torus network (see, e. g., [Ajima *et al.* \[2009\]](#) and the references therein). As a consequence, customized (vendor-specific) middleware (e. g., MPI implementations [[Adachi *et al.*, 2013](#)]) are usually provided in order to fully exploit the respective hardware. In the case of distributed systems we cannot rely on any regularity assumptions and can in principle face any kind of (connected) topology. Moreover, we do not insist on a static topology but also allow for dynamic changes. In fact, the algorithms we develop even work for topologies which are not strictly connected as long as they are “connected in expectation connected”. What this exactly means is discussed in [Section 3.3.4](#).

Local knowledge. The knowledge about the entire system available locally at each node is another crucial difference between parallel and distributed systems. In the case of parallel systems there are no limitations, i. e., every node can get, or a priori has, all (global) information it requires. In strong contrast to that, we assume the local knowledge in a distributed system to be very limited. More concretely, the nodes only know about their neighborhoods $\mathcal{N}_i$ and how to communicate with the neighbors. A somewhat crucial question is whether the nodes also know about the total number of nodes the entire system consist of (e. g., the nodes in an ad-hoc network do usually not know how many nodes form the system). Unless otherwise stated we always assume that the nodes do not have this knowledge.

Communication. In a parallel system the communication is (on the application level) assumed to be fully reliable. Beyond that there are in general also no limitations in terms of communication between the nodes, independently of the underlying physical network which interconnects the nodes. We consider distributed systems where communication capabilities are restricted to neighboring nodes. This means a node i can only communicate with its (physical) neighbors $\mathcal{N}_i$ and there are no mechanisms

(overlay networks, routing, etc.) which would allow for communication beyond the scope of the neighborhoods (i.e., there is no routing or other kinds of multi-hop communication). Moreover, we assume that the communication between two nodes can fail silently, i.e., messages can get dropped and neither the sender nor the intended receiver are informed about the loss.

Global control. In a parallel system global control mechanisms are available and also frequently used, e.g., barriers for the synchronization of (all) nodes. Due to the limited communication capabilities and the local focus of the nodes, we also assume that there are no global control mechanisms available within a distributed system. This means there are no mechanisms, e.g., for synchronization, which would enable some coordination between the actions of the nodes, i.e., the nodes in a distributed system are for the most part expected to act autonomously.

So far, we did not explicitly incorporate assumptions on the (un)reliability of the individual nodes in our differentiation because the unreliability of nodes is also a widely considered problem in the area of parallel computing (cf. [Section 4.1.3](#)). Consequently, we address in our investigations the problem that the nodes themselves are unreliable, e.g., that a node can crash in the course of a computation.

Altogether, our assumptions on distributed systems clearly rule out the direct utilization of parallel algorithms on distributed systems. However, the upcoming [Section 2.2](#) illustrates that—at least for the problem of aggregation—distributed methods can be obtained as a generalization of their parallel counterparts.

2.2. From Parallel Towards Distributed Computing

The dissemination of information and the computation of aggregate information are fundamental operations for both parallel and distributed computing. We consider in this section so-called *all-to-all* aggregations as particularly important representatives of collective operations. In an all-to-all aggregation every node i provides a piece of data a_i (e.g., $a_i \in \mathbb{R}$) and the objectives are to compute an aggregate function

$$\varphi(a_1, a_2, \dots, a_n) = a_1 \odot a_2 \odot \dots \odot a_n, \quad (2.1)$$

and to deliver its value to all nodes. Here, the operation $\odot$ can be any associative and commutative binary operation (e.g., $a \odot b = a + b$). In the MPI standard also the aggregation with non-commutative operators is specified. However, we always assume commutativity, because non-commutative aggregation is not feasible in a distributed setting, due to the lack of global control.

In a classical long-term field study, [Rabenseifner \[1999\]](#) profiled all executed MPI jobs on a production system and analyzed in which MPI routines these applications spent (most of) their time. A noteworthy result of this profiling study in the context of this thesis is that about forty percent of the overall runtime was spent in the execution of all-to-all operations. The profiling study of [Rabenseifner \[1999\]](#) more generally illustrated that almost all executed MPI jobs made extensive use of collective

operations. However, the importance of well-performing collective operations is not an artifact of past times. On the contrary, the design and implementation of collective operations seems even more crucial on nowadays and future extreme-scale systems [Adachi *et al.*, 2013; Pješivac-Grbović *et al.*, 2007].

Straightforwardly, an all-to-all aggregation can be composed of a simple (tree-based) aggregation, where a single node obtains the result, and a subsequent broadcast of the result to all other nodes. Among conceptual drawbacks such as the introduction of a single point of failure, this simple approach is also inefficient (in practice) [Thakur *et al.*, 2005]. In the following, we illustrate how to efficiently approach all-to-all aggregations from a parallel computing point of view as well as from a distributed computing point of view. In doing so, we especially emphasize that (in some sense) the distributed approach naturally evolves out of the parallel approach by weakening the assumptions on the capabilities of the environment.

2.2.1. Parallel All-To-All Aggregation

The choice of a particular all-to-all algorithm in practice (e.g., within an MPI implementation) depends on many factors including the physically available topology and the required message size [Thakur *et al.*, 2005]. Especially the physical topology of a parallel system needs to be taken into account in order to achieve high sustained performance [Sanders *et al.*, 2009]. For instance, the algorithm we present below has superior theoretical properties but may suffer from problems such as high contention at certain physical communication links.

For the sake of a concise presentation, we assume a system with $n = 2^d$ nodes which are consecutively numbered from $0, \dots, n - 1$. This special case allows for a particularly elegant treatment of collective operations [Johnsson and Ho, 1989]. The case where $n \neq 2^d$ can be handled similarly and does not cause substantially more effort. Since $n = 2^d$, the nodes in the system can be identified with the nodes of the d -dimensional hypercube, i.e., we can restrict³ the neighborhood of node i to $\mathcal{N}_i = \{i \oplus 2^k \mid 0 \leq k < d\}$, where the operator $\oplus$ denotes the binary XOR operator. Johnsson and Ho [1989] first observed that it suffices to communicate exactly once along each edge of the hypercube in order to implement an all-to-all aggregation. Algorithm 2.1 depicts the basic structure of the resulting algorithm. Implicitly, this algorithm is tantamount to n simultaneous tree aggregations rooted at the individual nodes. Clearly, Algorithm 2.1 shows the basic structure of a general all-to-all reduction. The logic required for a particular aggregate function has to be plugged into the step in line 5 of Algorithm 2.1. For instance, $a_i \leftarrow a_i + a_j$ or $a_i \leftarrow \max(a_i, a_j)$ for computing the sum or maximum over all a_k , respectively.

Algorithm 2.1 also neatly illustrates common properties of parallel algorithms. For instance, we see that the communication schedule is fixed and very regular, i.e., each

³Since in a parallel system all nodes can communicate with each other, we can consider an arbitrary “logical” communication network at the algorithmic level.

Algorithm 2.1 Skeleton of a parallel all-to-all aggregation (for $n = 2^d$)

```

1:  $a_i \leftarrow$  initial contribution from node  $i$ 
2: for  $k = 0$  to  $d - 1$  do
3:   send  $r_i$  to node  $i \oplus 2^k$ 
4:   wait for receiving  $a_j$  from node  $i \oplus 2^k$ 
5:    $a_i \leftarrow$  combine  $a_i, a_j$  according to the target aggregate function, cf. (2.1).
6: end for

```

node picks its communication partners according to a single global strategy. Moreover, the algorithm terminates by construction after $d = \log_2 n$ steps with the correct result. Of course, such a parallel approach has the drawback that a single failure may cause a globally wrong result. Depending on the point of time when a failure occurs, one, many, or even all nodes may compute a wrong result.

2.2.2. Distributed All-To-All Aggregation

In [Algorithm 2.1](#) we restricted the communication between the nodes according to the neighborhood structure of the d -dimensional hypercube. Hence, it is apparent to ask for algorithms which can cope with arbitrary neighborhoods (graphs). Since all algorithms we consider in the following chapters are of an iterative nature, we have to measure the quality of the locally computed estimates for the target aggregate.

Definition 2.1 (ε -accurate). Let α denote the target aggregate and a_i a local estimate of it. Then we say that a_i is ε -accurate if $|a_i - \alpha|/|\alpha| \leq \varepsilon$. //

Following the considerations above, we base our quest for more flexible algorithms on [Algorithm 2.1](#). An exemplary distributed counterpart of [Algorithm 2.1](#) which has much weaker requirements on the communication capabilities of the underlying system is given by [Algorithm 2.2](#). More precisely, instead of a fixed communication schedule a

Algorithm 2.2 Skeleton of a distributed aggregation

```

1:  $a_i \leftarrow$  initial contribution from node  $i$ 
2: while not  $\varepsilon$ -accurate do
3:   pick a node  $k \in \mathcal{N}_i$  and send  $a_i$  to node  $k$ 
4:   wait for a message  $a_j$  from a node  $j \in \mathcal{N}_i$ 
5:    $a_i \leftarrow$  combine  $a_i, a_j$  according to the target aggregate function, cf. (2.1).
6: end while

```

flexible scheme is employed. In terms of sending information (line 3), this means that the nodes can employ some (autonomous) local strategy for picking communication partners (e. g., random choices). Consequently, this implies that in terms of receiving information (line 4) the nodes have to listen for a message from any neighbor, since they do not know about a potentially upcoming transmission of a specific neighbor.

Of course, in this case it is less straightforward how to combine the results of different nodes in order to obtain correct results (line 5). The increased flexibility in the choice of communication partners comes for the price that we cannot expect anymore a *direct algorithm*⁴ as a result. Instead of that, we get an *iterative algorithm*, where we can only hope for convergence towards the correct result locally at each node. Needless to say, any kind of direct algorithm is doomed to failure under our assumptions on distributed environments (e.g., due to the unreliable communication). On the other hand, iterative algorithms offer enough flexibility to cope with the challenges posed by distributed systems.

Moreover, iterative algorithms in the spirit of [Algorithm 2.2](#) can still be very efficient. For instance, by employing the Push-Sum algorithm [[Kempe, Dobra, et al., 2003](#)] (where the nodes pick their communication partners uniformly at random), a distributed all-to-all aggregation requires (on well-connected topologies, see [Section A.4](#)) only $\mathcal{O}(\log n + \log \varepsilon^{-1})$ iterations [[Boyd et al., 2006](#)], where ε denotes the target accuracy. This bound emphasizes that the increased flexibility comes with (reasonable) additional costs which depend on the target accuracy ε , whereas the scaling with the number of nodes n is still logarithmic as in the case of parallel approaches. [Figure 2.1](#) depicts for the case of hypercubes how the number of iterations required to finish an accurate all-to-all scale with increasing system size. We see that both the parallel and the distributed approach scale logarithmically with the number of nodes, whereas the distributed approach has the expected (additive) overhead.

At this point it is crucial to recall our objective to employ all-to-all aggregations within accurate distributed matrix computations (cf. [Chapter 1](#)). Therefore, we commonly fix ε with respect to the limitations of floating-point arithmetic to either single precision $\varepsilon \approx 10^{-8}$ or double precision $\varepsilon \approx 10^{-16}$. Consequently, in an analysis, ε must not be treated as a constant in the context of this thesis, since ε^{-1} is always of a significant order of magnitude and the dependency of the convergence speed of algorithms on ε is very crucial.

2.2.3. Challenges

[Algorithm 2.1](#) and [Algorithm 2.2](#) illustrate the structure and properties of classical parallel and distributed aggregation algorithms. The principle structure of [Algorithm 2.2](#) allows for deriving many concrete algorithms, which is witnessed by numerous complying distributed aggregation approaches in the literature (cf. [Section 3.1](#)). However, in most of the cases, algorithmic flexibility is misleadingly equated with robustness and/or fault tolerance (see, e.g., [Kempe, Dobra, et al. \[2003\]](#)). More specifically, a flexible communication schedule alone by no means guarantees a resilient aggregate computation, especially with respect to our challenging assumption on a distributed environment. Another important (but commonly neglected) issue is how to decide

⁴We borrow here the terminology of *direct* vs. *iterative* methods from numerical linear algebra [[Trefethen and Bau, 1997](#), parts IV and VI]

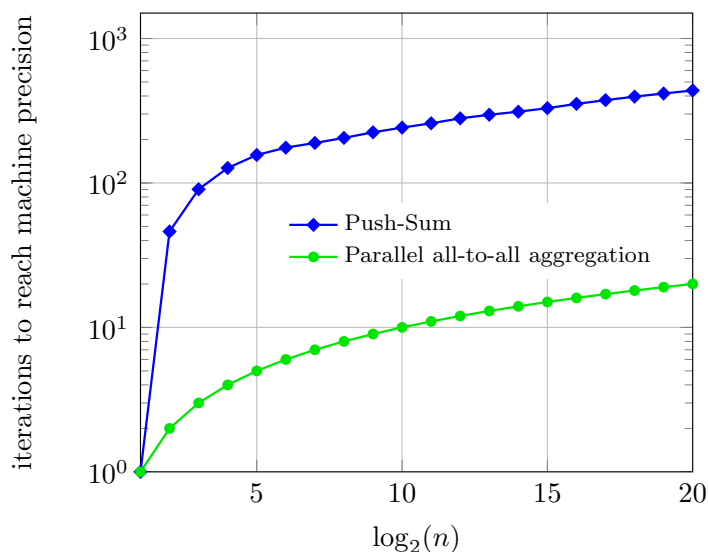


Figure 2.1.: Iterations required to compute a fully accurate ($\varepsilon \approx 10^{-16}$) all-to-all aggregation on a hypercube. We observe a logarithmic scaling for both Push-Sum (cf. Algorithm 2.2) and the parallel approach (cf. Algorithm 2.1).

locally when to terminate an iterative algorithm in the spirit of Algorithm 2.2. Consequently, the following two questions are apparent:

1. How can we exploit the flexibility of (certain) distributed aggregation algorithms, e. g., in order to really obtain high fault tolerance at the algorithmic level?
2. How can we decide locally when to terminate a distributed aggregation algorithm, i. e., how can a node autonomously deduce when its local result meets a predefined target accuracy ε ?

Most of the technical contributions in the following chapters are centered around these two questions: In Chapters 3 and 4 we generically develop and analyze fault tolerance mechanisms for a large class of distributed aggregation algorithms and in Chapter 5 we develop local convergence criteria for distributed iterative processes.

2.3. Epidemic Algorithms: Merits and Challenges

In most of the aggregation algorithms which we consider in this thesis the nodes pick their communication partners randomly within their neighborhoods. As a baseline on how efficiently such *epidemic algorithms* can spread information within distributed networks, we look at the problem of rumor spreading.

From real life we know that, for instance, viral infections can spread very quickly over populations and hence, can cause epidemics. Such a fast spread of a virus can still be observed when (a certain fraction of) the population takes actions against the

dissemination. From the virus’ point of view, this property can be seen as some kind of “robustness”. In computer science, we can turn this “efficiency” and “robustness” of infectious diseases into something positive—by imitating them in order to obtain efficient distributed algorithms.

In the remainder of this section, we review important instances of epidemic algorithms for the problems of rumor spreading and distributed aggregation. The purpose of this review is twofold: (i) we want to give an overview on how well simple epidemic algorithms can cope with certain distributed computing problems, and (ii) we provide a clear distinction between the problems of rumor spreading and distributed aggregation. More specifically, we complement our review of (recent) theoretical results regarding (the efficiency of) rumor spreading protocols by a clear intuition why distributed aggregation is substantially more difficult than rumor spreading, especially with respect to fault tolerance aspects.

2.3.1. Notes on Rumor Spreading

Throughout this section, we consider the problem of spreading a piece of information (“the rumor”) in a distributed system with n nodes: Initially a single node knows the rumor and the objective is to spread it to all other nodes. We consider this problem, because the deep analytical results regarding the efficiency of simple rumor spreading algorithms deliver a good intuition, when we can also expect distributed aggregation algorithms to be efficient. Our presentation is restricted to the line of research which concerns methods in the spirit of [Algorithm 2.2](#). In particular, we exclude results where the message size is not constant (see, e.g., [Haeupler \[2013\]](#)) and more generally, results which are not applicable under our assumptions on a distributed environment (cf. [Table 2.1](#)). Another big advantage of the kind of rumor spreading methods we consider is that their analysis usually yields strong robustness guarantees as a byproduct. In particular, often a substantial amount of nodes (picked by an adversary) can crash without effecting the overall performance significantly.

First, we review some standard terminology in the context of rumor spreading, which originates in the seminal works of [Demers et al. \[1987\]](#) and [Karp et al. \[2000\]](#). A well-established model for the communication between the nodes is the so-called *random phone call model*. This model is round-based and at every round each node *calls* a neighbor, which is determined by a local strategy. We distinguish between two kinds of communication: *push* and *pull*. In the case that the calling node knows the rumor, it can *push* the rumor to the called node. If the called node knows the rumor and the calling node does not, then the calling node can *pull* the rumor from the called node.

Besides the communication model also the (global) knowledge available locally at each node is restricted. An algorithm is called *address-oblivious*, whenever the decision of a node if and what to communicate with a particular neighbor in a certain round does not depend on the neighbor’s address [[Karp et al., 2000](#)]. The notion of address-oblivious algorithms supports our distinction between parallel and distributed

algorithms, since non-trivial parallel algorithms are never address-oblivious, because they rely on specific communication patterns where the identification via (unique) addresses is essential. In particular, recall [Algorithm 2.1](#) as an example of a parallel all-to-all aggregation. There, each parallel process picked its communication partners in a very specific way. On the other hand, algorithms where nodes pick their communications partners uniformly at random are naturally address-oblivious.

Unless otherwise stated all algorithms in the following literature review are address-oblivious. Moreover, all bounds we refer to hold *with high probability*. Depending on the respective authors and contexts there are minor differences in the formal definition of this notion. Common definitions are that a bound holds with high probability if it holds at least with probability $1 - \mathcal{O}(n^{-c})$ for an arbitrary constant $c > 0$, or that it holds at least with probability $1 - o(1)$ (if $g(n) \geq 0$ for almost all n , then $f(n) = o(g(n))$ means $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$).

For many graph topologies even the most basic strategy, where all nodes who know the rumor push it to a randomly chosen neighbor, is very efficient. For instance, a $\Theta(\log n)$ bound on the number of iterations required to inform all nodes is known for complete graphs [[Pittel, 1987](#)], hypercubes [[Feige et al., 1990](#)] and (certain) Erdős-Rényi graphs [[Frieze and Molloy, 1994](#)]. Moreover, there is a symmetry between protocols using only push or only pull operations [[Doerr, Friedrich, et al., 2009](#)], i. e., graphs for which push protocols are efficient, pull protocols are also efficient. Nevertheless, the example of *preferential attachments graphs* [[Barabási and Albert, 1999](#)]—a common model for social networks—reveals that simple push (pull) protocols can be substantially less efficient than a combined push-pull protocol [[Chierichetti et al., 2011](#)]. Already [Karp et al. \[2000\]](#) observed the superiority of combined push-pull schemes, where each node calls a random neighbor and pushes or pulls the rumor according to its status. More precisely, [Karp et al. \[2000\]](#) showed for complete graphs that push-pull spreads a rumor in $\mathcal{O}(\log n)$ rounds using only $\mathcal{O}(n \log \log n)$ messages⁵. Roughly speaking, push is superior over pull “at the beginning”, until about half of the nodes is informed, whereas pull is superior over push thereafter [[Karp et al., 2000](#)]. Hence, by employing a push-pull scheme one aims at combining the benefits of both worlds in order to improve the time and number of messages required to inform all nodes.

We consider rumor spreading as a benchmark on the efficiency of epidemic processes. Therefore, it is very natural to ask whether the efficiency of rumor spreading protocols can be generically expressed in terms of graph connectivity measures such as *conductance* and *vertex expansion* (cf. [Section A.4](#)). Today this question is entirely settled (see the given references for more background on the historical developments): [Giakkoupis \[2011\]](#) proved that for every graph with conductance at least $\phi \in (0, 1]$, an upper bound of $\mathcal{O}(\log(n)/\phi)$ rounds holds. This bound is tight, because there

⁵In the light of standard communication models which account for latency, such as LogGP [[Alexandrov et al., 1995](#)], practitioners may disagree with the stated bound on the message complexity, because each node *calls* a node in each of the $\mathcal{O}(\log n)$ iterations. Hence, the $\mathcal{O}(n \log \log n)$ bound on the communication complexity can only be observed in practice, when transmitting the rumor is substantially more expensive than the process of calling a node.

exist graphs with diameter $\Omega(\log(n)/\phi)$ [Chierichetti *et al.*, 2010]. Very recently, Giakkoupis [2014] also proved a $\mathcal{O}(\log^2(n)/\chi)$ bound on the number of rounds required to spread a rumor in any graph with vertex expansion at least $\chi \in (0, 1]$. This bound is tight as well, since the corresponding lower bound $\Omega(\log^2(n)/\alpha)$ is also known [Giakkoupis and Sauerwald, 2012].

Besides the most basic algorithms where the nodes pick their communication uniformly at random, also slightly more complex algorithms for rumor spreading have been investigated. For instance, the exclusion of the most recently contacted neighbor(s) from the selection of the next communication partner proved to be useful. For instance, such slightly more clever algorithms were shown to improve the message complexity [Elsässer and Sauerwald, 2008] as well as the runtime [Doerr, Fouz, *et al.*, 2011] of rumor spreading processes on certain graph structures. The results of Doerr, Fouz, *et al.* [2011] also illustrate that rumors can even spread in sub-logarithmic time. More recently, Avin and Elsässer [2013] showed, under the assumption of a slight extension to the random phone call model, an $\mathcal{O}(\sqrt{\log n})$ bound on the number of rounds required to spread a rumor in a complete graph. While this improved efficiency comes for the price of a slightly increased communication complexity, it is particularly noteworthy that the algorithm of Avin and Elsässer [2013] is still robust, e. g., with respect to crashing nodes.

2.3.2. From Rumor Spreading to Aggregation

Protocols for quickly spreading a rumor and protocols for aggregating information share many similarities. Moreover, the efficiency of either of the approaches relies similarly on the underlying graph properties. On the other hand, the problem of aggregation is provably more complex than the problem of spreading a rumor. In particular, Chen and Pandurangan [2012] showed a lower bound of $\Omega(n \log n)$ on the message complexity for any address-oblivious aggregation algorithm, as opposed to the $\mathcal{O}(n \log \log n)$ bound on the message complexity for address-oblivious rumor spreading in the complete graph [Karp *et al.*, 2000]. Besides efficiency, we also concern fault tolerance properties. While the efficient rumor spreading protocols we discussed in Section 2.3.1 are naturally robust, we argue below that this does not hold for distributed aggregation algorithms, i. e., fault tolerance does not come for free in this case. We refrain from giving technical details on the formal differentiation between rumor spreading and distributed aggregation of Chen and Pandurangan [2012]. Instead, we shed light on this subject from a more practical point view and reveal that their result appears very naturally (at least on an informal level).

In a nutshell, computing aggregates is less efficient than spreading a rumor, because the nodes cannot judge whether the information they hold is relevant for their neighbors. Hence, they always have to communicate (“just in case”). Moreover, the robustness gets lost in certain aggregations due to the lack of natural redundancy in the course of the computation. As it is common in the literature, Chen and Pandurangan [2012] treat all kinds of aggregations the same. In contrast to that, we distinguish

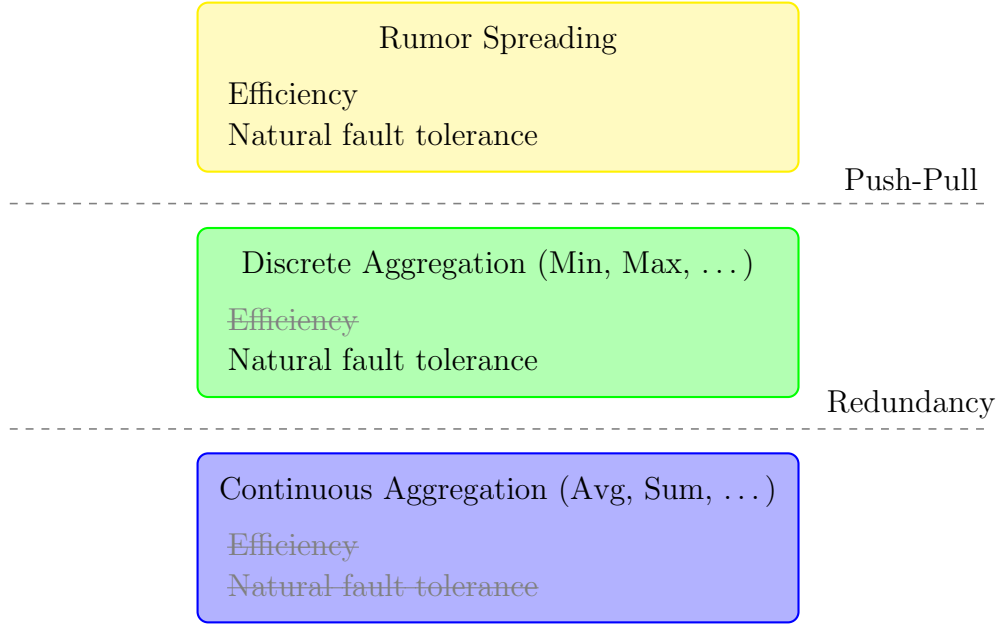


Figure 2.2.: Gradual differentiation between rumor spreading and aggregation. The inapplicability of push-pull causes the inefficiency of (discrete) aggregation and the missing redundancy causes the lack of fault tolerance in case of continuous aggregates

two kinds of aggregations. With respect to our definition of an aggregation in (2.1), we talk about a *discrete aggregation* if φ fulfills $\varphi(a_1, \dots, a_n) = a_i$, for some $1 \leq i \leq n$. For instance, the computation of a minimum is an example of a discrete aggregation. Furthermore, we call all other forms of aggregations, such as sum and average, *continuous aggregations*.

Our distinction between discrete and continuous aggregation is especially motivated by the fact that it illustrates the gradually increasing difficulty of the problem of aggregation over the problem of rumor spreading, which is summarized in Figure 2.2. The efficiency of rumor spreading algorithms heavily relies on the utilization of both push and pull communication. More precisely, the fact that the nodes can decide when a push or pull operation yields an information gain is crucial. In case of an aggregation this ability gets entirely lost, because the nodes can not judge whether their local information is the target aggregate.

Recall the classical result of Karp *et al.* [2000] that by using a push-pull scheme, a rumor can be spread in a complete graph in $\mathcal{O}(\log n)$ rounds using $\mathcal{O}(n \log \log n)$ messages. The missing ability to decide whether a communication yields to an improvement implies that all nodes “have to” communicate in each round in order to allow for an efficient computation of the target aggregate. Therefore, the lower bound of $\Omega(n \log n)$ on the message complexity of address-oblivious aggregation due to Chen and Pandurangan [2012] appears very naturally.

Fault tolerant computations are almost always coupled to some form of redundancy⁶. In the case of rumor spreading or discrete aggregation, we observe that a natural redundancy appears because (i) at the beginning (at least) one node knows the result, and (ii) the result spreads (exponentially fast) over the network. Hence, in the course of the computation more and more nodes learn about the result, which leads (together with the independence from a specific communication schedule) to an extraordinary high degree of natural robustness.

The situation drastically changes in the case of a continuous aggregate such as a sum, where the input of all nodes is required for computing the correct result. In other words, none of the nodes holds initially the correct result and therefore, there is also no redundancy which could be exploited. Hence, even a single failure (e. g., a lost message or a crashed node) immediately leads to wrong results at (potentially) all nodes. Concerning this issues, in the following chapters we develop and analyze purely algorithmic fault tolerance mechanisms for the computation of continuous aggregates, which can provably cope with several kinds of failures. Opposed to our efforts which are based on diffusion-based iterative algorithms in the spirit of [Algorithm 2.2](#), [Mosk-Aoyama and Shah \[2006\]](#) showed how to compute (among others) continuous aggregates solely via the computation of minima. While this approach yields a high degree of natural robustness, it also comes with the severe drawback of a very bad scaling with the target accuracy ε (cf. [Section 3.1](#)), which is crucial with respect to our high accuracy demands.

⁶This fact will become more explicit in [Section 4.1.3](#) when we review existing fault tolerance mechanisms (at the algorithmic level).

Lessons Learned: Parallel vs. Distributed Computing

In this thesis, we strictly distinguish between the notions of parallel and distributed computing. In particular, we do not call a distributed memory parallel system a distributed system. In the course of our differentiation between parallel and distributed systems we pointed out substantial differences in terms of the topology, communication capabilities, global control mechanisms and local knowledge (cf. [Table 2.1](#)). While distributed algorithms can be executed on parallel systems, the converse does not hold, because parallel algorithms impose many requirements on the environment which distributed systems cannot fulfill (according to our definition).

All-to-all aggregations are fundamental for both parallel and distributed computing. In the course of such an operation, a global aggregate (e. g., a sum) involving contributions from all nodes is computed and its value is delivered to all nodes. On parallel systems, all-to-all aggregation is a classical problem and many efficient algorithms are known. In particular, dedicated all-to-all aggregation algorithms are superior to the combination of a tree-based aggregation and a subsequent broadcast. In the spirit of dedicated parallel all-to-all aggregation algorithms we sketched how to compute an all-to-all aggregation within a distributed environment. Despite the fact that such distributed approaches are always very flexible, they can very efficient. However, the change from direct methods (parallel algorithms) to iterative methods raises additional challenges. With respect to our quest for reliable and practically applicable distributed all-to-all aggregation algorithms, we identified the following two central questions which we address in the following chapters:

1. How can we exploit the flexibility of (certain) distributed aggregation algorithms, e. g., in order to obtain high fault tolerance at the algorithmic level?
2. How can we decide locally when to terminate, i. e., how can a node autonomously deduce when its local result meets a predefined target accuracy ε ?

Also recall that the target accuracy ε is a crucial parameter in our considerations which can (due to its scale) never be treated as a constant in an analysis.

Most of the distributed algorithms we investigate in this thesis are based on random communication schedules. As a baseline on how well such epidemic algorithms can work in practice we reviewed recent theoretical results concerning the efficiency of rumor spreading. Moreover, the deep insights on how the efficiency of rumor spreading relates to common graph connectivity measures gives a good outline for which network structures we can expect efficient aggregation. We completed this chapter by several insights on how and why the (somewhat similar) problems of rumor spreading and aggregation differ in their complexity. Especially the requirement of explicit fault tolerance mechanisms in the case of (continuous) aggregations further motivates our technical contributions in the following chapters.

3 | Vector Iterations vs. Distributed Algorithms

Assumptions and Implications

By an appropriate discretization of time, distributed all-to-all aggregation algorithms in the spirit of [Algorithm 2.2](#) can be precisely analyzed by means of simple mathematical models. In particular, many distributed algorithms can be analytically expressed¹ in terms of a linear iteration

$$x(t) = C(t)x(t-1), \quad x(t) \in \mathbb{R}^n, \quad C(t) \in \mathbb{R}^{n \times n}. \quad (3.1)$$

Roughly speaking, the elements of the vectors $x(t)$ represent the local information of the n individual nodes at time t and the matrices $C(t)$ describe the communication which happens between the nodes at time t . The simple mathematical model (3.1) immediately gives rise to many questions concerning theoretical as well as practical issues. Needless to say, the convergence analysis of this kind of iterations is a central theme and due to the widespread appearance of iterations of the form (3.1) in the literature, a large body of analytical results is available. However, common assumptions made on the (distribution of the) matrices $C(t)$ prevent the practical application of the respective algorithms, as we argue later in [Section 3.3](#). Besides a thorough discussion about practical problems caused by particular assumptions, our central contribution in this chapter (cf. [Section 3.4](#)) is a convergence analysis for iterations of the form (3.1) with particularly weak (“realistic”) assumptions on the shape of the matrices $C(t)$. Our analysis is not only a valuable generalization of existing convergence results, but it is also a central pillar of our work on fault tolerance algorithms in [Chapter 4](#). More specifically, our fault tolerance results cannot be built upon existing convergence analyses and strictly require (at least) the more general theory we develop in this chapter.

¹From a practical point of view this step is in most cases not loss-free, i. e., there is a gap between the algorithm as it is executed in practice and its formal analysis. Concerning this issue we distinguish in this thesis thoroughly between algorithms and their formal analysis in order to derive theoretical results which also cover the practical execution of the respective algorithms.

3.1. Background and Related Problems

In all our investigations and in the related work we refer to, the matrices $C(t)$ in (3.1) are assumed to be drawn independently from a fixed distribution $\mathcal{C}$, i.e., the matrices $C(t)$ are *i.i.d.* (independent and identically distributed) random variables. Therefore, whenever an algorithm is analyzed in terms of a linear iteration of the form (3.1), the algorithm together with the assumed environment determines the properties of the distribution $\mathcal{C}$. Consequently, there is a natural quest for proving that certain distribution properties (determined by particular algorithms and assumptions on the environment) guarantee the desired convergence. The other way round, the fewer assumptions a convergence theory requires on a potential distribution, the wider is the range of applications of the corresponding algorithms. For instance, simple approaches may require that the nodes exchange messages in well defined rounds, whereas more subtle theories may guarantee convergence even for less regular communication patterns. In this chapter we focus on providing generic convergence results under “minimal” assumptions, i.e., we do not strive for optimality with respect to specific scenarios and algorithms.

3.1.1. Linear Iterations and Beyond

The in-network computation of aggregate information, in particular the distributed computation of averages, is a well-studied problem whose investigation goes back as far as to DeGroot [1974] and Tsitsiklis [1984]. Due to the widespread appearance of problems of this kind, relevant contributions have been made in several scientific fields, especially in signal processing and in (theoretical) computer science. Frequently used terms for distributed aggregation algorithms are *consensus algorithms* [Olfati-Saber et al., 2007] and *gossip algorithms* [Dimakis, Kar, et al., 2010]. Furthermore, the problem of computing an average in a distributed manner is known as *average consensus problem* [Olfati-Saber et al., 2007]. The notion of consensus algorithms refers to the fact that all nodes eventually “agree on” the same aggregate, i.e., they reach a consensus on the result. Opposed to that, the notion of gossip algorithms points towards the epidemic nature and dynamic communication schedules which characterize algorithms of this kind. Due to the importance of the topic, several thorough review articles on the subject have appeared, covering theory [Olfati-Saber et al., 2007; Dimakis, Kar, et al., 2010] as well as applications [Kar and Moura, 2013].

Existing work regarding the analysis of linear iterations can be put into two groups. First, there exist specific results where a particular algorithm and potentially also specific network topologies are studied in detail (cf. Section 4.1.3). Second, there exist generic results which guarantee convergence for entire classes of algorithms and arbitrary² topologies [Boyd et al., 2006; Benezit, Blondel, et al., 2010]. In this chapter,

²Arbitrary means here that the topology is not specified. However, nonrestrictive (implicit) assumptions regarding the connectivity have to be made (cf. Section 3.3).

we focus on providing the latter kind of analysis, which naturally may be in certain cases less tight than a specific analysis of a particular setup of algorithm and topology. However, the generality of our analysis is crucial for the derivation and analysis of the novel fault tolerant algorithms we derive in [Chapter 4](#).

Especially in signal processing authors commonly do not distinguish between an algorithm and its formal analysis (cf. [Section 3.3](#)). In such cases, the considered linear iteration is supposed to define the respective algorithm. The other way round, there are also algorithms which can be analyzed in terms of linear iterations but whose original analysis required a custom approach. For instance, a complete linear iteration-based convergence proof of the Push-Sum algorithm [[Kempe, Dobra, et al., 2003](#)] was for a long time beyond the reach of generic convergence results due to (too) restrictive assumptions made therein. More recently, much of this gap was closed by [Benezit, Blondel, et al. \[2010\]](#). However, for our purposes the assumptions of [Benezit, Blondel, et al. \[2010\]](#) are still too restrictive (and we provide a more general convergence theory in [Section 3.4](#)).

Of course, not all distributed aggregation algorithms in the literature can be expressed and analyzed in terms of linear iterations. For instance, [Mosk-Aoyama and Shah \[2006\]](#) introduced a method for the distributed computation of separable functions which relies on properties of exponentially distributed random variables³. Since this method builds upon the computation of minima, it is also naturally robust (cf. [Section 2.3.2](#)). Moreover, for certain topologies it can provide rough estimates of a target function substantially faster than methods based on linear iterations [[Mosk-Aoyama and Shah, 2006](#), § 5]. However, all complexity results of [Mosk-Aoyama and Shah \[2006\]](#) for computing ε -accurate results include an ε^{-2} factor, i. e., they show bounds of the form $\mathcal{O}(\varepsilon^{-2}f(\cdot))$ for a function $f(\cdot)$ depending on certain parameters. This implies that accurate computations (e. g., in single or double precision) require a tremendous amount of time and are thus not feasible with this approach.

Another common approach towards distributed aggregation are optimization-based methods. Such methods are especially employed in the context of the average consensus problem because the average $\alpha = \sum_{i=1}^n x_i/n$ of scalars x_i minimizes the value of the particularly simple functional $F(y) = \sum_{i=1}^n (x_i - y)^2$, since

$$F'(y) = 0 \Leftrightarrow -2 \sum_{i=1}^n (x_i - y) = 0 \Leftrightarrow \sum_{i=1}^n x_i - ny = 0 \Leftrightarrow y = \sum_{i=1}^n x_i/n,$$

and $F''(y) = 2n$. Recently, optimization-based aggregation algorithms have been frequently considered (see, e. g., [Rabbat and Nowak \[2004\]](#); [Erseghe et al. \[2011\]](#); [Mota et al. \[2012\]](#); [Sayed \[2013\]](#)). Even though the concept of optimization-based computations is interesting, such methods require assumptions on the environment which go far beyond what we consider to be possible in a distributed environment (cf. [Table 2.1](#)).

³Let $X_1, X_2, \dots, X_m$ be independent random variables and let X_i be exponentially distributed with parameter λ_i . Then the random variable $Y = \min(X_1, X_2, \dots, X_m)$ is exponentially distributed with parameter $\sum_{i=1}^m \lambda_i$

For instance, all of the aforementioned methods require a synchronization between (certain subsets of) the nodes in the system.

3.1.2. Related Problems

In the existing literature, the matrices $C(t)$ in iteration (3.1) are commonly assumed to be nonnegative. Moreover, they are assumed to be column, row, or even doubly stochastic, depending on the respective context. Therefore, there is a close connection to discrete Markov chains and their steady state analysis (see also Section A.3 for examples). In the context of Markov chains, one does not speak about nodes but about states, and the elements of the matrices $C(t)$ are interpreted as probabilities for a state transition from one state to the other. Therefore, the initial vector $x(0)$ determines the initial distribution (i.e., the probabilities that the Markov chain is initially in a particular state) and the overall objective is to compute the stationary distribution of the Markov chain.

Another way of reinterpreting a distributed averaging process described by a linear iteration appears in the context of *load balancing*. In load balancing the goal is to distribute the initial load of the nodes within the network such that all nodes have (roughly) the same load. Conceptually, there is a distinction whether the loads are assumed to be divisible (“continuous load balancing”) or not (“discrete load balancing”). With respect to distributed aggregate computations, especially results regarding continuous load balancing are relevant. Even though the analysis of the discrete case is more subtle, it turned out over the last years that the discrete case is not much harder than the continuous case (see, e.g., Elsässer and Sauerwald [2010]; Friedrich *et al.* [2012]; Sauerwald and Sun [2012]).

Due to the close relationship between the aforementioned problems and distributed aggregation, results regarding either of the problems can often be employed in the other world as well. For instance, the fault tolerance results we develop in Chapter 4 on top of our convergence results from this chapter (see Section 3.4), as well as our convergence results themselves directly apply also to continuous load balancing schemes. On the other hand, it is less straightforward how to interpret failures (and their recovery) or negative elements in $C(t)$ in the world of Markov chains. However, in Section 4.2 an interesting interpretation of these results appears.

3.2. Two Faces of the Power Iteration

The analysis of time-heterogeneous iterations of the form $x(t) = C(t)x(t-1)$ (see (3.1)) builds upon simpler time-homogenous iterations of the form $x(t) = Cx(t-1)$. For a square matrix $C \in \mathbb{R}^{n \times n}$ and an initial vector $x(0) \in \mathbb{R}^n$, the *power iteration* is defined for all $t > 0$ by

$$x(t) = \frac{Cx(t-1)}{\|Cx(t-1)\|} = \frac{C^t x(0)}{\|C^t x(0)\|}, \quad (3.2)$$

where the norm $\|\cdot\|$ can be chosen arbitrarily. In the following, we depict how the power iteration can be employed as an algorithm for computing the extremal eigenpair and how it appears as an ingredient in the analysis of distributed algorithms. Unless otherwise stated, in this section we always assume matrices whose eigenvalues fulfill $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \dots \geq |\lambda_n|$. Hence, it is valid to call λ_1 the dominating eigenvalue without causing ambiguities. However, whenever we employ the power iteration later on, we have to verify/guarantee that $|\lambda_1| > |\lambda_2|$ indeed holds. For nonnegative matrices the separation of the dominating eigenvalue can be often conveniently verified by utilizing results from Perron-Frobenius theory (cf. [Section A.3](#)).

3.2.1. Power Iteration as Eigenvalue Solver

In many areas, partial spectral information such as knowledge of the dominating eigenpair is a key information with important applications. A particularly famous application of this kind is Google's PageRank (see [Brin and Page \[1998\]](#) and references therein). Up to a moderate scale, the computation of (partial) spectral information of a matrix does not cause any difficulties by using today's tools and hardware. However, things get more subtle whenever the required matrix dimension prevents the utilization of standard tools. At this point it is important to note that the power iteration can be implemented in a matrix-free fashion, which means that only a routine for evaluating $x \mapsto Cx$ is required, but not the matrix C itself. Hence, by employing (developing) structure-aware special purpose algorithms, the computationally tractable matrix dimension can be huge in certain applications. For instance, in [Chapter 6](#) we utilize the power iteration together for specifically structured dense matrices up to a dimension of 2^{25} .

In the power iteration nothing else than the repeated evaluation of a matrix vector product takes place (which itself may be sophisticated routine). Simply tracking the executed operations yields the following convergence proof: For the sake of simplicity we assume⁴ that C has n linearly independent eigenvectors $v_1, \dots, v_n$ with corresponding eigenvalues $\lambda_1, \dots, \lambda_n$. Furthermore, we assume without loss of generality that all eigenvectors v_i are normalized, i. e., $\|v_i\| = 1$. Then, we can write the iteration $x(t) = C^t x(0) / \|C^t x(0)\|$ as follows.

$$\begin{aligned}
 x(t) \cdot \|C^t x(0)\| &= C^t x(0) \\
 &= C^t (d_1 v_1 + d_2 v_2 + \dots + d_n v_n) \\
 &= \lambda_1^t d_1 v_1 + \lambda_2^t d_2 v_2 + \dots + \lambda_n^t d_n v_n \\
 &= \lambda_1^t d_1 \left(v_1 + \underbrace{\frac{\lambda_2^t d_2}{\lambda_1^t d_1} v_2 + \dots + \frac{\lambda_n^t d_n}{\lambda_1^t d_1} v_n}_{\rightarrow v_1 \text{ for } t \rightarrow \infty} \right)
 \end{aligned} \tag{3.3}$$

⁴In a general proof one has to work with the Jordan canonical form of C , but otherwise analogously to the proof we give here [[Saad, 2003](#), §1.8.4].

Altogether, we see that the eigenvector approximation $x(t)$ converges linearly with rate $|\lambda_2/\lambda_1|$, i. e., $\|x(t) \pm v_1\| = \mathcal{O}(|\lambda_2/\lambda_1|^t)$ whenever the choice of $x(0)$ ensures that $d_1 \neq 0$ in (3.3). In other words, $x(0)$ must not be orthogonal to v_1 . In the course of the power iteration, an approximation $\lambda(t)$ of the dominating eigenvalue can be computed by computing the so-called *Rayleigh quotient*

$$\lambda(t) = \frac{x(t)^T C x(t)}{x(t)^T x(t)}.$$

While the eigenvector approximation converges linearly, the eigenvalue approximation converges (with the same rate) quadratically, i. e., $\|\lambda(t) \pm \lambda_1\| = \mathcal{O}(|\lambda_2/\lambda_1|^{2t})$ [Trefethen and Bau, 1997, Theorem 27.1]. In its basic form, the success of the power iteration heavily relies on the ratio $|\lambda_2/\lambda_1|$ (i. e., the gap between λ_1 and λ_2). However, convergence acceleration schemes which cause only low additional costs can be applied in order to lower the number of iterations required for convergence (cf. Section 3.5).

In Chapter 6, we investigate a concrete application problem from biochemistry where the efficient computation of the dominating eigenpair of extreme-scale matrices is the computational bottleneck. In particular, we design novel structure-aware algorithms which yield a fast and accurate power iteration-based solver. Moreover, we also provide implementations of our methods on current parallel hardware.

Notes on Orthogonality and Floating-Point Arithmetic

Since the convergence of the power iteration relies on an orthogonality condition we shall also point out the non-trivial relationship between orthogonality and floating-point arithmetic. The concept of orthogonality in mathematics is an elegant and extremely fruitful notion which represents an integral part of numerous fundamental results in linear algebra, functional analysis and many other (mathematical) disciplines. However, it is crucial to observe that the theoretical beauty of orthogonality turns into subtle pitfalls at the transition from “pen and paper” to nowadays digital computers and the floating-point arithmetic implemented therein.

For the sake of simplicity, in the following we consider the power iteration for the exemplary 2×2 matrix

$$M = \begin{pmatrix} 0.7 & 0.4 \\ 0.4 & 1.3 \end{pmatrix} \tag{3.4}$$

whose eigenvalues are $\lambda_1 = 1.5$ and $\lambda_2 = 0.5$ with corresponding eigenvectors $v_1 = (1, 2)^T$ and $v_2 = (-2, 1)^T$. From a theoretical point of view, we have seen so far that the power iteration converges whenever the initial vector $x(0)$ is not orthogonal to the dominating eigenvector. In the concrete case at hand this means that the convergence of the power iteration is guaranteed whenever $x(0)$ is not a multiple of $(-2, 1)^T$. Hence, one expects that choosing $x(0) = (-2, 1)^T$ as a start vector in (3.2) precludes the convergence of the power iteration to the dominating eigenvector. Figure 3.1 depicts results from a simple MATLAB experiment which shows that this expectation is not

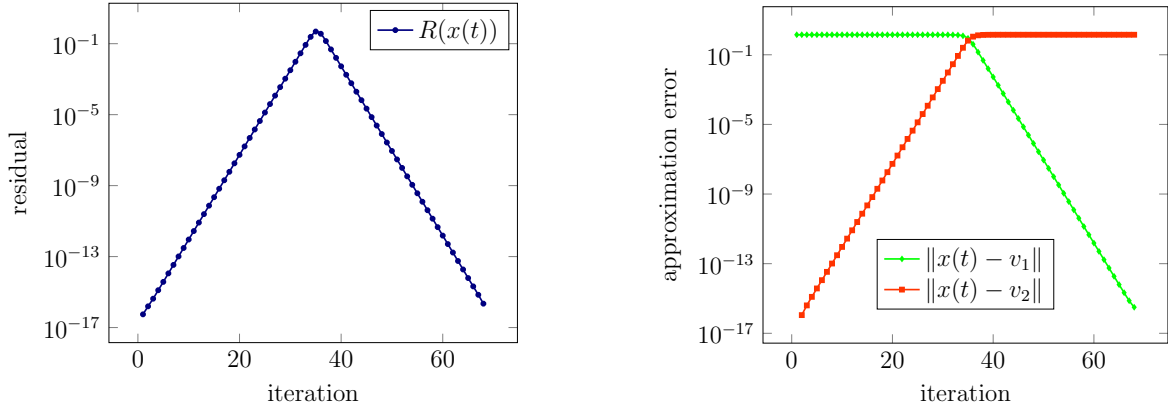


Figure 3.1.: Floating-point implementation of the power iteration with the matrix M from (3.4) and start vector $x(0) = (-2, 1)^T$. On the left, we depict the eigenvector residual $R(x(t))$ of the approximation $x(t)$, where $R(x) = \|Mx - (x^T Mx)x / (x^T x)\|$. On the right, the approximation errors $\|x(t) - v_1\|$ and $\|x(t) - v_2\|$ are shown.

fulfilled in practice and that the power iteration converges also to the dominating eigenvector in this case. The reason for this behavior is that due to the floating-point arithmetic, the vector Mv_2 is not a multiple of v_2 (as in exact arithmetic) but a linear combination of v_1 and v_2 . Hence, over the course of many iterations the contribution of v_1 in $M^t v_2$ increases and eventually the approximation $x(t)$ converges to v_1 —even though the start vector was chosen as $x(0) = v_2$.

At a first glance, the message of this toy example is an entirely positive one: The power iteration converges in practice also when the start vector is ill chosen, i.e., when the choice of the start vector precludes convergence in exact arithmetic. On the other hand, from a theoretical point of view this means that proofs which rely on the fact that the convergence to the dominating eigenvector can be prevented by choosing a specific start vector may break down in practice. Such proofs commonly appear in the analysis of distributed algorithms and also our convergence results contain an argument of this kind (see the proof of Theorem 3.7). However, in our work and the work we refer to, no substantial problems occur and that the algorithms behave as predicted in theory because $\lambda_1 \leq 1$ holds. Therefore, the inaccuracies introduced by floating-point arithmetic do not build up significantly.

Another particularly widely known field where the interplay of floating-point arithmetic and orthogonality is critical are Krylov subspace methods of the Lanczos type [Saad, 2003, §§ 6, 7]. In exact arithmetic such methods implicitly produce (i.e., without explicit orthogonalization) an orthogonal basis of the respective Krylov subspace. However, in practice the basis computed by standard methods deviates the further from being an orthogonal basis the higher the dimension gets. Accordingly, the quest for accurate and efficient Lanczos type methods received a lot of attention over the last decades (see, e.g., Cullum and Willoughby [2002] and references therein).

3.2.2. Power Iteration as Analytical Tool

Many distributed aggregation algorithms analytically boil down to (some form of) the power iteration, where the vector $x(t)$ contains the local data of the nodes and the respective matrix encodes the communication between the nodes. More specifically, in an iteration step $x(t) = Cx(t-1)$ a nonzero element $c_{j,i}$ of C means that node i sends $C_{j,i} \cdot x(t-1)_i$ to node j at time t (and node j receives it). Later, we consider flexible time-dependent communication schedules which yield iterations of the form (3.1). However, in the following illustration of how simple iterative processes can be utilized to compute meaningful functions (in a distributed way), we consider—for the sake of simplicity—the time-homogenous case.

To shed light on computational aspects appearing behind the scenes of the power iteration, we give another proof of its convergence. Under the assumption that the matrix C has n linearly independent eigenvectors $v_1, \dots, v_n$, the eigendecomposition $C = V\Lambda V^{-1}$ exists. More specifically, the columns of V are given by the eigenvectors of C , i. e., $V(:, i) = v_i$. Now we additionally assume $\lambda_1 = 1$ and by employing the eigendecomposition of C we see that

$$\lim_{t \rightarrow \infty} C^t = \lim_{t \rightarrow \infty} (V\Lambda V^{-1})^t = \lim_{t \rightarrow \infty} V\Lambda^t V^{-1} = \lim_{t \rightarrow \infty} V \text{diag}(1, \lambda_2^t, \dots, \lambda_n^t) V^{-1} = v_1 u_1^T,$$

where $u_1^T = V^{-1}(1, :)$ denotes the left eigenvector of C which corresponds to λ_1 . Hence, we observe that the power iteration with start vector $x(0)$ yields

$$\lim_{t \rightarrow \infty} C^t x(0) = (v_1 u_1^T) x(0) = v_1 (u_1^T x(0)) = v_1 \alpha, \quad \alpha \in \mathbb{R}. \quad (3.5)$$

We see that the scalar α is a linear combination of the local initial values collected in the vector $x(0)$. Hence, assume that α is the target aggregate which all nodes want to compute. In order to make the result independent of the communication which takes place, we have to enforce that u_1^T is a multiple of $\mathbf{1}^T$ which means that α is a multiple of $\sum_{i=1}^n x(0)_i$. Unless v_1 is a multiple of $\mathbf{1}$ (which is unwanted from a practical point of view, cf. Section 3.3), every node ends up with a different value, in contrast to the objectives of an all-to-all aggregation. Concerning this issue, we consider so-called *sum-weight schemes*, which are characterized by two simultaneous iterations with the same matrix. This means, besides an iteration $s(t) = C(t)s(t-1)$ there is a second iteration $w(t) = C(t)w(t-1)$ and the vector of local estimates of the target aggregate $x(t)$ is elementwise defined by $x(t)_i = s(t)_i / w(t)_i$. In a sum-weight scheme every node i picks initially its data $s(0)_i$ and a corresponding weight $w(0)_i$ (which determines the type of function which gets computed). More specifically, by (3.5) we obtain for every node i that

$$\lim_{t \rightarrow \infty} x(t)_i = \lim_{t \rightarrow \infty} \frac{s(t)_i}{w(t)_i} = \lim_{t \rightarrow \infty} \frac{[C^t s(0)]_i}{[C^t w(0)]_i} = \frac{\sum_{i=1}^n s(0)_i}{\sum_{i=1}^n w(0)_i}.$$

This shows that for each node i the local quantity $s(t)_i / w(t)_i$ converges to an aggregate of the initial values of all nodes which can be determined by the choice of the weights. For instance, an average is computed for all initial weights $w(0)$ which satisfy $\sum_{i=1}^n w(0)_i = n$, e. g., $w(0)_i = 1$ for all nodes i .

3.3. (Un)Realistic Assumptions and Their Implications

Especially in the context of consensus algorithms in signal processing in most cases no distinction is made between a linear iteration as analytical tool and the distributed algorithm which it is intended to model (see, e. g., [Olfati-Saber *et al.* \[2007\]](#) and the references therein). As we argue below, this approach implies several (crucial) simplifications which may cause problems at the transition from theory to practice. Moreover, we have already seen that the distribution of the matrices $C(t)$ (which is determined by the assumed environment and algorithm) plays a central role in the analysis of linear iterations. Accordingly, in this section we elaborate in detail up to which extent assumptions which are commonly made in existing theoretical results are in compliance with the assumptions on distributed systems, which we made in [Chapter 2](#) (see [Table 2.1](#) for a concise summary), and where more general results are necessary.

3.3.1. Algorithm vs. Analysis

As mentioned above, in signal processing it is common practice to specify (distributed) algorithms in terms of linear iterations (see, e. g., [Boyd *et al.* \[2006\]](#); [Olfati-Saber *et al.* \[2007\]](#); [Dimakis, Kar, *et al.* \[2010\]](#); [Hadjicostis *et al.* \[2012\]](#) and references therein). Conceptually, this approach has the apparent drawback that the linear iteration point of view is inherently global whereas for the practical application a local formulation (for each node) is needed. Of course, it is not difficult to derive an algorithm for each node which corresponds to a particularly structured linear iteration. However, in doing so one encounters several subtleties. For the matter of concreteness, [Algorithm 3.1](#) shows the generic (distributed) algorithm corresponding to iterations of the form (3.1). In order to make this algorithm work, certain assumptions on the matrices $C(t)$ are commonly made in the literature. In the following section, we elaborate in detail which of these assumptions are in compliance with our definition of distributed systems and which are not. In the remainder of this section we investigate up to which extent not distinguishing between an algorithm and its analysis causes practical limitations. Roughly speaking, [Algorithm 3.1](#) consists of the following three parts:

1. The outer while-loop (lines 1–11), which states that we keep computing until the local estimate x_i reaches the target accuracy ε (see [Chapter 5](#) for a discussion of purely local convergence criteria)
2. The discrete time t (line 2), where “at time t ” means within “time slot t ” (the size of the time slots can vary arbitrarily). Within a time slot t it is crucial that exactly the actions described by the matrix $C(t)$ happen. However, it is not crucial when and in which order these actions happen. In [Section 3.3.2](#) we discuss common assumptions on the matrices $C(t)$ and their (practical) implications.
3. The actual communication (line 3–10), where the respective message exchange is determined by the actual algorithm.

Algorithm 3.1 Generic linear iteration-based algorithm

Local variables: x_i
Initial values: $x_i \leftarrow x(0)_i$

```

1: while not  $\varepsilon$ -accurate do
2:   ... at time  $t$  do ...
3:   for all  $C(t)_{j,i} \neq 0$  do
4:     Send  $C(t)_{j,i} \cdot x_i$  to  $j$ 
5:   end for
6:    $x_i \leftarrow C(t)_{i,i} \cdot x_i$ 
7:   for all  $C(t)_{i,j} \neq 0$  do
8:     receive  $C(t)_{i,j} \cdot x_j$  from  $j$ 
9:      $x_i \leftarrow x_i + C(t)_{i,j} \cdot x_j$ 
10:  end for
11: end while

```

The latter two parts are not in compliance with our assumptions on distributed systems. First, the utilization of discrete time slots means in a real system that (global) synchronization between the nodes is required in order to guarantee that all nodes are within the same time slot. Second, regarding the communication we have to distinguish between the send (line 3–6) and receive (line 7–10) part (see also the upcoming [Section 3.3.2](#)). On the one hand, a node can always decide locally how and with whom it shares its local data (send part). On the other hand, it is strictly against our assumptions on distributed systems that a node knows that it gets contacted by a certain neighbor (receive part). More specifically, the only receive operation which is in compliance with our assumptions is to “listen” for potential messages from neighboring nodes (cf. Algorithms [3.2](#) and [3.3](#)).

In general, we propose a distinction between an algorithm and its analysis. In contrast to deriving an algorithm out of an analysis, we aim at covering the real execution of the distributed algorithms by our theoretical results. This means, we provide results based on assumptions which are insofar “realistic enough” that there always exists such a discretization of time that the real execution trace is in compliance with the theoretical assumptions on the structure of potential traces. In particular, the existence of such discretizations can be guaranteed whenever the assumptions are of a purely local nature, i.e., assumptions which only restrict (independent) local decisions of the individual nodes. How such “realistic” assumptions on the potential communication patterns can look like is subject of the upcoming section.

3.3.2. Communication

As we have seen in [Section 3.2.2](#), the matrices $C(t)$ describe the communication between neighboring nodes from a bird’s eye of view. Therefore, every constraint on the matrices $C(t)$ restricts how potential communication patterns can look in prac-

tice in order to be covered by theoretical results. Of course, theory and practice are antagonists in this case, i. e., the more convenient assumptions are for a theoretical analysis the less they cover real environments. In the literature three kinds of constraints appear, namely constraints regarding the elements, columns, and rows of the matrices $C(t)$.

Recall the interpretation of an element $C(t)_{j,i}$ that node i sends $C(t)_{j,i} \cdot x(t-1)_i$ at time t to node $j \in \mathcal{N}_i$, i. e., node i “shares” a fraction of its local data with its neighbor j . In order to comply with this view, the elements $C(t)_{i,j}$ are commonly assumed to be chosen from the unit interval $[0, 1]$. The technical benefit of assuming that the matrices $C(t)$ are nonnegative lies in the regularity of the resulting convergence. More specifically, the nonnegativity of the matrices $C(t)$ implies that the vector of local approximation errors $(x(t) - \alpha \mathbf{1})$ converges, if it converges, monotonously in $\|\cdot\|_\infty$ (see, e. g., [Kempe, Dobra, et al. \[2003\]](#)). This is easy to see, since a (convex) combination of local estimates always lies between the smallest and the largest local estimate across all nodes. Consequently, the worst local estimate can only improve. On the other hand, all but the worst estimate can also worsen in the course of an iteration. Therefore, the monotonicity of the convergence implied by the nonnegativity of the matrices $C(t)$ is a purely global property. That the local convergence is not monotone is also one of the reasons why the purely local convergence detection is a non-trivial task (see [Chapter 5](#) for an elaboration on this subject).

Even though the utilization of nonnegative matrices $C(t)$ is natural and technically convenient, certain circumstances may require to weaken this assumption. For instance, cleverly chosen negative elements may enhance the convergence speed (cf. [Section 3.5](#)). In our convergence result in [Section 3.4](#) we allow “infrequently” occurring negative elements in the matrices $C(t)$ in the sense that we do not constrain the elements of the matrices $C(t)$, but instead we require that $\mathbb{E}[C(t)]$ is nonnegative. The possibility of coping with negative elements in the matrices $C(t)$ is not only interesting from a theoretical point of view, but also occurs in [Chapter 4](#) as a necessary requirement for our fault tolerance results.

Constraints on the rows and columns of the matrices $C(t)$ restrict the possible communication patterns for each node. First and foremost, it is crucial to note that constraints on the rows of the matrices $C(t)$ restrict the actions of the nodes as receivers whereas constraints on the columns of the matrices $C(t)$ restrict the actions of the nodes as senders. With respect to our assumptions on distributed environments it is particularly crucial to note that the nodes have no global knowledge. Thus, they can only influence what (and to whom) they send but not what (and from whom) they receive. For instance, the most fundamental constraint on the columns is that $\mathbf{1}^T C(t) = \mathbf{1}^T$. This constraint is known as *mass conservation* since it guarantees that in the course of the computation the initial data is preserved in the sense that $\sum_{i \in \mathcal{V}} x(t)_i = \sum_{i \in \mathcal{V}} x(0)_i$. Needless to say, mass conservation can be easily ensured by local means, since every node can decide how and with whom it shares its local data.

In strong contrast to mass conservation, constraints on the rows of the matrices $C(t)$ are unrealistic from a practical point of view, because a node cannot decide

Algorithm 3.2 Push-Sum [Kempe, Dobra, *et al.*, 2003].

Local variables: s_i, w_i
Initial values: $(s_i, w_i) \leftarrow (s(0)_i, w(0)_i)$

```

1: while not  $\varepsilon$ -accurate do
2:   ... at time  $t$  do ...
3:    $k \leftarrow$  choose a neighbor from  $\mathcal{N}_i$  uniformly at random
4:   Send  $(s_i, w_i)/2$  to  $k$ 
5:    $(s_i, w_i) \leftarrow (s_i, w_i)/2$ 
6:   for all received pairs  $r$  do
7:      $(s_i, w_i) \leftarrow (s_i, w_i) + r$ 
8:   end for
9: end while

```

locally what it will receive from which neighbors at a certain point of time. The second standard assumption in classical results regarding consensus algorithms is that $C(t)\mathbf{1} = \mathbf{1}$. This constraint means that all nodes receive a convex combination of local estimates of their neighbors in each time slot, which certainly requires a lot of coordination between the nodes. Such a coordination is often unrealistic in practice and it severely restricts the applicability of the resulting methods. Nevertheless, the constraint $C(t)\mathbf{1} = \mathbf{1}$ is omnipresent in the literature concerning linear iteration-based methods and only recently, avoiding this constraint was emphasized to be important from a practical point of view [Benezit, Blondel, *et al.*, 2010; Tsianos *et al.*, 2012].

In order to exemplify an algorithm which solely requires mass conservation and where the nodes do not impose constraints on what they receive, we depict the Push-Sum algorithm [Kempe, Dobra, *et al.*, 2003] in Algorithm 3.2. Analogously to Algorithm 3.1 the algorithm is round-based and hence, its corresponding analytical expression in terms of a linear iteration (sum-weight scheme) can be derived straightforwardly. However, the original analysis of Kempe, Dobra, *et al.* [2003] is not carried out in terms of linear iterations. Furthermore, it is noteworthy that the analysis of Kempe, Dobra, *et al.* [2003] is restricted to a round-based execution on a complete graph, whereas the algorithm works under much weaker assumptions on the environment as, e. g., our results also rigorously prove.

3.3.3. Asynchronicity

In the context of distributed (aggregation) algorithms, asynchronicity is a frequently recurring term. However, only some of the various sources of asynchronicity within distributed systems are considered in the existing literature. Most commonly, asynchronicity is considered in terms of differently scheduled activities of the nodes, i. e., instead of proceeding in (globally synchronized) rounds, the nodes are assumed to be (in)active independently of each other. Nowadays, there exists a great bunch of literature considering this kind of asynchronicity and well-understood formal models

are available. The standard model in this context is that each node has a clock which ticks at the times of a Poisson process. [Boyd et al. \[2006\]](#) analyzed a class of averaging algorithms for this asynchronous time model and also illustrated that the formal treatment of this kind of asynchronicity does not cause technical difficulties (compared to a synchronous, round-based execution). Analogously, many other round-based results can be lifted to this more realistic setup where the actions of the nodes are independent of each other.

It is important to note that there exist also further sources of asynchronicity, which received little attention so far. A simplification which can be incorporated fairly easily in existing results is to consider the time local computations take. In contrast to that, it is less clear how to cope with the crucial simplification that the transmission of data is by definition assumed to take no time. Recall that an element $C(t)_{j,i}$ not only specifies that node i sends $C(t)_{j,i} \cdot x(t-1)_i$ at time t to node j , but also that node j receives $C(t)_{j,i} \cdot x(t-1)_i$ at time t .⁵ In reality, the communication between nodes takes substantial amounts of time and a sender of a message can potentially proceed with its local computations long time before the intended communication partner received the message at the algorithmic level. Moreover, message exchanges between nodes can overlap, i. e., two nodes can (“at the same time”) send each other messages before either of the nodes received the message of the other node. While it seems to be fairly natural that the kind of aggregation algorithms we consider should also work under more realistic assumptions on message transmission, rigorous proofs are missing in the literature. In [Section 4.3](#) we close this gap for certain models. Recently, [Georgiou et al. \[2013\]](#) investigated the problem of rumor spreading (among others) for the case that message transmission takes finite amount of time. Our results also include a model of asynchronicity similar to the one of [Georgiou et al. \[2013\]](#). While our generic results are of an existential nature, [Georgiou et al. \[2013\]](#) derive explicit (tight) bounds in their setting. Analogous to the rumor spreading results we referred to in [Section 2.3.1](#), these bounds heavily rely on the simplicity of the rumor spreading problem and do not extend to aggregation problems.

Being able to cope with asynchronicity at the level of analytical results is important but not sufficient in practice. For a practical application we also need truly asynchronous algorithms which are (e. g., in contrast to [Algorithm 3.2](#)) not inherently round-based, but consist solely of local commands. In order to illustrate how such algorithms can look, we further refine [Algorithm 3.2](#) to [Algorithm 3.3](#) which constitutes a truly asynchronous variant of [Algorithm 3.2](#). Instead of rounds [Algorithm 3.3](#) uses a “blocking” receive command (line 5) to coordinate when a particular node is (in)active. Care is needed at this place since now a message has to be sent for each received message in order to preserve the invariant that n messages (i. e., one for each node) are floating in the system. Otherwise, the number of messages floating in the

⁵Again, “at time t ” means “in time slot t ”, i. e., there can be some time between a send and its corresponding receive operation. However, in general one cannot expect that there always exists a discretization of time such that a send and its corresponding receive operation lie within the same time slot

Algorithm 3.3 An implementable truly asynchronous variant of [Algorithm 3.2](#).

Local variables: s_i, w_i

Initial values: $(s_i, w_i) \leftarrow (s(0)_i, w(0)_i)$

```

1: while not  $\varepsilon$ -accurate do
2:    $k \leftarrow$  choose a neighbor from  $\mathcal{N}_i$  uniformly at random
3:   Send  $(s_i, w_i)/2$  to  $k$ 
4:    $(s_i, w_i) \leftarrow (s_i, w_i)/2$ 
5:   wait until pair  $r$  received from a neighboring node
6:    $(s_i, w_i) \leftarrow (s_i, w_i) + r$ 
7: end while

```

system might steadily decrease until a deadlock is reached. In the round-based formulation this invariant is preserved differently, namely by the fact that in every round each node sends exactly one message, no matter how many messages were received.

3.3.4. Topology

Independently of the algorithm used, an all-to-all aggregation can only be computed when the topology of the underlying communication network fulfills certain connectivity properties. In principle, the fundamental property required for an all-to-all aggregation is that local information from each node can reach every other node in the system. In the language of graph theory this property is known as strong connectivity⁶. Consequently, strong connectivity is—one way or another—an inevitable assumption on the topology in the context of all-to-all aggregations.

Due to the interpretation of the elements of the matrices $C(t)$ it is clear that their nonzero elements reflect the underlying communication network in so far that an element $C(t)_{j,i}$ can only be nonzero if there is an edge between the nodes i and j . However, expecting that the elements of the matrices $C(t)$ entirely capture the underlying topology implies that in every time slot all edges are used. Even though this assumption is very restrictive, it frequently occurs in the literature, especially in cases where wireless communication is assumed (see, e.g., [Olfati-Saber et al. \[2007\]](#); [Hadjicostis et al. \[2012\]](#) and references therein)⁷. While it is very restrictive to assume that the entire topology is exploited at every point of time, it is reasonable to assume that the existing communication links are utilized in the course of the computation. Among others, our convergence analysis in [Section 3.4](#) depicts that it suffices to assume connectivity properties in terms of $\mathbb{E}[C(t)]$, instead of the individual matrices

⁶We call a directed graph strongly connected if there exists (in each direction) a path between every pair of nodes.

⁷The modeling of wireless communication via linear iterations is somewhat troublesome, since it models the wireless communication in terms of concurrent point-to-point communications. This approach has drawbacks from a practical point of view. See [Section 7.2](#) for a discussion of this issue and related open questions for future research.

$C(t)$. In particular, whenever available links are not excluded algorithmically, the matrix $\mathbb{E}[C(t)]$ entirely reflects the effective topology of the communication network by its pattern of nonzero elements. Thus, the topology induced by $\mathbb{E}[C(t)]$ has to be strongly connected. However, strictly speaking this assumption is still insufficient for guaranteeing convergence, e. g., the matrix

$$C = \mathbb{E}[C(t)] = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

induces a strongly connected topology but $\lim_{t \rightarrow \infty} \mathbb{E}[C]^t x(0)$ does not exist. From Perron-Frobenius theory (cf. [Section A.3](#)) it is known that so-called *primitive* matrices ([Definition A.10](#)) induce strongly connected topologies and at the same time guarantee convergence. It is important to observe that considering primitive matrices is by no means a restriction. For instance, a matrix is primitive if it induces a strongly connected topology and it has at least one nonzero diagonal element ([Lemma A.13](#)). In our context this means that whenever at least one node does not at all times push all of its local data to its neighbors, convergence can be guaranteed. To the best of our knowledge, this property is straightforwardly fulfilled by all meaningful linear iteration-based algorithms in the literature.

While generic convergence results require only rudimentary connectivity properties, the specific shape of the considered topology plays a particularly important role in terms of the actual achievable convergence speed. In general, the convergence speed of a distributed aggregation algorithm is for the most part determined by the underlying topology. In particular, distributed algorithms cannot overcome bottlenecks by clever methods due to the lack of global knowledge and control. As we argued in [Section 2.3.1](#), the performance and dependence on connectivity measures of rumor spreading protocols can be seen as a rough estimate on how well distributed aggregation routines can perform. Moreover, the efficiency of our fault tolerance techniques we develop in [Chapter 4](#) depends also to some extent on the size of the neighborhoods $\mathcal{N}_i$.

3.3.5. Convergence Behavior

A meaningful analysis of a distributed algorithm requires results regarding its convergence to the intended target value. Care is needed at this place in the comparison of different approaches, due to the many notions of convergence in mathematical proofs. In particular, also different sorts of convergence results for distributed algorithms appear in the literature. At their heart the algorithms we consider are very flexible and allow for various kinds of random or deterministic communication schedules. However, with respect to the application of distributed matrix computations, our objective is that every node obtains an ε -accurate result of the target aggregate α in finite amount of time for (up to the limitations of floating-point arithmetic) arbitrary $\varepsilon > 0$.

In the time-homogenous case—where all $C(t)$ are equal—one expects that the limit $\lim_{t \rightarrow \infty} x(t) = \alpha \mathbf{1}$ exists at least point-wise, or preferably uniformly. In the time-

inhomogeneous case $x(t)$ is a random variable whose distribution depends on the distribution of the communication matrices $C(t)$ and thus, on the respective algorithm and assumptions on the environment. For our purposes, *second moment convergence* results are most common and well-suited. This means, if $x(t)$ denotes the approximate solution at time t and α the exact solution, then $\lim_{t \rightarrow \infty} \mathbb{E}[x(t)] = \alpha \mathbf{1}$ and $\lim_{t \rightarrow \infty} \mathbb{E}[\|x(t) - \alpha \mathbf{1}\|_2^2] = 0$. Another notion of convergence which leads to accurate results is *almost sure convergence*, i. e., $\Pr(\lim_{t \rightarrow \infty} \|x(t) - \alpha \mathbf{1}\| = 0) = 1$. Note that both second moment convergence and almost sure convergence imply *convergence in probability*, which means that for all $\varepsilon > 0$ it holds that $\lim_{t \rightarrow \infty} \Pr(\|x(t) - \alpha \mathbf{1}\| \geq \varepsilon) = 0$. However, in general neither of second moment convergence and almost sure convergence implies the other [Elstrodt, 2011, Chapter VI, § 4].

In the analysis of distributed aggregation algorithms a frequently occurring (see, e. g., Aysal *et al.* [2009]) notion of convergence which is not sufficient for our purposes is *convergence in expectation*, which means that $\lim_{t \rightarrow \infty} \mathbb{E}[x(t)] = \alpha \mathbf{1}$. Results dealing with convergence in expectation commonly do not (and cannot) include insights into how the second moment behaves. Hence, in general the variance is nonzero and therefore, the computed result is in general not the target value (the expectation $\mathbb{E}[x(t)]$) but it deviates from it.

3.4. Convergence Under Realistic Assumptions

[Niederbrucker and Gansterer, 2014, § 2]

In this section we generically analyze the convergence of sum-weight schemes on “arbitrary” (cf. Section 3.3.4) topologies $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $n = |\mathcal{V}|$. More specifically, for arbitrary input vectors $s(0)$ and $w(0)$ we consider the sum-weight scheme

$$s(t) = C(t)s(t-1), \quad w(t) = C(t)w(t-1), \quad (3.6)$$

where the matrices $C(t)$ are drawn i. i. d. from a fixed distribution $\mathcal{C}$ over the $n \times n$ matrices. In compliance with our discussion regarding “realistic” assumptions on the communication matrices $C(t)$, we assume that $\mathcal{C}$ is a so-called push-distribution.

Definition 3.1 (Push-distribution). We call $\mathcal{C}$ a *push-distribution* if

1. every matrix $C(t)$ drawn from $\mathcal{C}$ satisfies $\mathbf{1}^T C(t) = \mathbf{1}^T$, and $C(t)_{j,i} = 0$ for all $(i, j) \notin \mathcal{E}$,
2. $\mathbb{E}[C(t)]$ is a primitive matrix and $\mathbb{E}[C(t)]_{i,j} > 0$ for all edges $(i, j) \in \mathcal{E}$.

//

Push-distributions generalize the distributions Benezit, Blondel, *et al.* [2010] analyze in their closely related work with respect to two properties concerning the (individual) matrices $C(t)$. More specifically, Definition 3.1 does not impose that the matrices $C(t)$ are nonnegative and it does not impose that the matrices $C(t)$ have positive diagonal

elements. Both of these additional assumptions are crucial for the convergence result of Benezit, Blondel, *et al.* [2010]. In analogy to less general existing results concerning the convergence of sum-weight schemes, we prove in the remainder of this section the following assertion.

Theorem 3.1. *If the matrices $C(t)$ in (3.6) are i. i. d. drawn from a push-distribution, then $\lim_{t \rightarrow \infty} \mathbb{E}[\|s(t) - \alpha w(t)\|_2^2] = 0$, with $\alpha = (\sum_{i \in \mathcal{V}} s(0)_i) / (\sum_{i \in \mathcal{V}} w(0)_i)$. In particular, it holds for all $i \in \mathcal{V}$ that $\lim_{t \rightarrow \infty} \mathbb{E}[(s(t)_i/w(t)_i - \alpha)^2] = 0$.*

Note that our proof of Theorem 3.1 is a generalization of a result of Boyd *et al.* [2006], who analyzed the case when the matrices $C(t)$ are doubly-stochastic. We obtain this generalization by employing Perron-Frobenius theory (cf. Section A.3) at various places in the proof. Henceforth, we denote by $C := \mathbb{E}[C(t)]$ the expectation of the matrices $C(t)$. As a prerequisite, note the following immediate consequence of the Perron-Frobenius theorem (Theorem A.10).

Corollary 3.2. *For the matrix C it holds that $\rho(C) = 1$, where $\rho(C)$ denotes the spectral radius of C .*

Proof. By Definition 3.1 all matrices $C(t)$ fulfill $\mathbf{1}^T C(t) = \mathbf{1}^T$. Hence, also C fulfills $\mathbf{1}^T C = \mathbf{1}^T$. Therefore, it immediately follows that $\mathbf{1}^T$ is an eigenvector of C corresponding to the eigenvalue one. By Theorem A.10 we know that there exists exactly one left eigenvector of C without negative entries and that this eigenvector corresponds to the largest eigenvalue in magnitude of C . Thus, since C is primitive, $\mathbf{1}^T > 0$ and $\mathbf{1}^T C = \mathbf{1}^T$, the claim $\rho(C) = 1$ is proved. $\square$

Let $\varphi(t)$ denote the product $C(t) \cdot C(t-1) \cdot \dots \cdot C(0)$, then $s(t)$ is related to its initial value by $s(t) = \varphi(t)s(0)$. Hence, to study the convergence of $s(t)$, it suffices to study the convergence of the matrix product $\varphi(t)$.

Theorem 3.3. *Let v be the stochastic dominating eigenvector of the matrix $\mathbb{E}[C(t)]$, i. e., $\mathbb{E}[C(t)]v = v$ and $\|v\|_1 = 1$. Then the product of matrices $\varphi(t)$ converges in expectation to $v\mathbf{1}^T$, i. e., $\lim_{t \rightarrow \infty} \mathbb{E}[\varphi(t)] = v\mathbf{1}^T$.*

Proof. First, since the matrices $C(t)$ are drawn i.i.d., we can conclude that

$$\mathbb{E}[\varphi(t)] = \mathbb{E}\left[\prod_{i=0}^t C(i)\right] = \prod_{i=0}^t \mathbb{E}[C(i)] = C^t.$$

Thus, we have to study the convergence of $\lim_{t \rightarrow \infty} C^t$. By Definition 3.1 the matrix C is primitive and by Corollary 3.2 we know that the dominating eigenvalue is one with corresponding left eigenvector $\mathbf{1}^T$. Thus, as an immediate consequence of the Perron-Frobenius theorem we conclude that $\lim_{t \rightarrow \infty} \mathbb{E}[\varphi(t)] = \lim_{t \rightarrow \infty} C^t = v\mathbf{1}^T$. $\square$

Corollary 3.4. *It holds that $\lim_{t \rightarrow \infty} \mathbb{E}[s(t) - \alpha w(t)] = 0$, i. e., the sum-weight scheme (3.6) converges in expectation.*

Proof. By [Theorem 3.3](#), we know that $\lim_{t \rightarrow \infty} \mathbb{E}[s(t)] = v \cdot \sum_{i \in \mathcal{V}} s(0)_i$, as well as that $\lim_{t \rightarrow \infty} \mathbb{E}[w(t)] = v \cdot \sum_{i \in \mathcal{V}} w(0)_i$. Hence, by linearity of expectation we obtain $\lim_{t \rightarrow \infty} \mathbb{E}[s(t) - \alpha w(t)] = 0$. $\square$

In the following we show the second moment convergence result we posed in [Theorem 3.1](#). For this purpose we define the error residual $y(t)$ at time t as $y(t) = s(t) - \alpha w(t)$. Thus, $y(t)$ also evolves according to $C(t)$ because of

$$\begin{aligned} y(t+1) &= s(t+1) - \alpha w(t+1) \\ &= C(t+1)s(t) - \alpha C(t+1)w(t) \\ &= C(t+1)(s(t) - \alpha w(t)) \\ &= C(t+1)y(t). \end{aligned}$$

In order to guarantee the desired second moment convergence of the error residual $y(t)$, we prove that $\lim_{t \rightarrow \infty} \mathbb{E}[\|y(t)\|_2^2] = 0$, where $\|y(t)\|_2^2 = y(t)^\top y(t)$. [Boyd et al. \[2006, p.2512\]](#) showed that $\mathbb{E}[y(t)^\top y(t)] \rightarrow 0$ if and only if $\mathbb{E}[y(t)y(t)^\top] \rightarrow 0$. For proving the convergence of $\mathbb{E}[y(t)y(t)^\top]$, we require two auxiliary results regarding Kronecker products.

Lemma 3.5. *Let A and B be primitive matrices. Then the Kronecker product $A \otimes B$ is also a primitive matrix.*

Proof. Due to our assumptions, there exists a $k \in \mathbb{N}$ such that $A^k > 0$ and $B^k > 0$. By the generalized mixed product property of the Kronecker product (cf. [Corollary A.7](#)), $(A \otimes B)^k = A^k \otimes B^k$. This is a positive matrix, since both A^k and B^k are positive matrices. Hence, $A \otimes B$ is a primitive matrix. $\square$

Lemma 3.6. *Let $C(t) \in \mathbb{R}^{n \times n}$ be a random matrix drawn from a push-distribution. Then $\mathbb{E}[C(t) \otimes C(t)]$ is a primitive matrix.*

Proof. By definition of the expectation of a matrix, $\mathbb{E}[C(t) \otimes C(t)]$ reads as

$$\mathbb{E}[C(t) \otimes C(t)] = \begin{pmatrix} \mathbb{E}[C(t)_{1,1}C(t)] & \cdots & \mathbb{E}[C(t)_{1,n}C(t)] \\ \vdots & \ddots & \vdots \\ \mathbb{E}[C(t)_{n,1}C(t)] & \cdots & \mathbb{E}[C(t)_{n,n}C(t)] \end{pmatrix}.$$

The expectation of a Kronecker product is in general not a Kronecker product. But note that since $C(t)_{i,j}$ is either zero for all t or a random variable with positive expectation, $\mathbb{E}[C(t) \otimes C(t)]$ and $\mathbb{E}[C(t)] \otimes \mathbb{E}[C(t)]$ have the same pattern of non-zero entries, i.e., $[\mathbb{E}[C(t)] \otimes \mathbb{E}[C(t)]]_{i,j} \neq 0$ if and only if $\mathbb{E}[C(t) \otimes C(t)]_{i,j} \neq 0$ for all $1 \leq i, j \leq n^2$. Since the property of a matrix to be primitive solely relies on the pattern of non-zero entries, $\mathbb{E}[C(t) \otimes C(t)]$ is primitive if $\mathbb{E}[C(t)] \otimes \mathbb{E}[C(t)]$ is primitive. By applying [Lemma 3.5](#) we can conclude that $\mathbb{E}[C(t)] \otimes \mathbb{E}[C(t)]$ is primitive which completes the proof. $\square$

It is a well known result in the theory of the Kronecker product that the matrix equation $Y = AXB^T$ is equivalent to the linear system $\text{vec}(Y) = (B \otimes A)\text{vec}(X)$, as long as the product AXB^T exists (cf. [Example A.5](#)). By definition of the error residual $y(t)$ we see that

$$y(t+1)y(t+1)^T = C(t+1)y(t)y(t)^T C(t+1)^T.$$

By $Y(t) := y(t)y(t)^T$ we get the matrix iteration $Y(t+1) = C(t+1)Y(t)C(t+1)^T$. Then, by vectorization $\tilde{y}(t) := \text{vec}(Y(t))$, we get the vector iteration

$$\tilde{y}(t+1) = (C(t+1) \otimes C(t+1))\tilde{y}(t).$$

Let $\tilde{C} := \mathbb{E}[C(t) \otimes C(t)]$ denote the expected value of the matrices $C(t) \otimes C(t)$. Then, by $\mathbb{E}[\tilde{y}(t+1) | \tilde{y}(t)] = \tilde{C}\tilde{y}(t)$ repeated conditioning results in the linear iteration

$$\mathbb{E}[\tilde{y}(t+1)] = \tilde{C}^{t+1}\tilde{y}(0). \quad (3.7)$$

Theorem 3.7. *The error residual $\tilde{y}(t)$ converges to zero, i. e., $\lim_{t \rightarrow \infty} \mathbb{E}[\tilde{y}(t)] = \mathbf{0}$ and the rate of convergence is governed by $\rho(\tilde{C} - \tilde{v}\mathbf{1}^T)$ where $\tilde{v}$ denotes the stochastic dominating right eigenvector of $\tilde{C}$, i. e., $\tilde{C}\tilde{v} = \tilde{v}$ and $\|\tilde{v}\|_1 = 1$.*

Proof. First note that for all t the error residual $y(t)$ is orthogonal to $\mathbf{1}$. To see this, observe that

$$\sum_{i \in \mathcal{V}} s(t)_i = \sum_{i \in \mathcal{V}} s(0)_i, \quad \sum_{i \in \mathcal{V}} w(t)_i = \sum_{i \in \mathcal{V}} w(0)_i$$

holds for all t because all $C(t)$ are assumed to be column stochastic. Thus, we get

$$\begin{aligned} \sum_{i \in \mathcal{V}} s(t)_i - \alpha w(t)_i &= \sum_{i \in \mathcal{V}} s(t)_i - \alpha \sum_{i \in \mathcal{V}} w(t)_i \\ &= \sum_{i \in \mathcal{V}} s(0)_i - \sum_{i \in \mathcal{V}} s(0)_i = 0. \end{aligned}$$

Moreover, from $y(t) \perp \mathbf{1}$ with $\mathbf{1} \in \mathbb{R}^n$, we can also conclude $\tilde{y}(t) \perp \mathbf{1}$ with $\mathbf{1} \in \mathbb{R}^{n^2}$, because

$$\sum_{i=1}^{n^2} \tilde{y}(t)_i = \sum_{i \in \mathcal{V}} \left(y(t)_i \cdot \sum_{j \in \mathcal{V}} y(t)_j \right) = \sum_{i \in \mathcal{V}} y(t)_i \cdot 0 = 0.$$

Since all $C(t)$ are column stochastic, also all $C(t) \otimes C(t)$ are column stochastic and moreover, $\tilde{C}$ is column stochastic. By [Lemma 3.6](#) we can also apply the Perron-Frobenius theorem to $\tilde{C}$ and thus, we can conclude that the dominating eigenvalue of $\tilde{C}$ is one with strictly positive left eigenvector $\mathbf{1}^T$ and strictly positive right eigenvector $\tilde{v}$. In particular, $\mathbf{1}^T \tilde{v} > 0$ and one is a simple eigenvalue. Thus, the vector iteration of the form (3.7) converges to the eigenvector corresponding to the largest eigenvalue of the respective matrix unless the input vector is orthogonal to this eigenvector. Since for all

vectors $\tilde{y}(t)$, it holds that $\tilde{y}(t) \perp \mathbf{1}$ we see that $\tilde{y}(t)$ cannot converge to the dominating eigenvector $\tilde{v}$, since $\tilde{v}\mathbf{1}^T > 0$. The Perron-Frobenius theorem further delivers that $\rho(\tilde{C}) < 1$ and therefore, we see that the error residual $\tilde{y}(t)$ converges to zero, with a rate governed by $\rho(\tilde{C} - \tilde{v}\mathbf{1}^T)$. $\square$

Summarizing, the considerations above constitute a proof [Theorem 3.1](#), because the assertion of [Theorem 3.1](#) is a combination of [Corollary 3.4](#) and [Theorem 3.7](#).

3.5. Notes on Convergence Acceleration

A traditional thread of numerical mathematics is concerned with the convergence acceleration of iterative processes. For certain iterative processes simply the clever combination of the results of previous iterations can yield a faster convergence. Even though we considered time-dependent vector iterations of the form $x(t) = C(t)x(t-1)$, we have seen that the actual convergence analysis was mostly carried out with constant matrices, e. g., with $C = \mathbb{E}[C(t)]$. Consequently, we outline methods which can be used to increase the convergence speed of the power iteration and closely related methods. These convergence acceleration schemes guarantee improvements with respect to the constant matrix which they get as an input, thus it is a priori not clear how much of this improvement can also be observed when the matrix is time-dependent. However, by appropriate parameter choices one can achieve improvements of the convergence speed also in the time-dependent case.

As we have seen, the efficiency of the power iteration for a matrix C relies on the ratio $|\lambda_2/\lambda_1|$, where λ_1, λ_2 denote the two largest (in magnitude) eigenvalues of C . In sequential and parallel computing this issue is commonly tackled by appropriate shifts (i. e., $C - \mu I$) as well as by more subtle shift-and-invert methods which iterate with the matrix $(C - \mu I)^{-1}$ (cf. [Section 6.2.4](#) for more details). However, this type of acceleration approach is not suitable for yielding faster distributed aggregation algorithms due to the lack of global knowledge and control of the nodes.

The power iteration is closely related to stationary iterative methods for solving linear systems $Cx = b$. Such methods are based on a linear iteration of the form $x(t+1) = Mx(t) + Nb$, where the matrices M, N have to be carefully chosen with respect to C . The exact solution in these schemes is the stationary point of the iteration. Well-known methods of this type are, e. g., the Jacobi and Gauss-Seidl method as well as SOR [[Saad, 2003](#), § 4.1]. A stationary iterative method constitutes usually not only an iterative algorithm for solving a linear system but it also directly implies a more general convergence acceleration scheme for other iterative processes such as the ones we consider. Roughly speaking, such methods provide in each step t a polynomial $p_t(C)$ of degree t such that $p_t(C)x(0)$ converges faster than $C^t x(0)$ (e. g., with respect to some norm $\|\cdot\|$). Note that the polynomial $p_t(C)$ is not explicitly computed and it only appears in the formal analysis via recurrence relationships.

Because we consider a distributed environment where the nodes act autonomously, also the choice of potential recurrences is limited. More specifically, by our assumptions

on a distributed environment, the nodes can only keep old local values and employ them to improve the convergence speed. A suitable acceleration scheme which is in compliance with this restriction is the second-order Richardson method [Golub and Varga, 1961]:

$$x(1) = Cx(0), \quad x(t) = \beta Cx(t-1) + (1-\beta)x(t-2), \quad t > 1. \quad (3.8)$$

In load balancing this scheme is known as *second-order scheme* [Muthukrishnan et al., 1998] whereas the basic iteration (3.1) is known as *first-order scheme*. Naturally, this nomenclature is motivated by the respective convergence rates which are achieved [Muthukrishnan et al., 1998]. In principle, the iteration (3.8) converges whenever β is chosen from the open interval $(0, 2)$. However, fastest convergence is achieved by fixing β to

$$\beta = \frac{2}{1 + \sqrt{1 - \gamma^2}}, \quad \gamma = \max_{\lambda \in \sigma(A) \setminus \rho(A)} |\lambda|.$$

Further—but not substantial—improvements of the convergence speed can be achieved by using time-dependent scalars $\beta(t)$ instead of a fixed factor β . As often in polynomial approximation, the optimal solution is related to the theory of Chebyshev polynomials [Golub and Varga, 1961]. In particular, the optimal (with respect to $\|\cdot\|_2$ convergence) choice for the scalars $\beta(t)$ is given by

$$\beta(1) = 1, \quad \beta(2) = \frac{2}{2 - \gamma^2}, \quad \beta(t) = \frac{4}{4 - \gamma^2 \beta(t-1)}.$$

Iterations of the form (3.8) can also be conveniently expressed in terms of a simple vector iteration by observing

$$x(1) = Cx(0), \quad \begin{pmatrix} x(t) \\ x(t-1) \end{pmatrix} = \underbrace{\begin{pmatrix} \beta(t)C & (1-\beta(t))I \\ I & \mathbf{0} \end{pmatrix}}_{=: \tilde{C}} \begin{pmatrix} x(t-1) \\ x(t-2) \end{pmatrix}. \quad (3.9)$$

It is noteworthy that in (3.9) the matrix $\tilde{C}$ does not (fully) inherit the structure of the matrix C . In particular, if $\mathbf{1}^T C = \mathbf{1}^T$, then $\mathbf{1}^T \tilde{C} \neq \mathbf{1}^T$. Even though this property is crucial in the analysis of linear iterations such as (3.1), it is not required in the context of convergence acceleration schemes.

Faster Convergence by Less Restrictive Assumptions

Besides classical generic convergence acceleration schemes (from numerical analysis), faster convergence can also be obtained via additional assumptions on the distributed environment and methods which exploit these assumptions. Several conceptually different approaches towards the optimization of the convergence speed of linear iteration-based algorithms have been employed. For instance, one can optimize the communication patterns described by the matrices $C(t)$ with respect to a fixed topology [Boyd

et al., 2006] or optimize the topology itself [Kar and Moura, 2008]. Moreover, by assuming that the nodes have additional knowledge about the topology of the system, substantial improvements can be achieved by exploiting this additional (global) knowledge algorithmically [Dimakis, Sarwate, *et al.*, 2008; Benezit, Dimakis, *et al.*, 2010]. However, such methods rely (among other aspects) on multi-hop communication, which is not feasible under our assumptions on distributed environments (e.g., due to the unreliable communication, cf. Table 2.1).

3.6. Beyond Computing Simple Aggregates

While this thesis almost exclusively deals with problems regarding simple aggregate computations, real world computations involve also more complex operations which can be built on top of the simple aggregations we investigate. Straková [2013] investigated in detail how the methods we develop in this thesis can be employed in distributed matrix computations and showed among others that the improved robustness of our methods carries over to higher level operations such as matrix factorizations. For the sake of completeness, in the following we outline how to transfer frequently occurring patterns in parallel matrix computations to the distributed regime.

3.6.1. Repeated Aggregations

Frequently, matrix computations boil down to series of consecutive distributed aggregate computations (between the nodes) interleaved by local computations. For instance, the (modified) Gram-Schmidt orthogonalization procedure is a particularly simple computation of this kind [Straková, Gansterer, *et al.*, 2011; Gansterer *et al.*, 2013]. Algorithm 3.4 schematically depicts how such algorithms look (see Straková, Gansterer, *et al.* [2011] for an explicit illustration). In a straightforward implementa-

Algorithm 3.4 Schema of an aggregation-based distributed computation

```

1: for  $i \leftarrow 1, 2, \dots$  do
2:   perform local computations
3:   compute (distributed) global aggregate (e.g., a dot product)
4:   perform local computations
       $\vdots$ 
5: end for
```

tion one can simply employ the methods in this thesis whenever an aggregate computation is required. Already with such a simple approach, the higher level computation inherits beneficial properties from the employed aggregation algorithms [Straková, Niederbrucker, *et al.*, 2013].

To improve the overall performance one has to incorporate more flexibility, e.g., by allowing some degree of asynchronicity in the higher level operation. For instance, a

node which obtained already an accurate result in its current aggregation can move on to its local computations and can initiate the next aggregation. In the course of that, it is important that the nodes insure that the result of a particular aggregation spreads to all of the contributing nodes.

3.6.2. Broadcasting-based Algorithms

Especially block matrix computations often involve broadcasts of matrix blocks. A well-known algorithm of this kind is the parallel matrix-matrix multiplication algorithm SUMMA [Van De Geijn and Watts, 1997]. Solely for the sake of simplicity we assume that the nodes are numbered according to a $p \times p$ grid and that each node holds a block of the (square) input matrices as well as of the output matrix. Then, the most basic form of the SUMMA algorithm is depicted in Algorithm 3.5. As we can

Algorithm 3.5 SUMMA for computing $C = AB$ at node (i, j)

Input: Blocks $A_{i,j} = A(k(i-1):ki, k(j-1):kj)$, $B_{i,j}$ of A and B

Output: Block $C_{i,j}$ of $C = AB$

```

1:  $C_{i,j} \leftarrow \mathbf{0}$ 
2: for  $r \leftarrow 1$  to  $p$  do
3:   if  $r = i$  broadcast block  $A_{i,r}$  to all nodes  $(i, k)$  with  $1 \leq k \leq p$ 
4:   if  $r = j$  broadcast block  $B_{r,j}$  to all nodes  $(k, j)$  with  $1 \leq k \leq p$ 
5:    $C_{i,j} \leftarrow C_{i,j} + A_{i,r}B_{r,j}$ 
6: end for

```

see, the entire computation is done locally and only broadcasts of certain blocks are required in terms of communication between the nodes of a row/column in the grid. This utilization of broadcasts is a common pattern which can be found in parallel matrix computations. Hence, it is important to provide this functionality also in a distributed way.

Of course, every broadcast can be straightforwardly implemented in terms of an aggregation, e. g., in terms of a summation where the initiator of the broadcast contributes the respective data as input and all other nodes set their contribution to zero. However, in light of the resulting communication complexity and additional computations (over a parallel broadcast) this approach appears to be inappropriate. A better approach to reliably implement a broadcast is the utilization of rumor spreading schemes which are very efficient and naturally robust at the same time. More precisely, an aggregation-based implementation of a broadcast of a k -dimensional vector requires $\Theta(kn)$ floating-point operations per iteration, whereas rumor spreading schemes do not involve any. Moreover, aggregation-based approaches are in any case inferior in terms of communication complexity (cf. Section 2.3.2).

Lessons Learned: Algorithm, Iteration, Analysis

Distributed (aggregation) algorithms are commonly described at the node level. This means that every node runs locally the same algorithm which yields in total the desired result, e. g., the computation of an all-to-all aggregation. The precise mathematical analysis of such distributed algorithms requires an idealization of the runtime environment. Hence, it is crucial to question every assumption made in theoretical results regarding their practical consequences. Needless to say, properties which are convenient for a theoretical analysis often severely restrict the practical applicability of the respective method.

The analysis of distributed aggregation algorithms which are suited for our purposes is typically carried out in terms of linear iterations. More specifically, we pointed out that realistic environments require the usage of sum-weight schemes of the form

$$s(t) = C(t)s(t-1), \quad w(t) = C(t)w(t-1),$$

where the matrices $C(t)$ are drawn independently from a fixed distribution $\mathcal{C}$. The properties of $\mathcal{C}$ are determined by the respective algorithm and assumptions on the environment. In compliance with our correctness and accuracy requirements, we have seen that in sum-weight schemes the local estimates $s(t)_i/w(t)_i$ converge to the target aggregate α in the sense that for all nodes $i \in \mathcal{V}$

$$\lim_{t \rightarrow \infty} x(t)_i = \lim_{t \rightarrow \infty} \frac{s(t)_i}{w(t)_i} = \frac{\sum_{k \in \mathcal{N}_i} s(0)_k}{\sum_{k \in \mathcal{N}_i} w(0)_k} = \alpha.$$

Hence, the particular aggregate function (sum, average, ...) which gets computed by the nodes is determined by the initial choice of the weights $w(0)$.

Regarding technical contributions, we substantially generalized the generic convergence theory of [Boyd *et al.* \[2006\]](#) and eliminated all (from a practical point of view) problematic assumptions made therein, e. g., that $C(t)\mathbf{1} = \mathbf{1}$. Our only assumption on the individual matrices $C(t)$ was that $\mathbf{1}^T C(t) = \mathbf{1}^T$ which is known to be necessary. In contrast to the closely related work of [Benezit, Dimakis, *et al.* \[2010\]](#), we allow negative elements in the matrices $C(t)$ and do not impose assumptions on their diagonals. These generalizations are not only interesting from a theoretical point of view, but we also exploit them within our fault tolerance results in [Chapter 4](#). Our second premise was that $\mathbb{E}[C(t)]$ is primitive, which guarantees a minimal degree of connectivity of the network.

We have seen that—at its heart—the analysis of linear iterations boils down to the power iteration, which enables the utilization of well-established convergence acceleration techniques. Some of these methods cause only additional local operations and can therefore also be directly applied in practice. Last, but not least, we also outlined how basic aggregation routines can be employed in a more general context as integral part of distributed matrix computations. For an extensive study on how the fault-tolerant algorithms we develop in the upcoming [Chapter 4](#) can be employed as building block of reliable distributed matrix computations see [Straková \[2013\]](#).

4 | Computing in Unreliable Environments

Theory, Algorithms & Practice

Definition 4.1 (Push-based aggregation algorithm). We call an aggregation algorithm *push-based* if it can be modeled by a sum-weight scheme (3.6) with matrices $C(t)$ drawn from a push-distribution (cf. Definition 3.1). //

If the matrices $C(t)$ in (3.6) are drawn from a push-distribution, then $\mathbf{1}^T C(t) = \mathbf{1}^T$ holds by definition. This fundamental constraint directly implies the validity of the so-called *mass conservation* property over time

$$\sum_{i \in \mathcal{V}} (s(t)_i, w(t)_i) = \sum_{i \in \mathcal{V}} (s(0)_i, w(0)_i). \quad (4.1)$$

Note that the target aggregate is given by $\alpha = (\sum_{i \in \mathcal{V}} s(0)_i) / (\sum_{i \in \mathcal{V}} w(0)_i)$, whereas push-based algorithms approach the aggregate determined by the current overall mass $\tilde{\alpha} = (\sum_{i \in \mathcal{V}} s(t)_i) / (\sum_{i \in \mathcal{V}} w(t)_i)$. This means, whenever mass conservation is hurt, a perturbed result is computed by the nodes¹. In particular, we consider a range of failures which all lead (implicitly) to a change of mass. The difficulty of preserving mass conservation in the presence of failures is that it is a global property and that we can only employ local mechanisms in a distributed environment. Moreover, we assume failures (cf. Table 4.1) where the nodes do not know that (and by how much) the mass changed.

Our main contributions in this chapter are a fault tolerance mechanism, which is inspired by the graph theoretical flow concept, and its rigorous analysis (cf. Section 4.2). While also other fault tolerance mechanisms which can cope with changes of mass are known (cf. Section 4.1.2), only very little theoretical insights have been provided so far. Hence, we use our analytical techniques also to analyze existing fault tolerance mechanisms which have not been (fully) analyzed before. Moreover, in Section 4.3 we present formal results concerning the commonly neglected issue of truly asynchronous communication. In Section 4.4 we reveal some practical issues shared by the considered fault tolerance mechanisms and show how to (partially) overcome them.

¹For the sake of simplicity, we ignore the special case that the mass at time t is proportional to the initial mass. In this case the target aggregate is clearly not affected by the change of mass.

4.1. Fault Tolerance in Theory and Practice

As a prerequisite for our technical contributions concerning fault tolerance at the algorithmic level, we first discuss what we mean by fault tolerance and which kinds of failures we target with our methods. Moreover, we review existing fault tolerance approaches used in parallel and distributed computing.

4.1.1. Fault Tolerance: Notions & Requirements

[Niederbrucker and Gansterer, 2013, § 2.2]

What it means for a certain method to be fault tolerant depends on the respective application and user requirements. Hence, there exist many notions of fault tolerance which are employed in different contexts. For instance, one can define that an algorithm A tolerates a failure F if one of the properties (T1)–(T5) holds.

- (T1) The computation does not crash in an uncontrolled way and information about the occurrence of the failure is available (after termination).
- (T2) The computation terminates successfully, but the computed result is not necessarily correct.
- (T3) The correct result is computed with a, possibly failure-dependent, a priori unknown overhead. Concretely, handling a failure may be in some situations less efficient than starting over the computation.
- (T4) The correct result is computed with a bounded, reasonably small, overhead. By reasonably small we mean that the process of recovering from a failure takes (much) less effort than restarting the computation.
- (T5) The correct result is computed without noticeable overhead.

The nature of push-based algorithms guarantees fault tolerance at least in the sense of (T2). In contrast to discrete aggregates, the requirement (T5) is out of reach for continuous aggregates (cf. Section 2.3.2). Hence, the interesting question is up to which extent we can tolerate failures within a continuous aggregation in practice, i. e., if we have to pay an arbitrarily high price for a correct result (cf. (T3)), or if we can efficiently compute correct results despite failures occurring during the computation (cf. (T4)).

In Table 4.1 we describe the types of failures we target with the fault tolerance mechanisms which we investigate in this chapter. More specifically, we develop techniques which can handle these types of failures in the sense of (T4). Note that in case of permanent node failures we exclude the initial contribution of the failed nodes from the result. This means, we expect that the delivered result is the target aggregate function computed over all nodes which did not permanently fail throughout

Failure	Description
Message loss between nodes i and j	A message from node i to node j gets silently lost and neither of the nodes takes notice of it. This means, node i believes in the successful transmission of its message and node j does not know that node i intended to communicate with it.
Temporary failure of node $i \in \mathcal{V}$	From time t_f to t_r node i does not operate. Consequently, all messages sent to node i get silently lost, i.e., we assume that the neighbors of node i do not know about its temporary failure.
Permanent failure of link $(i, j) \in \mathcal{E}$	From time t_f on the communication link (i, j) does not work anymore and from time $t_k \geq t_f$ on both involved nodes know about the (permanent) failure.
Permanent failure of node $i \in \mathcal{V}$	From time t_f on node i does not operate anymore and from time $t_k \geq t_f$ on all nodes $j \in \mathcal{N}_i$ know about the (permanent) failure.

Table 4.1.: Description of failures we aim to handle at the algorithmic level.

the computation. The failure descriptions in Table 4.1 imply that if we can handle message loss and a permanent link failure, then we can also handle temporary and permanent node failures, respectively. Besides the failures described in Table 4.1, we show also some experimental results regarding the handling of silently occurring bit flips in Section 4.4.3.

4.1.2. Robust Distributed Aggregation Algorithms

[Gansterer *et al.*, 2013, § 2] [Niederbrucker and Gansterer, 2014, § 1]

Out of the types of failures listed in Table 4.1, silent message loss is the most challenging one (from the algorithmic point of view), because the nodes have to recover without any knowledge about the occurring failures. This is also reflected by the methods we discuss below, which are (i) designed for tolerating silently lost messages, and (ii) able to tolerate other types of failures as a byproduct of the techniques used to overcome lost messages. In a failure-prone environment push-based aggregation algorithms suffer from directly communicating “mass” (cf. Algorithm 3.1). Hence, if a message is lost, mass conservation is hurt and cannot be recovered, because neither of the nodes knows about the failure and the (amount of) lost mass.

In compliance with the considerations above, fault tolerant aggregation methods avoid in one way or the other the direct communication of mass. The commonly applied strategy to preserve mass conservation in a failure-prone environment is to keep extra information for each communication link in order to preserve the mass communicated locally for each link. This means, every node $i \in \mathcal{V}$ keeps for each

of its neighbors $j \in \mathcal{N}_i$ certain extra information in order to be able to recover from failures. Note that the failure recovery itself takes place implicitly whenever a message is successfully communicated.

Eyal *et al.* [2011] and Hadjicostis *et al.* [2012] independently used (for different aggregation algorithms) the idea of keeping cumulative information (*histories*) about the data sent and received across each link. By always sending the full history, the receiver of a message can easily tolerate previously lost messages by comparing its local history of received data with the transmitted history of sent data. Obviously, this kind of recovery from lost messages is *one-sided*, i. e., if a message from node i to node j was lost, it can only be recovered by a successfully transmitted message from node i to node j . An apparent problem of history-based algorithms is keeping the steadily growing histories reasonably small in order to avoid (severe) numerical inaccuracies. Concerning this issue, Eyal *et al.* [2011] propose a bidirectional cancellation operation, whereas Hadjicostis *et al.* [2012] leave this problem open. On the other hand, Eyal *et al.* [2011] only prove the convergence of their algorithm to the target aggregate in the case that from some point on no messages are lost. In contrast to that Hadjicostis *et al.* [2012] analyze their method without this constraint.

A competing approach to history-based algorithms is inspired by the flow concept from graph theory. Flow-based algorithms have been considered by Jesus *et al.* [2009, 2010] and Gansterer *et al.* [2013, 2011]. In principle, the underlying idea of flow-based algorithms is that the nodes keep their initial values local and share only *flows* with their neighbors. For each communication link, both attached nodes maintain a *flow variable* which represents the overall balance of communicated local values along this link. For communicating local values along a link, the sender does not transmit the local values directly, but instead adds them to the corresponding flow variable which is then sent. The receiver updates its own flow variable by the negated received flow in order to ensure the skew-symmetry of flows. By ensuring skew-symmetry flow-based algorithms do not only recover from failures but also preserve the initial mass. Moreover note that this kind of recovery of lost messages is also *two-sided*, i. e., if a message from node i to node j was lost, it cannot be recovered only by a successfully transmitted message from node i to node j but also by a message sent from node j to node i .

It is important to note that the usage of flows within the Flow Updating algorithm due to Jesus *et al.* [2009, 2010] crucially differs from the way we use flows [Gansterer *et al.*, 2013, 2011] (cf. Section 4.2.1). In particular, in Flow Updating not only flows are communicated, but also local estimates. This is necessary, since in Flow Updating the nodes obtain their local estimates of the target aggregate by averaging the (most recently received) estimates of their neighbors. It is easy to see that this kind of averaging leads to an inferior convergence speed [Gansterer *et al.*, 2013]. Moreover, neither in the original work [Jesus *et al.*, 2009] nor in later studies [Jesus *et al.*, 2010; Almeida *et al.*, 2011], Flow Updating has been formally shown to converge to the target aggregate in the presence of message loss. Hence, our analytical results (cf. Section 4.2.2) are the first theoretical results regarding the capabilities of flow-based aggregation

algorithms in the presence of failures. In general, flow-based algorithms are harder to analyze than history-based algorithms due to their capability of two-sided failure recovery. Our proof techniques are similar in spirit to the approach used by [Hadjicostis et al. \[2012\]](#) in the context of a history-based algorithm. Consequently, we also provide (as a byproduct) generic convergence results for history-based algorithms. In [Section 4.2.5](#) we present the first analytical results which quantify the intuition that the two-sided recovery of flow-based algorithms is more efficient than the one-sided recovery of history-based algorithms.

[Chen, Tron, et al. \[2010\]](#) consider a mechanism for keeping extra information which is similar to flow-based approaches. However, the severe drawback of their *corrective consensus* algorithm is that a failure recovery has to be explicitly initiated (e. g., periodically) in terms of “corrective iterations”. Therefore, one either recovers infrequently from failures, or a large overhead is caused by many corrective iterations. Moreover, it is questionable how to initiate (global) “corrective iterations” in a distributed environment.

Even though the aggregation algorithm proposed by [Mosk-Aoyama and Shah \[2006\]](#) (cf. [Section 3.1](#)) does not contain explicit fault tolerance mechanisms, it can tolerate many types of failures, since it is based solely on discrete aggregations which are very robust by definition (cf. [Section 2.3.2](#)). However, recall that this method comes with an inferior convergence speed (for accurate computations). Furthermore, there exist many results where failures lead to a non-zero second moment (see, e. g., [Kar and Moura \[2009, 2010\]](#)). Hence, such methods always compute a (bounded) perturbation of the target aggregate and therefore, they are not in compliance with our requirement of computing the exact result (despite failures).

In the large body of literature concerning distributed aggregation algorithms there are still further algorithms which are considered to be (in some sense) fault tolerant (see, e. g., [Olfati-Saber et al. \[2007\]](#); [Dimakis, Kar, et al. \[2010\]](#) and references therein). However, to the best of our knowledge, there are no further (reasonably efficient) methods which can cope with the assumptions on the environment (cf. [Table 2.1](#)) and failures (cf. [Table 4.1](#)) we consider in this thesis.

4.1.3. Fault Tolerance Methods in Parallel Computing

[[Niederbrucker, Straková, et al., 2012](#), § 1]

Fault tolerant methods for large scale computer systems have a long tradition. Solution approaches span a wide range from entirely hardware based solutions to methods working exclusively at the algorithmic level. The two main types of failures which are investigated in parallel computing are node crashes and so-called *soft-errors* (i. e., failures which do not affect the execution but the correctness of the result) like silently occurring bit flips. For future large scale systems, especially soft-errors are anticipated to become a crucial issue (see, e. g., [Dongarra, Beckman, et al. \[2011\]](#); [Cappello et al. \[2009\]](#)).

A classical generic, entirely hardware-based approach is TMR (triple modular redundancy) [Dhillon, 1987, § 5.3], where three identical systems are run in parallel and the results are determined by a two-out-of-three voting. Due to resource and energy constraints this approach is not suitable for nowadays parallel computers and cannot cope with the challenges raised by future extreme scale systems. Contrary to TMR, more recent approaches incorporate redundancy at the message passing level, i. e., in MPI libraries, resulting in much more efficient solutions [Fiala *et al.*, 2012]. However, such approaches focus only on communication correctness, i. e., the correctness of the messages sent over MPI.

A widely used fault tolerance paradigm is checkpoint/restart (C/R), where the system state is—either automatically or by a user request—written to stable storage, such that the application can be continued from a previously stored checkpoint. The major drawback shared by all flavors of C/R approaches is the overhead introduced by the expensive disk I/O. To avoid this bottleneck and to reduce the overhead of C/R in the absence of system failures, diskless checkpointing was introduced (see, e. g., Plank *et al.* [1998]). Another property shared by C/R methods is that they require information about occurring failures such that the recovery procedures can be launched. Consequently, silently occurring soft errors like bit-flips are out of scope of purely C/R-based solutions and require a more sophisticated logic, e. g., at the algorithmic level.

Due to the ever shrinking integrated circuit technology, soft errors are expected to become the rule rather than the exception. Even though hardware solutions for soft error resilience (like ECC memory) are available, they are very expensive and limited to a small number of soft errors due to the overhead they introduce. Therefore, fault tolerance already at the algorithmic level is a necessity in order to meet the fault tolerance challenges which are anticipated for the future.

A classical technique which targets the emerging need of soft error resilience in the context of matrix operations is *ABFT* (*algorithm-based fault tolerance*) [Huang and Abraham, 1984]. In ABFT, soft error resilience is achieved by extending the input matrices by checksums and by adapting the algorithms such that these checksums are updated correctly. Consequently, the checksums are used for detecting and recovering from soft errors. The original ABFT work of Huang and Abraham [1984] was restricted to a single soft error and matrix-matrix multiplication. Later work extended the principle ABFT concept to handle several (simultaneous) failures [Anfinson and Luk, 1988] and to integrate it with matrix factorizations [Luk and Park, 1988]. Moreover, in the last years the ABFT concept was extended to so-called *fail-stop failures* [Chen and Dongarra, 2008], where a failed process cannot be continued and all its data is lost.

The utilization of naturally available redundancy in iterative solvers for increasing fault tolerance was recently investigated by Chen [2011, 2013]. In contrast to classical ABFT work where failures are detected and corrected in the final result, recent ABFT-based approaches focus on detecting soft errors as early as possible to avoid their propagation [Wu *et al.*, 2011; Davies and Chen, 2013]. Furthermore, ABFT-based

factorization algorithms were shown to perform well on hybrid (CPU—GPU) [Du, Luszczek, Tomov, *et al.*, 2011] as well as on large-scale [Du, Luszczek, and Dongarra, 2012] systems.

While ABFT is a promising approach within the context of matrix computations, this concept is not attractive for operations as simple as aggregations due to the missing structure, the low complexity and the resulting high relative overhead of checksum strategies in this kind of computation. On the other hand, other fault tolerance approaches known in parallel computing are even less suitable due to the low complexity and atomicity of an aggregation. Consequently, the fault tolerant computation of aggregations in a distributed environment requires naturally fault tolerant, self-healing algorithms.

Since ABFT-based methods operate at the matrix level and they are directly encoded into the respective algorithms, the corresponding algorithmic modifications needed for achieving fault tolerance are very specific for each matrix operation. In contrast to ABFT, by building matrix operations on top of fault tolerant aggregation algorithms (cf. Section 3.6), the fault tolerance achieved on the aggregation level directly translates to the higher (matrix) level [Straková, 2013]. Of course, ABFT-based methods could still be used on top of this approach to provide an additional layer of fault tolerance.

4.2. Fault Tolerant Algorithms: Derivation & Analysis

In this section, we show how to generically derive and analyze fault tolerant variants of push-based algorithms (cf. Definition 4.1). First, we analyze the flow-based technique which was introduced together with the Push-Flow algorithm [Gansterer *et al.*, 2013, 2011]. Furthermore, we apply our analytical techniques to the generic analysis of history-based algorithms as they were (independently) considered by Eyal *et al.* [2011] and Hadjicostis *et al.* [2012]. Last but not least, in Section 4.2.5 we investigate the failure recovery overhead of flow-based and history-based algorithms theoretically as well as empirically.

4.2.1. Derivation of Flow-based Algorithms

[Niederbrucker and Gansterer, 2014, § 3]

Recall the flow concept from graph theory where a flow function assigns a flow $f_{i,j}$ to each edge $(i, j) \in \mathcal{E}$. Among other properties, flows are enforced to be *skew-symmetric*, i.e., $f_{i,j} = -f_{j,i}$ for all connected nodes i, j . Observe that the (local) skew-symmetry of flows directly implies the global property $\sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} f_{i,j} = 0$, known as *flow conservation*. In the same spirit, we show how to derive a locally preservable property which implies global mass conservation (4.1).

Inspired by graph theoretical flows, assume that we can represent the local data of node i at time t in the form

$$(s(t)_i, w(t)_i) = (s(0)_i, w(0)_i) + \sum_{j \in \mathcal{N}_i} f(t)_{i,j}$$

with the intended meaning that the current data of node i is its initial data combined with the data which “flows” at time t from/to its neighbors. In particular, a negative flow variable $f(t)_{i,j} < 0$ indicates that data is flowing from node i to node j (the local data of node i *decreases*). Accordingly, a positive flow variable $f(t)_{i,j}$ indicates that data is flowing from node j to node i (the local data of node i *increases*). If we further assume that the flows are skew-symmetric, i. e., $f(t)_{i,j} = -f(t)_{j,i}$, we observe (by flow conservation) that

$$\sum_{i \in \mathcal{V}} (s(t)_i, w(t)_i) = \sum_{i \in \mathcal{V}} \left((s(0)_i, w(0)_i) + \sum_{j \in \mathcal{N}_i} f(t)_{i,j} \right) = \sum_{i \in \mathcal{V}} (s_i(0), w_i(0)). \quad (4.2)$$

Therefore, the simple local property of skew-symmetry implies global mass conservation which push-based aggregation algorithms require for producing correct results. Moreover, if we assume that the nodes solely communicate their local flows $f(t)_{i,j}$, skew-symmetry can also be easily (re-)established in case it was violated due to a failure in the system. Concretely, once node i receives a flow $f(t)_{j,i}$ from its neighbor j , it sets $f(t)_{i,j} = -f(t)_{j,i}$ to (re-)establish skew-symmetry.

For the sake of clarity, we discuss a concrete algorithm in order to exemplify the differences between push-based algorithms and their flow-based counterparts. In particular, we consider Push-Sum ([Algorithm 3.2](#)) and its flow-based variant Push-Flow ([Algorithm 4.1](#)). Obviously, Push-Flow and Push-Sum share the same principle structure and mechanisms. Moreover, we identify the three principles of flow-based algorithms we discussed previously.

1. The local data at node i at time t is expressed in terms of the flows to the neighboring nodes (line 5 in [Algorithm 4.1](#))

$$(s(t), w(t)) = (s(0)_i, w(0)_i) + \sum_{j \in \mathcal{N}_i} f(t)_{i,j}.$$

2. The nodes only communicate flows (line 6 in [Algorithm 4.1](#)).
3. Receivers (re-)establish skew-symmetry (line 2 in [Algorithm 4.1](#)).

It is crucial to observe that the fault tolerant Push-Flow sends messages of the same size as its non-fault-tolerant counterpart Push-Sum, i. e., the introduced fault tolerance mechanisms do not lead to a communication overhead in a failure-free environment. Of course, in order to reduce the local computations, the local summation of flows (line 5 in [Algorithm 4.1](#)) can be avoided by introducing an additional variable which is updated accordingly (cf. [Section 4.4.2](#)).

Algorithm 4.1 Push-Flow [Gansterer *et al.*, 2013, 2011], a flow-based variant of Push-Sum (see Algorithm 3.2).

Local variables: $f_{i,j} = (f_{i,j}^s, f_{i,j}^w) \in \mathbb{R}^2$
Initial values: $f_{i,j} \leftarrow (0, 0)$

... at time t ...

- 1: **for all** received pairs $f_{j,i}$ **do**
- 2: $f_{i,j} \leftarrow -f_{j,i}$
- 3: **end for**
- 4: $k \leftarrow$ choose a neighbor from $\mathcal{N}_i$ uniformly at random
- 5: $f_{i,k} \leftarrow f_{i,k} - \underbrace{((s(0)_i, w(0)_i) + \sum_{j \in \mathcal{N}_i} f_{i,j})/2}_{=(s(t), w(t))}$
- 6: send $f_{i,k}$ to k
- ... handling of known permanent failures ...
- 7: **if** link $(i, j) \in \mathcal{E}$ or node $j \in \mathcal{N}_i$ failed forever **then**
- 8: $f_{i,j} \leftarrow 0$
- 9: $\mathcal{N}_i \leftarrow \mathcal{N}_i \setminus \{j\}$
- 10: **end if**

Interpretation and capabilities of flows

Before we dip into the rigorous convergence analysis of flow-based algorithms in faulty environments, we give an overview on how to interpret the introduced flow variables and how they enable fault tolerance at the algorithmic level. First, we exemplify how flows are used to algorithmically overcome silent message loss.

Consider two connected nodes i and j . This means, node i holds a flow variable $f_{i,j}$ and node j holds a flow variable $f_{j,i}$ which are both initially zero. Now, assume that at a certain point of time node i wants to send some data δ to j and that this communication is successful, i. e., prior to communication, i and j hold the flow variables $f_{i,j}$ and $f_{j,i}$, respectively and after the communication the local skew-symmetry property $\tilde{f}_{i,j} = -\tilde{f}_{j,i}$ is (re-)established. The following three cases can occur:

- $f_{i,j} = -f_{j,i}$: Local skew-symmetry holds prior to communication and therefore, the previous communication between nodes i and j was successful. Hence, node i sends $\tilde{f}_{i,j} = f_{i,j} - \delta$ to node j . Moreover, node j sets $\tilde{f}_{j,i} = -\tilde{f}_{i,j}$, i. e., $f_{j,i} - \tilde{f}_{j,i} = \delta$, which means that δ is now included into node j 's local data.
- $f_{i,j} - \Delta_i = -f_{j,i}$, $\Delta_i \neq 0$: In this case previous messages from node i to node j got lost. Thus, node i sends $\tilde{f}_{i,j} = f_{i,j} - \Delta_i - \delta$ to node j and node j sets $\tilde{f}_{j,i} = -\tilde{f}_{i,j}$, i. e., $f_{j,i} - \tilde{f}_{j,i} = \Delta_i + \delta$. Therefore, the current data δ and the sum of *all* previously lost messages Δ_i are communicated from node i to node j .

- $f_{i,j} - \Delta_i = -f_{j,i} - \Delta_j$ with $\Delta_i \neq 0, \Delta_j \neq 0$: In this case previous messages from node i to node j as well as messages from node j to node i got lost. Again node i sends $\tilde{f}_{i,j} = f_{i,j} - \Delta_i - \delta$ and node j sets $\tilde{f}_{j,i} = -\tilde{f}_{i,j}$, i. e., $f_{j,i} - \tilde{f}_{j,i} = \Delta_j + \Delta_i + \delta$. This means that node j receives δ and the sum of all messages Δ_j which node i did not receive as well as all messages Δ_i from node i it did not receive so far.

Summarizing, whenever in a flow-based algorithm a message from node i to node j is successfully transmitted, *all* unsuccessful earlier communications since the last successfully transmitted message are recovered. In particular, node j implicitly receives all messages it intended to send to node i and all messages node i intended to send to node j .

Besides silent message loss and temporary failures which can be interpreted as message loss (cf. Table 4.1) we also consider known permanent failures. As we can see in Algorithm 4.2 a permanent failure of a neighboring node or a communication link is handled by setting the corresponding flow variables to zero. The correctness of this operation follows immediately, since it does not affect mass conservation (4.1). However, one should note that this kind of failure handling is essentially equivalent to a restart of the computation (cf. Section 4.4.1).

4.2.2. Analysis of Flow-based Algorithms [Niederbrucker and Gansterer, 2014, § 3]

Having the Push-Flow algorithm at hand, it is straightforward to apply the same ideas to the generic aggregation algorithm depicted in Algorithm 3.1. The resulting generic flow-based algorithm is shown in Algorithm 4.2. In this section we prove that Algorithm 4.2 converges to the target aggregate α despite arbitrarily high message loss. More precisely, we assume that (at any time) a message from node i to node j gets lost with probability $0 \leq p_{i,j} < 1$, where the probabilities $p_{i,j}$ can be chosen arbitrarily from the interval $[0, 1)$. Formally, we use Bernoulli distributed² random variables to indicate whether a message gets lost or not. This means for each potential message (i. e., $C(t)_{j,i} \neq 0$), we draw a random variable $L(t)_{i,j}$ from a Bernoulli distribution with parameter $p_{i,j}$. To indicate a potential communication, we use random variables $S(t)_{i,j}$ defined by

$$S(t)_{i,j} = \begin{cases} 1 & C(t)_{j,i} \neq 0 \\ 0 & C(t)_{j,i} = 0 \end{cases}$$

Hence, the random variables $S(t)_{i,j} \in \{0, 1\}$ follow a Bernoulli distribution with parameter $\pi_{i,j}$, where $\pi_{i,j}$ is the (algorithm dependent) probability that node i picks node j at at time t as a communication partner.

Our generic convergence theory in Section 3.4 assumed a perfect environment without any failures. In order to utilize this existing convergence result in the presence of failures, we proceed as follows: (i) we show how to theoretically model the execution of flow-based algorithms in the presence of message loss by an extended graph

²If L follows a Bernoulli distribution with parameter p , then $\Pr(L = 1) = p$ and $\Pr(L = 0) = 1 - p$.

Algorithm 4.2 Generic shape of the flow-based variant of a push-based algorithm (cf. Algorithm 3.1). See Algorithm 4.1 for the specific flow-based algorithm Push-Flow.

Local variables: $f_{i,j} = (f_{i,j}^s, f_{i,j}^w) \in \mathbb{R}^2$
Initial values: $f_{i,j} \leftarrow (0, 0)$

... at time t do ...

- 1: **for all** $C(t)_{i,j} \neq 0$ **do**
- 2: $f_{i,j} \leftarrow -f_{j,i}$
- 3: **end for**
- 4: $(s(t), w(t)) \leftarrow (s(0)_i, w(0)_i) + \sum_{j \in \mathcal{N}_i} f_{i,j}$
- 5: **for all** $C(t)_{j,i} \neq 0$ **do**
- 6: $f_{i,j} \leftarrow f_{i,j} - C(t)_{j,i} \cdot (s(t), w(t))$
- 7: send $f_{i,j}$ to j
- 8: **end for**
- ... handling of known permanent failures ...
- 9: **if** link $(i, j) \in \mathcal{E}$ or node $j \in \mathcal{N}_i$ failed forever **then**
- 10: $f_{i,j} \leftarrow 0$
- 11: $\mathcal{N}_i \leftarrow \mathcal{N}_i \setminus \{j\}$
- 12: **end if**

structure which describes a failure-free system, and (ii) we prove that this failure-free model satisfies all necessary conditions such that the convergence theory presented in Section 3.4 applies. Together, these two steps lead to a rigorous convergence analysis of flow-based algorithms in an unreliable environment.

Flow-Recovery Graphs

We discussed in Section 4.2.1 that the principle source of fault tolerance in flow-based aggregation algorithms is that every successful communication recovers all previously failed communications by re-establishing local skew-symmetry of the respective flows. In other words, this means that information (mass) which is temporarily lost is propagated back into the system. In the following, we define a graph model based on this simple idea. In particular, we extend the given graph (based on the communication network) for every pair of connected nodes i and j by two nodes $l_{i,j}$ and $l_{j,i}$. A silently lost message from node i to node j is modeled by a successfully transmitted message from node i to node $l_{i,j}$. Moreover, by appropriately correlating the actions of the nodes $l_{i,j}$ with the actions of the nodes i and j , the recovery mechanisms inherent in the flow concept can be modeled formally.

Definition 4.2. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an arbitrary connected network graph. Then we define its *flow-recovery graph* $\mathcal{G}_f = (\mathcal{V}_f, \mathcal{E}_f)$ as follows:

1. $\mathcal{V}_f := \mathcal{V} \cup \{l_{i,j} \mid (i, j) \in \mathcal{E}\}$, i. e., $\mathcal{G}_f$ contains all nodes which $\mathcal{G}$ contains and

additionally it contains for each edge in $\mathcal{G}$ an auxiliary node called *keeper*. In particular we call $l_{i,j}$ the *j-keeper* of i , since $l_{i,j}$ “keeps” the messages node i intended to send to node j .

2. $\mathcal{E}_f := \mathcal{E} \cup \{(i, l_{i,j}), (l_{i,j}, i), (l_{i,j}, l_{i,j}), (l_{i,j}, j) \mid (i, j) \in \mathcal{E}\}$, i.e., node i can send to its j -keeper $l_{i,j}$ and $l_{i,j}$ can send to itself as well as to node j .
3. Moreover, $\kappa := |\mathcal{E}|$, i.e., $|\mathcal{V}_f| = |\mathcal{V}| + |\mathcal{E}| = n + \kappa$.

An example for the flow-recovery graph of a single connected pair of nodes is shown in Figure 4.1. //

We aim at simulating the aggregate computation in $\mathcal{G}$ by an aggregate computation in $\mathcal{G}_f$. The principle idea is that the original nodes $\mathcal{V}$ behave identically in $\mathcal{G}$ and $\mathcal{G}_f$, whereas the keeper nodes simulate the fault tolerance mechanisms inherent in the flow concept. More specifically, we obtain the following extended iteration matrices.

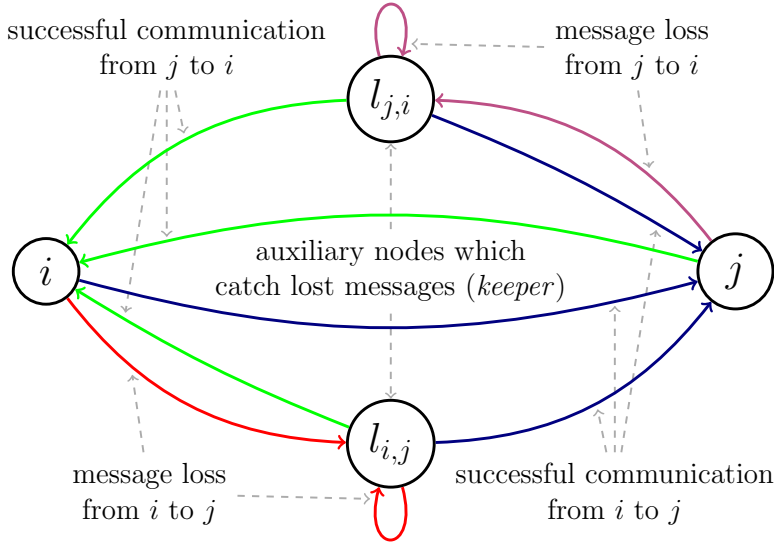


Figure 4.1.: Recovery graph for two connected nodes i and j (annotated with the intended meaning of the edges). Dependent edges (actions) are characterized by having the same target node and they are depicted in the same color.

Definition 4.3. Let $C(t)$ be drawn from a push-distribution and for each $C(t)_{j,i} \neq 0$ let $L(t)_{i,j}$ be drawn from a Bernoulli distribution with parameter $p_{i,j}$. Then we define

$$C_f(t) = \begin{pmatrix} \tilde{C}(t) & P(t) \\ R(t) & K(t) \end{pmatrix} \in \mathbb{R}^{(n+\kappa) \times (n+\kappa)}, \quad (4.3)$$

where the matrices $R(t) \in \mathbb{R}^{\kappa \times n}$, $P(t) \in \mathbb{R}^{n \times \kappa}$, and $K(t) \in \mathbb{R}^{\kappa \times \kappa}$ are defined as

$$\tilde{C}(t)_{j,i} = \begin{cases} C(t)_{i,i} + \sum_{k \in \mathcal{N}_i} L(t)_{i,k} (1 - L(t)_{k,i}) S(t)_{k,i} C(t)_{k,i} & i = j \\ C(t)_{j,i} & C(t)_{j,i} \neq 0 \wedge L(t)_{i,j} = 0 \\ 0 & \text{otherwise} \end{cases} \quad (4.4)$$

$$R(t)_{l_{i,j},k} = \begin{cases} C(t)_{j,i} & k = i \wedge C(t)_{j,i} \neq 0 \wedge L(t)_{i,j} = 1 \wedge (S(t)_{j,i} = 0 \vee L(t)_{j,i} = 1) \\ 0 & \text{otherwise} \end{cases} \quad (4.5)$$

$$P(t)_{k,l_{i,j}} = \begin{cases} 1 & (k = i \wedge C(t)_{i,j} \neq 0) \vee (k = j \wedge C(t)_{j,i} \neq 0) \\ 0 & \text{otherwise} \end{cases} \quad (4.6)$$

$$K(t)_{l_{i,j},l_{i',j'}} = \begin{cases} 1 & i = i' \wedge j = j' \wedge (C(t)_{i,j} = 0 \vee (C(t)_{i,j} \neq 0 \wedge L(t)_{i,j} = 1)) \wedge \\ & (C(t)_{j,i} = 0 \vee (C(t)_{j,i} \neq 0 \wedge L(t)_{j,i} = 1)) \\ -1 & i = i' \wedge j = j' \wedge C(t)_{i,j} \neq 0 \wedge L(t)_{i,j} = 0 \wedge \\ & C(t)_{j,i} \neq 0 \wedge L(t)_{j,i} = 0 \\ 0 & \text{otherwise} \end{cases} \quad (4.7)$$

//

It is easy to verify that the matrices $C_f(t)$ model the execution of flow-based algorithms in the presence of message loss: First, the matrices $\tilde{C}(t)$ originate in the matrices $C(t)$ and non-zero elements of $C(t)$ are only modified if messages are lost.

The matrices $R(t) \in \mathbb{R}^{\kappa \times n}$, represent the “redirection” of lost messages, i. e., if a message from node i to node j is lost in the original network $\mathcal{G}$, it is redirected to the keeper $l_{i,j}$ in the extended network $\mathcal{G}_f$. The only case where a lost message from node i to node j is not redirected to the keeper $l_{i,j}$ is when at the same time node j sends successfully to node i . In this case the “lost” message is immediately recovered (see $\tilde{C}(t)$).

The matrices $P(t) \in \mathbb{R}^{n \times \kappa}$, represent the back-propagation of lost messages, i. e., they reflect the recovery of all previously lost messages by a successful communication. More precisely, the local data of the keeper node $l_{i,j}$ consists of all lost messages from node i to node j (in $\mathcal{G}$) and hence, if $l_{i,j}$ sends all its data to node i or node j the lost messages are recovered.

Finally, the diagonal matrices $K(t) \in \mathbb{R}^{\kappa \times \kappa}$ reflect the fact that the keeper nodes $l_{i,j}$ keep all their data whenever no back-propagation takes place. Moreover, in case two nodes successfully send each other a message at the same time, the respective keeper send themselves their negated local data. It is clear that this behavior is required for ensuring $\mathbf{1}^T = C_f(t)\mathbf{1}^T$, however, its interpretation is somewhat subtle: Note that in case of a successful bidirectional communication local skew symmetry does

not hold (except of rare constellations). This means that after the next unidirectional communication, one of the two successful transmissions is undone. Hence, the behavior of the keeper is due to the fact that eventually one of the two successful transmissions is undone.

For being able to apply our convergence results we have to investigate the expectation of the matrices $C_f(t)$ obtained in (4.3).

Lemma 4.1. *Let $C_f(t)$ be the matrices defined in (4.3), then their expectation $\mathbb{E}[C_f(t)]$ is given by*

$$\mathbb{E}[C_f(t)] = \begin{pmatrix} \mathbb{E}[\tilde{C}(t)] & \mathbb{E}[P(t)] \\ \mathbb{E}[R(t)] & \mathbb{E}[K(t)] \end{pmatrix} \in \mathbb{R}^{(n+\kappa) \times (n+\kappa)}, \quad (4.8)$$

where the matrices $\mathbb{E}[R(t)] \in \mathbb{R}^{\kappa \times n}$, $\mathbb{E}[P(t)] \in \mathbb{R}^{n \times \kappa}$, and $\mathbb{E}[K(t)] \in \mathbb{R}^{\kappa \times \kappa}$ satisfy

$$\begin{aligned} \mathbb{E}[\tilde{C}(t)]_{j,i} &= \begin{cases} \mathbb{E}[C(t)]_{i,i} + \sum_{k \in \mathcal{N}_i} \mathbb{E}[C(t)]_{k,i} p_{i,k} \pi_{k,i} (1 - p_{k,i}) & i = j \\ \mathbb{E}[C(t)]_{j,i} (1 - p_{i,j}) & i \neq j \wedge \mathbb{E}[C(t)]_{j,i} \neq 0 \\ 0 & \text{otherwise} \end{cases} \\ \mathbb{E}[R(t)]_{l_{i,j},k} &= \begin{cases} \mathbb{E}[C(t)]_{j,i} p_{i,j} (1 - \pi_{j,i} (1 - p_{j,i})) & k = i \\ 0 & \text{otherwise} \end{cases} \\ \mathbb{E}[P(t)]_{k,l_{i,j}} &= \begin{cases} (1 - p_{j,i}) \pi_{j,i} & k = i \\ (1 - p_{i,j}) \pi_{i,j} & k = j \\ 0 & \text{otherwise} \end{cases} \\ \mathbb{E}[K(t)]_{l_{i,j},l_{i',j'}} &= \begin{cases} 1 - \pi_{i,j} (1 - p_{i,j}) - \pi_{j,i} (1 - p_{j,i}) & i = i' \wedge j = j' \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Proof. We simply take expectations in (4.3), where the random variables $L(t)_{i,j}$ and $S(t)_{i,j}$ are Bernoulli distributed with parameters $p_{i,j}$ and $\pi_{i,j}$, respectively. Moreover, for the matrix $\mathbb{E}[K(t)]$ note that

$$\begin{aligned} \mathbb{E}[K(t)]_{l_{i,j},l_{i,j}} &= (1 - \pi_{i,j} + \pi_{i,j} p_{i,j}) (1 - \pi_{j,i} + \pi_{j,i} p_{j,i}) - \pi_{i,j} (1 - p_{i,j}) \pi_{j,i} (1 - p_{j,i}) \\ &= (1 - \pi_{i,j} (1 - p_{i,j})) (1 - \pi_{j,i} (1 - p_{j,i})) - \pi_{i,j} (1 - p_{i,j}) \pi_{j,i} (1 - p_{j,i}) \\ &= 1 - \pi_{i,j} (1 - p_{i,j}) - \pi_{j,i} (1 - p_{j,i}). \end{aligned}$$

□

As a last technical ingredient for our analysis we have to confirm that the matrices $C_f(t)$ based on flow-recovery graphs are drawn from a push-distribution whenever the matrices $C(t)$ are drawn from a push-distribution.

Lemma 4.2. *Let the matrices $C(t)$ be drawn from a push-distribution such that the probabilities for a communication along link $(i, j) \in \mathcal{E}$ satisfy³ $0 < \pi_{i,j} \leq 1/2$. Moreover, let $0 \leq p_{i,j} < 1$ be the message loss probabilities for each link. Then the matrices $C_f(t)$ are also distributed according to a push-distribution.*

Proof. We have to verify the properties of push-distributions (cf. Definition 3.1). We verify the property $\mathbf{1}^T C_f(t) = \mathbf{1}^T$ in two steps. For the first n columns observe that the non-zero elements are exactly the non-zero elements of the respective column of the matrix $C(t)$. Thus, the first n columns of $C_f(t)$ sum up to one, since each of the columns of $C(t)$ sums up to one. For the remaining κ columns observe that in column $l_{i,j}$ only the elements $K(t)_{l_{i,j}, l_{i,j}}$, $P(t)_{j, l_{i,j}}$, $P(t)_{i, l_{i,j}}$ can be different from zero. The sum over these three elements is always one, because by definition either one of these elements is one while the others are zero or $P(t)_{j, l_{i,j}} = P(t)_{i, l_{i,j}} = 1$ and $K(t)_{l_{i,j}, l_{i,j}} = -1$. Therefore, also the sum over the entire column $l_{i,j}$ of $C_f(t)$ is one and hence, $\mathbf{1}^T C_f(t) = \mathbf{1}^T$ holds.

The primitivity of $\mathbb{E}[C_f(t)]$ can be seen as follows: First note that for each edge $(u, v) \in \mathcal{E}_f$ the corresponding element $\mathbb{E}[C_f(t)]_{u,v} > 0$ (cf. Lemma 4.1). Moreover, the flow-recovery graph $\mathcal{G}_f$ is strongly connected because all keeper nodes have a connection to and from a node in the original graph $\mathcal{G}$ (cf. Figure 4.1). Therefore, since $\mathbb{E}[C(t)]$ is primitive also $\mathbb{E}[C_f(t)]$ is primitive. $\square$

Together Lemma 4.2 and Theorem 3.1 show that flow-based algorithms converge in second moment to the target aggregate despite arbitrarily high message loss. Note that the dependency of the keeper nodes (of certain elements of $C_f(t)$) does not introduce any technical complications, because we did not make any restrictions on the (in)dependence of the nodes in Theorem 3.1.

4.2.3. Derivation of History-based Algorithms

In contrast to flow-based algorithms in history-based algorithms the nodes keep two additional variables per neighbor. More specifically, node i stores for each of its neighbors $j \in \mathcal{N}_i$ the overall data it sent to ($h^s(t)_{i,j}$) and received from ($h^r(t)_{j,i}$) this particular neighbor, i. e.,

$$h^s(t)_{i,j} = \sum_{\tau=1}^t C(\tau)_{j,i}(s(\tau)_i, w(\tau)_i), \quad h^r(t)_{j,i} = \sum_{\tau=1}^t C(\tau)_{i,j}(s(\tau)_j, w(\tau)_j).$$

Hence, the history-based representation of the local data of node i is given by

$$(s(t)_i, w(t)_i) = (s(0)_i, w(0)_i) + \sum_{j \in \mathcal{N}_i} h^s(t)_{i,j} - h^r(t)_{j,i}.$$

³This is a purely technical restriction which guarantees that the bidirectional communication at the same time (which cannot be handled by the flow concept) does not dominate. In reality, one does not have to fear this issue, since bidirectional communications of this kind are very rare, due to the several sources of asynchronicity.

In fact, the quantities $h^s(t)_{i,j} - h^r(t)_{j,i}$ correspond to the flow variables $f(t)_{i,j}$ we used before. However, substantial differences between flow-based and history-based algorithms arise in terms of the way how the recovery of lost messages happens. If node i intends to send a value v to node j , it first adds v to $h^s_{i,j}$ and thereafter transmits $h^s_{i,j}$ (instead of v). Consequently, if node j receives $h^s_{i,j}$ it computes the difference $h^s_{i,j} - h^r_{j,i}$ as new information and updates $h^r_{j,i}$ accordingly. Therefore, in contrast to flow-based methods which allow for a two-sided failure recovery, history-based approaches are characterized by an one-sided failure recovery. Applying this concept to Algorithm 3.1 leads to Algorithm 4.3, which constitutes a generic history-based algorithm. Moreover, Algorithm 4.4 shows the history-based Push-Sum variant which was called LiMoSense by Eyal *et al.* [2011].

Algorithm 4.3 Generic shape of a history-based algorithm. See Algorithm 3.1 for the generic shape of a distributed algorithm based on a linear iteration.

Local variables: $h^s_{i,j}, h^r_{j,i} \in \mathbb{R}^2$
Initial values: $h^s_{i,j}, h^r_{j,i} \leftarrow (0, 0)$

... at time t do ...

- 1: **for all** $C(t)_{i,j} \neq 0$ **do**
- 2: $h^r_{j,i} \leftarrow h^s_{i,j}$
- 3: **end for**
- 4: $(s(t), w(t)) \leftarrow (s(0)_i, w(0)_i) + \sum_{j \in \mathcal{N}_i} h^s_{i,j} - h^r_{j,i}$
- 5: **for all** $C(t)_{j,i} \neq 0$ **do**
- 6: $h^s_{i,j} \leftarrow h^s_{i,j} + C(t)_{j,i} \cdot (s(t), w(t))$
- 7: send $h^s_{i,j}$ to j
- 8: **end for**
- ... handling of known permanent failures ...
- 9: **if** link $(i, j) \in \mathcal{E}$ or node $j \in \mathcal{N}_i$ failed forever **then**
- 10: $h^s_{i,j}, h^r_{j,i} \leftarrow 0$
- 11: $\mathcal{N}_i \leftarrow \mathcal{N}_i \setminus \{j\}$
- 12: **end if**

The counterpart to the skew-symmetry of flows is that $h^s(t)_{i,j} = h^r(t)_{j,i}$ holds (is preserved) for histories, i. e., the overall data node i sent to node j equals the overall data node j received from node i . Therefore, we observe analogously to (4.2)

$$\sum_{i \in \mathcal{V}} (s(t)_i, w(t)_i) = \sum_{i \in \mathcal{V}} ((s(0)_i, w(0)_i) + \sum_{j \in \mathcal{N}_i} (h^s(t)_{i,j} - h^r(t)_{j,i})) = \sum_{i \in \mathcal{V}} (s_i(0), w_i(0)).$$

Interpretation and Capabilities of Histories

Consider again two connected nodes i and j . This means, node i holds history variables $h^s_{i,j}$, $h^r_{j,i}$ and node j holds history variables $h^s_{j,i}$, $h^r_{i,j}$ which are all initially zero. Now,

Algorithm 4.4 LiMoSense [Eyal *et al.*, 2011], a history-based variant of Push-Sum (see Algorithm 3.2).

Local variables: $h_{i,j}^s, h_{j,i}^r \in \mathbb{R}^2$
Initial values: $h_{i,j}^s, h_{j,i}^r \leftarrow (0, 0)$

... at time t ...

- 1: **for all** received pairs $h_{j,i}^s$ **do**
- 2: $h_{j,i}^r \leftarrow h_{j,i}^s$
- 3: **end for**
- 4: $k \leftarrow$ choose a neighbor from $\mathcal{N}_i$ uniformly at random
- 5: $h_{i,k}^s \leftarrow h_{i,k}^s + \underbrace{((s(0)_i, w(0)_i) + \sum_{j \in \mathcal{N}_i} h_{i,j}^s - h_{j,i}^r)/2}_{=(s(t), w(t))}$
- 6: send $h_{i,k}^s$ to k
- ... handling of permanent failures ...
- 7: **if** link $(i, j) \in \mathcal{E}$ or node $j \in \mathcal{N}_i$ failed forever **then**
- 8: $h_{i,j}^s, h_{j,i}^r \leftarrow 0$
- 9: $\mathcal{N}_i \leftarrow \mathcal{N}_i \setminus \{j\}$
- 10: **end if**

assume that at a certain point of time node i wants to send some data δ to j and that this communication is successful. In other words, prior to communication, i and j hold the history variables $h_{i,j}^s, h_{j,i}^r$ and $h_{j,i}^s, h_{i,j}^r$ and after the communication the local property $h_{i,j}^s = h_{i,j}^r$ is (re-)established. The following three cases can occur:

- $h_{i,j}^s = h_{i,j}^r$: In this case the previous communication between nodes i and j was successful. Hence, node i sends $\tilde{h}_{i,j}^s = h_{i,j}^s + \delta$ to node j . Moreover, node j sets $\tilde{h}_{i,j}^r = \tilde{h}_{i,j}^s$, i. e., $\tilde{h}_{i,j}^r - h_{i,j}^r = \delta$, which means that δ is now included into node j 's local data.
- $h_{i,j}^s + \Delta_i = h_{i,j}^r, \Delta_i \neq 0$: In this case previous messages from node i to node j got lost. Thus, node i sends $\tilde{h}_{i,j}^s = h_{i,j}^s + \Delta_i + \delta$ to node j and node j sets $\tilde{h}_{i,j}^r = \tilde{h}_{i,j}^s$, i. e., $\tilde{h}_{i,j}^r - h_{i,j}^r = \Delta_i + \delta$. Therefore, the current data δ and the sum of *all* previously lost messages Δ_i are communicated from node i to node j .

Summarizing, whenever a message from node i to node j is successfully transmitted, *all* unsuccessful earlier communications since the last successfully transmitted message are recovered in a history-based algorithm. In particular, node j implicitly receives all messages which node i intended to send to it (before). However, note that the recovery capability provided by histories is strictly one-sided.

Just as flow-based algorithms, history-based algorithms can recover from known permanent failures (cf. Table 4.1) by setting the respective histories to zero. Again,

this way of handling permanent failures is valid, since (eventually) the initial mass is preserved.

4.2.4. Analysis of History-based Algorithms

In this section, we prove the correctness of [Algorithm 4.3](#) in the presence of arbitrarily high message loss. Conceptually, we proceed analogously to the analysis of flow-based algorithms which we carried out in [Section 4.2.2](#). However, since histories provide only one-sided recovery, several technical details are simpler than in the case of flows. Note that our generic results imply the correctness of the history-based Push-Sum variant called LiMoSense (see [Algorithm 4.4](#)) which was introduced by [Eyal et al. \[2011\]](#) without a comprehensive analysis in failure-prone systems. In particular, [Eyal et al. \[2011\]](#) prove the convergence of LiMoSense only for the case that from some point on no messages are lost. Moreover, our results constitute an alternative analysis for the correctness of the history-based (consensus) algorithm considered by [Hadjicostis et al. \[2012\]](#).

History-Recovery Graphs

Similar to flow-recovery graphs, we define history-recovery graphs. The only difference is that in history-recovery graphs a keeper node $l_{i,j}$ does not have an edge back to node i .

Definition 4.4. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an arbitrary connected network graph. Then we define its *history-recovery graph* $\mathcal{G}_h = (\mathcal{V}_h, \mathcal{E}_h)$ as follows:

1. $\mathcal{V}_h := \mathcal{V} \cup \{l_{i,j} \mid (i,j) \in \mathcal{E}\}$, i.e., $\mathcal{G}_h$ contains all nodes which $\mathcal{G}$ contains and additionally it contains for each edge in $\mathcal{G}$ an auxiliary node called *keeper*. In particular, we call $l_{i,j}$ the *j-keeper* of i , since $l_{i,j}$ “keeps” the messages node i intended to send to node j .
2. $\mathcal{E}_h := \mathcal{E} \cup \{(i, l_{i,j}), (l_{i,j}, l_{i,j}), (l_{i,j}, j) \mid (i,j) \in \mathcal{E}\}$, i.e., node i can send to its j -keeper $l_{i,j}$ and $l_{i,j}$ can send to itself as well as to node j .
3. Moreover, $\kappa := |\mathcal{E}|$, i.e., $|\mathcal{V}_h| = |\mathcal{V}| + |\mathcal{E}| = n + \kappa$.

An example for the history-recovery graph of a single connected pair of nodes is shown in [Figure 4.2](#). //

With the same notational conventions as in [Section 4.2.2](#) we define extended iteration matrices $C_h(t)$ which model the execution of history-based algorithms in the presence of message loss.

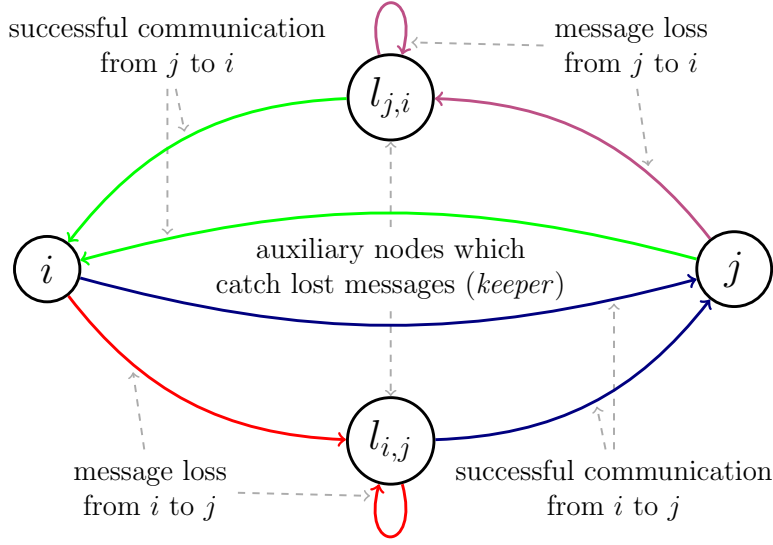


Figure 4.2.: History-recovery graph for two connected nodes i and j (annotated with the intended meaning of the edges). Dependent edges (actions) are characterized by having the same target node and they are depicted in the same color (cf. Figure 4.1).

Definition 4.5. Let $C(t)$ be drawn from a push-distribution and for each $C(t)_{j,i} \neq 0$ let $L(t)_{i,j}$ be drawn from a Bernoulli distribution with parameter $p_{i,j}$. Then we define

$$C_h(t) := \begin{pmatrix} \tilde{C}(t) & P(t) \\ R(t) & K(t) \end{pmatrix} \in \mathbb{R}^{(n+\kappa) \times (n+\kappa)}, \quad (4.9)$$

where the matrices $R(t) \in \mathbb{R}^{\kappa \times n}$, $P(t) \in \mathbb{R}^{n \times \kappa}$, and $K(t) \in \mathbb{R}^{\kappa \times \kappa}$ are defined as

$$\tilde{C}(t)_{j,i} = \begin{cases} C(t)_{i,i} & i = j \\ C(t)_{j,i} & C(t)_{j,i} \neq 0 \wedge L(t)_{i,j} = 0 \\ 0 & \text{otherwise} \end{cases} \quad (4.10)$$

$$R(t)_{l_{i,j},k} = \begin{cases} C(t)_{j,i} & k = i \wedge C(t)_{j,i} \neq 0 \wedge L(t)_{i,j} = 1 \\ 0 & \text{otherwise} \end{cases} \quad (4.11)$$

$$P(t)_{k,l_{i,j}} = \begin{cases} 1 & k = j \wedge C(t)_{k,i} \neq 0 \wedge L(t)_{i,j} = 0 \\ 0 & \text{otherwise} \end{cases} \quad (4.12)$$

$$K(t)_{l_{i,j},l_{i',j'}} = \begin{cases} 1 & i = i' \wedge j = j' \wedge (C(t)_{j,i} = 0 \vee (C(t)_{j,i} \neq 0 \wedge L(t)_{i,j} = 1)) \\ 0 & \text{otherwise} \end{cases} \quad (4.13)$$

//

In contrast to flow-based algorithms, we see that history-based algorithms can be modeled straightforwardly. In particular, if a message between node i and node j is lost, it is any case “redirected” to the respective keeper $l_{i,j}$ (by definition of $R(t)$). Moreover, also the recovery is very simple, since $l_{i,j}$ is only active if node i communicates successfully with node j (see $P(t)$ and $K(t)$).

Lemma 4.3. *Let $C_h(t)$ be the matrices defined in (4.9), then their expectation $\mathbb{E}[C_h(t)]$ is given by*

$$\mathbb{E}[C_h(t)] = \begin{pmatrix} \mathbb{E}[\tilde{C}(t)] & \mathbb{E}[P(t)] \\ \mathbb{E}[R(t)] & \mathbb{E}[K(t)] \end{pmatrix} \in \mathbb{R}^{(n+\kappa) \times (n+\kappa)}, \quad (4.14)$$

where the matrices $\mathbb{E}[R(t)] \in \mathbb{R}^{\kappa \times n}$, $\mathbb{E}[P(t)] \in \mathbb{R}^{n \times \kappa}$, and $\mathbb{E}[K(t)] \in \mathbb{R}^{\kappa \times \kappa}$ satisfy

$$\begin{aligned} \mathbb{E}[\tilde{C}(t)]_{j,i} &= \begin{cases} \mathbb{E}[C(t)]_{i,i} & i = j \\ \mathbb{E}[C(t)]_{j,i} (1 - p_{i,j}) & \mathbb{E}[C(t)]_{j,i} \neq 0 \\ 0 & \text{otherwise} \end{cases} \\ \mathbb{E}[R(t)]_{l_{i,j},k} &= \begin{cases} \mathbb{E}[C(t)]_{j,i} p_{i,j} & k = i \\ 0 & \text{otherwise} \end{cases} \\ \mathbb{E}[P(t)]_{k,l_{i,j}} &= \begin{cases} \pi_{i,j} (1 - p_{i,j}) & k = j \\ 0 & \text{otherwise} \end{cases} \\ \mathbb{E}[K(t)]_{l_{i,j},l_{i',j'}} &= \begin{cases} 1 - \pi_{i,j} (1 - p_{i,j}) & i = i' \wedge j = j' \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Proof. We simply take expectations in (4.9), where the random variables $L(t)_{i,j}$ and $S(t)_{i,j}$ are Bernoulli distributed with parameters $p_{i,j}$ and $\pi_{i,j}$, respectively. $\square$

Analogously to Lemma 4.4 we can prove that the matrices $C_h(t)$ are distributed according to a push-distribution and hence, history-based algorithms converge by Theorem 3.1 in second moment to the target aggregate despite arbitrarily high message loss.

Lemma 4.4. *Let the matrices $C(t)$ be drawn from a push-distribution and let $0 \leq p_{i,j} < 1$ be the message loss probabilities for link $(i, j) \in \mathcal{E}$. Then the matrices $C_h(t)$ are also distributed according to a push-distribution.*

Proof. We have to verify the properties of a push-distribution (cf. Definition 3.1). We verify the property $\mathbf{1}^T C_h(t) = \mathbf{1}^T$ in two steps. For the first n columns of $C_h(t)$ observe that the non-zero elements are exactly the non-zero elements of the respective column of the matrix $C(t)$. Thus, the first n columns of $C_h(t)$ sum up to one, since each of the columns of $C(t)$ sums up to one. For the remaining κ columns observe that in column $l_{i,j}$ only the elements $K(t)_{l_{i,j},l_{i,j}}$, $P(t)_{j,l_{i,j}}$ can be different from zero and that always either $K(t)_{l_{i,j},l_{i,j}}$ or $P(t)_{j,l_{i,j}}$ is one, while the other is zero. Therefore, also the sum over the entire column $l_{i,j}$ of $C_h(t)$ is one and hence, $\mathbf{1}^T C_h(t) = \mathbf{1}^T$ holds.

The primitivity of $\mathbb{E}[C_h(t)]$ can be seen as follows: First note that for each edge $(u, v) \in \mathcal{E}_h$ the corresponding element $\mathbb{E}[C_h(t)]_{u,v} > 0$ (cf. Lemma 4.3). Moreover, $\mathcal{G}_h$ is strongly connected because all keeper nodes have a connection to and from a node in $\mathcal{G}$ (cf. Figure 4.2). Therefore, since $\mathbb{E}[C(t)]$ is primitive also $\mathbb{E}[C_h(t)]$ is primitive. $\square$

4.2.5. Recovery Overhead: Insights and Bounds

As we have seen in the previous sections, flow-based as well as history-based algorithms recover from lost messages by subsequent successful transmissions. Hence, one would expect that the overhead caused by message loss depends mainly on the message loss probabilities $p_{i,j}$ and the probabilities for transmissions $\pi_{i,j}$.

In this section, we show that this intuition can be confirmed in theory and observed in practice. For the sake of a concise presentation we assume throughout this section that the underlying graph is regular with node degree d and that the message loss probability $p = p_{i,j}$ is equal for all edges $(i, j) \in \mathcal{E}$. Moreover, we consider the Push-Sum (Algorithm 3.2) variants Push-Flow (Algorithm 4.1) and LiMoSense (Algorithm 4.4) as exemplary algorithms, i. e., $\pi_{i,j} = 1/d$. This means, in this section we denote by $C(t)$, $C_f(t)$ and $C_h(t)$ the matrices corresponding to an execution of Push-Sum, Push-Flow and LiMoSense, respectively. Since we consider always expectations, we use abbreviations of the form $C_f = \mathbb{E}[C_f(t)]$.

Theoretical Insights

The convergence speed of sum-weight schemes is determined by spectral properties of the matrix $C = \mathbb{E}[C(t)]$. In particular, the second largest eigenvalue (in magnitude) of C determines the convergence speed (see Sections 3.4 and A.4). Hence, in order to quantify the overhead caused by message loss we aim at a characterization of the spectra $\sigma(C_f)$, $\sigma(C_h)$ of the extended matrices C_f , C_h in terms of the spectrum $\sigma(C)$ of C and the parameters p and d .

It is well known that for 2×2 block matrices the determinant can be computed by (see e. g., Silvester [2000, § 5])

$$\det\left(\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}\right) = \det(A_{2,2})\det(A_{1,1} - A_{1,2}A_{2,2}^{-1}A_{2,1}), \quad (4.15)$$

whenever $A_{2,2}^{-1}$ exists. With the identity (4.15) we obtain (cf. (4.3))

$$\det(C_f - \mu I) = \det(K - \mu I) \det(\tilde{C} - \mu I - P(K - \mu I)^{-1}R).$$

For both flow-based and history-based algorithms it holds that $K = c_k I$ for a certain constant c_k (see Lemmas 4.1 and 4.3). Therefore, we see that

$$\begin{aligned} \det(C_f - \mu I) &= \det((c_k - \mu)I) \det(\tilde{C} - \mu I - (c_k - \mu I)^{-1}PR) \\ &= (c_k - \mu)^{\kappa-n} \det((c_k - \mu)(\tilde{C} - \mu I) - PR) \end{aligned}$$

Hence, because $\kappa > n$, $\mu \in \mathbb{C}$ is an eigenvalue of C_f if and only if either $\mu = c_k$ holds or $\det((c_k - \mu)(\tilde{C} - \mu I) - PR) = 0$.

Lemma 4.5. *Let λ_2^f and λ_2^h denote the second largest eigenvalues of the matrices $C_f = \mathbb{E}[C_f(t)]$ and $C_h = \mathbb{E}[C_h(t)]$, respectively. Then it holds that*

$$|\lambda_2^f| \geq 1 - 2\frac{1-p}{d}, \quad |\lambda_2^h| \geq 1 - \frac{1-p}{d}.$$

Proof. We know that the dominating eigenvalue of the matrix C_f is one (and simple). Hence, all other eigenvalues can serve as lower bound for the magnitude of the second largest eigenvalue λ_2^f . Under the assumptions we make throughout this section, the matrix $K = (1 - 2(1-p)/d)I$ (see Lemma 4.1). Hence, by the considerations above $\det(C_f - \mu I) = 0$ for $\mu = 1 - 2(1-p)/d$. The bound for $|\lambda_2^h|$ follows analogously from Lemma 4.3. $\square$

Under the assumptions we made in this section, Lemma 4.1 implies for C_f that the respective non-zero elements of its sub-matrices R , P , K are equal. The same holds for C_h as well, due to Lemma 4.3. For the sake of a concise notation, we denote these (for C_f and C_h different) elements by c_r , c_p and c_k , respectively. Moreover, note that $\text{diag}(\tilde{C}) = c_d I$ and that all non-zero off-diagonal elements are equal to a certain constant c_c . As we have seen before, it holds that

$$\det(\tilde{C} - \mu I - P(K - \mu I)^{-1}R) = \det(\tilde{C} - \mu I - (c_k - \mu)^{-1}PR).$$

By assigning the node $l_{i,j}$ the row/column index $n + (i-1)d + j$ (in C_f , C_h), it is straightforward to verify that $PR = c_{pr}(\tilde{C} - c_d I)$ holds. As an example, consider for the case of histories (cf. Definition 4.5) a complete graph with four nodes

$$\begin{aligned} PR &= c_p \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} c_r \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} = c_{pr}(\tilde{C} - c_d I) = \frac{c_p c_r}{c_c}(\tilde{C} - c_d I). \end{aligned}$$

By the identity $PR = c_{pr}(\tilde{C} - c_d I)$, we observe

$$\begin{aligned}
 0 &= \det(K - \mu I) \det(\tilde{C} - \mu I - P(K - \mu I)^{-1}R) \\
 &= (c_k - \mu)^\kappa \det(\tilde{C} - \mu I - (c_k - \mu)^{-1}PR) \\
 &= (c_k - \mu)^\kappa \det(\tilde{C} - \mu I - (c_k - \mu)^{-1}c_{pr}(\tilde{C} - c_d I)) \\
 &= (c_k - \mu)^{\kappa-n} \det((c_k - \mu)(\tilde{C} - \mu I) - c_{pr}(\tilde{C} - c_d I)) \\
 &= (c_k - \mu)^{\kappa-n} \det((c_k - \mu - c_{pr})\tilde{C} - ((c_k - \mu)\mu - c_{pr}c_d)I).
 \end{aligned}$$

Hence, a general characterization of the spectra $\sigma(C_f)$ and $\sigma(C_h)$ is given by

$$\sigma(C_{\{f,h\}}) = \{c_k\} \cup \{\mu \in \mathbb{C} \mid (c_k - \mu)\mu - c_{pr}c_d = \lambda(c_k - \mu - c_{pr}), \text{ with } \lambda \in \sigma(C)\}. \quad (4.16)$$

Note that $\sigma(C_f)$ and $\sigma(C_h)$ are different, since the matrices C_f and C_h lead to different constants c_k , c_{pr} , and c_d .

Interpretation of Bounds

For algorithms whose convergence speed can be characterized by the spectral gap $1 - \lambda_2$ (cf. [Section A.4](#)), [Lemma 4.5](#) yields factors on the overhead compared to a failure-free scenario. For instance, [Boyd et al. \[2006\]](#) consider a certain kind of push-based algorithms which need $\mathcal{O}(\frac{\log n + \log \varepsilon^{-1}}{1 - \lambda_2})$ iterations to deliver an ε -accurate result at each node. Hence, flow-based and history-based variants of these algorithms require at least $\mathcal{O}(\frac{d}{1-p}(\log n + \log \varepsilon^{-1}))$ iterations to converge on d -regular graphs, where in case of histories the constant is larger by a factor of two. Note that this is the first bound of its sort, because in the existing literature regarding link-based fault tolerance mechanisms (cf. [Section 4.1.2](#)) no analytical results concerning the overhead caused by silent message loss are available.

In general, the bounds in [Lemma 4.5](#) indicate that the influence of the message loss probability p is relatively small compared to the node degree d (except of the unrealistic case that p is close to one). This intuition is also witnessed by the experiments we present below (see [Figure 4.3](#)). The matrices $K(t)$ encode the frequency of (implicit) recovery operations. Hence, the bounds obtained in [Lemma 4.5](#) are topology-independent, since they take only this frequency of local recovery operations into account. Consequently, the higher the message loss probability, the more important is the frequency of the failure recovery and thus, the tighter are the bounds obtained in [Lemma 4.5](#). If the failure recovery does not dominate (e.g., due to a low message loss probability), the topology determines (mostly) the convergence speed (cf. (4.16)).

Observed Overhead

In the following, we exemplify that the overhead caused by message loss indeed behaves (on the whole) as indicated by the bounds on the spectral gap we derived before.

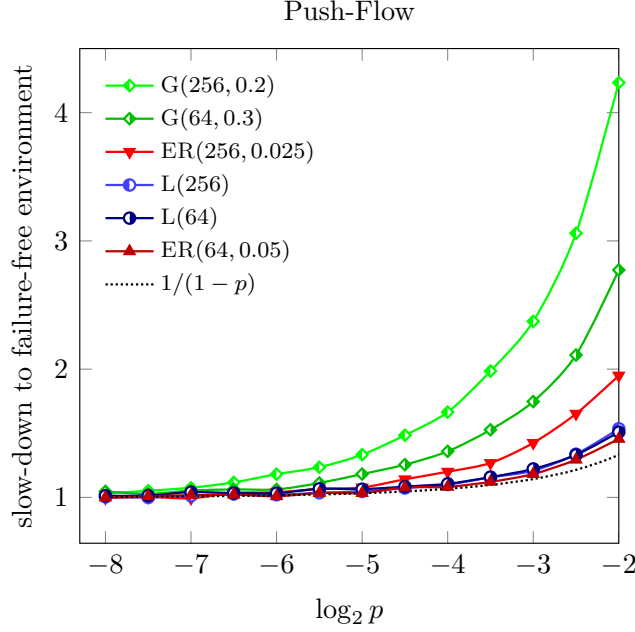


Figure 4.3.: Slow-down caused by silent message loss with uniform probability p is depicted for several topologies (cf. Table 4.2). As expected, the recovery of Push-Flow is the faster the smaller the neighborhoods of the nodes are.

First, in Figure 4.3 we depict the ratio (*slow-down*) between the number of iterations required in the presence of message loss with uniform probability p (with Push-Flow) and the respective number of iterations required to reach an accuracy of $\varepsilon = 10^{-8}$ in a failure-free environment (with Push-Sum). As a baseline, we consider a slow-down of $1/(1-p)$, because if the message loss probability is p , then $m/(1-p)$ messages are required to successfully transmit m messages.

We see that even for extremely high rates of silent message loss the flow-based algorithms compute accurate results efficiently. Moreover, Figure 4.3 highlights that the slow-down is influenced by the purely local property of a large node degree (cf. Table 4.2) rather than by global properties like the diameter which are known to strongly influence the convergence speed of distributed aggregation algorithms. Altogether, the slow-downs depicted in Figure 4.3 fully comply with our theoretical expectations.

As a second scenario we compare LiMoSense and Push-Flow in order to exemplify the different recovery performance of history-based and flow-based algorithms. In particular, Figure 4.4 illustrates the increase in the number of iterations required by LiMoSense and Push-Flow with increasing message loss. We see that Push-Flow indeed requires substantially fewer iterations for convergence than LiMoSense even though they are equivalent if no failures occur. However, the difference is (significantly) smaller than the factor of two, which we observed in our idealized bounds on the respective spectral gaps (cf. Lemma 4.5).

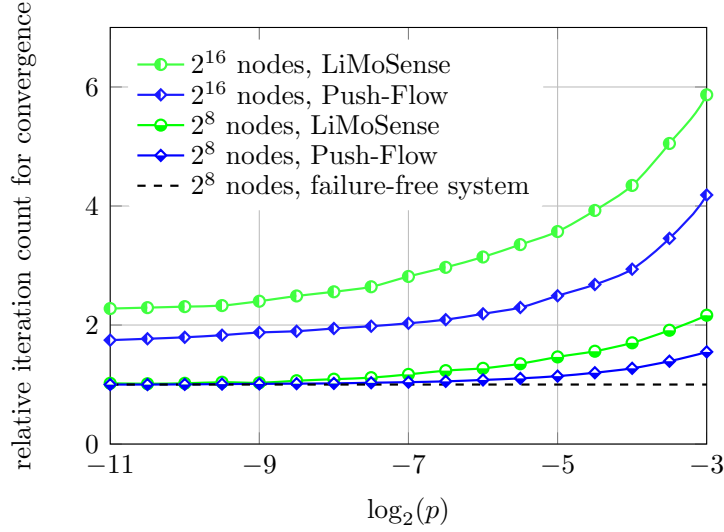


Figure 4.4.: Recovery overhead and scalability of LiMoSense and Push-Flow for averaging on hypercube topologies relative to a failure-free execution.

Graph	$\frac{\#Edges}{\#Nodes}$	Node degree distribution			Diameter
		Mean $\pm$ std	Min	Max	
G(256, 0.2)	14.59	29.18 ± 8.47	5	44	8
G(64, 0.3)	7.00	14.00 ± 5.14	4	25	6
ER(256, 0.025)	3.16	6.32 ± 2.32	1	12	6
L(256)	1.88	3.75 ± 0.47	2	4	30
L(64)	1.75	3.50 ± 0.62	2	4	14
ER(64, 0.05)	1.58	3.16 ± 1.62	1	7	8

Table 4.2.: Parameters of the graphs used in the experiments (cf. Figure 4.3).

4.3. Coping with Asynchronicity

As we pointed out before in Section 3.3.3, existing analytical results regarding linear iteration-based methods do not take into account that sending messages takes significant amount of time (cf. Section 3.4). In the simple model $x(t) = C(t)x(t-1)$, a nonzero element $C(t)_{i,j}$ indicates a send as well as receive operation at the same time. Hence, the assumption of negligible transmission time is inherent to this model. However, by proceeding analogously to our fault tolerance results in Sections 4.2.2 and 4.2.4, we can construct an extended graph structure which can be used to model the effects of asynchronous communication.

In the following, we analyze an exemplary model of asynchronous communication closely related to the one considered by Georgiou *et al.* [2013] in the context of rumor spreading. Later, we assume that the transmission of a message from node i to node j takes $\tau(t)_{i,j}$ iterations, where $\tau(t)_{i,j}$ is chosen randomly from the set $\{\tau_{\min}, \tau_{\min} +$

$1, \dots, \tau_{\max}$ (according to an arbitrary distribution). This means, if node i sends at time t a message to node j , then node j receives the message at time $t + \tau(t)_{i,j}$. Henceforth, we assume that we are given for each element $C(t)_{j,i} \neq 0$ (i.e., for each message sent) a transmission time $\tau_{\min} \leq \tau_{i,j}(t) \leq \tau_{\max}$ for the respective message to be delivered. Furthermore, we assume that all $\tau_{i,j}(t)$ are independently drawn. In particular, we do not require that the messages are delivered in the order they were sent.

Analogously to our fault tolerance results, we imitate asynchronous communication by an appropriately extended graph structure.

Definition 4.6. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an arbitrary connected network graph. Then we define its *asynchronicity graph* $\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a)$ as follows:

1. $\mathcal{V}_a := \mathcal{V} \cup \{l_{i,j}^{\tau,p} \mid (i,j) \in \mathcal{E}, \tau_{\min} \leq \tau \leq \tau_{\max}, 1 \leq p \leq \tau\}$
2. $\mathcal{E}_a := \{(i, l_{i,j}^{\tau,1}), (l_{i,j}^{\tau,p}, l_{i,j}^{\tau,p+1}), (l_{i,j}^{\tau,\tau}, j) \mid (i,j) \in \mathcal{E}, \tau_{\min} \leq \tau \leq \tau_{\max}, 1 \leq p < \tau\}$.

Moreover, we define $\kappa := |\mathcal{V}_a/\mathcal{V}|$, i.e., $|\mathcal{V}_a| = n + \kappa$. See Figure 4.5 for an illustration of the principle structure of $\mathcal{G}_a$. //

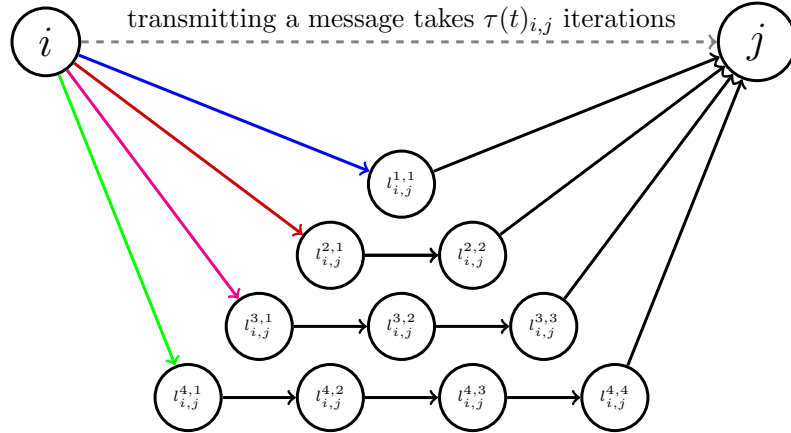


Figure 4.5.: Extension of edge $(i, j) \in \mathcal{E}$ in an asynchronicity graph $(\mathcal{V}_a, \mathcal{E}_a)$ with transmission times ranging from $\tau_{\min} = 1$ to $\tau_{\max} = 4$.

Definition 4.7. Let the matrices $C(t)$ be drawn from a push-distribution and $\tau(t)_{i,j}$ be the respective message transmission times for all $C(t)_{j,i} \neq 0$. Then we define an iteration

$$C_a(t) := \begin{pmatrix} \text{diag}(C(t)) & P(t) \\ R(t) & K(t) \end{pmatrix} \in \mathbb{R}^{(n+\kappa) \times (n+\kappa)}, \quad (4.17)$$

where the matrices $R(t) \in \mathbb{R}^{\kappa \times n}$, $P(t) \in \mathbb{R}^{n \times \kappa}$, and $K(t) \in \mathbb{R}^{\kappa \times \kappa}$ are defined as

$$\begin{aligned} R(t)_{l_{i,j}^{\tau,p},k} &:= \begin{cases} C(t)_{j,i} & k = i \wedge C(t)_{j,i} \neq 0 \wedge \tau = \tau(t)_{i,j} \wedge p = 1 \\ 0 & \text{otherwise} \end{cases}, \\ P(t)_{k,l_{i,j}^{\tau,p}} &:= \begin{cases} 1 & k = j \wedge p = \tau \\ 0 & \text{otherwise} \end{cases}, \\ K(t)_{l_{i,j}^{\tau,p}, l_{i',j'}^{\tau',p'}} &:= \begin{cases} 1 & p+1 = p' \wedge i = i' \wedge j = j' \wedge \tau = \tau' \\ 0 & \text{otherwise} \end{cases}. \end{aligned}$$

//

It is straightforward to see that a sum-weight scheme with the matrices $C_a(t)$ describes the model of asynchronous communication we introduced before: First, for each element $C(t)_{j,i} \neq 0$ we see that the respective message is sent to node $l_{i,j}^{\tau(t)_{i,j},1}$. Thereafter, the nodes $l_{i,j}^{\tau(t)_{i,j},p}$ propagate the message $\tau(t)_{i,j}$ steps until it reaches the intended receiver j . Moreover, a message can indeed “overtake” a previously sent message by construction of $\mathcal{G}_a$ (cf. Figure 4.5).

Lemma 4.6. *Let the matrices $C(t)$ be drawn from a push-distribution and let $\tau(t)_{i,j}$ be the corresponding message transmission times. Then the matrices $C_a(t)$ are also distributed according to a push-distribution.*

Proof. We show $\mathbf{1}^T C(t) = \mathbf{1}^T$ in two steps: For the first n columns note that the non-zero elements within the columns of $C(t)$ and the first n columns of C_a are identical by definition of the matrix $R(t)$ in Definition 4.7. For the remaining columns, this property follows immediately by the fact that all nodes $l_{i,j}^{\tau,p}$ have a single outgoing edge, which they use in each iteration according to the definition of the matrices $K(t)$, $P(t)$ in Definition 4.7. The remaining properties of a push-distribution follow analogously to the proof of Lemma 4.2. $\square$

Lemma 4.6 and Theorem 3.1 together show that push-based algorithms can cope with truly asynchronous message transfer. In the light of (positive) empirical results in asynchronous environments (see, e. g., Straková [2013, § 8]) this result does not surprise. On the other hand, a formal justification that push-based algorithms converge to the correct result in the presence of asynchronous communication is clearly valuable, since the existing (theoretical) literature neglects this aspect. A further more precise analysis of the impact of asynchronicity on the convergence speed appears to be delicate. In particular, the structure of the matrix $C_a(t)$ precludes a spectral analysis in the spirit of Section 4.2.5. Needless to say, there are further interesting models of asynchronous communication for which one can verify the convergence of push-based algorithms analogously to the results we presented in this section.

4.4. From Theory to Practice

Complementary to our (mostly) theoretical investigations, in this section we elaborate on practical aspects concerning the fault tolerant aggregation algorithms we analyzed before. In particular, we identify subtleties due to floating-point arithmetic and the recovery from permanent failures. For the special case of the Push-Flow algorithm, we show how to (partially) overcome these problems by a refined flow-based mechanism. However, note that our results apply straightforwardly to any other flow-based algorithm as well. As before, all our experimental results stem from round-based MATLAB implementations of the respective algorithms. Unless otherwise stated, all results are averages over 100 independent runs and the initial data is for each run (independently) drawn from a standard uniform distribution⁴. Since the individual deviations from the average are in neither of the experiments substantial (cf. [Niederbrucker and Gansterer \[2013\]](#)), we refrain from presenting them (visually) for the sake of a concise presentation.

4.4.1. Weaknesses of Existing Methods [[Niederbrucker, Straková, et al., 2012](#), § 2]

In the following, we reveal that in practice link-based fault tolerance mechanisms such as flows and histories share two common problems. On the one hand, we illustrate that both mechanisms lead to severe accuracy problem in floating-point arithmetic. On the other hand, we show that in both cases the handling of permanent failures is very inefficient since it is roughly equivalent to restarting the entire computation. From a practical point of view, we exemplify our investigations in terms of Push-Sum, its flow-based variant Push-Flow, and its history-based variant LiMoSense. In the presentation of experimental results we denote by PS, PF, LMS the algorithms Push-Sum, Push-Flow, and LiMoSense, respectively.

Numerical Inaccuracy

A central need for reliable scientific algorithms is to provide a pre-determined level of accuracy in the results. Therefore, we expect from a robust distributed aggregation algorithm that the local estimates $x(t)_i$ of the target aggregate α (can) satisfy at some time t the bound $\|x(t) - \alpha \mathbf{1}\|_\infty / |\alpha| \leq c(n)\varepsilon_{\text{mach}}$, where the quantity $c(n)$ grows only modestly with the number of nodes n .

Existing analyses of distributed reduction algorithms are commonly of a purely theoretical nature and they neglect the effects of floating-point arithmetic. Moreover, both flow-based and history-based algorithms contain operations which potentially lead to cancellation. In the following, we exemplify the accuracy problems of link-based fault tolerance mechanisms in terms of a simple case study.

⁴See [Niederbrucker and Gansterer \[2013\]](#) for other choices of the initial data distribution (which eventually lead to very similar results).

Case Study—Bus Network Consider a system consisting of n nodes, connected via a bus-style network which only allows for communication with neighboring nodes. I. e., for $1 < i < n$, node i can only communicate with its neighbors $i - 1$ and $i + 1$, node 1 can only communicate with node 2 and node n can only communicate with node $n - 1$. We assume that every node hosts a single scalar value s_i and that these local values are initialized as $s_1 = n + 1$ and $s_i = 1$ for all $1 < i \leq n$. Based on the described environment we consider that the nodes employ Push-Flow for computing the average over all s_i . The according initial states of the nodes are illustrated in the upper part of Figure 4.6. For the sake of simplicity, in this case study we omit the weights used in Push-Flow and assume them to be constantly one due to a regular, synchronous communication schedule (cf. Section 3.2.2). Moreover, note that the flow variables $f_{i,j}$ hosted by a node i are visually identified as outgoing arcs.

If Push-Flow is used to compute the global average (at each node) in the problem setup we described above and if the effects of floating-point arithmetic are ignored, the nodes reach the states depicted in the lower part of Figure 4.6. More specifically, convergence is reached after the flow variables approached⁵ the values $f_{i-1,i} = -n+i-1$ and $f_{i,i-1} = n-i+1$ for all $1 < i \leq n$. Note that for a growing number of nodes n , the target aggregate remains two in this case study, whereas the values of the local flows grow linearly with n . Consequently, several nodes certainly experience cancellation and accuracy problems in floating-point arithmetic, because the average (and therefore also the local sum of flows $\sum_{j \in \mathcal{N}_i} f_{i,j}$) is constant, whereas the individual flows grow linearly and cancel each other out. Even if the sum of flows is stored in a single variable (for efficiency reasons), the updates of this variable would still lead to inaccurate results due to the (linear) growth of the flow variables.

It is clear that the illustrated problems are not specific for Push-Flow but they are shared by all flow-based algorithms. In history-based algorithms even more crucial cancellation effects have to be expected, since after each successful communication the difference $h_{i,j}^s - h_{j,i}^r$ is computed, where neither $h_{i,j}^s$ nor $h_{j,i}^r$ are bounded. In case of flows, we at least know that the local flows are bounded (e. g., by the sum over the initial data).

The case study discussed above illustrates that various system parameters may influence the final accuracy of the results of the Push-Flow algorithm. More specifically, the achievable accuracy depends on (i) the number of nodes, (ii) the network topology, (iii) the initial data distribution, and (iv) the type of target aggregate. In Figure 4.7 we present experimental results which verify the aforementioned concerns about the achievable numerical accuracy. It is apparent that the resulting accuracy of LiMoSense and Push-Flow gets rapidly worse with increasing scale whereas Push-Sum provides accurate results also with increasing scale. Moreover, we see that also the type of aggregation appears to affect the achievable accuracy.

⁵Only because of the simplicity of the considered bus network and communication pattern, there exists a unique limit for the flow variables. In general, the flow variables approach (depending on the communication schedule) different values in different runs.

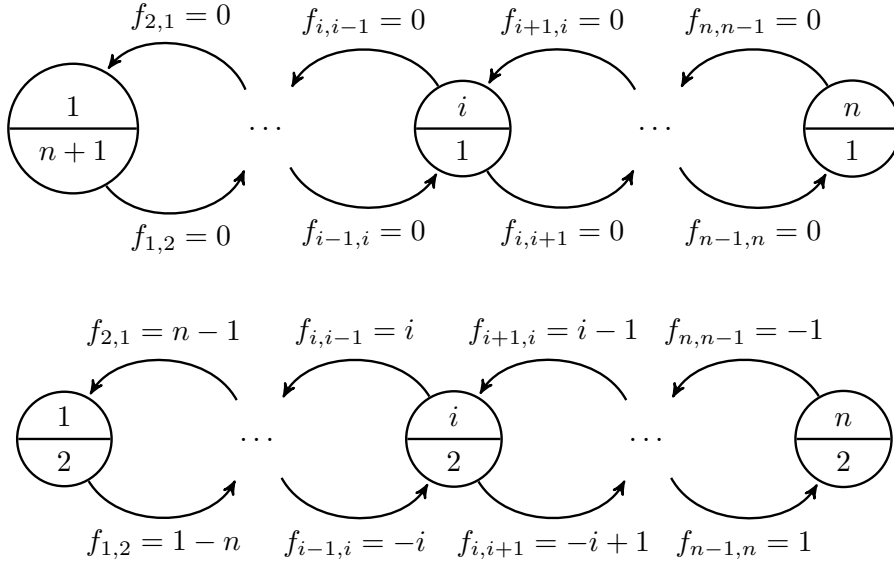


Figure 4.6.: Schematic representation of a bus network before (top) and after (bottom) running the Push-Flow algorithm for averaging the local initial data. Nodes are consecutively numbered from 1 to n from left to right and represented by circles. Each circle contains the index (upper part) and the current local data (lower part) of the corresponding node i which is computed as $s_i + \sum_{j \in \mathcal{N}_i} f_{i,j}$.

Restart Implied by Failure Handling

In [Section 4.2](#) we saw how system failures can be handled in two ways via link-based fault tolerance mechanisms. First, algorithms using these techniques recover from (silently) lost messages just by following the steps in the algorithm. Second, permanently failed components are excluded from the computation by setting the corresponding flows (histories) to zero: e. g., if the link between nodes i and j fails permanently, $f_{i,j}$ and $f_{j,i}$ ($h_{i,j}^{\{s,r\}}$ and $h_{i,j}^{\{r,s\}}$) are set to zero. In the following, we restrict our attention (without loss of generality, see [Table 4.1](#)) to handling a single permanent link failure.

After an algorithm equipped with a link-based fault tolerance mechanism converged, an equilibrium state is reached in which the flows (histories) describe a data flow which guarantees that every node can locally compute the global target aggregate. Moreover, we saw that the flows (histories) are not directly connected to the aggregate values and they might even differ from those by orders of magnitude. Hence, if we “artificially” set flows (histories) to zero, e. g., as a reaction to a failure in the system, the local values of the corresponding nodes may change arbitrarily within the range of the initial data. Therefore, this kind of recovery is roughly equivalent to a restart of aggregation. In [Figure 4.8](#) we show the impact of handling a permanent link failure at different stages of the convergence. We see that the later the failure recovery takes place the higher are the recovery costs. In particular, we observe that Push-Flow as well as

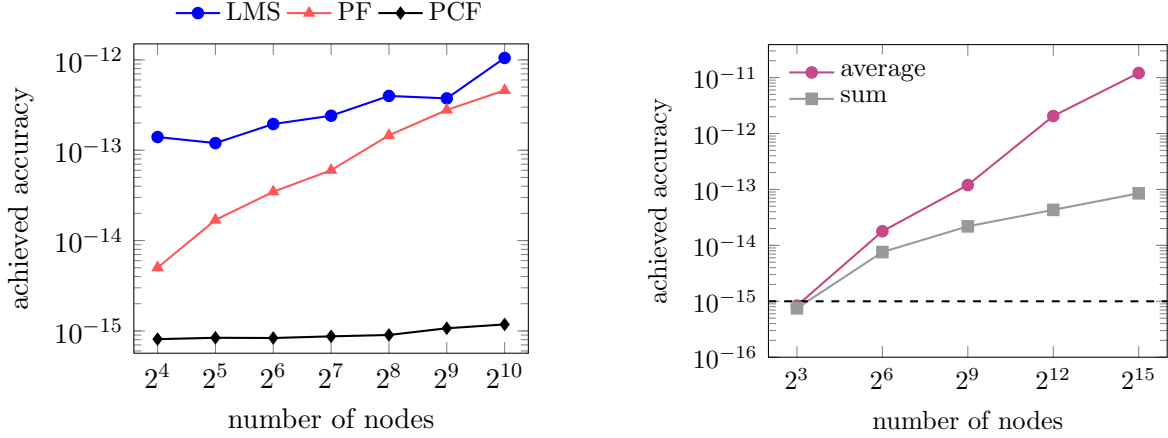


Figure 4.7.: On the left, we compare the accuracy achieved by Push-Sum (PS) and its fault-tolerant variants Push-Flow (PF) and LiMoSense (LMS). On the right, we depict the difference between average and sum computations with Push-Flow. Topology: hypercube, target accuracy $\varepsilon = 10^{-15}$.

LiMoSense cause almost identical overheads. This does not surprise, since in both cases the same “restarting” effect takes place. To exemplify this restarting behavior we also depict the result of a single run of Push-Flow where a failure handling takes place at iteration 75.

The basic problem of handling permanent failures in flow-based (history-based) algorithms is that—in contrast to the local estimates, which converge to the target aggregate—the flows (histories) converge to arbitrary values which depend on the respective communication schedules. Thus, if we set a flow (history) to zero, we have no control over the impact of this action on the local approximates of the target aggregate and we observe virtually a restart of the computation. Obviously, this behavior is not in compliance with our objective of efficient failure handling in the sense of requirement (T4) from Section 4.1.1.

4.4.2. Improving Fault Tolerance and Accuracy

[Niederbrucker, Straková, *et al.*, 2012, § 3]

In the previous section, we identified major issues in the failure handling and accuracy of link-based fault tolerance mechanisms in general and of the Push-Flow algorithm in particular. The key issue in flow-based algorithms is that the flow variables do not converge to the target aggregate but to some value which depends on the respective communication pattern. In the following, we introduce the *Push-Cancel-Flow* algorithm which operates similarly to the Push-Flow algorithm with the advantageous property that all flow variables converge to the target aggregate. Moreover, the fault tolerance properties of Push-Flow are preserved.

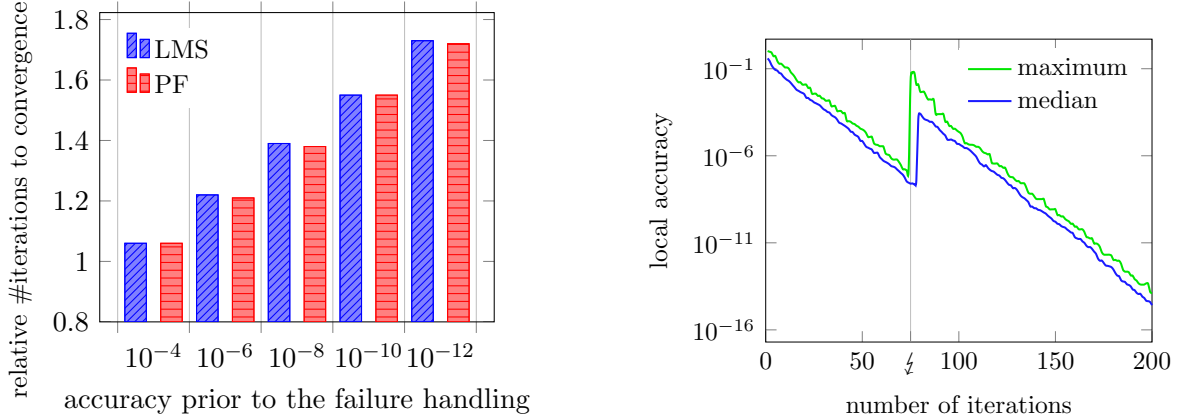


Figure 4.8.: On the left we show the relative overhead caused by handling a single permanent link failure. The recovery takes place after the local accuracy of node one reaches for the first time a threshold $\varepsilon_f = 10^{-x}$, $x \in \{4, 6, 8, 10, 12\}$. On the right, we show the impact of handling a single permanent link failure (at iteration 75) in a single experiment. Topology: hypercube with $n = 2^6$ nodes, target accuracy $\varepsilon = 10^{-13}$.

Push-Cancel-Flow (PCF)

In order to preserve the fault tolerance properties of Push-Flow we have to stick with the concept of exchanging flows, i. e., whatever data the nodes communicate has to be exchanged via flows. This restriction guarantees that the resulting algorithm has the same fault tolerance properties as the Push-Flow algorithm since its fault tolerance is based solely on utilizing the flow concept. In the following, we describe step by step how to derive Push-Cancel-Flow (Algorithm 4.5) from Push-Flow (Algorithm 4.1).

First of all, a local flow variable φ_i is introduced to store the sum over all local flows of a node i . Therefore, $\varphi_i = \sum_{j \in \mathcal{N}_i} f_{i,j}$ and the current local data is computed simply as $(s_i, w_i) + \varphi_i$. The basic idea behind the Push-Cancel-Flow algorithm is that if flow conservation holds, i. e., $f_{i,j} = -f_{j,i}$, the flows should be added to the sums φ_i , φ_j and afterwards, the flows should be set to zero. Proceeding this way, we ensure that the flow variables converge to the target aggregate because they only get updated by the current estimates of neighboring nodes (cf. Algorithm 4.1, line 10 and Algorithm 4.5, line 31) which converge to the target aggregate per definition.

To solve this problem, in Push-Cancel-Flow we use two flow variables $f_{i,j,1}$ and $f_{i,j,2}$ instead of a single flow variable $f_{i,j}$. At each point of time these two flows serve a different purpose. We speak about an *active flow* for exchanging current data, i. e., for imitating Push-Flow, and a *passive flow* where we want to achieve (local) skew-symmetry $f_{i,j,\cdot} = -f_{j,i,\cdot} = 0$. Hence, if both passive flows are zero, active and passive flows change their roles. This procedure is continuously repeated. Moreover, control variables $c_{i,j} \in \{1, 2\}$ and $r_{i,j} \in \mathbb{N}$ are introduced. The current active flow between i

Algorithm 4.5 Push-Cancel-Flow [Niederbrucker, Straková, *et al.*, 2012]

Local variables: $s_i, w_i, \varphi, f_{i,j,1}, f_{i,j,2}, c_{i,j}, r_{i,j}$
Initial values: $\varphi_i \leftarrow (0, 0), \{f_{i,j,1}, f_{i,j,2}\} \leftarrow (0, 0), \{c_{i,j}, r_{i,j}\} \leftarrow 1$

... at time t do ...

- 1: **for all** received tuples $\langle f_{j,i,1}, f_{j,i,2}, c_{j,i}, r_{j,i} \rangle$ **do**
- 2: **if** $c_{i,j} \neq c_{j,i}$ and $r_{i,j} = r_{j,i}$ **then**
- 3: $c_{i,j} \leftarrow c_{j,i}$
- 4: **end if**
- 5: **if** $c_{i,j} = c_{j,i}$ **then**
- 6: $\varphi_i \leftarrow \varphi_i - (f_{i,j,c_{i,j}} + f_{j,i,c_{i,j}})$
- 7: $f_{i,j,c_{i,j}} \leftarrow -f_{j,i,c_{i,j}}$
- 8: **if** $(f_{j,i,c_{i,j}}^c = -f_{i,j,c_{i,j}}^c$ and $r_{i,j} = r_{j,i})$ **then**
- 9: $f_{i,j,c_{i,j}}^c \leftarrow 0$
- 10: $r_{i,j} \leftarrow r_{i,j} + 1$
- 11: **else**
- 12: **if** $(f_{j,i,c_{i,j}}^c = 0$ and $r_{i,j} + 1 = r_{j,i})$ **then**
- 13: $c_{i,j} \leftarrow c_{i,j}^c$
- 14: $f_{i,j,c_{i,j}}^c \leftarrow 0$
- 15: $r_{i,j} \leftarrow r_{i,j} + 1$
- 16: **else**
- 17: **if** $r_{i,j} \leq r_{j,i}$ **then**
- 18: $\varphi_i \leftarrow \varphi_i - (f_{i,j,c_{i,j}}^c + f_{j,i,c_{i,j}}^c)$
- 19: $f_{i,j,c_{i,j}}^c \leftarrow -f_{j,i,c_{i,j}}^c$
- 20: **end if**
- 21: **end if**
- 22: **end if**
- 23: **end if**
- 24: **end for**
- 25: $k \leftarrow$ choose a neighbor uniformly at random
- 26: $f_{i,k,c_{i,k}} \leftarrow f_{i,k,c_{i,k}} + ((s_i, w_i) + \varphi_i)/2$
- 27: $\varphi_i \leftarrow \varphi_i - ((s_i, w_i) + \varphi_i)/2$
- 28: send $(f_{i,k,1}, f_{i,k,2}, c_{i,k}, r_{i,k})$ to k
- ... handling of known permanent failures ...
- 29: **if** link $(i, j) \in \mathcal{E}$ or node $j \in \mathcal{N}_i$ failed forever **then**
- 30: $f_{i,j} \leftarrow 0$
- 31: $\mathcal{N}_i \leftarrow \mathcal{N}_i \setminus \{j\}$
- 32: **end if**

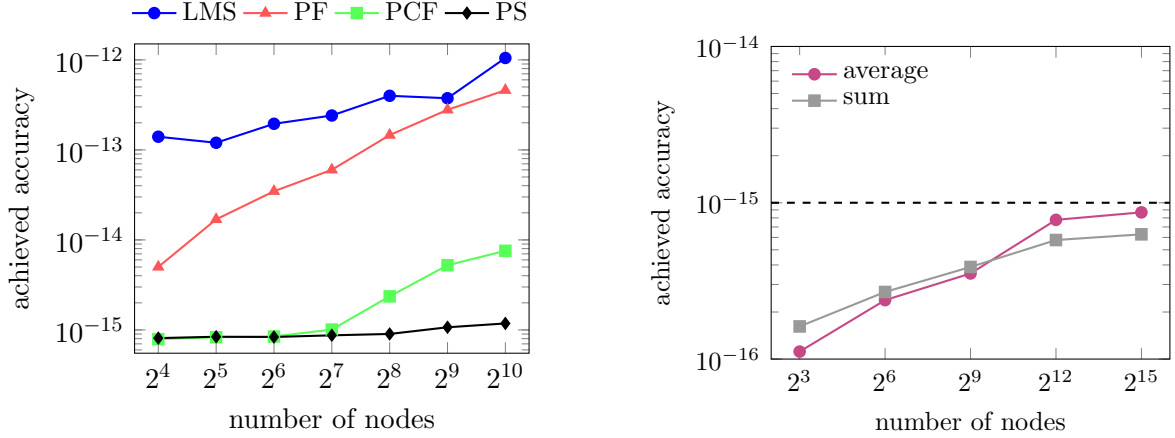


Figure 4.9.: The accuracy experiments shown in Figure 4.7 are repeated with Push-Cancel-Flow. In all cases we observe substantial improvements of the accuracy achieved.

and j is stored in $c_{i,j}$ and $r_{i,j}$ counts how often active and passive flows changed their roles. We use the notation $c_{i,j}^c$ for denoting the complementary value of $c_{i,j}$ in the sense that $c_{i,j}^c = 2$ for $c_{i,j} = 1$ and vice versa.

The additional logic for achieving the desired convergence properties of Push-Cancel-Flow affects only the handling of received messages (lines 6–29) whereas the initialization part (lines 1–5) and sending messages (lines 30–32) are completely analogous to Push-Flow. For a received message we distinguish two cases: In the first case (lines 7–9), the receiving node gets informed about a change of active and passive flow and updates its local values accordingly. In the second case (lines 10–29), the active flow is included into φ_i analogously to the Push-Flow algorithm and afterwards the passive flows are processed. In the handling of the passive flows we have to distinguish the following three cases: (i) lines 13–16: local skew-symmetry was achieved and the process of setting the passive flows to zero is started, (ii) lines 17–21: the process of setting the passive flows to zero was successful and the active/passive flows change their roles, and (iii) lines 22–25: local skew-symmetry does not hold for the passive flows and the passive flows are treated like the active flows.

Correctness and Convergence

By the fact that for the active flows nothing else than the Push-Flow algorithm is executed it is clear that the Push-Cancel-Flow algorithm and the Push-Flow algorithm are equivalent and produce (theoretically) identical results. Hence, the correctness and convergence of Push-Cancel-Flow follows immediately from the respective properties of Push-Flow. Moreover, since the aggregate data is only exchanged via flows, Push-Cancel-Flow directly inherits all fault tolerance properties from Push-Flow.

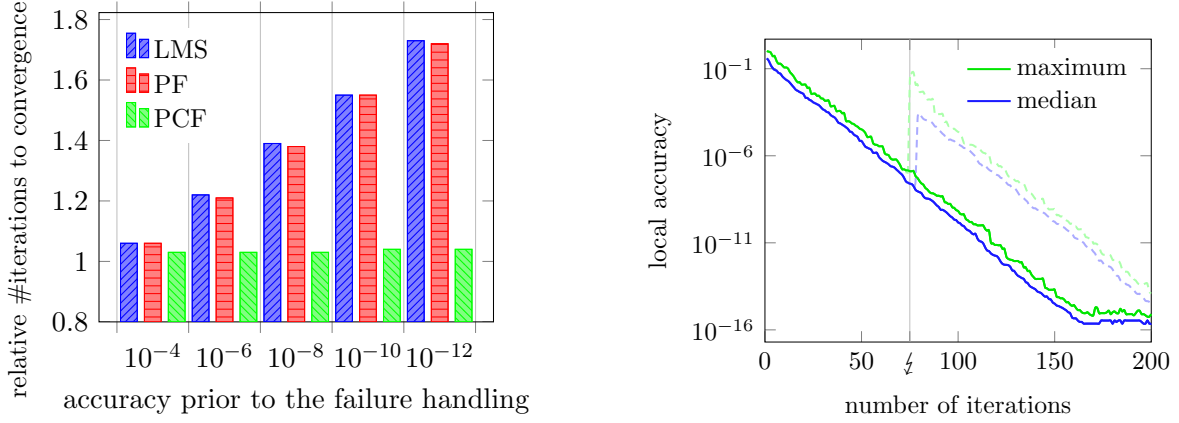


Figure 4.10.: The fault tolerance experiments shown in Figure 4.8 are repeated with Push-Cancel-Flow. On the right, from the point on where the results differ, i. e., the iteration where the failure handling takes place, the results for Push-Flow (depicted in Figure 4.8, right) are indicated in light colors.

Summarizing, the Push-Cancel-Flow algorithm is equivalent to the Push-Flow algorithm and provides the additional property that all flow variables converge to the target aggregate which yields a better numerical behavior and a more efficient handling of permanent failures. This is achieved by overlapping the actual computation (active flows) with incorporating conserved flows in a locally stored overall sum of flows (passive flows).

Experimental Results

To illustrate the superior fault tolerance and accuracy properties of the Push-Cancel-Flow algorithm over the Push-Flow algorithm (and LiMoSense) we repeat the experiments carried out before in Section 4.4.1.

First, in Figure 4.9 we study how the techniques introduced in Push-Cancel-Flow influence the resulting accuracy. We see that the error increase with growing network size is much slower than for Push-Flow and LiMoSense (cf. Figure 4.7). The second problem which we discussed is the inefficient failure handling of the Push-Flow algorithm, which leads to major fall-backs in the convergence. In Figure 4.10 the superior fault tolerance properties of Push-Cancel-Flow are depicted by a direct comparison to the results achieved by Push-Flow and LiMoSense. For the individual experiments in Figures 4.8 and 4.10 (right) we used the same initial random seed, i. e., the simulated random communication schedules are the same. Hence, since Push-Flow and Push-Cancel-Flow behave identically for the same communication schedules and initial data (if no failures occur), we see no difference between the two algorithms until the first failure occurs. After the failure handling, we see that Push-Cancel-Flow tolerates the failure without any fall-back in the convergence in strong contrast to Push-Flow.

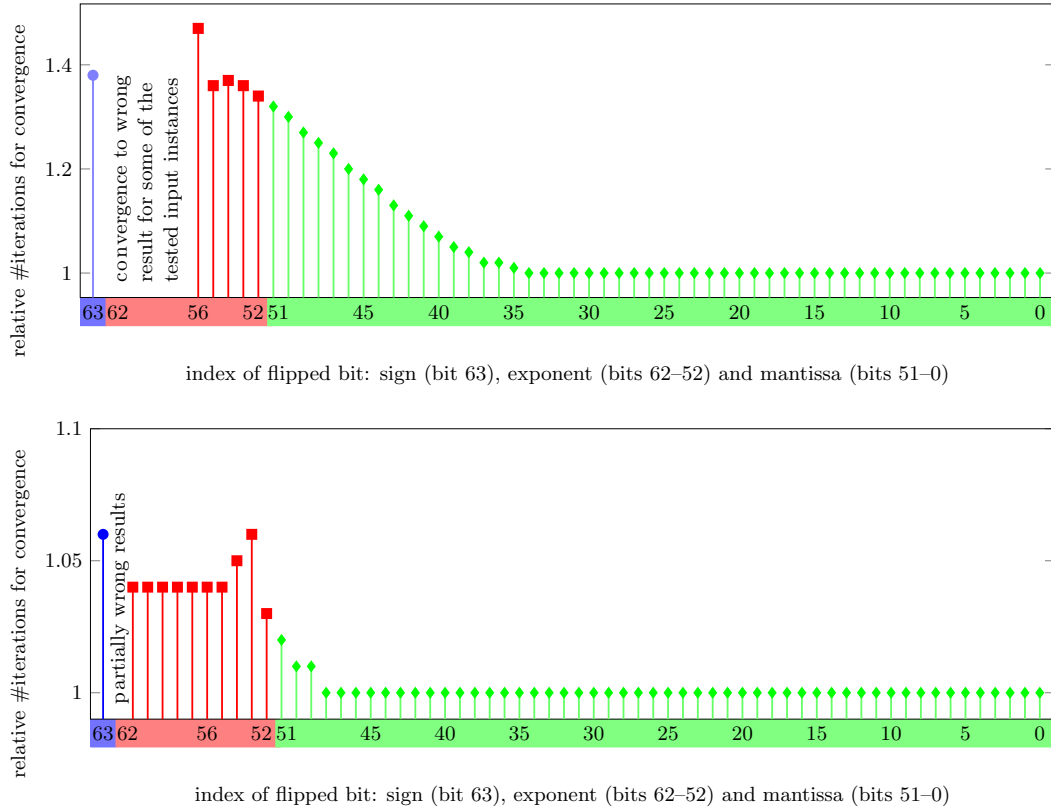


Figure 4.11.: A single bit b gets flipped in the flow variable of a fixed node i after its local error was $\leq 10^{-8}$ (top) and $\leq 10^{-4}$ (bottom) for the first time. Topology: hypercube $n = 2^8$ nodes, target accuracy $\varepsilon = 10^{-13}$.

4.4.3. An Excursion to Silent Data Corruption

[Niederbrucker and Gansterer, 2013, § 4.3]

The principle of flow-based aggregation algorithms (in contrast to history-based approaches) allows also to recover from silently occurring data corruptions as long as the original input data is not affected. That means, flow-based approaches can in principle tolerate bit flips in the flow variables $f_{i,j}$ but not in the initial data $(s(0)_i, w(0)_i)$. In Figure 4.11 we see for the case of Push-Flow how the position of a single injected bit flip as well as the point of time when the failure was injected influences the convergence speed. Concretely, we observe that many bit flips can be handled without any overhead whereas certain bit flips lead to high overheads and even convergence problems⁶. The positions where no overhead occurs can be explained by the fact that in the Push-Flow algorithm (without failures) the maximal local approximation error

⁶Whenever the target accuracy $\varepsilon = 10^{-13}$ was not achieved in one or more runs, we leave out the respective data point in Figure 4.11. The reason for these problems is that mass conservation is (substantially) hurt due to cancellation effects.

$\|x(t) - \alpha \mathbf{1}\|_\infty$ is monotonously decreasing, i. e., we have at each point of time a certain range of values (cf. [Section 3.3.2](#)). Consequently, whenever a bit flip does not affect this range, it is naturally tolerated. Clearly, the later a bit flip happens, the smaller the maximal local approximation error is and thus, the more bit flips potentially lead to an overhead (cf. [Figure 4.11](#)). Moreover, this argumentation also shows that not the amount of bit flips but the precise location and time of occurrence determines the resulting overhead.

To understand better why certain bit flips even preclude the computation of correct results we briefly recapitulate how real numbers are represented as (double precision) floating point numbers. A real number x in double precision is usually represented as

$$x = (-1)^s \cdot (1 + m \cdot 2^{-52}) \cdot 2^{e-1023}$$

with the *sign* $s \in \{0, 1\}$, the *mantissa* $0 \leq m < 2^{52}$ and the *exponent* $0 < e < 2^{11} - 1$. Obviously, flips in the most significant bits of the exponent can have tremendous impact, especially if the exponent is around the bias value 1023.

In [Figure 4.11](#) we observe for the earlier occurring bit flip only problems with the flip of the most significant exponent bit, whereas the later occurring bit flip leads to problems with several of the most significant bits. This observation can be explained by the nature of flow-based algorithms where we know that the flows gently develop (e. g., grow) over time to reach a stable equilibrium. In the case of an early bit flip, we observe in our experiment always an exponent smaller than the bias value of 1023 (due to the choice of the initial data). Therefore, only a change of the most significant bit leads to a (in absolute values) huge change of the respective floating-point variable. On the other hand, for the case of the later bit flip, we observe that the exponent sometimes exceeds the bias and thus also flipping the next few bits has a big impact in certain test cases.

Summarizing, we see a very diverse behavior in the handling of bit flips (depending on several parameters) which ranges from no overhead to a convergence to wrong results. Hence, a fully reliable protection against arbitrary bit flips can not be provided by flow-based methods. On the other hand, it is interesting to see how and up to which extent such an easy mechanism at the algorithmic level can cope with the problem of silent data corruption due to bit flips.

Lessons Learned: Provable Robustness on Detours

In this chapter we introduced and analyzed fault tolerance mechanisms for push-based distributed aggregation algorithms (see [Definition 4.1](#)). The analysis (correctness) of this kind of algorithms relies on the so-called *mass conservation* property

$$\sum_{i \in \mathcal{V}} (s(t)_i, w(t)_i) = \sum_{i \in \mathcal{V}} (s(0)_i, w(0)_i).$$

We considered a range of important failures (cf. [Table 4.1](#)) which hurt mass conservation and therefore, they lead to incorrect results. In particular, if mass conservation is once hurt in a push-based algorithm, it can not be reestablished, because the nodes do not have mechanisms for undoing a change of mass. In general, the difficulty in preserving mass conservation over time originates from the fact that it is a global constraint, whereas (in a distributed context) only local mechanisms can be used.

Our approach for coping with this problem is inspired by the flow concept from graph theory (cf. [Section 4.2.1](#)). We proposed that every node i keeps for each of its neighbors $j \in \mathcal{N}_i$ a flow variable $f(t)_{i,j}$ which describes the overall (i. e., until time t) balance of data communicated between the nodes i and j . We showed that based on a local data representation of

$$(s(t)_i, w(t)_i) = (s(0)_i, w(0)_i) + \sum_{j \in \mathcal{N}_i} f_{i,j}(t),$$

it suffices for the nodes to communicate their local flow variables $f(t)_{i,j}$. When receiving data, it is crucial to ensure (local) skew-symmetry by setting $f(t)_{j,i} = -f(t)_{i,j}$, which yields the desired mass conservation. Hence, it is (indirectly) also responsible for the ability to recover from failures. However, in this way mass conservation is (in general) not strictly preserved globally, because it depends on all local recoveries which (in general) do not take place at the same point of time. Therefore, it is a priori by no means clear that this technique succeeds at providing fault tolerance at the algorithmic level. In [Section 4.2.2](#) we proved the correctness of flow-based algorithms in the presence of failures by introducing so-called recovery graphs which we used to represent the execution of flow-based algorithms in the presence of failures in a (theoretical) failure-free environment. Analogously, we applied this analytical approach to also analyze history-based algorithms, which have not been fully analyzed before. Moreover, we also derived (the first known) bounds for the overhead silent message loss causes when using either of the aforementioned fault tolerance techniques. Similarly to our fault tolerance results, in [Section 4.3](#) we also analyzed an exemplary model of asynchronous communication by means of an appropriately extended graph structure.

Last but not least, our empirical investigations in [Section 4.4](#) revealed several practical subtleties which are common to link-based fault tolerance mechanisms. In particular, we experienced severe inaccuracies and large overheads in the handling of permanent failures. For flow-based algorithms, we showed how to (partially) overcome these issues by introducing a refined variant of the flow concept.

5 | while (not ε -accurate) do ...

When to stop a computation (locally)?

So far our results concerned the convergence of distributed aggregation algorithms (Chapter 3) as well as the development of fault tolerance mechanisms at the algorithmic level (Chapter 4). Throughout the previous chapters we left open how to terminate (in practice) the algorithms we investigated. For the sake of concreteness, recall Algorithm 2.2 which we introduced as a prototypical skeleton of the kind of distributed aggregation algorithms we elaborate on in this thesis.

```
1:  $a_i \leftarrow$  initial contribution from node  $i$ 
2: while not  $\varepsilon$ -accurate do
3:   pick a node  $k \in \mathcal{N}_i$  and send  $a_i$  to node  $k$ 
4:   wait for a message  $a_j$  from a node  $j \in \mathcal{N}_i$ 
5:    $a_i \leftarrow$  combine  $a_i, a_j$  according to the target aggregate function
6: end while
```

Even though local convergence criteria for distributed algorithms of an iterative nature are obviously necessary and crucial for real-world applications, the need for such criteria is—to the best of our knowledge—almost entirely neglected in the existing literature. According to our definition of the problem (cf. Section 5.1), the only method for local convergence detection we are aware of is due to Kempe and McSherry [2004]. However, their approach is not in compliance with our assumptions on distributed environments, because among others it itself involves particular aggregations where the problem of convergence detection is left open (again). Moreover, existing work concerning the convergence detection of asynchronous parallel iterative algorithms (see, e.g., Bertsekas and Tsitsiklis [1989]; Savari and Bertsekas [1996]; Bahi *et al.* [2005]; Charr *et al.* [2010]) does not apply in our context.

Implicitly formulated convergence criteria are by no means unique to the analysis of distributed (aggregation) algorithms and frequently appear—most commonly for the sake of convenience—in algorithm descriptions in numerical analysis. However, in case of (parallel) numerical algorithms, often explicit and easily computable convergence

criteria can be straightforwardly found. For instance, the evaluation of an appropriate residual leads in many cases to a meaningful convergence criterion (see [Section 6.2.3](#) for an example).

In this chapter, we address the problem of deciding locally at a node whether its local estimate is ε -accurate (see [Definition 2.1](#)). Needless to say, the effort for estimating the currently achieved accuracy should not dominate the overall complexity. This means that we are restricted to methods with a constant time (per iteration) and space complexity, due to the simplicity of the distributed aggregation algorithms we consider. In a truly distributed environment, the only information which a node can exploit is its local data. In particular, we assume that the nodes do not have any global information, e. g., about the (entire) topology or the initial data. Hence, by definition it appears impossible to provide generic methods which provably provide always the right decision. Consequently, we aim at developing well-performing heuristics based on a solid theoretical background (see [Section 5.3](#)). Moreover, we complement our theoretical results by a comprehensive set of experiments which illustrate the performance of the methods we propose (see [Section 5.4](#)). Our considerations build upon our generic convergence results (cf. [Section 3.4](#)), which provide essential a priori information on the convergence behavior of the distributed algorithms we address.

5.1. Objectives and Prerequisites

As we have seen in the previous chapters, the distributed aggregation algorithms we consider in this thesis produce locally at each node a sequence of estimates of the target aggregate α . For the sake of a concise representation we restrict our attention in this chapter to a fixed node without referring explicitly to it. This means, instead of a sequence of local estimates $(x(0)_i, x(1)_i, \dots)$ for an arbitrary but fixed node i , throughout this chapter we simply consider a sequence of local estimates $(x_m) = (x_0, x_1, \dots)$, which converges to the target aggregate α . Moreover, we assume without loss of generality that $x_i \neq x_{i+1}$ for all i , i. e., we ignore iterations where a node does not change its local estimate. Our objective is to provide an estimate $\tilde{\varepsilon}$ for the accuracy of x_i , i. e., $\varepsilon = \|x_i - \alpha\|/\|\alpha\|$, solely based on the respective local knowledge $\{x_0, x_1, \dots, x_i\}$. Recall that we can only effort constant additional time and space. Therefore, we can never have the entire set $\{x_0, x_1, \dots, x_i\}$ at hand, but only a fixed number of (its most recent) elements. From a practical point of view, the computed estimate $\tilde{\varepsilon}$ has to be judged with respect to two criteria (cf. [Section 5.4](#)): *correctness* ($\tilde{\varepsilon} \geq \varepsilon$) and *quality* ($\tilde{\varepsilon} \approx \varepsilon$). Note that the correctness requirement implies that we do not want to terminate too early (without having reached the target accuracy), whereas the quality requirement implies that we want to keep the number of additional (“unnecessary”) iterations as low as possible. Moreover, in an application context we have to expect not only scalar aggregations, but most commonly vector aggregations¹.

¹We did not explicitly investigate vector aggregations so far, because in the context of the previous chapters a vector aggregation is simply a simultaneous execution of (many) scalar aggregations.

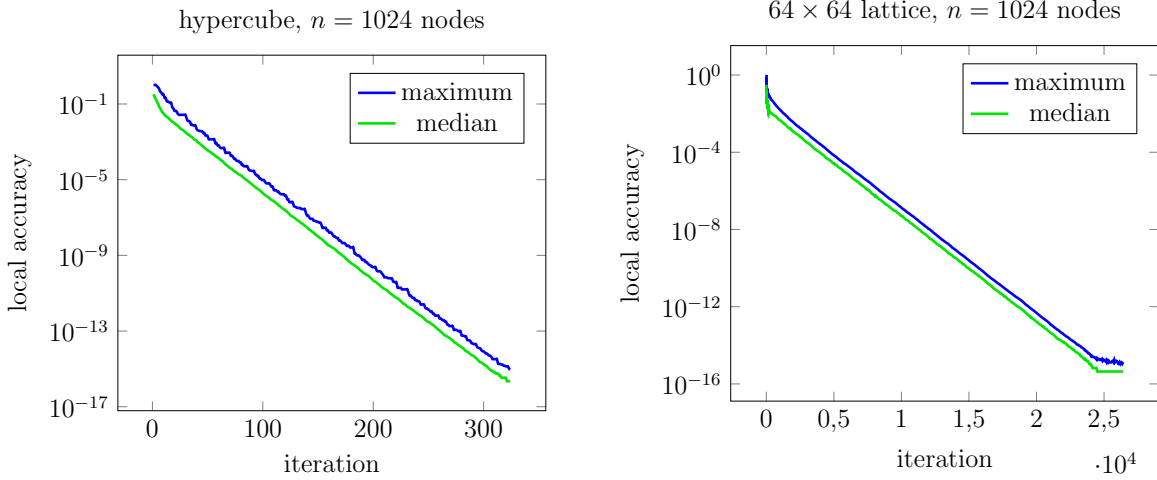


Figure 5.1.: We depict the maximum and the median over all local approximation errors for a single run of Push-Sum on a hypercube (left) and a square lattice (right), both with $n = 1024$ nodes.

Therefore, we assume that (x_m) is a sequence of k -dimensional vectors, i. e., $x_i \in \mathbb{R}^k$ for all $i \in \mathbb{N}$. Regarding the convergence of the sequence (x_m) , we know from our convergence results (cf. [Section 3.4](#)) that there exist constants $c \in (0, \infty)$ and $r \in (0, 1)$ such that

$$\|x_i - \alpha\| \leq cr^i. \quad (5.1)$$

See [Figure 5.1](#) for exemplary illustrations of the convergence behavior of the Push-Sum algorithm (cf. [Algorithm 3.2](#)). Beyond this upper bound on the convergence speed, our convergence results do not provide further insights which could be exploited locally. Note that the bound (5.1) does not only hold in the failure-free case, because we proved the convergence of our fault tolerant algorithms by the convergence results for the failure-free case (cf. [Section 4.2](#)).

In our formal treatment, we do not refer to a specific norm in (5.1). However, we only consider norms which fulfill for vectors x, y (besides the usual axioms) that $x \leq y$ (elementwise) implies $\|x\| \leq \|y\|$. This holds, for instance, for the Euclidean norm $\|\cdot\|_2$ and the uniform norm $\|\cdot\|_\infty$. Hence, for the sake of simplicity, we can (in theoretical results) always assume $c = 1$, since $\|\cdot\|_c = \|\cdot\|/c$ is for any norm $\|\cdot\|$ a norm as well. Nevertheless, we have to (approximately) determine this constant within our methods. The upcoming [Section 5.2](#) contains an elaboration on this subject.

Since we know that the convergence rate of the algorithms we target is bounded by r , it is important to understand how to approximate this quantity by local information only. As a first step, we show in the following lemma how to bound r from below.

5. while (not ε -accurate) do ...

Lemma 5.1. *Given a sequence (x_m) of local estimates, then for all indices $i < j$ it holds for the convergence rate r that*

$$r \geq \left(\frac{\|x_i - x_j\|}{2} \right)^{\frac{1}{i}}$$

Proof. By the triangle inequality we obtain

$$\begin{aligned} \|x_i - x_j\| &= \|(x_i - \alpha) - (x_j - \alpha)\| \\ &\leq \|x_i - \alpha\| + \|x_j - \alpha\| \\ &\leq r^i + r^j = r^i(1 + r^{j-i}) \\ &\leq 2r^i. \end{aligned}$$

Moreover, by taking logarithms we get

$$\begin{aligned} \log \left(\frac{\|x_i - x_j\|}{2} \right) &\leq i \log(r) \\ \frac{1}{i} \log \left(\frac{\|x_i - x_j\|}{2} \right) &\leq \log(r) \\ \left(\frac{\|x_i - x_j\|}{2} \right)^{\frac{1}{i}} &\leq r. \end{aligned}$$

□

[Lemma 5.1](#) shows that we easily get a lower bound of r by comparing the results of subsequent iterations, i.e., by measuring $\|x_i - x_{i+1}\|$. However, so far we have no information about the quality of this bound. As the following lemma shows, the influence of an observed distance $\|x_i - x_j\|$ in the lower bound for r we obtained in [Lemma 5.1](#) decreases fast with an increasing number of iterations.

Lemma 5.2. *In the setting of [Lemma 5.1](#), let $\|x_i - x_j\| = \delta 2r^i$ with $\delta \in (0, 1)$. Then, the convergence rate r is given by*

$$r = \left(\frac{\|x_i - x_j\|}{2} \right)^{\frac{1}{i}} \delta^{\frac{1}{i}}.$$

Proof. Use the proof of [Lemma 5.1](#) with $\|x_i - x_j\| = \delta 2r^i$ instead of $\|x_i - x_j\| \leq 2r^i$. □

From [Lemma 5.2](#) we can see the regulating effect of an increased number of iterations, because even if the (observed) quantity $\|x_i - x_j\|$ is far away from the theoretically assumed maximal value, we get a bound of high quality due to $\lim_{i \rightarrow \infty} \delta^{1/i} = 1$ for $0 < \delta < 1$.

5.2. Cooperative Parameter Estimation

As a prerequisite, we consider the problem of estimating the parameters c and r in the bound $\varepsilon = \|x_i - \alpha\| \leq cr^i$ (cf. (5.1)). As always in this chapter, we only permit a constant (additional) effort per processing a local estimate x_i . Besides local computations, we can also exploit that the nodes run a distributed aggregation. In particular, we can append small amounts of data to the messages used within the aggregation without increasing the overall complexity significantly. More specifically, we consider appending data to be appropriate as long as the appended data does not depend on the dimension of the data to be aggregated.

5.2.1. Estimation of c

Obviously, the constant c is fully determined by the initial data, since $\|x_0 - \alpha\| \leq c$. In case of a scalar aggregation, a bound for the constant c can be determined straightforwardly by computing the minimum $x_{\min}$ and maximum $x_{\max}$ over the initial data of all nodes. As mentioned before, these discrete aggregations can be done simultaneously to the actual continuous aggregation by appending data to the messages. It is easy to see that discrete aggregations converge much faster than continuous aggregations, whenever the same communication schedule (and topology) is used. Therefore, the nodes will always obtain the correct value for $x_{\min}, x_{\max}$ in reasonable amount of time. However, a node never knows whether the obtained values represent the global extrema. Having the extremal values $x_{\min}, x_{\max}$ at hand, it is clear that $c \leq x_{\max} - x_{\min}$. In case of vector aggregations, the nodes can first find the extremal values within their initial vector (locally) and subsequently proceed analogously to the case of scalar aggregations. Depending on the intended norm to be used, it can be required to account for the dimension of the aggregation in the local estimation of c . For instance, if $x_i \in \mathbb{R}^k$, it holds that $c \leq x_{\max} - x_{\min}$ with respect to $\|\cdot\|_{\infty}$, whereas $c \leq \sqrt{k}(x_{\max} - x_{\min})$ with respect to the Euclidean norm $\|\cdot\|_2$.

5.2.2. Estimation of r

In principle, we can estimate the convergence rate r in the same spirit as the constant c . By Lemma 5.1 we can compute for each newly computed local estimate x_{i+1} an upper bound for r by measuring $\|x_{i+1} - x_i\|$. Since we obtain in each step an upper bound for r , it makes sense to keep the maximum over all these bounds as the approximation to be used. This involves two steps: First, whenever a node computes a new bound $\tilde{r}$ for r , it keeps the maximum of $\tilde{r}$ and the (best) bound it obtained so far. Second, the nodes append their (currently) best bound of r to the exchanged messages in order to provide all nodes with the best known bound for r . Since r is a scalar, this additional computation comes in any case for minor additional costs. Due to the fact that the bounds we can compute for r by Lemma 5.1 tend to improve over time (cf. Lemma 5.2),

5. while (not ε -accurate) do ...

this aggregation is unlikely to converge in a strict sense, but nevertheless it provides the nodes with better and better bounds over time.

5.2.3. A Minimalist Approach for Accuracy Estimation

Basically, the ability of being able to estimate the parameters c and r already yields a simplistic way of convergence detection, since one can easily deliver $\tilde{c}\tilde{r}^i$ as an estimate for accuracy at iteration i , whenever $\tilde{c}, \tilde{r}$ are (reasonably) good estimates of c, r . Of course, this approach considers inherently the worst case, because it bounds the entire range of potential local estimates. Therefore, the computed bounds can be seen as global rather than local bounds (even though they are computed locally). This means that this simplistic approach emphasizes correctness ($\varepsilon \leq \tilde{\varepsilon}$) rather than quality ($\varepsilon \approx \tilde{\varepsilon}$). On the other hand, since receiving once a “bad” value can substantially worsen a local estimate which was “good” before, such a worst case strategy is by no means inappropriate. The only drawback of the approach is that it heavily relies on the quality of the estimation of r which improves with increasing number of iterations (cf. [Lemma 5.2](#)). Therefore, this approach works the better the slower the aggregation converges (i.e., the larger r is). In [Section 5.4](#) we illustrate this intuition by experiments on various network topologies (which lead to different convergence rates).

5.3. Keeping Means: Bound and Float

On top of the parameter estimation from the previous section, in this section we propose a refined method for accuracy estimation which exploits several properties of means. Given a sequence of local estimates (x_m) with limit α , the *sample mean* for its initial subsequence $\{x_1, \dots, x_m\}$ is defined as $\bar{x}_m = \sum_{i=1}^m x_i / m$. In the context of this chapter we have never the entire sequence (x_i) at hand. Therefore, it is important to see that the sample mean can be computed “online” via the identity

$$\bar{x}_{m+1} = \bar{x}_m + (x_{m+1} - \bar{x}_m) / (m + 1). \quad (5.2)$$

As one can see from [\(5.2\)](#), the online computation of a sample mean involves only a cheap update operation. Therefore, we do not have to keep (a range of) previous estimates when we consider means.

5.3.1. Overview

With our assumptions on the convergence of sequences of local estimates (x_i) it is easy to see that also $\lim_{m \rightarrow \infty} \bar{x}_m = \alpha$. The apparent drawback of means in our setting is, that the sequence $(\bar{x}_i)$ converges slowly compared to the sequence (x_i) ($\mathcal{O}(1/m)$ vs. $\mathcal{O}(r^m)$). On the other hand, this slow convergence leads (from some iteration on) to a monotone convergence (in $\|\cdot\|$). Among others, the utilization of this monotonicity

plays an important role in our considerations below. In order to bypass the inherent problem of slow convergence of means, we consider the following kind of sequences of means.

Definition 5.1. Given a sequence of local estimates (x_i) , we call a sequence $(\bar{x}_i)$ a *k-corresponding sequence of means* if for all $i \geq k$ the elements $\bar{x}_i$ are given by (5.2). //

Analogously to the sample mean, one can easily show that every *k*-corresponding sequence of means converges to the target aggregate α . In a nutshell, the method we introduce in the following sections aims at exploiting the beneficial properties of means and at overcoming their weaknesses by keeping four means. Two for getting accuracy bounds (*bounding means*) and two for improving the convergence speed (*floating means*). The two bounding means can be imagined as monotonically converging “upper” and “lower” bounds for the limit α . In contrast to that, the floating means are intended to be close to the most recently seen local estimates x_i . Hence, they are (first) neither expected to be bounds for α nor to converge monotonically. The acceleration property of the floating means comes from the effect that the floating means eventually also turn into bounds (due to the slow convergence of the mean). Thereafter, they can serve as improved bounding means and new floating means have to be appropriately chosen for a further improvement of the bounding means.

5.3.2. Auxiliary Results

In the following, we study which conditions are necessary to guarantee monotonic convergence of a *k*-corresponding sequence of means to the limit α . Afterwards we show how to derive estimates for α with corresponding accuracy bounds using such monotonically converging means. First, consider the following elementary assertion regarding the convergence behavior of means.

Lemma 5.3. *Let (x_m) be a sequence of local estimates and $(\bar{x}_m)$ a *k*-corresponding sequence of means. Then, for all $i \geq k$, $\|\bar{x}_i - \alpha\| \geq r^{(i+1)}$ implies $\|\bar{x}_{i+1} - \alpha\| \leq \|\bar{x}_i - \alpha\|$.*

Proof. By the definition of $\bar{x}_i$ we see that

$$\begin{aligned} \|\bar{x}_{i+1} - \alpha\| &= \left\| \frac{i\bar{x}_i + x_{i+1}}{i+1} - \alpha \right\| \\ &= \frac{1}{i+1} \|i(\bar{x}_i - \alpha) + (x_{i+1} - \alpha)\| \\ &\leq \frac{1}{i+1} (i\|\bar{x}_i - \alpha\| + \|x_{i+1} - \alpha\|) \\ &\leq \frac{1}{i+1} (i\|\bar{x}_i - \alpha\| + r^{(i+1)}) \\ &\leq \|\bar{x}_i - \alpha\|. \end{aligned}$$

□

5. while (not ε -accurate) do ...

Lemma 5.3 gives a sufficient condition that $\bar{x}_{i+1}$ cannot be worse than $\bar{x}_i$ (as estimate for α). As a logical next step we investigate which (additional) assumptions guarantee that a k -corresponding sequence of means converges monotonously to the target aggregate α .

Lemma 5.4. *Let (x_m) be a sequence of local estimates and $(\bar{x}_m)$ a k -corresponding sequence of means. If $i \geq k$, $i \geq -\log(r)^{-1}$ and*

$$\|\bar{x}_i - g\| \geq r^{(i+1)} \left(1 + \frac{1}{i(1-r)}\right),$$

then $\|\bar{x}_j - \alpha\| \geq \|\bar{x}_{j+1} - \alpha\|$ holds for all $j \geq i$.

Proof. According to **Lemma 5.3** we just have to show that $\|\bar{x}_j - \alpha\| \geq r^{(j+1)}$ holds for all $j \geq i \geq -\log(r)^{-1}$. First observe that by the reverse triangle inequality we get

$$\begin{aligned} \|\bar{x}_j - \alpha\| &= \left\| \frac{1}{j} \left(i\bar{x}_i + \sum_{k=i+1}^j x_k \right) - \alpha \right\| = \frac{1}{j} \left\| i(\bar{x}_i - \alpha) + \left(\left(\sum_{k=i+1}^j x_k \right) - (j-i)\alpha \right) \right\| \\ &= \frac{1}{j} \left\| i(\bar{x}_i - \alpha) - \sum_{k=i+1}^j (\alpha - x_k) \right\| \\ &\geq \frac{1}{j} \left| \left\| i(\bar{x}_i - \alpha) \right\| - \left\| \sum_{k=i+1}^j (\alpha - x_k) \right\| \right|. \end{aligned}$$

Again by the triangle inequality we see that

$$\left\| \sum_{k=i+1}^j (\alpha - x_k) \right\| \leq \sum_{k=i+1}^j \|(\alpha - x_k)\| \leq \sum_{k=i+1}^j r^k = \sum_{k=i+1}^j r^k$$

By applying the formulas for the (finite) geometric series (recall that $r < 1$) we get

$$\begin{aligned} \sum_{k=i+1}^j r^k &= \left(\sum_{k=0}^j r^k - \sum_{k=0}^i r^k \right) = \frac{(1 - r^{(j+1)}) - (1 - (r^{(i+1)}))}{1 - r} \\ &= \frac{r^{(i+1)} - r^{(j+1)}}{1 - r} \leq \frac{r^{(i+1)}}{1 - r}. \end{aligned}$$

By our assumptions on $\|\bar{x}_i - \alpha\|$ we get that

$$\begin{aligned} \frac{1}{j} \left| \left\| i\bar{x}_i - \alpha \right\| - \left\| \sum_{k=i+1}^j (\alpha - x_k) \right\| \right| &\geq \frac{1}{j} \left(i r^{(i+1)} \left(1 + \frac{1}{i(1-r)} \right) - \frac{r^{(i+1)}}{1-r} \right) \\ &= \frac{1}{j} (i r^{(i+1)}) \geq r^{(j+1)}. \end{aligned}$$

To show in the last step that $ir^{(i+1)} \geq jr^{(j+1)}$, we used that the function $\varphi(x) := xr^{(x+1)}$ is monotonously decreasing for $x \geq -\log(r)^{-1}$ since $x = -\log(r)^{-1}$ is the single extremal value of φ . $\square$

Obviously, it holds that $\lim_{i \rightarrow \infty} (1 + (i(1-r))^{-1}) = 1$. Therefore, the simple requirement in [Lemma 5.3](#) asymptotically leads to monotonic convergence. This highlights how slow and regular k -corresponding sequences of means converge. In the following, we additionally derive a similarly shaped requirement on the monotonic convergence which is already fulfilled after a finite number of iterations.

Corollary 5.5. *Let (x_m) be a sequence of local estimates and $(\bar{x}_m)$ a k -corresponding sequence of means. Then there exists an $i \geq k$ such that for all $j \geq i$, $\|\bar{x}_j - \alpha\| \geq r^j$ implies the monotonic convergence of (x_m) .*

Proof. According to [Lemma 5.4](#) we have to find an i such that $r(1 + (i(1-r))^{-1}) \leq 1$. For $r \in (0, 1)$ we get

$$\begin{aligned} ri(1-r) + r &\leq i(1-r) \\ i - ir + 1 &\leq ir^{-1} - i \\ i^{-1} &\leq r^{-1} - 2 + r. \end{aligned}$$

Since $r^{-1} - 2 + r = r^{-1}(1-r)^2 > 0$, we can always choose $i = \lceil (r^{-1} + r - 2)^{-1} \rceil$. $\square$

While monotonic convergence is the property which we exploit with the bounding means (valid bounds only get better over time), we also need criteria for deciding whether the floating means already turned into bounds or not. Therefore, in the following lemma we derive conditions which are sufficient to gain bounds for the limit α in terms of two means.

Lemma 5.6. *Let (x_m) be a sequence of local estimates and $(\bar{v}_m), (\bar{w}_m)$ k -corresponding sequences of means with $\bar{v}_i \leq \bar{w}_i$. If $\bar{v}_i \leq x_i \leq \bar{w}_i$ and $\|x_i - \bar{v}_i\|, \|\bar{w}_i - x_i\| > r^i$, then $\bar{v}_i \leq \alpha \leq \bar{w}_i$ holds.*

Proof. We know that $\|x_i - \alpha\| \leq r^i$. Assume that $\bar{v}_i \leq \alpha \leq \bar{w}_i$ does not hold, then either $\alpha < \bar{v}_i \leq x_i$ or $x_i \leq \bar{w}_i < \alpha$ holds. For $\alpha < \bar{v}_i \leq x_i$ we get a contradiction, since

$$r^i < \|x_i - \bar{v}_i\| \leq \|x_i - \alpha\| \leq r^i.$$

Analogously, we get for $x_i \leq \bar{w}_i < \alpha$ the contradiction

$$r^i < \|\bar{w}_i - x_i\| \leq \|\alpha - x_i\| \leq r^i.$$

$\square$

Note that [Corollary 5.5](#) implies for the bounds deduced by [Lemma 5.6](#) that (for a sufficiently late iteration i) at least one of the means converges monotonously to α . By the ability to bound α in terms of two means $\bar{x}_i, \bar{y}_i$, we can also provide estimates for the limit α , e. g., $(\bar{x}_i + \bar{y}_i)/2$ with corresponding accuracy bound $\|(\bar{x}_i + \bar{y}_i)/2 - \alpha\| \leq \|(\bar{y}_i - \bar{x}_i)/2\|$.

5. *while* (not ε -accurate) *do* ...

5.3.3. Bound-and-Float Algorithm

Algorithm 5.1 Bound-and-Float algorithm for a k -dimensional aggregation

Local variables: $\bar{x}, \bar{y}, \bar{v}, \bar{w} \in \mathbb{R}^k, \tilde{r}, \tilde{c} \in \mathbb{R}$

```

1:  $\bar{x}, \bar{y}, \bar{v}, \bar{w} \leftarrow$  pick initial means (e.g.,  $\bar{x} \leftarrow \frac{1}{2}x_0, \bar{y} \leftarrow \frac{7}{8}x_0, \bar{v} \leftarrow \frac{9}{8}x_0, \bar{w} \leftarrow \frac{3}{2}x_0$ )
2: for each local estimate  $x_i$  do
3:   update estimates  $\tilde{r}, \tilde{c}$  of  $r$  and  $c$  (cf. Section 5.2)
4:   update  $\bar{x}, \bar{y}, \bar{v}, \bar{w}$  according to (5.2) by  $x_i$ 
5:   for  $j \leftarrow 1$  to  $k$  do
6:     if  $[x_i]_j < \bar{v}_j$  then
7:        $\bar{u}_j \leftarrow 2[x_i]_j - \bar{v}_j, \bar{x}_j \leftarrow \min(\bar{x}_j, \bar{v}_j)$ 
8:     end if
9:     if  $[x_i]_j > \bar{w}_j$  then
10:       $\bar{v}_j = 2[x_i]_j - \bar{w}_j, \bar{y}_j \leftarrow \max(\bar{w}_j, \bar{y}_j)$ 
11:    end if
12:  end for
13:  if  $\|\bar{u} - x_i\| > \tilde{c}\tilde{r}^i$  and  $\|\bar{v} - x_i\| > \tilde{c}\tilde{r}^i$  then
14:     $\bar{x} \leftarrow \bar{u}, \bar{y} \leftarrow \bar{v}$ 
15:     $\bar{u} \leftarrow (\bar{x} + \bar{y})/2 - (\bar{y} - \bar{x})/100$ 
16:     $\bar{v} \leftarrow (\bar{x} + \bar{y})/2 + (\bar{y} - \bar{x})/100$ 
17:  end if
18:  deliver  $\tilde{\alpha} = (\bar{x} + \bar{y})/2$  as estimate for  $\alpha$ 
19:  deliver  $\tilde{\varepsilon} = \|\bar{x} - \bar{y}\|/\|\bar{x} + \bar{y}\|$  as estimate for  $\|\alpha - \tilde{\alpha}\|/\|\alpha\|$ 
20: end for

```

In this section, we put together the insights which we previously obtained in order to derive a method for the local accuracy estimation. More concretely, [Algorithm 5.1](#) is an implementation of the idea of bounding and floating means (cf. [Section 5.3.1](#)). As mentioned earlier, we keep four means, the bounding means $(\bar{x}, \bar{y})$ and the floating means $(\bar{v}, \bar{w})$. Moreover, we employ our parameter estimation techniques from [Section 5.2](#) to obtain estimates for the constants r and c .

Initially, all four means have to be determined using only the initial local estimate x_0 . We propose to chose the four means symmetrically around x_0 (line 1). Subsequently, it is necessary to algorithmically ensure $\bar{x} \leq \bar{v} \leq \bar{w} \leq \bar{y}$ according to our objective and the interpretation of the different means. This can be easily done for each new local estimate x_i (lines 4–9). The decision whether the floating means turn into bounds (line 11) is based on [Lemma 5.6](#). Therefore, this decision relies on a good estimates of r and c (cf. [Section 5.2](#)). Due to the inaccuracies in the estimation of r and c , this decision can be (slightly) wrong for some iterations. However, the slow convergence of means leads over time to monotonic convergence and consequently to valid bounds. Therefore, it can happen that wrong conclusions are drawn temporarily, but these

mistakes get corrected quickly by the slow convergence of the (bounding) means. In case the floating means turn into bounding means, new floating means have to be determined. Analogously to our initial choices for the means, we propose to choose $\bar{u}, \bar{v}$ symmetrically around the arithmetic mean $(\bar{x} + \bar{y})/2$ of the bounding means $\bar{x}, \bar{y}$.

In general, both the initial choice of the means and the update of the floating means can be done in many ways, which certainly impacts the performance of the predictions. Our parameter choices in [Algorithm 5.1](#) are made such that we obtain good results on various topologies rather than excellent results on a particular topology (cf. [Section 5.4](#)). Finally, note that [Algorithm 5.1](#) indeed introduces only a constant additional time and space complexity (per iteration) as we required previously.

5.4. Experimental Results

In this section, we evaluate empirically how well the methods which we proposed in the previous sections work in practice. In particular, we investigate the influence of the topology as well as the difference between scalar and vector aggregation on the quality of the accuracy estimation. As extreme cases we include complete graphs and (square) lattices, since these topologies lead to a very fast and very slow convergence, respectively. Following a description of the methodology behind our experiments ([Section 5.4.1](#)), we depict results concerning both the simple parameter estimation approach ([Section 5.2.3](#)) and the bound-and-float algorithm ([Section 5.3.3](#)). Furthermore, in [Section 5.4.4](#) we give an interpretation of our empirical observations.

5.4.1. Methodology

Our objective was to provide methods which deliver (locally at each node) an estimate $\tilde{\varepsilon}$ of the current local accuracy ε . As mentioned earlier in [Section 5.1](#), we judge our methods for accuracy estimation (convergence detection) with respect to two criteria: correctness ($\tilde{\varepsilon} \geq \varepsilon$) and quality ($\tilde{\varepsilon} \approx \varepsilon$). In order to evaluate these criteria we focus in our experiments on the ratio $\chi = \varepsilon/\tilde{\varepsilon}$. This means that the requirements of correctness and quality translate to $\chi \leq 1$ and $\chi \approx 1$, respectively. We consider the following experimental setup:

Algorithm: As a concrete aggregation algorithm satisfying the model assumption regarding the convergence $\|x_i - \alpha\| \leq cr^i$ (cf. [\(5.1\)](#)), we employ in all our results the Push-Sum algorithm (see [Algorithm 3.2](#)). We expect that the utilization of this particular algorithm does not bias the results (qualitatively), since the employed algorithm only influences the convergence rate r , which can be equally well varied by considering different topologies.

Topology: We consider a selection of topologies which substantially differ in their nature. In [Table 5.1](#) we summarize the most important parameters of the topologies

5. *while (not ε -accurate) do ...*

we employ in our experiments. In particular, one can observe that there are huge differences in the achievable convergence speed (rate r).

	nodes	degree	diameter	iterations to reach $\varepsilon = 10^{-14}$
$G(64, 0.4)^1$	64	20.46 ± 1.82	4	299.25 ± 81.94
$G(256, 0.3)$	256	54.98 ± 2.46	5	484.24 ± 65.64
$G(1024, 0.2)$	1024	106.44 ± 0.66	8	1093.64 ± 61.50
Hypercube	64	6	6	152.22 ± 12.24
	256	8	8	178.40 ± 9.20
	1024	10	10	185.25 ± 8.45
Complete graph	64	63	1	92.17 ± 2.54
	256	255	1	97.14 ± 1.61
	1024	1023	1	99.31 ± 1.20
Square lattice	64	3.50 ± 0.62	14	1303.60 ± 27.64
	256	3.75 ± 0.47	30	5589.20 ± 87.74
	1024	3.88 ± 0.34	62	22880.20 ± 494.34

Table 5.1.: Graphs (and their properties) we use in our experiments. The rightmost column depicts mean and standard deviation over 100 independent runs with target accuracy $\varepsilon = 10^{-14}$.

Initial data: We consider scalar, as well as vector aggregations. In all cases the input data is drawn from the uniform distribution over $[0, 1]^k$. Again, we expect that this particular choice of the initial data does not bias the results significantly.

Results: For each experiment, we fix a topology, the dimension k of the aggregation, as well as the method for accuracy estimation. Based on this setup, we run a (round-based) implementation of the Push-Sum algorithm until all nodes reach a target accuracy of $\varepsilon = 10^{-14}$ and store the ratio $\chi(i)_j = \varepsilon(i)_j / \tilde{\varepsilon}(i)_j$ for all nodes j and all iterations i .

In order to illustrate up to which extent we can meet our correctness and quality requirements, we depict for each iteration i two quantities: the maximum (correctness) and the median (quality) of all local estimation ratios $\chi(i)_j$. Moreover, we do not present results from a single experiment but instead repeat all experiments 100 times³, in order to account for the randomness in the experiments. As an overall result, the illustrated data points in our plots are computed as statistics over all 100 experiments. In particular, we show the maximum of all maxima and the median of all medians.

² $G(n, d)$ denotes a random geometric graph with n nodes and radius d (cf. [Section A.4](#)).

³In case of random geometric graphs, we consider ten independently generated graphs and repeat each experiment ten times, in order to account for the randomness in the construction of random geometric graphs.

Note that in our experiments we do not tune parameters with respect to specific topologies. This means that we run our accuracy estimation methods in all our experiments with the standard parameters. Consequently, by well-chosen parameters one can further improve the presented results. Up to which extent the search for such parameters is feasible in practice depends on the respective application.

5. while (not ε -accurate) do ...

5.4.2. Experiments: Simple Parameter Estimation

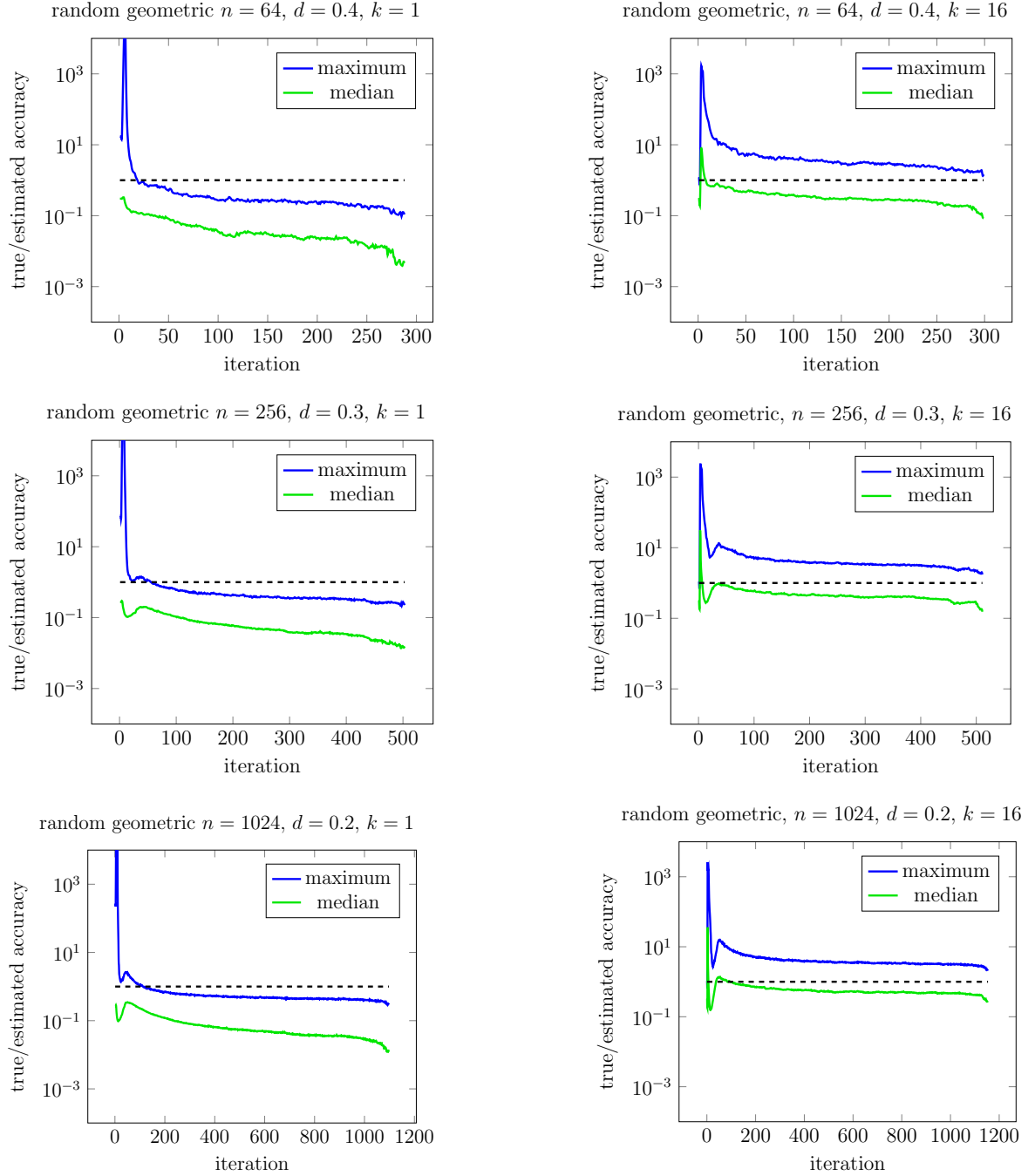


Figure 5.2.: Convergence detection based on simple parameter estimation, network topology: random geometric graph $G(n, d)$, $n \in \{64, 256, 1024\}$ nodes, communication range $d \in \{0.4, 0.3, 0.2\}$, and aggregate dimension $k \in \{1, 16\}$.

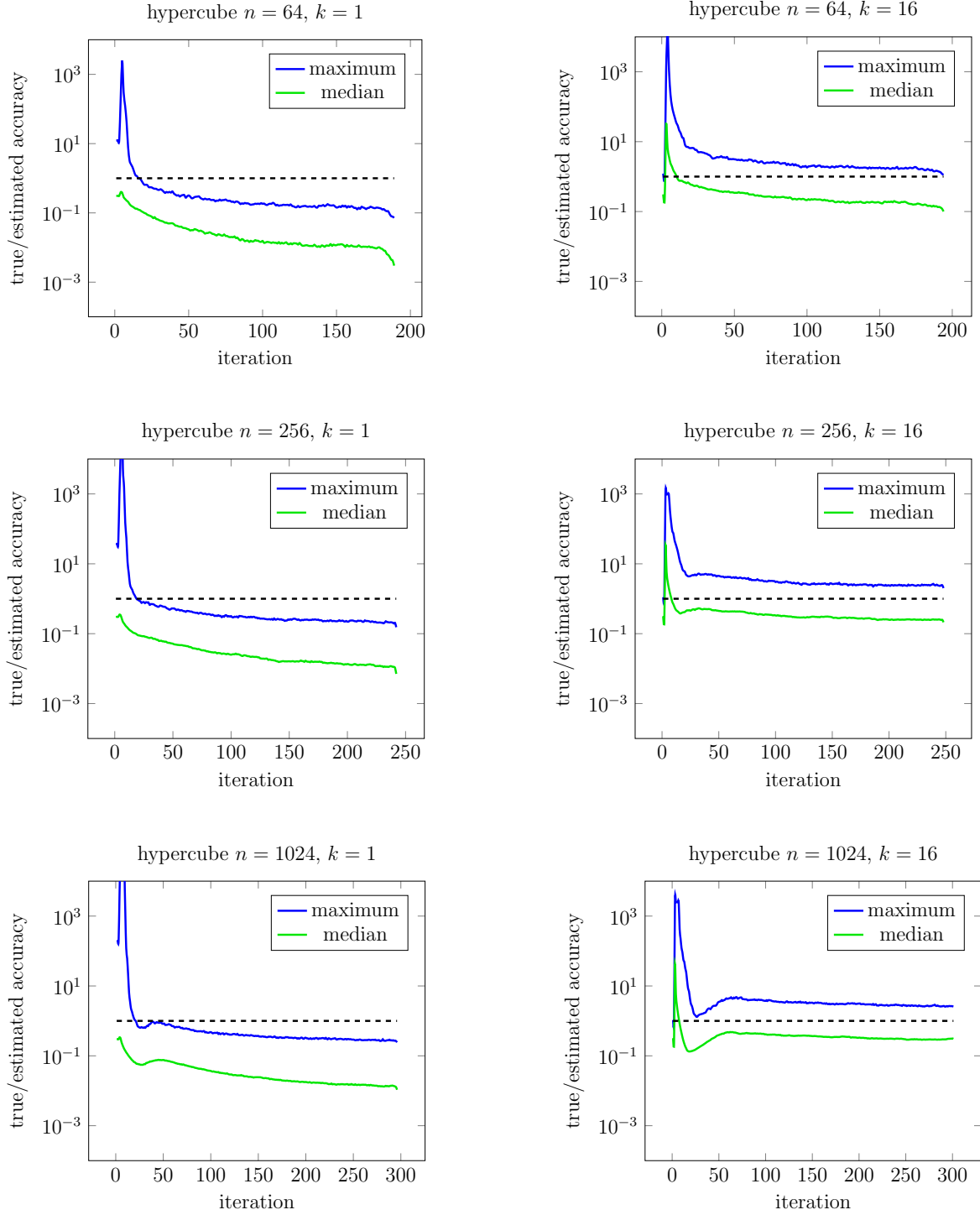


Figure 5.3.: Convergence detection based on simple parameter estimation, network topology: hypercube, $n \in \{64, 256, 1024\}$ nodes, and aggregate dimension $k \in \{1, 16\}$.

5. *while (not ε -accurate) do ...*

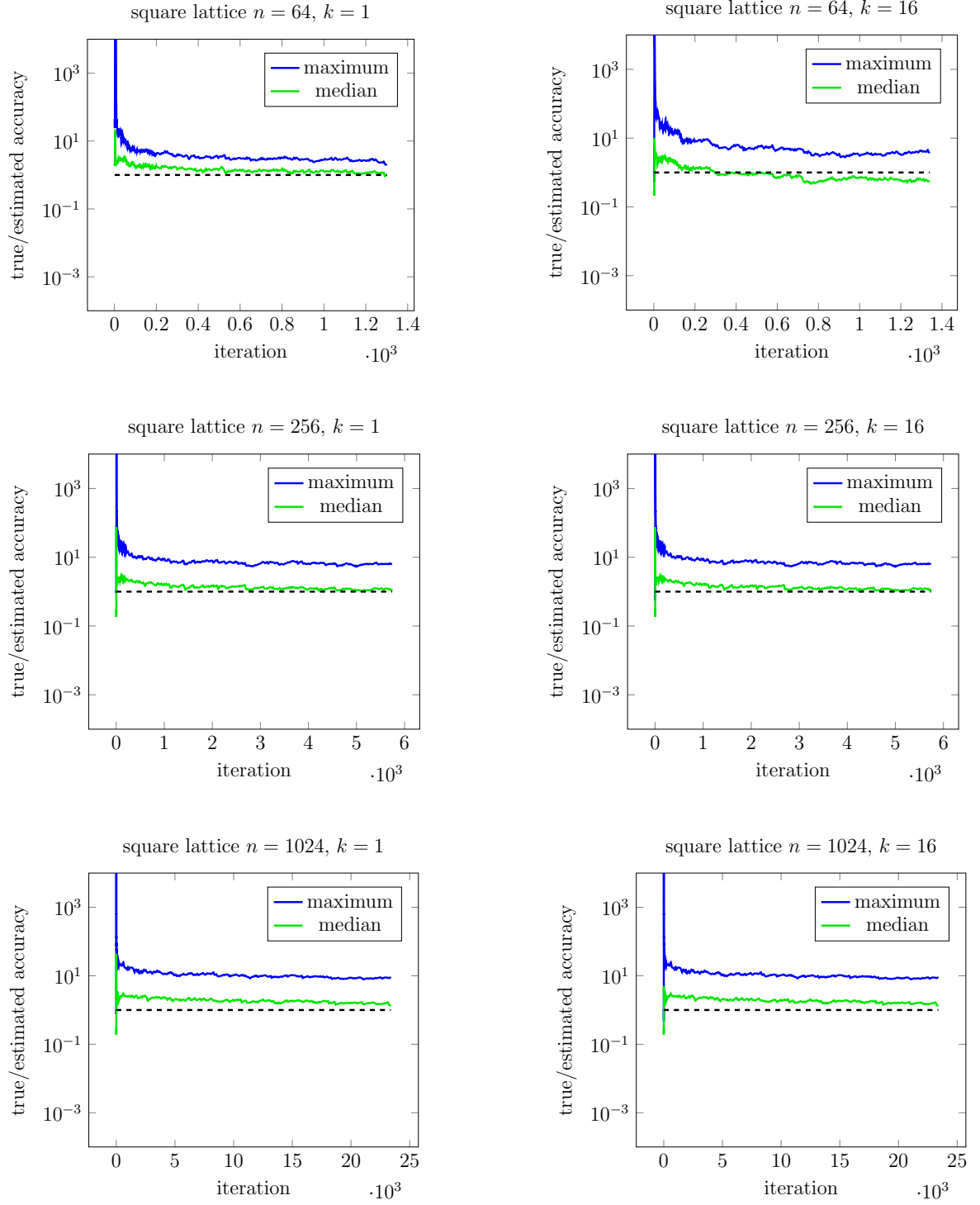


Figure 5.4.: Convergence detection based on simple parameter estimation, network topology: square lattice, $n \in \{64, 256, 1024\}$ nodes, and aggregate dimension $k \in \{1, 16\}$.

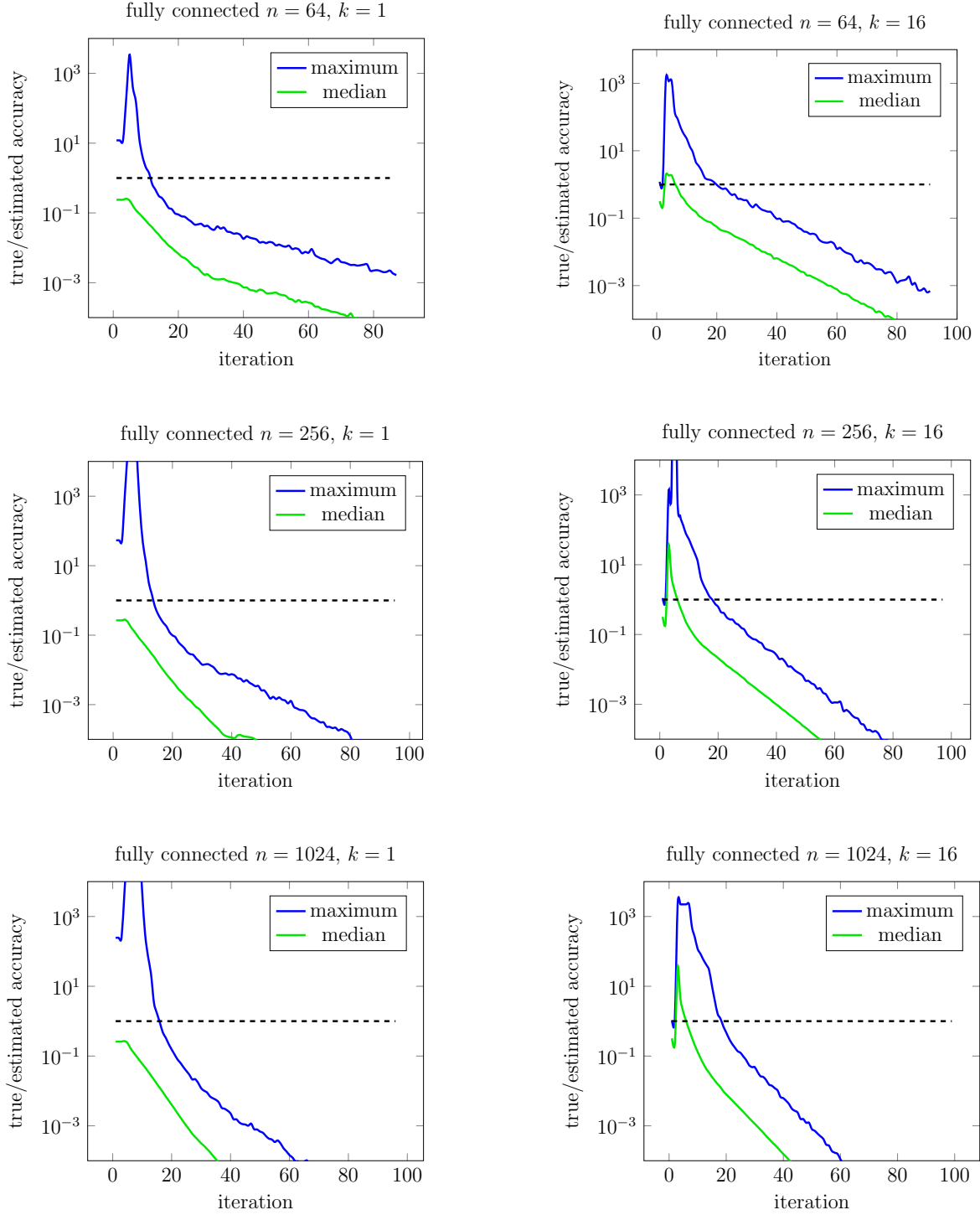


Figure 5.5.: Convergence detection based on simple parameter estimation, network topology: fully connected, $n \in \{64, 256, 1024\}$ nodes, and aggregate dimension $k \in \{1, 16\}$.

5. while (not ε -accurate) do ...

5.4.3. Experiments: Bound-and-Float Algorithm

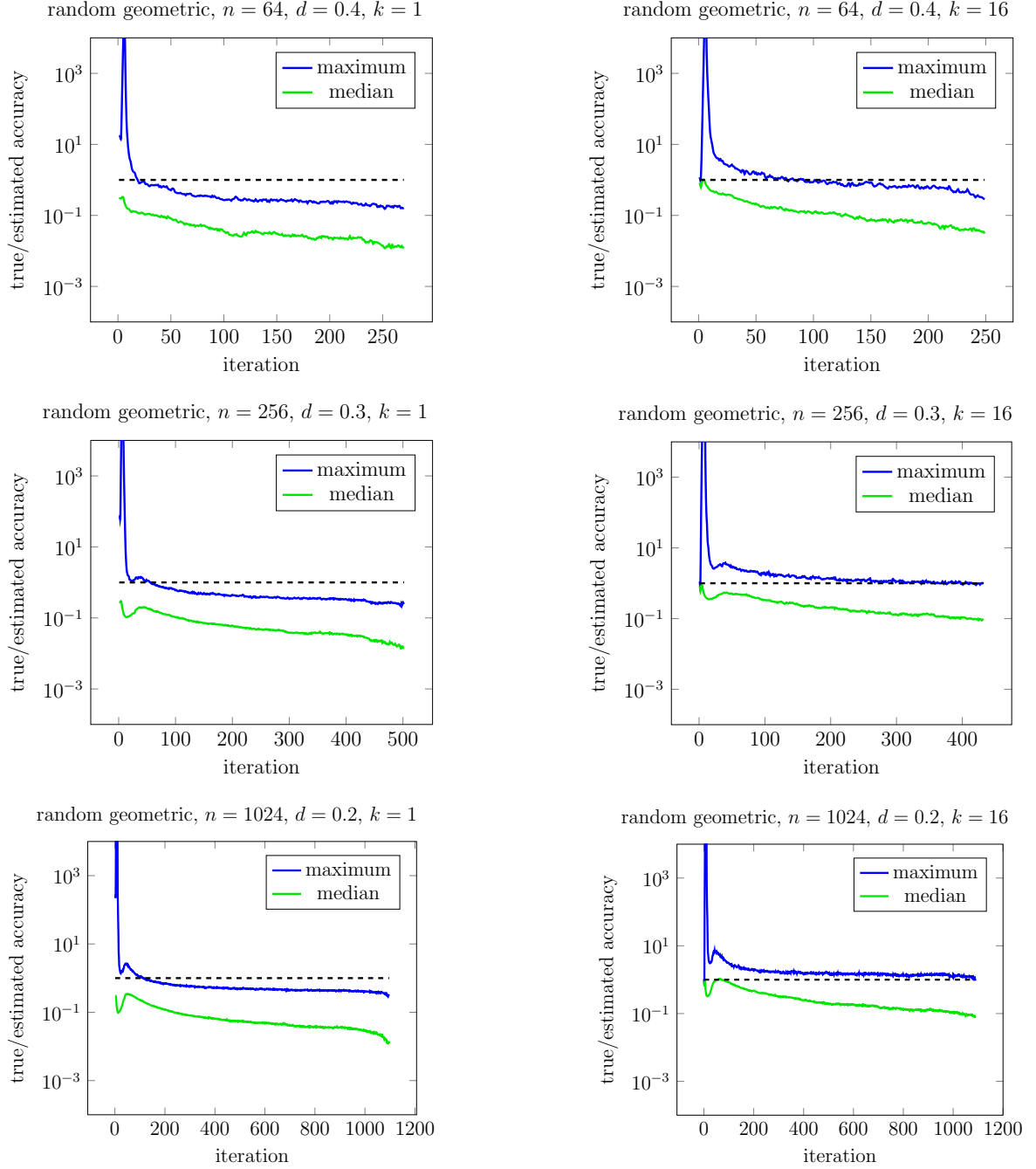


Figure 5.6.: Convergence detection based on bound-and-float algorithm, network topology: random geometric graph $G(n, d)$, $n \in \{64, 256, 1024\}$ nodes, communication range $d \in \{0.4, 0.3, 0.2\}$, and aggregate dimension $k \in \{1, 16\}$.

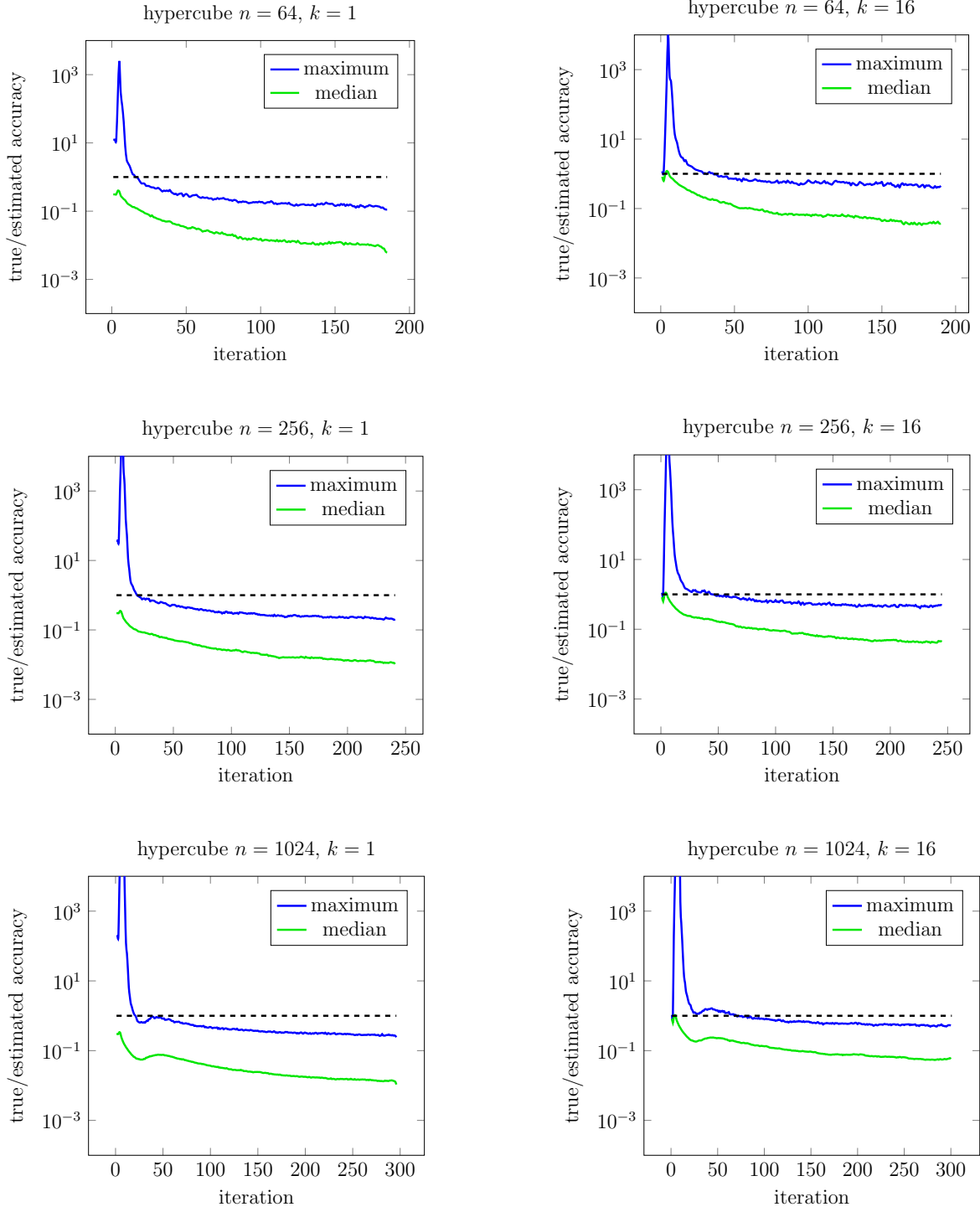


Figure 5.7.: Convergence detection based on bound-and-float algorithm, network topology: hypercube, $n \in \{64, 256, 1024\}$ nodes, and aggregate dimension $k \in \{1, 16\}$.

5. *while (not ε -accurate) do ...*

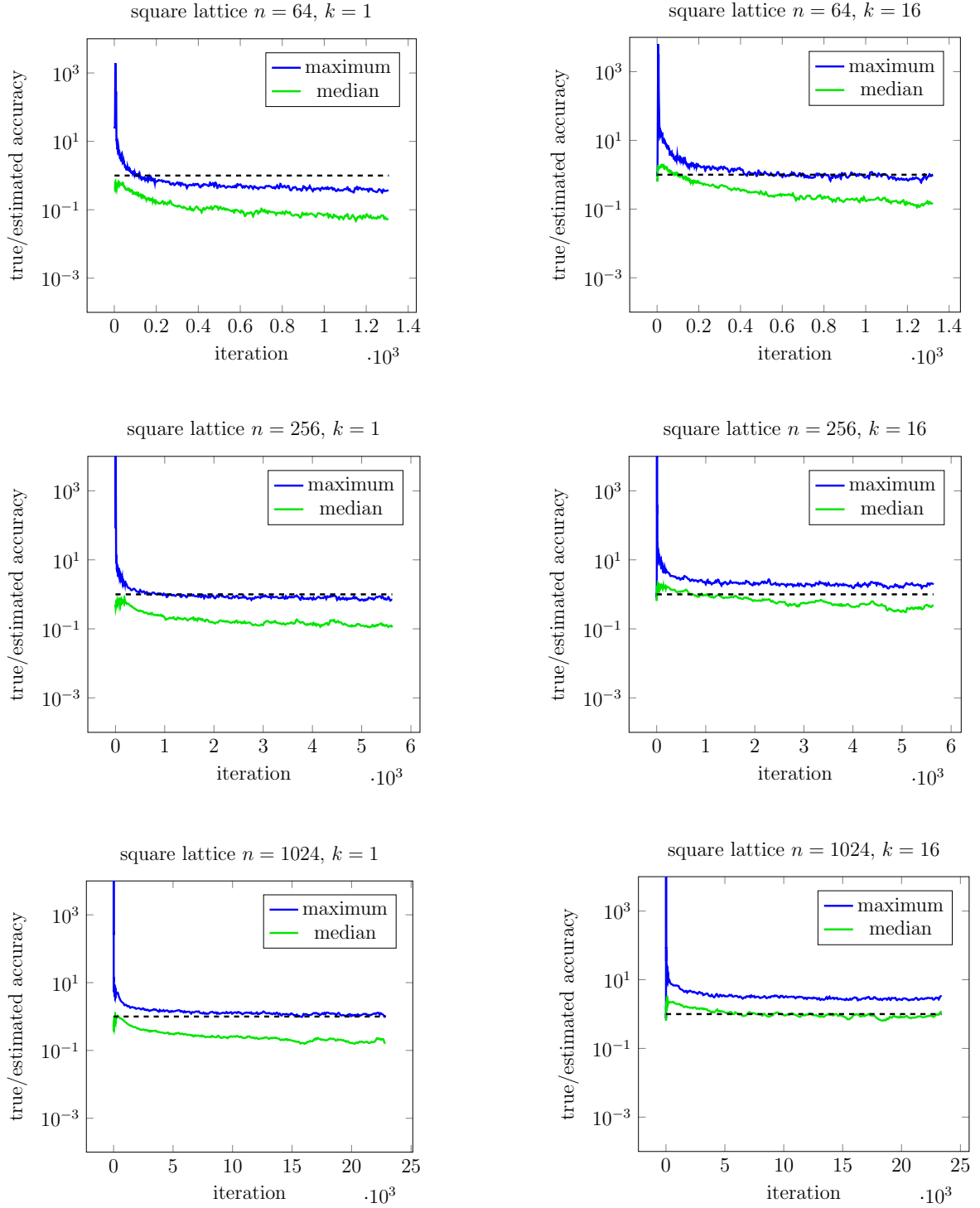


Figure 5.8.: Convergence detection based on bound-and-float algorithm, network topology: square lattice, $n \in \{64, 256, 1024\}$ nodes, and aggregate dimension $k \in \{1, 16\}$.

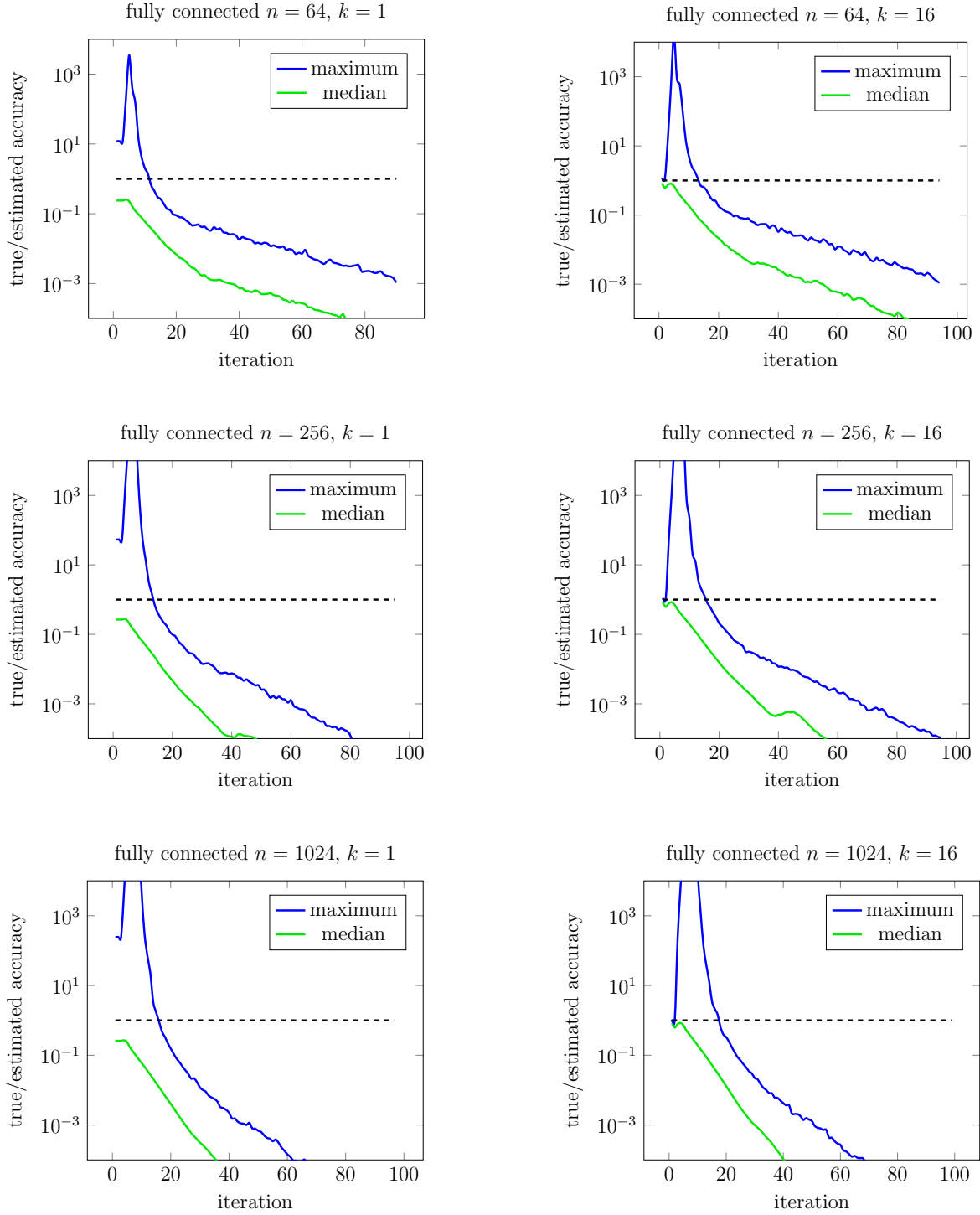


Figure 5.9.: Convergence detection based on bound-and-float algorithm, network topology: fully connected, $n \in \{64, 256, 1024\}$ nodes, and aggregate dimension $k \in \{1, 16\}$.

5.4.4. Interpretation of Results

In all of the experiments we show in Figures 5.2–5.9 it is apparent that there exists an initial period where the maximal ratios $\chi(i)_{j_{\max}}$ can be very big, i.e., the worst accuracy estimation is very incorrect. This behavior has to be expected, because it takes a while until all nodes obtain good local estimates for the (global) constants c and r in the bound (5.1), which are crucial for deriving good estimates for the local accuracy.

Both of the methods we developed in this chapter for accuracy estimation can be seen as methods for estimating the worst local accuracy (in the entire system) locally at each node. Therefore, there is a gap between the maximal ratios $\chi(i)_{j_{\max}}$ and the median ratios $\chi(i)_{j_{\text{med}}}$ is inevitable. Exactly as one would expect with respect to Figure 5.1, we see that this gap is (nearly) constant in all our experiments.

Regarding the four different topologies we utilized in our experiments, we observe a tremendous difference between the results obtained for complete graphs (Figures 5.5 and 5.9) and all other topologies. In particular, we see that the local accuracy estimates are in principle correct, whereas their quality decreases rapidly with an increasing number of iterations. On complete graphs we face (with respect to convergence detection) two problematic aspects. On the one hand, the convergence is very fast and on the other hand, all nodes can communicate with each other. This implies that there is no “locality” and hence, the local estimates for the target aggregate α can change quickly. Therefore, our accuracy estimation methods cannot cope with the high speed of convergence we observe in complete graphs.

In general, we can see that the bound-and-float algorithm delivers (slightly) better results than the simple parameter estimation. Our results reveal two different kinds of weaknesses within both of these methods. First, in certain situations the ratios $\chi(i)_j$ show a wrong trend (see, e.g., Figures 5.2 and 5.6). This wrong trend is due to the error-prone approximation of the convergence rate r and can be (partially) corrected by an appropriately parametrized estimation of r , i.e., either by employing Lemma 5.2 or by scaling the obtained values for r by a suitable factor. In particular, such a parametrized estimation of r is important for the case of complete graphs. In case the ratios $\chi(i)_j$ show the right trend (i.e., they are constant) one can still face the problem of a permanently biased estimation of the local accuracy (see, e.g., Figures 5.4 and 5.8). However, one can easily cope with this issue by scaling the accuracy estimations by a (topology-dependent) constant factor.

Altogether, we can conclude that the presented methods can estimate the local accuracy achieved so far reasonably well. This holds especially for topologies which lead to a reasonably slow convergence. Moreover, if additional topology-dependent parameters are available one can even expect (substantially) better results. Needless to say, the availability of such parameters depends on the particular application context. In particular in the context of distributed computing it is highly questionable whether appropriate parameters can be found. Hence, we refrained from giving detailed results for parametrized cases.

Lessons Learned: (In)Sufficiency of Local Methods

All distributed aggregation algorithms we investigate in this thesis are of an iterative nature (cf. [Algorithm 2.2](#)). More concretely, each node computes locally a sequence (x_m) which converges to the target aggregate α . In the existing literature, the problem of convergence detection (for the algorithms we consider) is almost entirely neglected, i. e., it is common practice to assume that a computation aborts as soon as the local error $\varepsilon = \|x_i - \alpha\|/\|\alpha\|$ reaches a predefined threshold. In this chapter, we developed methods for providing an estimate $\tilde{\varepsilon}$ of ε , which can be used to detect convergence in practice. With the application of convergence detection in mind, we judge estimates $\tilde{\varepsilon}$ of ε according to *correctness* ($\tilde{\varepsilon} \geq \varepsilon$) and *quality* ($\tilde{\varepsilon} \approx \varepsilon$). Due to the simplicity of the aggregation algorithms we target, our restriction to methods with constant space and time (per processing a local estimate x_i) complexity is self-evident. More complex methods for convergence detection would dominate the complexity of the overall aggregation, which is certainly unwanted from a practical point of view. Since we assume a distributed environment where the nodes are limited to their local knowledge, it appears impossible to provide generic methods which provably provide always correct estimates (of high quality). However, we have seen that theoretically well-founded heuristics can achieve good results in practice.

The starting point for our investigations was the a priori bound of $\|x_i - \alpha\| \leq cr^i$ on the convergence of the respective sequence of local estimates (for the target aggregate α), where $c \in (0, \infty)$ and $r \in (0, 1)$. Consequently, in [Section 5.2](#) we first investigated up to which extent we can find (good) estimates $\tilde{c}$, $\tilde{r}$ for the constants c and r , locally at each node. Having such estimates at hand, a straightforward accuracy estimate of $\tilde{\varepsilon} = \tilde{c}\tilde{r}^i$ (at iteration i) can be made. We have seen theoretically as well as empirically that this estimation gets more accurate with an increasing number of iterations. Hence, the slower the respective algorithm converges (i. e., the larger r is), the better becomes the estimation with respect to a fixed target accuracy. In order to provide better estimates we introduced the bound-and-float algorithm (cf. [Section 5.3](#)), which also employs the aforementioned parameter estimation. In the bound-and-float algorithm we utilize (sample) means in order to provide good accuracy estimates $\tilde{\varepsilon}$. In particular, we keep four means: two (*bounding means*) for obtaining correct bounds and two (*floating means*) for ensuring a high quality of the estimation. The interplay of these four means allows to exploit the slow convergence of means regarding correctness ($\tilde{\varepsilon} \geq \varepsilon$) and to overcome the same regarding quality ($\tilde{\varepsilon} \approx \varepsilon$).

Our experiments (see [Section 5.4](#)) illustrate that especially the bound-and-float algorithm provides (on various topologies) good accuracy estimates for reasonable costs. In particular, we have seen a high quality of the results without choosing determining topology-dependent parameters. Hence, further improvements can be easily achieved by more specific choices of the available parameters. By doing so, we can expect even good results for very specific topologies such as the complete graph.

6 | The Power Iteration in HPC.

A case study from biochemistry

In this chapter, we employ the power iteration in a parallel computing context and exemplify parallel computing research in terms of a large scale structured eigenvalue problem stemming from biochemistry. In particular, the motivation behind the presented results is to substantially increase the computationally tractable problem dimensions by a tandem of novel structure-aware algorithms together with their respective implementations on modern hardware.

6.1. The Quasispecies Model

[Niederbrucker and Gansterer, 2011a, § 1]
[Niederbrucker, 2013, § 1.3]

Solving Eigen’s quasispecies model [Eigen, 1971] for the evolution of virus populations involves the computation of the dominant eigenvector of a matrix whose dimension grows exponentially with the chain length of the virus to be modeled. Most of the biologically interesting chain lengths are well beyond the reach of general purpose solvers. Consequently, in order to raise the tractable chain lengths to a meaningful scale, specific solvers which adequately exploit certain problem properties are required.

6.1.1. Overview

Under some assumptions on the environment, the evolution and the long-term behavior of a virus population can be modeled by the *quasispecies model* [Eigen, 1971]. In this model, each virus is represented by an RNA molecule and each RNA molecule is represented as a string over a finite alphabet with a fixed length ν , the so-called *chain length*. Unless otherwise stated, we consider a binary alphabet, i. e., RNA molecules are represented as binary strings of length ν . Commonly, the probability of a single point mutation in an RNA sequence is modeled by a (uniform) *error rate* p satisfying $0 < p \leq 1/2$. Given a fixed chain length ν , all $N := 2^\nu$ possible RNA sequences have to be considered, since any RNA molecule can potentially mutate into any other one (although the overall probability for some of these mutations may be very low).

Given that initially a single species exists, the goal is to study how this initial population evolves over time by tracking the relative concentration of each species with respect to the entire population. In the limit, the population converges to an equilibrium which means that a stationary distribution of the relative concentrations is reached. This stationary distribution which is obtained by taking the limit over time is the so-called *quasispecies* [Eigen and Schuster, 1977]. The name “quasispecies” is motivated by the fact that the quasispecies is not a particular species but an equilibrium distribution which is reached in the model with respect to the employed model parameters. Our overall goal is to numerically compute the relative concentrations of the quasispecies which describes a stationary distribution in the model. More specifically, the computation of these relative concentrations is tantamount to the numerical computation of the eigenvector corresponding to the largest eigenvalue of a nonnegative matrix $W \in \mathbb{R}^{N \times N}$, which describes the constitution and the mutation probabilities of the N different RNA molecules with chain length ν (see Section 6.1.2). This eigenvector contains the information about the relative concentrations within the quasispecies. Since the dimension N of this eigenvalue problem grows *exponentially* with the chain length ν of the RNA molecules to be modeled, numerical solutions of the model based on general purpose methods are restricted to very small chain lengths. Hence, more specific algorithmic approaches are required in order to utilize the problem structure such that the range of computationally

6.1.2. Model Definition and Biochemical Background

In the following, we review the quasispecies model which leads to the large scale eigenvalue problem we investigate in this chapter. For more details regarding the chemical background we refer to survey articles [Schuster, 2006, 2008, 2011].

As stated above, each RNA molecule is formally represented by a string over a binary alphabet. In particular, by X_i with $1 \leq i \leq N$ we denote the RNA molecule and the corresponding species which is represented by the ν -digit binary encoding $(b_1, \dots, b_\nu)$ of the integer $i - 1$. Moreover, the error-free sequence X_1 associated with the integer 0 is called the *master sequence*. As a distance measure between species X_i and X_j we use the Hamming distance $d_H(X_i, X_j)$ which represents the minimal number of elementwise mutations required to transform X_i into X_j (see Section 6.4.1 for a rigorous definition of $d_H(\cdot, \cdot)$). When the meaning is clear from the context, we henceforth also use just i instead of X_i for denoting the binary string corresponding to the integer i . Due to Eigen [1971] the non-linear system of ODEs

$$\begin{aligned} \frac{dx_i}{dt} &= \sum_{j=1}^N f_j \cdot q_{i,j} \cdot x_j(t) - x_i(t) \cdot \Phi(t), \\ \Phi(t) &= \sum_{j=1}^N f_j \cdot x_j(t), \quad \sum_{j=1}^N x_j(t) = 1, \end{aligned} \tag{6.1}$$

with $i = 1, \dots, N$, models the evolution of RNA molecules. In this model the variables $x_i(t)$ denote the relative concentration of the molecular species X_i at time t . Initially, only the master sequence exists, i.e., $x_1(0) = 1$ and $x_i(0) = 0$ for $i \neq 1$. The positive fitness value f_i describes the constitution of the molecular species X_i and the (i, j) -th entry $q_{i,j}$ of the *mutation matrix* $Q \in \mathbb{R}^{N \times N}$ represents the probability that sequence X_j mutates into sequence X_i .

In order to explicitly determine the mutation probabilities $q_{i,j}$, the so-called *uniform error model* [Eigen, 1971] is commonly applied, which leads to

$$q_{i,j} := p^{d_H(X_i, X_j)} \cdot (1 - p)^{\nu - d_H(X_i, X_j)}. \quad (6.2)$$

By definition, the mutation probabilities $q_{i,j}$ depend only on $d_H(X_i, X_j)$ and therefore, the entire matrix Q contains only $\nu + 1$ different values. It should be pointed out that this standard model assumes that p is an *average* error rate over all possible mutations since it does not depend on the position or on the overall number of the mutations. More realistic models are believed to substantially increase the computational complexity and are therefore rarely considered in the literature [Schuster, 2011]. In contrast to that, we show in Section 6.4.2 how to handle more general models without sacrificing performance.

According to Thompson and McBride [1974], the ODE system (6.1) can be straightforwardly transformed into a linear system with constant coefficients $\dot{z} = Wz$ by a suitable change of variables. More specifically, the matrix W is given in terms of a factorization $W = QF$, where Q is defined by (6.2) and F , the so-called *fitness landscape*, is a diagonal matrix with the fitness values $f_i > 0$ along its diagonal. The problem of searching the quasispecies then reduces to the computation of the eigenvector corresponding to the dominating eigenvalue of W [Schuster, 2008]. In fact, several mathematically equivalent formulations of the problem with slightly differing structure can be obtained via similarity transformations (see Definition A.3). In particular, by similarity transforms with $F^{\frac{1}{2}}$ (i.e., the elementwise square root of F) and F we obtain

$$QFx_R = \lambda x_R, \quad F^{\frac{1}{2}}QF^{\frac{1}{2}}x_S = \lambda x_S, \quad FQx_L = \lambda x_L. \quad (6.3)$$

Since F is diagonal, the solutions of these problems can easily be transformed into each other:

$$x_R = F^{-\frac{1}{2}}x_S, \quad x_S = F^{-\frac{1}{2}}x_L, \quad x_R = F^{-1}x_L.$$

Note that in the special case where all values in F are equal, the problem reduces to the computation of the dominating eigenvector of a bistochastic matrix, which is trivial and leads to an eigenvector where all entries are equal. This is not surprising, since for equally fit sequences we expect the uniform distribution as result. Because the variables $x_i(t)$ in (6.1) represent relative concentrations, we are only interested in solutions where all components of the computed eigenvector are nonnegative (negative concentrations do not have a physically meaningful interpretation). Since the

matrix W is by definition a positive matrix, it satisfies the conditions of the Perron-Frobenius theorem (cf. [Section A.3](#)) and therefore, the aforementioned nonnegativity property is guaranteed.

In the quasispecies model not only the concentrations of individual species are studied, but also cumulative concentrations of so-called *error classes*. The error class $\Gamma_{k,i}$ contains all sequences j , which have a certain Hamming distance k from the fixed sequence i :

$$\Gamma_{k,i} := \{j \mid 1 \leq j \leq N \wedge d_H(X_i, X_j) = k\} \quad (6.4)$$

Since the error classes with respect to the master sequence are particularly relevant, we define $\Gamma_k := \Gamma_{k,1}$. By definition the error class Γ_k contains $\binom{\nu}{k}$ sequences (cf. [Lemma 6.2](#)). In the style of the definition of the error classes Γ_k we denote the $\nu + 1$ different values of Q by Q_{Γ_k} , i. e., $Q_{\Gamma_k} := p^k \cdot (1 - p)^{\nu-k}$ for $0 \leq k \leq \nu$. After the dominating eigenvector x of W has been computed, the cumulative concentrations $[\Gamma_k] := \sum_{j \in \Gamma_k} x_j$ of the error classes Γ_k in the stationary distribution are obtained. Plotting these cumulative concentrations for different error rates p leads to curves as the ones shown in [Figure 6.1](#).

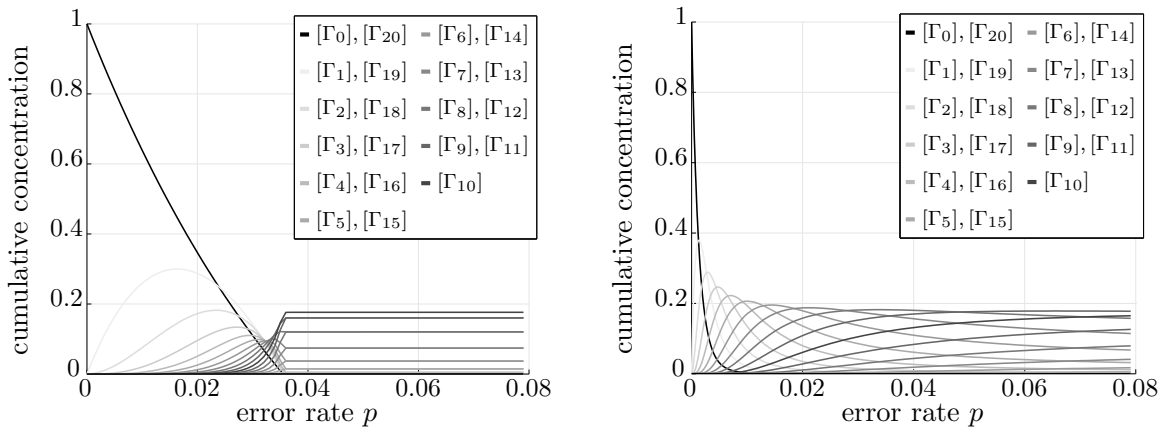


Figure 6.1.: A visualization of the error threshold phenomenon is shown on the left for $\nu = 20$, in case of a *single peak fitness landscape* with $f_1 = 2$, $f_i = 1$ for all $1 < i \leq N$. An ordered stationary distribution results up to $p_{\max} \approx 0.035$, and for $p > p_{\max}$ a sudden change to the uniform distribution of all sequences occurs. Equally large error classes (i. e., Γ_k and $\Gamma_{\nu-k}$) are depicted in the same color. This implies that the curves of equally large error classes meet when the uniform distribution is reached at the error threshold. The right plot depicts the behavior for $\nu = 20$ in case of the so-called *linear landscape*, which is defined as $f_i = f_1 - (f_1 - f_\nu) \cdot d_H(i, 1)/\nu$ for all $1 \leq i \leq N$ with $f_1 = 2$ and $f_\nu = 1$. For this landscape a smooth transition into the uniform distribution is observed and the error threshold phenomenon does not occur.

Such figures can be used to get a better understanding of how a certain fitness landscape influences the evolution of the virus population. Similar plots at the level of single sequences would be even more interesting, but such investigations appear rarely in the literature, due to the limitations in chain lengths which can be handled computationally.

Depending on the concrete fitness values, the *error threshold phenomenon* may occur or not. If it occurs, there is a non-uniform stationary distribution of the concentrations up to a critical value $p_{\max}$ for the error rate p , where certain error classes clearly dominate, whereas others do not appear at all due to low cumulative concentrations. For $p > p_{\max}$, the structure of the population changes suddenly into a uniform stationary distribution where all sequences occur in the same concentration, which is equivalent to random replication. Note that in Figure 6.1 cumulative concentrations are shown for the error classes. This implies that although all sequences have the same concentration for $p > p_{\max}$, the cumulative concentrations of the error classes differ because their cardinality differ, i.e., $|\Gamma_k| = \binom{\nu}{k}$. Typical values for $p_{\max}$ on certain fitness landscapes are in the range 0.01–0.1 [Schuster, 2008, 2011], depending on the concrete fitness values and the chain length. Such small values for $p_{\max}$ are quite surprising since random replication as exact solution of the ODE system (6.1) is obtained only for $p = 0.5$ [Schuster, 2008]. This sudden change from an ordered distribution to random replication is of potential interest as a building block for new antiviral strategies [Eigen, 2002] because the error rates of RNA viruses are usually close to this critical value [Drake, 1993] and an increase of the error rate p is possible by the use of pharmaceutical drugs.

6.1.3. Existing Approaches vs. Contributions

Traditionally, computational investigations in the large body of literature about the quasispecies model were limited to the case where the fitness landscape F is defined via the Hamming distance $f_{i,i} := \varphi(d_H(i, 1))$ for some function $\varphi: \{0, \dots, \nu\} \rightarrow \mathbb{R}$ [Schuster, 2011]. Practically, this means that all sequences with the same distance to the master sequence are equally fit, which is often a rather unrealistic assumption. With this simplifying assumption one expects that the problem reduces from an $N \times N$ to a $(\nu + 1) \times (\nu + 1)$ problem, since all sequences in the same error class Γ_k are considered equivalent. For this special setup efficient approximation methods have been developed [Swetina and Schuster, 1982; Nowak and Schuster, 1989]. Therefore, also studies of the error threshold phenomenon have been restricted to this simplified problem formulation.

In contrast to these approximation methods, our objective in this work is to deal with the general form of the problem without any special assumptions, where no fast solvers are available [Schuster, 2011]. For the general problem without assumptions on the structure of the fitness landscape we employ a power iteration-based approach and introduce several novel fast matrix vector multiplication routines for the occurring matrices. In particular, all matrix vector routines we consider are of a matrix-

free (implicit) nature in order to enable a substantial increase of the computationally tractable chain lengths ν . Moreover, we also show how to generalize the computationally tractable model assumptions and how to exploit specifically structured instances.

6.2. Theoretical Insights and Fast Algorithms

Most important algorithms for computing the extreme eigenpair of a (large scale) matrix A require solely a routine for evaluating the matrix vector product $v \mapsto Av$. In particular, large data-sparse problems like the problem at hand require to evaluate $v \mapsto Av$ in a matrix-free fashion without storing the elements of A explicitly. Consequently, we derive in this section efficient implicit matrix vector products with $W = QF$, which fully exploit the underlying problem structure. Since F is diagonal, it suffices to focus on a matrix vector product with the mutation matrix Q .

6.2.1. Structured Implicit Sparsification [Niederbrucker and Gansterer, 2011b, §3]

A straightforward approach towards an efficient matrix vector routine for $W = QF$ is to sparsify the problem. Since Q is defined via powers of p and the practically relevant choices for p are usually not significantly larger than 10^{-2} [Swetina and Schuster, 1982] we see that there are many elements below machine precision which we can eliminate already for small ν . Due to the problem structure, we refrain from specifying a numerical bound on the elements of W we want to keep. Instead, we use a bound $d_H^{\max}$ for the maximal Hamming distance we take into account in (6.2). This restriction can easily be applied on the matrix Q , since the (i, j) -th element of Q is defined via the Hamming distance between the sequences corresponding to i and j , respectively. In total, by multiplying the sparsified version of Q by F , we get a sparse version of W since F is diagonal.

In the quasispecies model, this approach corresponds to restricting the possible mutants of an RNA molecule to near relatives. A byproduct of this sparsification strategy is that the requirements for Perron-Frobenius theorem remain satisfied. In particular, the sparsified matrix W is indeed primitive even for $d_H^{\max} = 1$ because in this minimal case the non-zero values of Q correspond to the non-zero values of the adjacency matrix of the ν -dimensional hypercube. Therefore, there exists a mutation path between any two sequences X_i and X_j which guarantees the condition for primitivity, namely that there exists a value of m such that W^m contains only positive elements. The border case $d_H^{\max} = 0$ is meaningless, since in this case no mutations are possible and therefore, we have to compute the dominating eigenvector of a diagonal matrix, which is trivial. Concretely, it gives us the expected result that only the master sequence (more precisely, the sequence with the dominating fitness value) appears in the stationary distribution.

The fact that the entries of F are not considered in the sparsification is not a problem, since they usually do not vary as much as the entries of Q do. Thus, our

simple strategy of restricting the maximal Hamming distance which is considered in (6.2) delivers almost the same results as specifying a numerical threshold for the entries of W . Obviously, this sparsification technique implies a certain loss of accuracy in the eigenvector computed. More precisely, the sparsification error is at least of the order of the largest (in magnitude) element of W we set to zero [Bai *et al.*, 2004].

So far we only dealt with the structured sparsification of the matrix W . As a next step, we design an implicit matrix vector product with W which does not require the elements of W to be stored in memory. Recall that we did not specify a specific numerical bound for the elements of Q but instead we used a bound $d_H^{\max}$ for the maximal Hamming distance we want to take into account in (6.2). In order to obtain an implicit matrix vector product with W , we only have to appropriately reorder the operations of the standard matrix vector product. Usually, we traverse the rows of W from left to right. Instead, we traverse the elements (r, i) of row r sorted according to $d_H(X_i, X_r)$. In the notation of Section 6.1.2 this means that we run through the different error classes (cf. (6.4)) of the row index. More specifically, for computing $y = Wx$ we get

$$y_i = \sum_{j=1}^N q_{i,j} \cdot f_{j,j} \cdot x_j = \sum_{k=0}^{\nu} \sum_{j \in \Gamma_{k,i}} q_{i,j} \cdot f_{j,j} \cdot x_j = \sum_{k=0}^{\nu} Q_{\Gamma_k} \cdot \left(\sum_{j \in \Gamma_{k,i}} f_{j,j} \cdot x_j \right).$$

Again, we obtain a “sparse” matrix vector product by replacing ν as upper bound for the sum by some limit $d_H^{\max}$ for the Hamming distance we want to consider. The sorting according to d_H has to happen implicitly during the multiplication, otherwise we would end up with a significant increase of the time complexity.

In general, the crucial problem is to sort the sequences X_i according to $d_H(X_i, X_r)$, where X_r denotes a fixed sequence. For simplicity we first use the element $X_1 = (0, \dots, 0)$, because computing the Hamming distance to X_1 just means counting bits. We also know in advance the number of elements with a certain Hamming distance. Therefore, we can use a bucket sort-like procedure [Cormen *et al.*, 2001, §8.4] to create the sorted sequence with respect to $d_H(\cdot, X_1)$ in linear time without any additional memory needed.

Up to now we have a properly sorted sequence for one base element $X_1 = (0, \dots, 0)$ but we need such a sequence for every possible base element X_i . Computing these sequences separately is highly inefficient. Thus, we use the single sequence with base $(0, \dots, 0)$ and transform this sequence into the one we actually need during the multiplication. For this purpose recall the truth table of the binary XOR $\oplus$:

$\oplus$	0	1
0	0	1
1	1	0

An informal interpretation of $A \oplus B$ is that the bit A gets flipped if and only if the bit B is true. As we know, the extension of $\oplus : \{0, 1\} \mapsto \{0, 1\}$ for bit vectors to a bitwise

$\text{XOR } \oplus : \{0, 1\}^\nu \mapsto \{0, 1\}^\nu$ is one of the commonly available operations for basic data types like integers on today's digital computers. By our informal interpretation of $\oplus$ we can conclude that for two binary sequences X_i and X_j it holds that

$$d_H(X_i, X_i \oplus X_j) = d_H(X_j, X_1) \quad (6.5)$$

Now we have all what we need for an efficient (implicit) matrix vector product: We can compute the sorted sequence with base element $(0, \dots, 0)$ efficiently once as a preprocessing step and the $\oplus$ operator lets us transform this sequence to the sorted sequence for an arbitrary base element during the multiplication. Using (6.5) we get

$$y_i = \sum_{k=0}^{\nu} Q_{\Gamma_k} \cdot \left(\sum_{j \in \Gamma_{k,i}} f_{j,j} \cdot x_j \right) = \sum_{k=0}^{\nu} Q_{\Gamma_k} \cdot \left(\sum_{j \in \Gamma_k} f_{i \oplus j} \cdot x_{i \oplus j} \right) \quad (6.6)$$

as new formula for computing the elements of the output vector y .

Putting everything together leads to [Algorithm 6.1](#) as an efficient procedure for computing $y = Wx$ using a specific sparsification parameter $d_H^{\max}$. It is easily verified

Algorithm 6.1 Sparse matrix vector multiplication $y = Wx$ (XMVP($d_H^{\max}$))

Input: $x \in \mathbb{R}^N$, $Q_{\Gamma} \in \mathbb{R}^{\nu+1}$, $F \in \mathbb{R}^N$, $S \in \mathbb{N}^N$ (sorted according d_H to $(0, \dots, 0)$)

Output: $y = Wx \in \mathbb{R}^N$ (according to (6.6))

```

1: for all  $i \in \{1, \dots, N\}$  do
2:    $y(i) \leftarrow 0$ ,  $j \leftarrow 1$ 
3:   for  $k \leftarrow 0$  to  $d_H^{\max}$  do
4:      $\text{sum} \leftarrow 0$ 
5:     for  $l \leftarrow 1$  to  $\binom{\nu}{k}$  do
6:        $\text{sum} \leftarrow \text{sum} + F(i \oplus S(j)) \cdot x(i \oplus S(j))$ 
7:        $j \leftarrow j + 1$ 
8:     end for
9:      $y(i) \leftarrow y(i) + Q_{\Gamma}(k) \cdot \text{sum}$ 
10:  end for
11: end for

```

that [Algorithm 6.1](#) is an algorithmic encoding of (6.6). The occurring fixed values $S(k)$, $\binom{\nu}{k}$ and Q_{Γ_k} should be precomputed once before starting the power iteration. In the following we always refer to the (sparse) matrix vector product developed in this section by XMVP($d_H^{\max}$), where $0 < d_H^{\max} \leq \nu$ controls the degree of sparsification. Hence, XMVP(ν) can be seen as an implicit variant of the standard matrix vector multiplication which we later refer to by SMVP.

A natural question arising now is how this newly defined sparse matrix vector product XMVP($d_H^{\max}$) compares to standard methods for different $d_H^{\max}$. The maximum

possible degree of sparsification in $\text{XMVP}(d_H^{\max})$ is achieved by setting $d_H^{\max} = 1$ (consider only the “neighboring” sequences with Hamming distance one). This leads to a time complexity of $\Theta(N \cdot (\nu + 1)) = \Theta(N \log_2 N + N)$ for evaluating the matrix vector product. This means, we can provide efficient (implicit) matrix vector products with W , by trading off accuracy for complexity. At this point one must not forget that the intended sparsification leads to an error in the computed eigenvector of the order of the largest element of W which becomes zero [Bai *et al.*, 2004]. Summarizing, Figure 6.2 illustrates how $\text{XMVP}(d_H^{\max})$ scales with the chain length ν for different choices of the sparsification parameter $d_H^{\max}$.

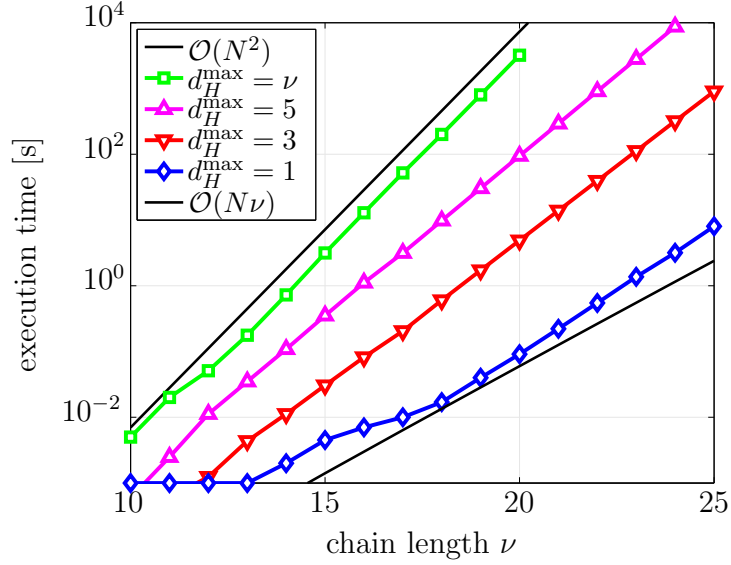


Figure 6.2.: Runtimes for computing $y = Wx$ by employing $\text{XMVP}(d_H^{\max})$ with random input vector x , using different levels of sparsification $d_H^{\max}$. The depicted times are averages over ten individual runs on a INTEL I5 750 @ 2.67Ghz processor with 4Gb memory.

6.2.2. Fast Mutation Matrix Product

In contrast to the previous section, where we traded off accuracy for efficiency, we develop in the following an *exact* matrix vector product for W , which requires only an effort of $\Theta(N \log_2 N)$ floating-point operations. This means that we derive an algorithm which outperforms even the coarsest approximation $\text{XMVP}(1)$, while computing fully accurate results.

According to the definition of Q in (6.2), mutation is modeled in the quasispecies model by independent coin flips (with heads up probability p) for each position in the sequence. Each entry $q_{i,j}$ is the result of the product of ν independent stochastic processes (coin flips). This property allows for using a Kronecker product representation which avoids the use of the Hamming distance and combines the ν independent

stochastic single point mutation processes implicitly [Dress and Rumschitzki, 1988]. In particular,

$$Q(\nu) = \bigotimes_{i=1}^{\nu} \begin{pmatrix} (1-p) & p \\ p & (1-p) \end{pmatrix}. \quad (6.7)$$

This representation can also be described recursively

$$\begin{aligned} Q(0) &:= 1 \\ Q(\nu) &:= \begin{pmatrix} (1-p) \cdot Q(\nu-1) & p \cdot Q(\nu-1) \\ p \cdot Q(\nu-1) & (1-p) \cdot Q(\nu-1) \end{pmatrix}. \end{aligned} \quad (6.8)$$

Rumschitzki [1987] noted that as an immediate consequence of the Kronecker product-based representation (6.7) the eigenvalues $\Lambda(\nu)$ and corresponding eigenvectors $V(\nu)$ of $Q(\nu)$ can be directly computed as

$$\begin{aligned} \Lambda(\nu) &:= \bigotimes_{i=1}^{\nu} \begin{pmatrix} 1 & 0 \\ 0 & 1-2p \end{pmatrix} \\ V(\nu) &:= 2^{-\frac{\nu}{2}} \cdot \bigotimes_{i=1}^{\nu} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \end{aligned}$$

The symmetry of the matrix of eigenvectors $V(\nu)$ leads to a simplified eigendecomposition $Q(\nu) = V(\nu)\Lambda(\nu)V(\nu)$. This implicit description of the eigendecomposition can be transformed back into a componentwise representation

$$\begin{aligned} (\Lambda(\nu))_{i,i} &= (1-2p)^{d_H(i,0)} \\ (V(\nu))_{i,j} &= 2^{-\frac{\nu}{2}} \cdot (-1)^{1/2 \cdot (d_H(i,0) + d_H(j,0) - d_H(i,j))}. \end{aligned}$$

One can conclude that $Q(\nu)$ has the eigenvalues $(1-2p)^k$ for $0 \leq k \leq \nu$ with multiplicities $\binom{\nu}{k}$. For $p < 1/2$ this implies that the mutation matrices Q are always positive definite. By $(F^{\frac{1}{2}}x)^T Q(F^{\frac{1}{2}}x) > 0$ for any landscape F and vector x , we also see that the eigenvalue problems from (6.3) involve positive definite matrices, and therefore all eigenvalues of W are positive real numbers. Moreover, the eigendecomposition of $Q(\nu)$ also indicates that we may expect an efficient matrix vector product with $Q(\nu)$, since multiplication with the eigenvector matrix $V(\nu)$ can be performed by the well known fast Walsh-Hadamard transform (FWHT) with $\Theta(N \log_2 N)$ operations (see Section A.2.2 for further details). Similar to the FWHT, we derive an efficient matrix vector product with W in the following.

Fast transformations (matrix vector products) usually appear hand in hand with a Kronecker product representation of the underlying matrix [Van Loan, 2000]. Based on (6.8), the $N \times N$ matrix vector product

$$Q(\nu)v = \begin{pmatrix} (1-p) \cdot Q(\nu-1) & p \cdot Q(\nu-1) \\ p \cdot Q(\nu-1) & (1-p) \cdot Q(\nu-1) \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

can be computed in two ways. We can either first compute the two $N/2 \times N/2$ matrix vector products $\bar{v}_1 := Q(\nu - 1) \cdot v_1$ and $\bar{v}_2 := Q(\nu - 1) \cdot v_2$ and then

$$Q(\nu)v = \begin{pmatrix} (1-p)\bar{v}_1 & + & p\bar{v}_2 \\ p\bar{v}_1 & + & (1-p)\bar{v}_2 \end{pmatrix}, \quad (6.9)$$

or we directly compute

$$Q(\nu)v = \begin{pmatrix} Q(\nu - 1) \cdot ((1-p)v_1 + pv_2) \\ Q(\nu - 1) \cdot (pv_1 + (1-p)v_2) \end{pmatrix}. \quad (6.10)$$

Lemma 6.1. *Computing the matrix vector product $Q(\nu)v$ using (6.9) or (6.10) recursively has a runtime complexity $\Theta(N \log_2 N)$.*

Proof. The complexity result can be easily deduced by the Master Theorem [Cormen et al., 2001, p.73]. Setting $a = b = 2$ and $f(n) \in \Theta(n^{\log_b(a)})$ in the recurrence $T(n) = aT(n/b) + f(n)$ ($f(n)$ describes the cost for combining the solutions of the recursively solved subproblems) and applying the theorem leads to a runtime complexity $\Theta(N \log_2 N)$. $\square$

Due to the Kronecker product representation of Q we also observe that both matrix vector products (6.9) and (6.10) work *in situ* without requiring additional storage, as it is common for such transformations. Since F is diagonal, the matrix vector products with the matrices QF , $F^{\frac{1}{2}}QF^{\frac{1}{2}}$ and FQ (cf. (6.3)) also have a complexity of $\Theta(N \log_2 N)$. Henceforth, we refer to the fast matrix vector product based on (6.9) or (6.10) as *fast mutation matrix product* (FMMP). The efficient implementation of FMMP on GPUs is discussed in Section 6.3.

6.2.3. Computing the Quasispecies

Up to now we derived efficient algorithms for computing $y = Wx$, but we did not yet address the problem of computing the desired extremal eigenvector. Recall that we are free to choose any of the problem formulations shown in (6.3). In particular, this implies that without loss of generality we may restrict our investigation to a symmetric eigenproblem with real eigenvalues.

As already noted, at this point we are considering problems without any specific assumptions on the fitness landscape beyond diagonality. The matrix F is therefore a general diagonal matrix and all its N values have to be stored. An unstructured landscape F also implies an unstructured resulting extremal eigenvector with $N = 2^\nu$ values to be stored. Due to this exponential growth with ν it is crucial to consider methods with minimum additional storage requirements beyond the unavoidable parameters, i.e., the fitness landscape F , the elements of Q , and an approximation of the dominating eigenvector of $W = QF$.

Potentially relevant methods for computing an extremal eigenvector on the basis of an (implicitly available) matrix vector product are the basic power iteration or

Lanczos/Arnoldi iterations [Trefethen and Bau, 1997, parts V and VI]. The latter require storing more intermediate vectors than the simpler power iteration and are thus less attractive for very large scale instances of the problem considered here. A completely different approach to the problem of computing the extremal eigenvector of a matrix is to use randomized (Monte Carlo) methods [Mascagni and Karaivanova, 2002; Drineas *et al.*, 2003]. Although such approaches allow for a strong reduction of the problem dimension, this reduction tends to strongly reduce the accuracy of the result. In the problem at hand, increasing ν not only leads to growing storage requirements, but also to growing accuracy requirements since the eigenvector to be computed contains relative concentrations. Consequently, randomized methods are not the first choice in this context.

In the following, we focus on the investigation of a power iteration (PI) approach, which provides the best balance between storage requirements and accuracy. The particular matrix vector product used is stated as an argument in the notation, e.g., PI(FMMP) denotes the power iterations on the basis of the fast FMMP matrix vector product introduced in Section 6.2.2. Since W is positive definite and the Perron-Frobenius theorem is applicable, it is assured that the eigenvalues of W satisfy

$$\lambda_1 > \lambda_2 \geq \lambda_3 \geq \dots \geq \lambda_N > 0.$$

Therefore, convergence of the power iteration is guaranteed in practice (cf. Section 3.2).

As starting vector s for the power iteration we use the fitness landscape itself $s = \text{diag}(F)/\|\text{diag}(F)\|_1$, since we can expect the extremal eigenvector of $W = QF$ to be of a similar shape (remember that the extremal eigenvector of Q is a multiple of $(1, 1, \dots, 1)^T$). As stopping criterion we monitor the residual $R(\tilde{\lambda}, \tilde{x}) = \|W\tilde{x} - \tilde{\lambda}\tilde{x}\|_2$.

Convergence Acceleration

As we have seen in Section 3.2.1, the eigenvector approximation of the power iteration converges linearly with convergence rate λ_2/λ_1 . In the following we derive a generally applicable shift μ , in order to enhance the convergence rate to $(\lambda_2 - \mu)/(\lambda_1 - \mu)$. Of course, the shift μ has to be chosen such that $\lambda_1 - \mu$ remains the largest (in magnitude) eigenvalue of $W - \mu I$. For deriving a generally applicable shift μ , we again examine the Kronecker product representation of $Q(\nu)$ in (6.7). By exploiting the property $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$ we can represent the inverse $Q(\nu)^{-1}$ as

$$Q(\nu)^{-1} = (1 - 2p)^{-\nu} \bigotimes_{i=1}^{\nu} \begin{pmatrix} (1-p) & -p \\ -p & (1-p) \end{pmatrix}. \quad (6.11)$$

According to (6.11), the absolute row and column sums of $Q(\nu)^{-1}$ are all $(1 - 2p)^{-\nu}$. With this knowledge we can compute $\|\cdot\|_1$ of the matrices $W = Q \cdot F$ and $W^{-1} = F^{-1}Q^{-1}$ in order to derive bounds for the largest eigenvalue λ_1 as well as for the smallest eigenvalue λ_N of W :

$$\lambda_1 \leq \|W\|_1 \leq \|Q\|_1 \cdot \|F\|_1 = f_{\max} := \max_{i=1}^N F_{i,i}$$

and

$$\begin{aligned}\lambda_N^{-1} \leq \|W^{-1}\|_1 &\leq \|F^{-1}\|_1 \cdot \|Q^{-1}\|_1 \\ &= f_{\min}^{-1} \cdot (1 - 2p)^{-\nu},\end{aligned}$$

where $f_{\min} := \min_{i=1}^N F_{i,i}$. Since $f_{\min} > 0$ and $p < 1/2$, we can conclude that $\lambda_N \geq (1 - 2p)^\nu f_{\min}$. Therefore, applying the shift $\mu := (1 - 2p)^\nu f_{\min}$ is in any case possible, because $\lambda_1 - \mu$ remains the largest eigenvalue and hence, the shifted power iteration still converges to the desired eigenvector. Although this choice of the shift μ is very conservative, it results in a clearly measurable reduction of the number of iterations of about ten percent and more for the random landscapes we considered. Additionally, in many cases efficient shifts which exploit the structure of a particular landscape F can be derived via the Gershgorin circle theorem [Golub and Van Loan, 1996, Theorem 7.2.1].

6.2.4. Limitations of Power-Iteration Approaches [Niederbrucker, 2013, § 1.3.4]

The aforementioned power iteration-based approaches are very simple and efficient in cases the dominating eigenvalue of the matrix W is sufficiently well separated from the other eigenvalues, since the power iteration converges linearly of the order $\mathcal{O}(\lambda_2/\lambda_1)$ where λ_1, λ_2 denote the two largest (in magnitude) eigenvalues of W [Trefethen and Bau, 1997, Theorem 27.1]. In cases of a bad separation the power iteration suffers from a very slow convergence. In our case, the Perron-Frobenius theory (cf. Section A.3) solely guarantees that $\lambda_1 > \lambda_2$. and thus, a well-behaved spectrum (with respect to the power iteration) can not be guaranteed in practice. As we have seen in Section 6.2.3, applying a shift μ to the power iteration can lead to better convergence properties. The drawback of the simple power iteration is that the choice of this shift is limited by the constraint that the dominating eigenvalue must remain the largest eigenvalue in magnitude. Otherwise the dominating eigenvector—our objective we want to compute—changes.

In contrast to the simple power iteration, *shift-and-invert* methods such as the *inverse iteration* or the *Rayleigh quotient iteration* [Trefethen and Bau, 1997, Algorithms 27.2 and 27.3] allow (conceptually) for an “arbitrary good” shift. This is due to the fact that such iterations run a power iteration on a shifted and inverted variant of the input matrix. This means that such algorithms proceed by solving linear systems of the form $(A - \mu I)x_{k+1} = x_k$ in each iteration. Thus, since the eigenvalues of the inverse of a matrix A are the reciprocals of the eigenvalues of A , the shift μ has to be chosen such that $\lambda_1 - \mu$ is the smallest eigenvalue of $A - \mu I$, in order that $(\lambda_1 - \mu)^{-1}$ is the largest eigenvalue of $(A - \mu I)^{-1}$. For a more detailed treatment of the utilization and limitations of shift-and-invert approaches for the problem at hand see Niederbrucker [2013].

6.3. Parallel Quasispecies Computations

[Niederbrucker and Gansterer, 2011a, § 4]

In this section, parallel implementations of the power iteration based on FMMP and based on XMVP($d_H^{\max}$) are compared. We illustrate that the new FMMP-based approach is well suited for exploiting the potential of state-of-the-art GPU hardware. For the sake of portability, we implemented our algorithms using OpenCL². This implies that our codes can also make use of modern multi-core CPUs besides GPUs.

6.3.1. Implementation

Note that the implementation presented below uses only straightforward techniques and is not intended to compete with (auto-)tuned codes produced by program generators such as SPIRAL [Püschel *et al.*, 2005] (see Section A.2.2 for some more details on this subject). Nevertheless, in view of the exponential growth of the problem at hand, the pivotal aspect is the algorithmic improvement. Relatively speaking, the improvements achievable by further code optimizations do not result in a noteworthy increase of the tractable problem dimensions. Our implementation of the fast matrix vector product FMMP shown in Algorithm 6.2 is an iterative formulation of the algorithm given in (6.9). The corresponding algorithm for (6.10) can be obtained by reversing the index sequence in the outermost i -loop. Algorithm 6.2 is—up to multiplications with

Algorithm 6.2 FMMP based on (6.9)

Input: $v \in \mathbb{R}^N$

Output: $Q(\nu)v$

```

1: for  $i \leftarrow 1$  to  $N/2$  by  $2 \cdot i$  do
2:   for  $j \leftarrow 0$  to  $N - 1$  by  $2 \cdot i$  do
3:     for  $k \leftarrow 0$  to  $i - 1$  do
4:        $t_1 \leftarrow v[j + k], t_2 \leftarrow v[j + k + i]$ 
5:        $v[j + k] \leftarrow (1 - p) \cdot t_1 + p \cdot t_2$ 
6:        $v[j + k + i] \leftarrow p \cdot t_1 + (1 - p) \cdot t_2$ 
7:     end for
8:   end for
9: end for

```

constants—identical with basic implementations of the FWHT and could therefore be optimized analogously.

For execution on a GPU or, more general, for the computations with OpenCL we need to provide a so-called kernel function where the current thread id is queried for

²Open Computing Language, Khronos group, see <http://www.khronos.org/opencl/>

deciding which computations have to be executed. A corresponding formulation is shown in [Algorithm 6.3](#), where ID denotes the thread id and “&” denotes the bitwise AND operation. Like [Algorithm 6.2](#), [Algorithm 6.3](#) performs exactly $\log_2 N$ iterations

Algorithm 6.3 GPU/OpenCL ready implementation of (6.9)

Input: $v \in \mathbb{R}^N$
Output: $Q(\nu) \cdot v$

```

1: for  $i \leftarrow 1$  to  $N/2$  by  $2 \cdot i$  do
2:   for ID  $\leftarrow 0$  to  $N/2 - 1$  by 1 do
3:      $j \leftarrow 2 \cdot \text{ID} - (\text{ID} \& (i - 1))$ 
4:      $t_1 \leftarrow v[j]$ 
5:      $t_2 \leftarrow v[j + i]$ 
6:      $v[j] \leftarrow (1 - p) \cdot t_1 + p \cdot t_2$ 
7:      $v[j + i] \leftarrow p \cdot t_1 + (1 - p) \cdot t_2$ 
8:   end for
9: end for

```

of the outer i -loop, but in contrast to [Algorithm 6.2](#) the ID-loop has a constant range of $N/2$. However, we see that these $N/2$ iterations of the ID-loop are entirely independent of each other and therefore a very high degree of parallelism is available. The index computation in line 3 can be easily verified starting from the usual formula for this task

$$\begin{aligned}
& 2 \cdot i \cdot \lfloor \text{ID}/i \rfloor + \text{ID} \bmod i \\
&= 2 \cdot (\text{ID} - (\text{ID} \bmod i)) + \text{ID} \bmod i \\
&= 2 \cdot \text{ID} - \text{ID} \bmod i \\
&= 2 \cdot \text{ID} - (\text{ID} \& (i - 1)).
\end{aligned}$$

The last step exploits that i is always a power of two and that in this case the modulo computation can be performed via the cheaper bitwise AND operation. When using [Algorithm 6.3](#) as base for an OpenCL implementation, the i -loop runs at the host. In each iteration of the i -loop, the host invokes the kernel with $N/2$ threads, which itself is given by lines 3–7. In a real implementation, ID is determined by the proper OpenCL command.

This implementation of the fast matrix vector product FMMP scales nicely with the parallelization potential of the underlying hardware. Since there is a relatively high number of memory operations compared to floating-point operations, one can expect that the memory bandwidth of the hardware has an important influence on the overall performance. This can be observed in the experiments, since the performance achieved on the GPU exactly corresponds to its memory bandwidth.

Beyond the matrix vector product, the power iteration only needs a fast procedure for the summation of the components of a vector for computing norms and the residual

after each iteration. Since summing the components of a vector can be relatively well parallelized [Breshears, 2009, § 6], this part of the power iteration method has almost no influence on the overall execution time.

6.3.2. Experimental Results

For our experiments, we do not make any special assumptions on the values of the (diagonal) fitness landscape F , and use (similar to Schuster [2011]) landscapes of the form

$$f_0 = c, \quad f_i = \sigma \cdot (\eta_{\text{rnd}}(i) + 0.5), \quad (6.12)$$

where $c > 0$, $\sigma \in (0, c/2)$ and $\eta_{\text{rnd}}(i)$ is a call to a uniform random number generator on the interval $[0, 1]$.

For evaluating the methodical improvement provided by our fast matrix vector product we compare PI(FMMP) with PI(XMVP(ν)) and PI(XMVP(5)). The choice $d_H^{\text{max}} = \nu$ in the approximate (sparse) matrix vector product XMVP(d_H^{max}) illustrates how FMMP compares to the standard matrix vector product SMVP $\equiv$ XMVP(ν). The choice $d_H^{\text{max}} = 5$ in XMVP(d_H^{max}) has been shown to yield an approximation error around 10^{-10} , whereas the accuracy achieved with smaller values for d_H^{max} is usually too low [Niederbrucker and Gansterer, 2011b]. Consequently, these values are not considered here. Choices of $d_H^{\text{max}} > 5$ lead to a smaller error, for the price of a higher complexity. Recall that FMMP is faster than XMVP(d_H^{max}) even for the lowest possible accuracy with $d_H^{\text{max}} = 1$.

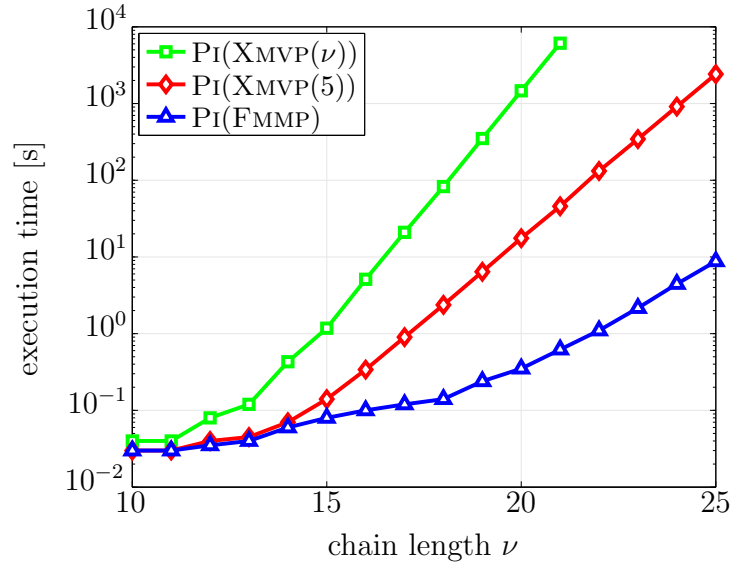


Figure 6.3.: Overall execution times on a single NVIDIA TESLA C2050 GPU for finding the dominating eigenvector of $W = QF$ ($p = 0.01$) on a random landscape (6.12) with $c = 5$ and $\sigma = 1$ for increasing chain length ν .

Figure 6.3 shows the improvement FMMP delivers when used as the building block of the power iteration. In all cases $p = 0.01$ was chosen and the iteration was stopped when the residual $R(\tilde{\lambda}, \tilde{x})$ for the approximated eigenpair $(\tilde{\lambda}, \tilde{x})$ was less than an appropriate threshold τ ($\tau = 10^{-10}$ for $d_H^{\max} = 5$ and $\tau = 10^{-15}$ otherwise). All computations were performed on an NVIDIA TESLA C2050 with 3Gb memory. The runtimes shown are overall execution times which include the data transfer from and to the GPU.

In order to better differentiate the influence of different algorithms from the influence of different hardware platforms, we depict the speedups for solving the quasispecies model for various combinations of algorithms and hardware in Figure 6.4. As reference value for evaluating the speedup we use the execution times of $\text{PI}(\text{XMVP}(\nu))$ running on a CPU (INTEL I5 750 @ 2.67GHZ with 4Gb memory), which achieves the same accuracy as $\text{PI}(\text{FMMP})$. For $\nu \geq 22$ the execution times for $\text{PI}(\text{XMVP}(\nu))$ are so long that they had to be extrapolated based on the curves in Figures 6.2 and 6.3 for estimating the speedup values.

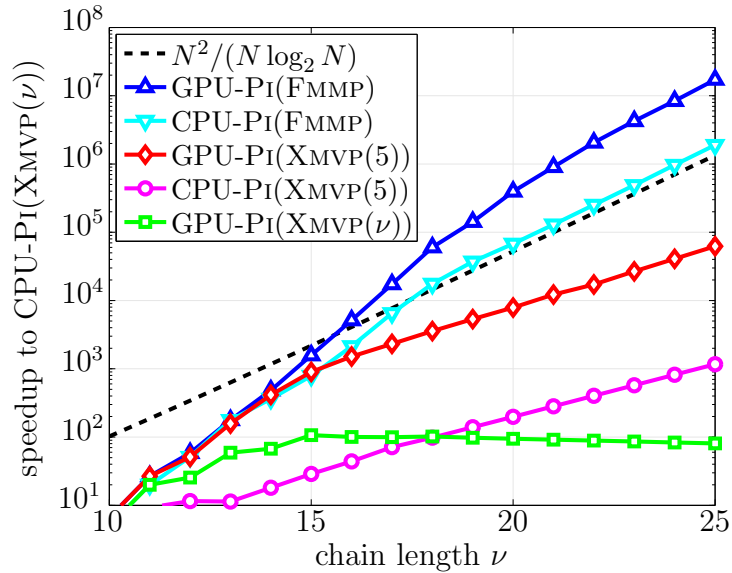


Figure 6.4.: Speedup factors for solving the quasispecies model based on various combinations of algorithms and hardware over the power iteration based on $\text{XMVP}(\nu)$ running on a single CPU core. For a given algorithm, different hardware platforms lead to (asymptotically) parallel speedup curves with the same slope, whereas the curves for different algorithms have different slopes.

Figure 6.4 constitutes a concise summary of the results obtained. For different algorithms on the same hardware we always observe a different slope of the speedup curve as we would expect from theory. This means that our algorithms scale well with the underlying hardware. We also see that the speedup delivered by FMMP over

$\mathcal{O}(N^2)$ algorithms is of the expected order of magnitude. The speedup values of FMMP on the CPU are partly above the theoretically expected value of $N^2/(N \log_2 N)$, which is caused by the computational overhead in XMVP(ν). For the same algorithms on different hardware we observe parallel shifted speedup curves and great performance benefit from exploiting the potential of GPUs. One should keep in mind that the reference computation and the fastest combination, PI(FMMP) on the GPU, deliver the same results (up to the limitations of floating-point arithmetic). The observed speedup factor of about $2 \cdot 10^7$ for $\nu = 25$ underlines the combined benefits of algorithmic improvements and efficient implementation which exploits the potential of GPUs.

6.4. More General vs. More Structured Models

Throughout the previous sections, we always considered a mutation matrix Q given by the uniform error model (6.2) and an unconstrained fitness landscape F (i.e., the matrix F is diagonal and positive, but otherwise arbitrary). With respect to these assumption the contributions depicted in this section are twofold: (i) we generalize the computationally tractable mutation models without sacrificing performance, and (ii) we investigate how additional structure in the fitness landscape can be utilized to reduce the overall complexity.

6.4.1. Hamming Distance-based Matrices

[Niederbrucker, 2013, § 4.1]

In the following we give a coarse overview of the theory of Hamming distance-based matrices which was recently developed by Niederbrucker [2013, § 4]. Subsequently, we exemplify in Section 6.4.2 how these results can be utilized in order to enable the efficient treatment of mutation models beyond the simple uniform error rate model.

Definition 6.1. Let $A_1, A_2, \dots, A_n$ be non-empty sets and $x, y \in A_1 \times A_2 \times \dots \times A_n$, i.e., $x = (x_1, x_2, \dots, x_n)$ and $y = (y_1, y_2, \dots, y_n)$ with $x_i, y_i \in A_i$. Then, we define by

$$d_H(x, y) = \sum_{i=1}^n \delta(x_i, y_i), \quad \delta(u, v) = \begin{cases} 1 & u = v \\ 0 & u \neq v \end{cases}$$

the *Hamming distance* on the set $A_1 \times A_2 \times \dots \times A_n$. //

Note that the Hamming distance $d_H(\cdot, \cdot)$ is a metric on $A_1 \times A_2 \times \dots \times A_n$ [cf. Hamming, 1950, § 5]. This follows directly from the definition, since the Hamming distance is an extension of the *discrete metric* ($\delta(\cdot, \cdot)$ in Definition 6.1) to multi-dimensional sets. If A is an arbitrary finite set with $|A|$ elements, we can always identify A with the set $\mathbb{Z}_{|A|} = \{0, 1, \dots, |A| - 1\}$ by a natural bijection between A and $\mathbb{Z}_{|A|}$. Therefore, we consider henceforth without loss of generality only signatures of the form $\Omega = \mathbb{Z}_{b_1} \times \mathbb{Z}_{b_2} \times \dots \times \mathbb{Z}_{b_n}$ with $b_i \in \mathbb{N}$ and call $\dim(\Omega) = n$ their respective dimension. It is well-known how to convert numbers from an arbitrary mixed radix

numeral system to integers. Concretely, for $\Omega = \mathbb{Z}_{b_1} \times \mathbb{Z}_{b_2} \times \cdots \times \mathbb{Z}_{b_n}$ this is done by the mapping.

$$\iota_\Omega: \Omega \rightarrow \mathbb{Z}_{|\Omega|}, (z_1, z_2, \dots, z_n) \mapsto z_n + \sum_{i=1}^{n-1} \left(z_{n-i} \cdot \prod_{j=n-i+1}^n b_j \right)$$

Let A be a matrix of dimension n and let $n = b_1 \cdot b_2 \cdot \dots \cdot b_k$ be a factorization of n . Then, since ι_Ω is a bijection, we can interpret by $\iota^{-1}(i-1)$ each row and column index of A as a string over $\mathbb{Z}_{b_1} \times \mathbb{Z}_{b_2} \times \cdots \times \mathbb{Z}_{b_n}$. Accordingly, we use in the following for a fixed signature Ω and indices $1 \leq i, j \leq |\Omega|$, the abbreviation $d_H(i, j) := d_H(\iota_\Omega^{-1}(i-1), \iota_\Omega^{-1}(j-1))$. Thus, we interpret $d_H(\cdot, \cdot)$ as a function $d_H: \{1, 2, \dots, |\Omega|\}^2 \rightarrow \mathbb{Z}_{\dim(\Omega)+1}$ instead of a function $d_H: \Omega \times \Omega \rightarrow \mathbb{Z}_{\dim(\Omega)+1}$.

Definition 6.2 (Hamming distance-based matrices). For an arbitrary finite signature Ω , we define the set of *Hamming distance-based* (d_H -based) matrices over Ω by

$$\mathcal{H}(\Omega) := \left\{ H \in \mathbb{R}^{|\Omega| \times |\Omega|} \mid \exists \varphi: \mathbb{Z}_{\dim(\Omega)+1} \rightarrow \mathbb{R} \text{ with } h_{i,j} = \varphi(d_H(i, j)) \right\}.$$

Therefore, $\mathcal{H}(\Omega)$ contains all matrices whose (i, j) -th entry solely depends on the Hamming distance between i and j , i.e., it solely depends on the Hamming distance between the strings $\iota_\Omega^{-1}(i-1)$ and $\iota_\Omega^{-1}(j-1)$. For a matrix $H \in \mathcal{H}(\Omega)$ we call the function φ which determines the entries of H its *associated function*. Furthermore, we define $\mathcal{H}(\emptyset) := \mathbb{R}$. //

Example 6.1. Consider arbitrary matrices $H \in \mathcal{H}(\mathbb{Z}_2)$ and $G \in \mathcal{H}(\mathbb{Z}_2^2)$. Then we observe the following generic structure in the matrices H and G :

$$H = \begin{pmatrix} a & b \\ b & a \end{pmatrix} \rightsquigarrow \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & a & b \\ 1 & b & a \end{array}, \quad G = \begin{pmatrix} c & d & d & e \\ d & c & e & d \\ d & e & c & d \\ e & d & d & c \end{pmatrix} \rightsquigarrow \begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ \hline 00 & c & d & d & e \\ 01 & d & c & e & d \\ 10 & d & e & c & d \\ 11 & e & d & d & c \end{array}.$$

Moreover, we can observe how the ordering of the integers determines the rich structure of the matrices H and G . //

Definition 6.3 (Hamming distance-based vector). For an arbitrary finite signature Ω , we say that a vector $x \in \mathbb{R}^{|\Omega|}$ is d_H -based, if there exists a matrix $H \in \mathcal{H}(\Omega)$ such that x is the first column of H , i.e., $x = H(:, 1)$. In particular, it holds that $x_i = \varphi(d_H(i, 1))$ where φ denotes the associated function of H . //

First, we state an auxiliary result which we previously referred to. It deals with the number of elements with a fixed Hamming distance in d_H -based vectors, for the case of signatures $\Omega = \mathbb{Z}_c^n$.

Lemma 6.2 (see [Niederbrucker \[2013, Lemma 4.1\]](#)). *Consider natural numbers $c, n \in \mathbb{N}$ with $c > 1$ and let $x \in \mathbb{R}^{|\Omega|}$ be a d_H -based vector over $\Omega = \mathbb{Z}_c^n$ with associated function φ_x . Then, for $0 \leq k \leq c$, the value $\varphi_x(k)$ appears $\binom{n}{k}(c-1)^k$ times in x . In particular, this implies that $|\{i \mid 1 \leq i \leq c^n \wedge d_H(i, 1) = k\}| = \binom{n}{k}(c-1)^k$.*

At first glance, it seems difficult to analytically cope with d_H -based matrices, due to their definition via the Hamming distance. It turns out that the opposite holds, since the class of d_H -based matrices can be well analyzed in-depth. In particular, the existing theory of d_H -based matrices [[Niederbrucker, 2013, § 4](#)] is comprehensive in the sense that it considers the treatment of these matrices from a theoretical as well as from a computational (practical) point of view. Concretely, it contains a complete characterization of the structure of these matrices and generically shows, e. g., how the eigenvalues and the elements of d_H -based matrices are related to each other. Most importantly, there exists a fast (subquadratic) matrix vector product with d_H -based matrices with complexity $\Theta(|\Omega| \dim(\Omega))$ [[Niederbrucker, 2013, Theorem 4.30](#)]. Besides that, the algebraic structure of the set of all d_H -based matrices as well as the analytical solution of several approximation problems involving d_H -based matrices has been investigated [[Niederbrucker, 2013, § 4](#)].

6.4.2. More General Mutation Processes

The assumption of a uniform error rate is a well known shortcoming of existing studies of the quasispecies model [[Schuster, 2011](#)]. With respect to this assumption, mutations in different positions of a sequence are entirely independent and they happen with the same probability p , which leads only to a coarse approximation of realistic mutation processes. In contrast to that, the ODE formulation of the model (6.1) requires only that Q is column stochastic [[Schuster, 2008](#)]. In the following, we discuss how to apply our fast algorithms in a more general setting, with more realistic assumptions on the structure of Q .

Hamming Distance-Based Mutation Models

[[Niederbrucker, 2013, § 5.1](#)]

There is a huge gap between the requirements of the ODE model (6.1), namely that Q is column stochastic, and the special case that Q is given by the uniform error model (6.2). As a substantial generalization, we can assume that the mutation matrix Q is an arbitrary d_H -based matrix, which leads to a fairly general modeling of the mutation process. The available theory of d_H -based matrices depicts that d_H -based matrices do (despite their generality) not cause an increase of the computational complexity (compared to Kronecker product representations of Q) [[Niederbrucker, 2013](#)]. In the following example we consider an exemplary mutation model beyond the uniform error rate model.

Example 6.2. We consider two potential mutation matrices Q_1 and Q_2 for a chain length of $\nu = 2$. The matrix Q_1 is based on the uniform error rate model with $p = 0.01$

and $Q_2 \in \mathcal{H}(\mathbb{Z}_2^2)$ is a particular example of a d_H -based matrix which does not allow for a simple Kronecker product representation.

$$Q_1 = \begin{pmatrix} 0.81 & 0.09 & 0.09 & 0.01 \\ 0.09 & 0.81 & 0.01 & 0.09 \\ 0.09 & 0.01 & 0.81 & 0.09 \\ 0.01 & 0.09 & 0.09 & 0.81 \end{pmatrix} = \bigotimes_{i=1}^2 \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$$

$$Q_2 = \begin{pmatrix} 0.90 & 0.04 & 0.04 & 0.01 \\ 0.04 & 0.90 & 0.01 & 0.04 \\ 0.04 & 0.01 & 0.90 & 0.04 \\ 0.01 & 0.04 & 0.04 & 0.90 \end{pmatrix} \neq \bigotimes_{i=1}^2 \begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix}$$

Another example of a mutation model which can be expressed in terms of d_H -based matrices is to choose the mutation error rate exponentially decaying, i.e., choosing the associated function of Q as $\varphi_Q(d) = e^{-d}/c$ where c is a normalization constant. //

It can be easily seen from the structure of d_H -based matrices that they have constant column sums. This means any Hamming distance-based matrix, i.e., any model function of the form $\varphi(d_H(i, j))$, can be used within the quasispecies model (up to normalization). Hence, it is useful to approximate a given (realistic) mutation matrix with a d_H -based matrix.

Example 6.3. Assume we are given an arbitrary column stochastic mutation matrix Q obtained, e.g., by practical observations or expert knowledge. Then we can efficiently find the d_H -based matrix $\tilde{Q}$ which approximates Q best, i.e., which minimizes $\|\tilde{Q} - Q\|_F$ [Niederbrucker, 2013, Theorem 4.16]. The utilization of the approximation $\tilde{Q}$ enables the investigation of (an approximation) of Q within the quasispecies model.

$$Q = \begin{pmatrix} 0.460 & 0.210 & 0.263 & 0.112 \\ 0.214 & 0.362 & 0.082 & 0.241 \\ 0.256 & 0.159 & 0.416 & 0.303 \\ 0.070 & 0.270 & 0.240 & 0.344 \end{pmatrix}, \quad \tilde{Q} = \begin{pmatrix} 0.3955 & 0.2496 & 0.2496 & 0.1058 \\ 0.2496 & 0.3955 & 0.1058 & 0.2496 \\ 0.2496 & 0.1058 & 0.3955 & 0.2496 \\ 0.1058 & 0.2496 & 0.2496 & 0.3955 \end{pmatrix}$$

Note that the best-approximating d_H -based matrix of an arbitrary column stochastic matrix is by definition also a column stochastic matrix [Niederbrucker, 2013, Theorem 4.16]. //

Another aspect which the known theory of d_H -based matrices allows to generalize is that we can also consider larger alphabets, e.g., the usual four letter RNA alphabet [Schuster, 2011], besides binary alphabets. Even more generally, we can consider specifically constrained sequences where different elements of the sequences are taken from different alphabets. This might be of interest whenever additional a priori knowledge is available for certain positions in the RNA sequence.

Summarizing, we can cover substantially more general model scenarios by applying d_H -based matrices for relatively low costs compared to the more specific existing methods. This holds in terms of the mutation model as well as in terms of the manageable alphabets.

Kronecker Product-based Mutation Models [Niederbrucker and Gansterer, 2011a, § 5]

In the Kronecker product representation of Q in (6.7) there is actually no need for the single point mutations to have the same properties. Moreover, the ODE model (6.1) by itself solely requires that Q is column stochastic. As it is easily verified, the Kronecker product of two column stochastic matrices is column stochastic as well. Consequently, for arbitrary diagonal fitness landscapes F , a fast matrix vector product with $W = QF$ can be constructed as long as the mutation process is modeled as the composition of ν independent processes, one for each position in the RNA sequence. For such generalizations, Q may lose some properties such as symmetry, but this does not affect the performance of our approach since it just relies on the Kronecker product based definition of Q . Even the restriction to 2×2 column stochastic matrices can be relaxed when needed for modeling more general situations.

In principle, our approach allows for efficiently handling mutation matrices Q of the more general structure

$$Q = \bigotimes_{i=1}^g Q_{G_i}, \quad Q_{G_i} \in \mathbb{R}^{2^{g_i} \times 2^{g_i}}, \quad \sum_{i=1}^g g_i = \nu, \quad (6.13)$$

which models $1 \leq g \leq \nu$ groups of independent mutation processes, where each group contains g_i single point mutations, which are dependent on each other. Again, the single requirement is that the matrices Q_{G_i} are column stochastic such that the model is valid. Clearly, the integers g_i influence the complexity of the resulting algorithms, but as long as the maximal g_i is not too large, we still get efficient methods also for this more general setting. More specifically, this approach is the more efficient the smaller dimension the Q_{G_i} have.

6.4.3. Utilizing Structured Fitness Landscapes

[Niederbrucker and Gansterer, 2011a, § 5]

The methods presented so far are designed for the most general formulation in terms of fitness landscapes F without any assumptions beyond diagonality. We already made the observation that the mutation matrix Q may be defined as a d_H -based matrix or as a Kronecker product. In case the fitness landscape also has such a representation, one can expect that the matrices Q and F in some sense “fit together” and that their structure translates to the product $W = QF$. This additional structure of the matrix W can subsequently be used to derive further enhancements to the methods presented so far. In fact, the structure of F does not only translate into the matrix W , it even translates into the dominating eigenvector, which emphasizes that such structure should also translate into our algorithms and their complexity, respectively. The resulting structure in the dominating eigenvector reflects the loss of generality in the results since the usually N degrees of freedom in the fitness landscape F are drastically reduced by further assumptions on the structure of F . How well such

specially structured landscapes agree with observations in nature has to be clarified with application experts.

Hamming Distance-Based Landscapes

The existing quasispecies literature deals almost exclusively with d_H -based landscapes, due to the unavailability of efficient solvers for general landscapes and especially because of the availability of good approximation methods for d_H -based landscapes [Swetina and Schuster, 1982; Nowak and Schuster, 1989]. In this section we show that given a fitness landscape F , where $\text{diag}(F)$ is a d_H -based vector, it is sufficient to solve a $(\nu + 1) \times (\nu + 1)$ eigenproblem to get the *exact* eigenvector of the full $N \times N$ eigenproblem. Therefore, approximation methods are not really needed in this case. One should note that this dimensionality reduction also occurs in the extremal eigenvector which corresponds to a reduction from N to $\nu + 1$ degrees of freedom in the structure of the eigenvector.

The usual way to treat d_H -based landscapes (see, e. g., Schuster [2008]) is by defining a reduced mutation matrix Q . This reduction is based on the observation that mutations do not longer have to be considered on the granularity of single molecules, but instead, as mutations from some fixed element of an error class Γ_d to any element of another error class Γ_k . For $0 \leq d, k \leq \nu$ the reduced mutation matrix is defined as

$$[Q_\Gamma]_{d,k} := \sum_{j=k+d-\nu}^{\min(k,d)} \binom{\nu-d}{k-j} \binom{d}{j} \frac{p^{k+d-2j}}{(1-p)^{(k+d-2j)-\nu}}, \quad (6.14)$$

which can be derived by explicitly computing the probability that a fixed molecule from error class Γ_d mutates into a molecule from error class Γ_k . Based on this definition, approximation methods for computing the quasispecies are available [Swetina and Schuster, 1982; Nowak and Schuster, 1989].

In order to develop a better understanding of this problem reduction, we present a more computationally oriented perspective. We start by considering the power iteration for the full $N \times N$ problem as we did so far for the general case, and then we simplify it step by step. At the end, we basically obtain the same problem reduction but with the advantage of a clear interpretation and understanding of the structures arising. This knowledge can then be used to efficiently compute the desired eigenvector. The following lemma shows that in case both Q and (the diagonal of) F are d_H -based, their product maps d_H -based vectors to d_H -based vectors.

Lemma 6.3 (Niederbrucker [2013, Lemma 4.21]). *Let $c, n \in \mathbb{N}$ be arbitrary, $\Omega = \mathbb{Z}_c^n$, $H \in \mathcal{H}(\Omega)$ a d_H -based matrix and $x \in \mathbb{R}^{|\Omega|}$ a d_H -based vector (with respect to Ω). Then $y = Hx$ is also a d_H -based vector.*

Using Lemma 6.3 we can conclude that if $\text{diag}(F)$ is a d_H -based vector, then the dominating eigenvector of $W = QF$ is a d_H -based vector as well. This follows by using an arbitrary positive d_H -based vector as starting vector for the power iteration, since

(i) the product of F with a d_H -based vector is (as an elementwise product of d_H -based vectors) a d_H -based vector, and (ii) the product of a d_H -based vector with Q is by [Lemma 6.3](#) a d_H -based vector. We also know now that all intermediate vectors in the power iteration are d_H -based vectors. Consequently, we should not compute the resulting value for all vector components during an iteration, but instead of that, it is more efficient to compute the resulting value only for an arbitrary representative index from each error class. The natural and most obvious choices of such representatives is given by the set $\{2^k \mid 0 \leq k \leq \nu\}$.

So far, our insights reduced the original $N \times N$ problem

$$\bar{v}_i = \sum_{j=0}^{N-1} q_{i,j} \cdot F_{j,j} \cdot v_j$$

to a $(\nu + 1) \times N$ problem

$$\bar{v}_{\Gamma_d} = \sum_{k=0}^{\nu} F_{\Gamma_k} \cdot v_{\Gamma_k} \cdot \sum_{j \in \Gamma_k} q_{2^d j}.$$

We defined $Q_{\Gamma_d,k}$ in (6.14) as the probability that a fixed element from Γ_d mutates into an element of Γ_k . Observe that this is exactly what the last sum in the previous equation computes and therefore we obtain the $(\nu + 1) \times (\nu + 1)$ problem

$$\bar{v}_{\Gamma_d} = \sum_{k=0}^{\nu} Q_{\Gamma_d,k} \cdot F_{\Gamma_k} \cdot v_{\Gamma_k}.$$

Thus, efficiently computing the eigenvector of the original problem by the power iteration shows that the desired eigenvector can be computed out of the eigenvector of the reduced $(\nu + 1) \times (\nu + 1)$ matrix. But note that when v denotes the original and v_{Γ} the eigenvector of the reduced problem we do not get $[\Gamma_k] = v_{\Gamma_k} = \sum_{j \in \Gamma_k} v_j$, because the reduced matrix itself does not correspond to a change of variables from single molecules to error classes. As we have seen in the course of this section, it corresponds to a change from single molecules to representatives of classes. In summary, this means that given v_{Γ} , the dominating eigenvector of the reduced problem, we can compute the cumulative concentrations for the original problem by

$$[\Gamma_k] = \binom{\nu}{k} \frac{v_{\Gamma_k}}{\sum_{j=0}^{\nu} \binom{\nu}{j} v_{\Gamma_j}}.$$

This rescaling emphasizes that the eigenvector of the reduced problem contains the concentrations of some representatives of the error classes Γ_k .

Besides its low computational complexity, the main advantage of this exact problem reduction is that we do not have to worry about approximation aspects or about the details of the perturbation theory used. It is sufficient to set up the needed matrices by the well known formulas and to use a standard solver for computing the extremal eigenvector of a small $(\nu + 1) \times (\nu + 1)$ matrix. However, this low computational complexity comes for the price of very simple (unrealistic) model assumptions.

Kronecker Product-Based Landscapes

In [Section 6.2.2](#) we considered using a Kronecker product-based definition of the mutation matrix Q which also resulted in a significant increase in the generality of Q without causing higher computational cost. Analogously, if we have a Kronecker product representation of the fitness landscape F , we can derive more efficient solution methods than the ones for arbitrary landscapes.

The Kronecker product representation of Q in [Section 6.2.2](#) resulted in a great reduction of the complexity in the matrix vector product with Q . The product of Q with a diagonal matrix F is inherently fast, but a Kronecker product representation of F has additional benefits: on the one hand, it leads to a reduction of the memory requirements, on the other hand, it allows for splitting up the problem into independent subproblems which can be solved in parallel.

Following the same scheme as for Q in [\(6.13\)](#), we define a Kronecker product-based landscape F via the Kronecker product of matrices F_{G_i} of dimension $2^{g_i} \times 2^{g_i}$ by

$$F = \bigotimes_{i=1}^g F_{G_i}, \quad F_{G_i} \in \mathbb{R}^{2^{g_i} \times 2^{g_i}}, \quad \sum_{i=1}^g g_i = \nu. \quad (6.15)$$

The big advantage of a Kronecker product representation is that it decouples the problem entirely into independent subproblems, which leads to an implicit description of the solution of the entire problem. In terms of complexity this means that the usual multiplicative connection becomes an additive one. More concretely, if all integers g_i are equal, we can reduce the problem for chain length ν to g subproblems of size $2^{\nu/g}$ and solve them independently, instead of solving one problem of size 2^ν . For example, the quasispecies model for chain length $\nu = 100$ (which occurs in existing viruses of interest) is by far out of reach of any of the currently available computational technology. However, for a Kronecker fitness landscape with $g = 4$ it could be reduced to four subproblems of dimension 2^{25} , which can be solved fast with the methods introduced in this chapter.

We also observe that Kronecker product-based landscapes have a much richer structure than the ones based on Hamming distances, since they have $\sum_{i=1}^g 2^{g_i}$ degrees of freedom compared to only $\nu + 1$ in the case of Hamming distance based landscapes.

If both Q and F have a Kronecker product representation and if these representations are also compatible (the simplest case would be that Q and F have the same g_i in [\(6.13\)](#) and [\(6.15\)](#)), then it is not even needed to compute the entire eigenvector. By results from Kronecker product theory, we get an implicit description of the desired eigenvector, since the eigenvalues and eigenvectors of a matrix $M = \bigotimes_{i=1}^g M_i$ can be computed out of the eigenvalues and eigenvectors of the component matrices M_i (cf. [Theorem A.8](#)).

Up to which extent the matrices Q and F “fit together” is limited by the applicability of the so-called mixed product formula (cf. [\(A.1\)](#)). This is exactly the point where we can see a possible increase in the joint group sizes of Q and F when considering

$W = QF$. But since our new solvers easily handle chain lengths of $\nu \approx 25$ within a few seconds, one can still expect efficient computations for fairly complex structures.

Based on an implicit description of the eigenvalues and eigenvectors of the component matrices one could also develop algorithms for retrieving (partial) information about the desired eigenvector of the full matrix W . More specifically, it is quite simple to compute the extremal eigenvalue and relevant parts of its corresponding eigenvector efficiently. For example, one could compute the minimum and maximum concentration for each error class which should provide sufficient information for investigating the question whether the error threshold phenomenon occurs or not. So far, we did not investigate such methods in great detail because the biological relevance of Kronecker product-based fitness landscapes is not yet clear. From a computational point of view they obviously have many interesting properties and together with efficient methods for extracting information out of the implicit eigenvector description it would be possible to model chain lengths far beyond what is possible today, easily including complex viruses with long RNA chains. Moreover, for Kronecker product-based landscapes it is relatively easy to extend the quasispecies model beyond a binary alphabet to the full four element RNA alphabet.

Lessons Learned: Quasispecies Model Meets HPC

Throughout this chapter we considered—as an HPC case study—the problem of solving the extreme scale eigenvalue problem occurring in the quasispecies model. More concretely, the objective was to compute the dominating eigenvector of extremely large dense but structured matrices. The treatment of the intended huge problem dimensions was only possible because the involved matrices are “data-sparse”. We depicted a tandem of fast algorithms and their respective implementations on modern (parallel) platforms which enables, among others, a substantial increase of the computationally tractable problem dimension (compared to existing work).

A crucial step in our considerations was the development of a deep understanding of the structure inherent in the eigenvalue problem stemming from the quasispecies model. In order to exploit the special structure and properties of the eigenvalue problem, we developed two fast matrix vector products $\text{XMVP}(d_H^{\max})$ [Niederbrucker and Gansterer, 2011b] and FMMP [Niederbrucker and Gansterer, 2011a], where the latter allows for a reduction of the arithmetic complexity from $\Theta(N^2)$ to $\Theta(N \log_2 N)$. In contrast to the sparsification-based approach based on $\text{XMVP}(d_H^{\max})$ and earlier approximation approaches [Swetina and Schuster, 1982; Nowak and Schuster, 1989], our solver based on FMMP produces results as accurate as possible within the limitations of floating-point arithmetic. With an efficient parallel implementation of a power iteration approach based on FMMP and by utilizing modern GPU hardware we achieved enormous speedups up to $2 \cdot 10^7$ (for a chain length $\nu = 25$) over equivalent standard solvers.

We also showed how to efficiently handle mutation models beyond the ones considered in the existing quasispecies literature by: (i) considering Kronecker product representations with heterogeneous factors, and (ii) by employing the rich theory of Hamming distance-based matrices [Niederbrucker, 2013, § 5]. Moreover, beyond the improvements for general fitness landscapes in the quasispecies model, we also outlined how to reduce the computational cost even further for structured fitness landscapes, leading to further substantial reductions of the space and runtime complexity as well as to an even higher parallelization potential.

Given the new solvers presented in this chapter, the main limiting factor in computationally solving the quasispecies model is not any more (only) the runtime, but (rather) the space complexity.

7 | Summary and Conclusions

Throughout this thesis, we strictly distinguished between the notions of parallel and distributed computing. In the course of our differentiation between parallel and distributed systems we pointed out substantial differences in terms of the topology, communication capabilities, global control mechanisms and local knowledge of the individual nodes (cf. [Table 2.1](#)). While distributed algorithms can be executed on parallel systems, the converse does not hold, because parallel algorithms impose many requirements on the environment which distributed systems do not fulfill (according to our definition). In particular, computations which can be easily performed on a parallel system are in most cases hard (or even impossible) to perform on a distributed system.

Our high level motivation was to provide numerical linear algebra functionality under conditions where existing approaches cannot be applied or fail. We saw that such distributed matrix computations require robust and flexible all-to-all aggregation routines as a building block. Hence, in this thesis we focused on the development and analysis of powerful variants of these fundamental low-level operations. For a detailed elaboration on how the methods we developed can be employed as building block for distributed matrix computations see [Straková \[2013\]](#) and references therein. In particular, the results of [Straková \[2013\]](#) illustrate that the (beneficial) properties of the methods we developed in this thesis are carried over to higher level matrix computations, e.g., to a distributed QR factorization and an orthogonal iteration-based eigensolver.

In our considerations we constantly aimed at taking theoretical as well as practical points of view into account. In doing so, we did not only reveal several issues concerning the practical applicability of existing methods and their analyses, but also developed results which partially closed the identified gaps between theory and practice. In particular, we based our theoretical investigations solely on assumptions which we believe to be applicable in real distributed systems. While these assumptions are (technically) less convenient than commonly employed assumptions, we could still provide all required (convergence) results. However, due to their generality, our results are mostly of an existential nature and do not provide tight bounds.

Summarizing, the results in this thesis cover a wide range of aspects ranging from theory to practice. Concretely, we introduced generic techniques for deriving provably robust (distributed) aggregation algorithms which can cope with the challenges imposed by real distributed systems.

7.1. Contributions

It is important to point out that all our contributions concerning distributed aggregation algorithms (Chapters 2–5) are of a generic nature. In particular, we never derived results for specific algorithms or environments (topology, etc.) and all our results can be applied to a large class of distributed aggregation algorithms which covers many well-known methods. Algorithms belonging to this class can be modeled in terms of a linear iteration $x(t) = C(t)x(t-1)$, where the matrices $C(t)$ are drawn independently from a fixed distribution $\mathcal{C}$. We have seen that algorithms which can cope with realistic environments require the usage of so-called sum-weight schemes

$$s(t) = C(t)s(t-1), \quad w(t) = C(t)w(t-1),$$

where for all nodes i the ratio $x(t)_i = s(t)_i/w(t)_i$ converges to the target aggregate α .

The properties of the distribution $\mathcal{C}$ are determined by the respective algorithm and assumptions on the concrete environment. In [Section 3.3](#) saw that properties which are convenient for a theoretical analysis often severely restrict the practical applicability of the respective distributed method. Therefore, we aimed at deriving a generic convergence theory for sum-weight schemes which requires as little assumptions as possible. Accordingly, in [Section 3.4](#) we substantially generalized the generic convergence theory of [Boyd et al. \[2006\]](#) and eliminated all (from a practical point of view) problematic assumptions made therein. Moreover, in contrast to the closely related work of [Benezit, Dimakis, et al. \[2010\]](#), we allow negative elements in the matrices $C(t)$ and we do not impose assumptions on their diagonals. These generalizations are not only interesting from a theoretical point of view, but primarily they are fundamental for our analyses of fault tolerance mechanisms (cf. [Section 4.2](#)). We have also seen that—at its heart—the analysis of linear iterations boils down to the power iteration, which enables the utilization of well-established convergence acceleration techniques (cf. [Section 3.5](#)). Several of these methods cause only additional local operations and therefore, they can also be directly applied in practice.

In [Chapter 4](#) we presented our core results regarding fault tolerance mechanisms for distributed aggregation algorithms. In order to guarantee convergence to the target aggregate α , it is necessary to preserve (globally) the so-called *mass conservation* property over time, i. e., $\sum_{i \in \mathcal{V}} (s(t)_i, w(t)_i) = \sum_{i \in \mathcal{V}} (s(0)_i, w(0)_i)$. The difficulty in preserving mass conservation over time originates from the fact that it is a global constraint, whereas (in a distributed context) only local mechanisms can be used. Our approach for coping with this problem was inspired by the flow concept from graph theory. We proposed that every node i keeps for each of its neighbors $j \in \mathcal{N}_i$ a flow variable $f(t)_{i,j}$ which describes the overall (i. e., until time t) balance of data communicated between the nodes i and j . We showed that by an appropriate representation of the local data the nodes can communicate solely their local flow variables $f(t)_{i,j}$. When receiving data, it is crucial to always ensure (local) skew-symmetry by setting $f(t)_{j,i} = -f(t)_{i,j}$, which yields the desired mass conservation. Hence, it is (indirectly)

also responsible for the ability to recover from failures. However, in this way mass conservation is (in general) not strictly preserved globally, because it depends on all local recoveries which (in general) do not take place at the same point of time. Therefore, it is a priori by no means clear that this technique succeeds at providing fault tolerance at the algorithmic level. We rigorously proved the correctness of our flow-based method in the presence of failures by introducing a model which represents the execution of flow-based algorithms in the presence of failures in a (theoretical) failure-free environment. Analogously, we applied this analytical approach also to another fault tolerance mechanism [Eyal *et al.*, 2011], which has not been fully analyzed before. Moreover, we also derived first bounds for the overhead (arbitrary high) message loss cause when using either of the aforementioned fault tolerance techniques. Last but not least, our empirical investigations revealed several subtleties of (existing) fault tolerant distributed aggregation algorithms. For flow-based algorithms, we are able to overcome most of these by introducing a refined variant of the flow concept.

In an algorithm from the class of distributed aggregation algorithms which we investigated in this thesis, each node computes locally a sequence (x_m) converging to the target aggregate α . However, the problem of convergence detection (for the class of algorithms we consider) is almost entirely neglected in the existing literature, i. e., it is common practice to assume that a computation aborts as soon as the local error $\varepsilon = \|x_i - \alpha\|/\|\alpha\|$ reaches a predefined threshold. In Chapter 5, we developed methods for providing locally at each node an estimate $\tilde{\varepsilon}$ of ε , which can be used to detect convergence in practice. Since we assumed a distributed environment where the nodes are limited to their local knowledge, it appears impossible to provide generic methods which provably provide always correct estimates (of high quality). However, we have seen that theoretically well-founded heuristics can achieve good results in practice.

In Chapter 6 we elaborated (in addition to the main thread of this thesis) on solvers for the extreme scale eigenvalue problem occurring in the quasispecies model. More concretely, our objective was to compute the dominating eigenvector of extremely large dense but structured matrices whose dimension N grows exponentially with the practically important model parameter. In order to exploit the specific problem structure, we developed two fast structure-aware matrix vector products $\text{XMVP}(d_H^{\max})$ and FMMP , where the latter allows for a reduction of the arithmetic complexity from $\Theta(N^2)$ to $\Theta(N \log_2 N)$. In contrast to the sparsification-based approach based on $\text{XMVP}(d_H^{\max})$ and existing approximation approaches, our solver utilizing FMMP produces fully accurate results (within the limitations of floating-point arithmetic). Besides the development of structure-aware algorithms we also considered their (parallel) implementation on modern hardware. Furthermore, we investigated whether the currently employed model assumptions can be generalized without sacrificing performance. In particular, we introduced efficient methods for mutation models beyond the ones considered in the existing quasispecies literature. Moreover, we also outlined how to reduce the computational cost even further for structured fitness landscapes. Altogether, our results yield a substantial raise of the computationally tractable problem dimensions and a generalization of the applicable model assumptions.

7.2. Open Problems and Future Work

In this thesis, we covered many topics and introduced several new concepts. Hence, there are also many interesting questions left open for future work. In the following, we highlight those we believe to be the most important.

First of all, it is somewhat unpleasant that our convergence results are based on indirect proofs and thus, we gained only relatively little insights into the methods covered by these convergence results. In particular, we gave indirect proofs in the analysis of our fault tolerance techniques which did not reveal background information about how these techniques work and which overhead potential failures cause. Even though we obtained some bounds concerning the overhead of message loss, a more precise analysis is highly desirable. However, such results most likely require direct convergence proofs and a deep understanding of the underlying processes.

Besides the analysis of the overhead caused by failures, it would also be of high interest to extend the range of failures which can be handled (efficiently) at the algorithmic level. This desire especially includes various kinds of permanent failures such as (silently) crashing nodes.

In [Chapter 5](#) we discussed the problem of convergence detection within distributed aggregation algorithms and introduced reasonably efficient heuristics for this task. In this context, it is an interesting question whether one can find theoretical models where provably correct and efficient convergence detection (accuracy estimation) algorithms can be developed. Moreover, it is also a highly non-trivial issue how to safely terminate entire distributed algorithms (based on local decisions to terminate) in practice.

The algorithms we considered in thesis are often proposed to be suitable for wireless sensor networks. Due to the wireless communication in such systems, *broadcasting* is the natural (and most efficient) way of communication. This means in case a node sends a message, it does not determine a particular receiver, but all nodes within its communication range are intended to receive the message. For failure-free environments many broadcasting-based methods are known. However, especially in wireless sensor networks messages (broadcasts) often do not reach all potential receivers (e.g., due to interference). The problem at this point with our fault tolerance techniques is that they heavily rely on point-to-point communication between pairs of nodes. Hence, it would be very interesting to find similarly-suited efficient fault tolerance techniques which can naturally¹ cope with a broadcasting environment. Moreover, it is questionable whether the assumption that the nodes know about their neighborhoods, which is commonly made in the respective existing literature, is applicable in (potentially dynamic) wireless networks. If this assumption is dropped, only methods based on computing extrema can be used, which converge impracticably slow (cf. [Section 3.1](#)). Therefore, there is a high demand for robust and efficient broadcasting-based aggregation methods.

¹Our flow-based methods can be applied in a broadcasting environment, but at the cost of large messages, since each message would have to contain the flows for all neighbors of its sender.

A | Mathematical Background

A.1. Notation and Fundamentals

A.1.1. Linear Algebra

In the following we explicitly state notations, definitions and facts from linear algebra which we employ throughout the thesis (cf. [Golub and Van Loan \[1996, §§ 1.1, 7.1\]](#)).

Notation A.1 (Matrices). For a matrix $A \in \mathbb{R}^{m \times n}$ we denote its entries by $a_{i,j}$, i. e.,

$$A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix}, \quad a_{i,j} \in \mathbb{R}.$$

Moreover, we also use $[A]_{i,j} = a_{i,j}$ to index the (i, j) -th entry of A . By $\text{tr}(A)$, $\text{rank}(A)$ and $\det(A)$, we denote the *trace*, *rank* and *determinant* of A , respectively. Moreover, in case A is a *block matrix*, we denote by $A_{i,j}$ its blocks, i. e.,

$$A = \begin{pmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & \ddots & \vdots \\ A_{m,1} & \cdots & A_{m,n} \end{pmatrix}, \quad A_{i,j} \in \mathbb{R}^{m_i \times n_j}.$$

The transpose of a matrix $A \in \mathbb{R}^{m \times n}$ is denoted by $A^T \in \mathbb{R}^{n \times m}$. //

Notation A.2 (Diagonal Matrices). Let $A \in \mathbb{R}^{n \times n}$ be a square matrix, then $\text{diag}(A)$ denotes the vector formed by the entries on the main diagonal of A , i. e., $\text{diag}(A) = (a_{1,1}, a_{2,2}, \dots, a_{n,n})$. Conversely, if $v \in \mathbb{R}^n$ is a column or row vector, then

$$\text{diag}(v) = \text{diag}(v_1, v_2, \dots, v_n) = \begin{pmatrix} v_1 & 0 & \cdots & 0 \\ 0 & v_2 & 0 & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & v_n \end{pmatrix}.$$

Moreover, if $B_i \in \mathbb{R}^{m_i \times n_i}$ are matrices, then $\text{diag}(B_1, B_2, \dots, B_n)$ denotes the block diagonal matrix with the blocks B_i on its diagonal. By $I_n \in \mathbb{R}^{n \times n}$ we denote the identity matrix in $\mathbb{R}^{n \times n}$, i. e., $I_n = \text{diag}(1, 1, \dots, 1)$. //

Notation A.3 (Submatrix Specification). For a matrix $A \in \mathbb{R}^{m \times n}$ and index sets $u \subseteq \{1, \dots, m\}$ and $v \subseteq \{1, \dots, n\}$, we denote by $A(u, v)$ the submatrix of A formed by picking the rows u and columns v of A , e. g.,

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{pmatrix} \in \mathbb{R}^{4 \times 4}, \quad A(\{1, 4\}, \{2, 3\}) = \begin{pmatrix} a_{1,2} & a_{1,3} \\ a_{4,2} & a_{4,3} \end{pmatrix} \in \mathbb{R}^{2 \times 2}.$$

We further use the *colon notation* to abbreviate index sets with consecutive indices. Concretely, for integers i and j , we denote by $i:j$ the set $\{i, i+1, \dots, j-1, j\}$. More generally, $i:k:j$ denotes the set $\{i, i+k, \dots, j-k, j\}$. To select all rows or columns from a matrix A we use $A(:, v)$ and $A(u, :)$, respectively. //

Notation A.4 (Matrices vs. Vectors). Let A be an arbitrary $m \times n$ matrix, then the *vectorization* of A is the column vector obtained by concatenating the columns of A .

$$\text{vec}(A) = \begin{pmatrix} A(:, 1) \\ A(:, 2) \\ \vdots \\ A(:, n) \end{pmatrix}$$

The natural inverse to the vectorization of a matrix is to construct a matrix out of a vector. Given a vector $x \in \mathbb{R}^{rc}$ we will denote by $x_{r \times c}$ the matrix

$$x_{r \times c} = \begin{pmatrix} x_1 & x_{r+1} & \cdots & x_{(c-1)r+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_r & x_{2r} & \cdots & x_{cr} \end{pmatrix}$$

//

Definition A.1 (Spectrum, spectral radius). Let $A \in \mathbb{R}^{n \times n}$ be a matrix then we call $\sigma(A) = \{\lambda \in \mathbb{C} \mid \exists v \neq \mathbf{0} \text{ with } Av = \lambda v\}$ the *spectrum* of A . Moreover, we call $\rho(A) := \max_{\lambda \in \sigma(A)} |\lambda|$ the *spectral radius* of A . (This definition is sound since $|\sigma(A)| \leq n$.) //

Definition A.2 (Eigenvalues, eigenvectors). A scalar $\lambda \in \sigma(A)$ is called an *eigenvalue* of a matrix $A \in \mathbb{R}^{n \times n}$ and its associated vector $v \in \mathbb{R}^n$, satisfying $Av = \lambda v$, is called (*right*) *eigenvector*. Moreover, $\sigma(A) = \sigma(A^T)$ and for each $\lambda \in \sigma(A)$ there exists also a *left eigenvector* w associated with it, which satisfies $w^T A = \lambda w^T$. Consequently, in case of symmetric matrices, left and right eigenvectors coincide. //

By definition, a scalar multiple of an eigenvector v associated with eigenvalue λ is an eigenvector associated with λ as well, i. e., eigenvectors are only unique up to normalization. Moreover, many important properties of a (square) matrix A can be expressed in terms of its eigenvalues. For instance, $\text{tr}(A) = \sum_i \lambda_i$ and $\det(A) = \prod_i \lambda_i$. Hence, a regular matrix A cannot have an eigenvalue $\lambda_i = 0$, since it satisfies $\det A \neq 0$.

Definition A.3 (Similarity). Let $A, B \in \mathbb{R}^{n \times n}$ such that $B = P^{-1}AP$ for a regular matrix P . Then the matrices A and B are called *similar* and the matrix P is called a *similarity transformation* //

A particularly important property of similar matrices is that they share the same set of eigenvalues and consequently, also all other properties which can be expressed in terms of the eigenvalues, such as the rank or the determinant.

Theorem A.1 (Eigendecomposition, [Golub and Van Loan \[1996, § 7.1.1\]](#)). Assume a matrix $A \in \mathbb{C}^{n \times n}$ has n linearly independent eigenvectors $v_i \in \mathbb{C}^n$ and let $V \in \mathbb{C}^{n \times n}$ denote the matrix $(v_1, v_2, \dots, v_n)$. Then V diagonalizes A , where $\Lambda = V^{-1}AV$ is the diagonal matrix with the eigenvalues of A on its diagonal. In particular, $A = V\Lambda V^{-1}$ is called the eigendecomposition of A .

From a computational point of view, having the eigendecomposition of a matrix A at hand has the advantage that subsequently many (a priori expensive) functions of A , such as A^{-1} or A^k , can be computed efficiently.

A.1.2. Miscellaneous

Notation A.5 (Asymptotic bounds). Let $f(n)$ and $g(n)$ be arbitrary functions in $n \in \mathbb{N}$. Then we define the following asymptotic notations.

1. *Upper bound*: $f(n) = \mathcal{O}(g(n))$, i. e., there exists a constant $c > 0$ such that there exists an $n_0 \geq 0$ such that $|f(n)| \leq c|g(n)|$ for all $n \geq n_0$.
2. *Lower bound*: $f(n) = \Omega(g(n))$, i. e., $g(n) = \mathcal{O}(f(n))$.
3. *Tight bound*: $f(n) = \Theta(g(n))$, i. e., $f(n) = \mathcal{O}(g(n))$ and $g(n) = \mathcal{O}(f(n))$.

//

A.2. The Kronecker Product

[[Niederbrucker, 2013, § 3](#)]

Definition A.4. For matrices $A \in \mathbb{R}^{m_1 \times n_1}$ and $B \in \mathbb{R}^{m_2 \times n_2}$ we define the *Kronecker product* (or *tensor product*) of A and B as the $m_1 m_2 \times n_1 n_2$ block matrix

$$A \otimes B := \begin{pmatrix} a_{1,1}B & \cdots & a_{1,n_1}B \\ \vdots & \ddots & \vdots \\ a_{m_1,1}B & \cdots & a_{m_1,n_1}B \end{pmatrix}.$$

//

In this section we review the most important properties of the Kronecker product and show how to utilize it as a “language” for describing and manipulating efficient algorithms on an abstract level. Moreover, we discuss how to approximate given matrices by Kronecker products of matrices and show up relations to low rank approximation problems and the singular value decomposition. For proofs see [Niederbrucker \[2013, §3\]](#) and the references therein.

A.2.1. Basic Properties

Proposition A.2. *Let $A \in \mathbb{R}^{m_1 \times n_1}$, $B \in \mathbb{R}^{m_2 \times n_2}$ and $C \in \mathbb{R}^{m_3 \times n_3}$ be arbitrary real matrices and $\alpha \in \mathbb{R}$ a real constant. Then the following properties hold.*

1. $I_m \otimes I_n = I_{mn}$
2. $(\alpha A) \otimes B = \alpha(A \otimes B) = A \otimes (\alpha B)$
3. $(A \otimes B)^T = A^T \otimes B^T$
4. $A \otimes (B \otimes C) = (A \otimes B) \otimes C$

Since the Kronecker product is associative the expression $A_1 \otimes A_2 \otimes \dots \otimes A_n$ is unambiguous. Thus, we will henceforth use $\bigotimes_{i=1}^n A_i$ as abbreviation for $A_1 \otimes A_2 \otimes \dots \otimes A_n$. Due to its inherently asymmetric definition, it is not surprising that the Kronecker product is not commutative as the following example confirms.

Example A.1.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \otimes I_3 = \begin{pmatrix} 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 \\ 3 & 0 & 0 & 4 & 0 & 0 \\ 0 & 3 & 0 & 0 & 4 & 0 \\ 0 & 0 & 3 & 0 & 0 & 4 \end{pmatrix} \neq \begin{pmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 3 & 4 \end{pmatrix} = I_3 \otimes \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Nevertheless, this counterexample gives already some evidence that $A \otimes B$ and $B \otimes A$ are closely related. Indeed, we will see that $A \otimes B = P(B \otimes A)Q$ for certain permutation matrices P and Q . //

Definition A.5 (Perfect Shuffle). For positive integers p, q and $n = pq$, we define the *perfect shuffle* matrix $\mathcal{S}_{p,q} \in \mathbb{R}^{n \times n}$ as block matrix¹

$$\mathcal{S}_{p,q} := \begin{pmatrix} I_n(1:q:n, :) \\ I_n(2:q:n, :) \\ \vdots \\ I_n(q:q:n, :) \end{pmatrix},$$

consisting of q blocks $I_n(i:q:n, :) \in \mathbb{R}^{p \times n}$. //

¹We use colon notation for submatrix specification (cf. [Notation A.3](#))

Since $\mathcal{S}_{p,q}$ contains exactly one entry 1 in each row and column, perfect shuffles are permutation matrices. Therefore, all matrices $\mathcal{S}_{p,q}$ are orthogonal, i. e., $\mathcal{S}_{p,q}\mathcal{S}_{p,q}^T = I_n$. Moreover, the transpose of a perfect shuffle is a perfect shuffle as well.

Example A.2. Let $p = 3$ and $q = 2$, then $\mathcal{S}_{p,q}$ and $\mathcal{S}_{q,p}$ are each others transpose.

$$\mathcal{S}_{3,2} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}^T = \mathcal{S}_{2,3}^T$$

//

Lemma A.3. Let $p, q \in \mathbb{N}$ be arbitrary integers. Then the identity $\mathcal{S}_{p,q} = \mathcal{S}_{q,p}^T$ holds.

The name perfect shuffle is motivated by the operation of $\mathcal{S}_{p,q}$ on vectors. Suppose we are given a deck of $n = pq$ cards represented by an n -dimensional vector x , then $y = \mathcal{S}_{p,q}x$ can be imagined as follows. The deck of n cards is first split up into p piles of height q and then we successively take one card from each pile to obtain y . The matrix vector multiplication $y^T = x^T \mathcal{S}_{p,q}$ can be interpreted analogously.

Example A.3. Consider the perfect shuffle $\mathcal{S}_{3,2}$ and an arbitrary vector $x \in \mathbb{R}^6$. Then,

$$y = \mathcal{S}_{3,2}x = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} \xrightarrow{\text{"split"}} \left[\begin{array}{c|c|c} x_1 & x_3 & x_5 \\ x_2 & x_4 & x_6 \end{array} \right] \xrightarrow{\text{"take"}} y = \begin{pmatrix} x_1 \\ x_3 \\ x_5 \\ x_2 \\ x_4 \\ x_6 \end{pmatrix}.$$

//

Perfect shuffle permutations allow to relate the columnwise vectorization $\text{vec}(A)$ of a matrix A with the vectorization of its transpose as well as $A \otimes B$ to $B \otimes A$.

Proposition A.4. Let $A \in \mathbb{R}^{m_1 \times n_1}$ and $B \in \mathbb{R}^{m_2 \times n_2}$

1. $\text{vec}(A) = \mathcal{S}_{m_1, n_1} \cdot \text{vec}(A^T)$
2. $A \otimes B = \mathcal{S}_{m_1, m_2}^T \cdot (B \otimes A) \cdot \mathcal{S}_{n_1, n_2}$

Example A.4. Let A be an arbitrary 3×2 matrix. Then one can directly confirm that

$$\text{vec}(A) = \begin{pmatrix} a_{1,1} \\ a_{2,1} \\ a_{3,1} \\ a_{1,2} \\ a_{2,2} \\ a_{3,2} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{1,1} \\ a_{1,2} \\ a_{2,1} \\ a_{2,2} \\ a_{3,1} \\ a_{3,2} \end{pmatrix} = \mathcal{S}_{3,2} \cdot \text{vec}(A^T).$$

Moreover, recall the counterexample for the commutativity of the Kronecker product given in [Example A.1](#). According to [Proposition A.4](#) these two matrices are related in the form

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \otimes I_3 = \mathcal{S}_{3,2} \cdot \left[I_3 \otimes \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \right] \cdot \mathcal{S}_{2,3}.$$

This claim is verified by the following step-by-step computation².

$$\begin{array}{c|c|c} \begin{pmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 3 & 4 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \\ \hline \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 \\ 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 4 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 \\ 3 & 0 & 0 & 4 & 0 & 0 \\ 0 & 3 & 0 & 0 & 4 & 0 \\ 0 & 0 & 3 & 0 & 0 & 4 \end{pmatrix} \end{array}$$

In the following we give a proof for the general assertions given in [Proposition A.4](#). //

A key property of the Kronecker product is that the product $A \otimes B$ inherits most of the structure of its factors A and B . Concretely, if A and B have the property X then usually also $A \otimes B$ has the property X . In particular, if A and B are *nonnegative*, *symmetric*, *triangular*, *orthogonal*, *stochastic* or *diagonal matrices*, their product $A \otimes B$ inherits the respective properties.

Theorem A.5 (Mixed Product Property). *Let A, B, C and D be arbitrary matrices such that AC and BD are defined, then*

$$(A \otimes B)(C \otimes D) = AC \otimes BD. \quad (\text{A.1})$$

² In order to explicitly compute the matrix product $P = A \cdot B \cdot C$ we utilize the scheme

$$P = A \cdot B \cdot C = \frac{\begin{array}{c|c} & B \\ \hline A & A \cdot B \end{array}}{\begin{array}{c|c} & C \\ \hline A \cdot B & A \cdot B \cdot C \end{array}}$$

In fact, the mixed product property states that for many computations it suffices to perform the respective computations on the individual factors instead of the entire Kronecker product. The following corollary states several useful properties of this kind.

Corollary A.6. *Computations on Kronecker products usually require only independent computations on its factors. For example the following properties hold.*

1. Let A and B be regular matrices, then $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.
2. Let $A = Q_A R_A$ and $B = Q_B R_B$ be the QR decompositions of A and B , respectively. Then $A \otimes B = (Q_A \otimes Q_B)(R_A \otimes R_B)$ is the QR decomposition of $A \otimes B$.
3. Let $A = L_A U_A$ and $B = L_B U_B$ be the LU decompositions of A and B , respectively. Then $A \otimes B = (L_A \otimes L_B)(U_A \otimes U_B)$ is the LU decomposition of $A \otimes B$.
4. Let $A = U_A \Sigma_A V_A^T$ and $B = U_B \Sigma_B V_B^T$ be the SVDs of A and B , respectively. Then $A \otimes B = (U_A \otimes U_B)(\Sigma_A \otimes \Sigma_B)(V_A^T \otimes V_B^T)$ is the SVD of $A \otimes B$.

Example A.5. Given matrices B , C and X such that CXB^T is defined. Then the matrix equation $Y = CXB^T$ is equivalent to the linear system $\text{vec}(Y) = (B \otimes C)\text{vec}(X)$. Consider the case $B \in \mathbb{R}^{2 \times 2}$, $C \in \mathbb{R}^{2 \times 3}$ and $X \in \mathbb{R}^{3 \times 2}$. Moreover, let x_1 and x_2 denote the first and second column of X , respectively, i.e., $x_1 = X(:, 1)$, $x_2 = X(:, 2)$ and $X = (x_1, x_2)$. For this case, the stated equivalence is verified as follows.

$$\begin{aligned} Y &= CXB^T = C \cdot (x_1, x_2) \cdot B^T = (Cx_1, Cx_2) \cdot \begin{pmatrix} b_{1,1} & b_{2,1} \\ b_{1,2} & b_{2,2} \end{pmatrix} \\ &= (C(b_{1,1}x_1 + b_{1,2}x_2), C(b_{2,1}x_1 + b_{2,2}x_2)) \end{aligned}$$

$$(B \otimes C)\text{vec}(X) = \begin{pmatrix} b_{1,1}C & b_{1,2}C \\ b_{2,1}C & b_{2,2}C \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} C(b_{1,1}x_1 + b_{1,2}x_2) \\ C(b_{2,1}x_1 + b_{2,2}x_2) \end{pmatrix} = \text{vec}(Y)$$

Assume that B and C are square matrices of dimension n . In case we ignore the structure of $B \otimes C \in \mathbb{R}^{n^2 \times n^2}$, solving the n^2 -dimensional linear system $\text{vec}(Y) = (B \otimes C)\text{vec}(X)$ is tantamount to $\mathcal{O}(n^6)$ arithmetic operations. Contrary to that, by computing the LU decompositions of B and C independently to obtain a LU decomposition of $B \otimes C$ (cf. [Corollary A.6](#)), this system can be solved by $\mathcal{O}(n^3)$ arithmetic operations only. //

Corollary A.7. *The mixed property allows for the following two natural generalizations.*

1. Let A_i, B_i be matrices such that the products $A_i B_i$ are defined, then

$$\left(\bigotimes_{i=1}^n A_i \right) \cdot \left(\bigotimes_{i=1}^n B_i \right) = \bigotimes_{i=1}^n (A_i B_i)$$

2. Let A_i, B_i be matrices such that the products $\prod_{i=1}^n A_i$ and $\prod_{i=1}^n B_i$ are defined, then

$$\prod_{i=1}^n (A_i \otimes B_i) = \left(\prod_{i=1}^n A_i \right) \otimes \left(\prod_{i=1}^n B_i \right)$$

Example A.6. In order to illustrate the two generalizations of the mixed product property from [Corollary A.7](#), consider matrices A_i, B_i with $1 \leq i \leq 3$. First, assume that all products $A_i B_i$ exist. Then, by the mixed product property and associativity of $\otimes$,

$$\begin{aligned} (A_1 \otimes A_2 \otimes A_3)(B_1 \otimes B_2 \otimes B_3) &= ((A_1 \otimes A_2) \otimes A_3)((B_1 \otimes B_2) \otimes B_3) \\ &= ((A_1 \otimes A_2)(B_1 \otimes B_2)) \otimes (A_3 B_3) \\ &= (A_1 B_1) \otimes (A_2 B_2) \otimes (A_3 B_3) \end{aligned}$$

Similarly, by assuming that all products $\prod_{i=1}^3 A_i$ and $\prod_{i=1}^3 B_i$ exist, we see that

$$(A_1 \otimes B_1) \otimes (A_2 \otimes B_2) \otimes (A_3 \otimes B_3) = (A_1 A_2 \otimes B_1 B_2)(A_3 B_3) = (A_1 A_2 A_3) \otimes (B_1 B_2 B_3).$$

//

Clearly, the definition of the Kronecker product covers also vectors. Assume we are given matrices $A_1, \dots, A_n$ and vectors x_i such that $A_i x_i$ is defined for all $1 \leq i \leq n$. Then we get by [Corollary A.7](#) that $y = (\bigotimes_{i=1}^n A_i)(\bigotimes_{i=1}^n x_i) = \bigotimes_{i=1}^n (A_i x_i)$.

Theorem A.8. Let $A, B \in \mathbb{R}^{n \times n}$ be square matrices and let (λ_i, v_i) and (μ_j, w_j) be the eigenpairs of A and B , respectively. Then the eigenpairs of $A \otimes B$ are given by $(\lambda_i \mu_j, v_i \otimes w_j)$ for all $1 \leq i, j \leq n$. Analogous results also hold for the singular values and vectors of $A \otimes B$. Moreover, all results extend accordingly to products of the form $\bigotimes_{i=1}^k A_i$.

Corollary A.9. Let $A_i \in \mathbb{R}^{n_i \times n_i}$ be square matrices and B_i arbitrary matrices, then

1. $\text{tr}(\bigotimes_{i=1}^k A_i) = \prod_{i=1}^k \text{tr}(A_i)$
2. $\det(\bigotimes_{i=1}^k A_i) = \prod_{i=1}^k \det(A_i)^{n_i}$
3. $\text{rank}(\bigotimes_{i=1}^k B_i) = \prod_{i=1}^k \text{rank}(B_i)$

A.2.2. Kronecker Products and Fast Algorithms

In general, computing a matrix vector product $y = Ax$ for a dense square matrix $A \in \mathbb{R}^{n \times n}$ requires $\Theta(n^2)$ floating point operations. However, for certain specially structured matrices it is possible to break this barrier and to provide, e. g., algorithms requiring only $\Theta(n \log n)$ floating point operations. Matrix vector products with a subquadratic complexity are often called *fast transforms*. For instance, [Cooley and](#)

Tukey [1965] showed in their seminal paper that the computation of the discrete Fourier transformation of a signal of length $n = 2^k$ requires only $\Theta(n \log_2 n)$ instead of $\Theta(n^2)$ arithmetic operations. Today, we are aware of many fast transforms and almost all of them are closely related to the Kronecker product [Van Loan, 2000].

Modern computing architecture became highly complex and diverse, since every single processor provides several levels of parallelism. While it gets more and more difficult to produce a hand-written code which fully utilizes the theoretical potential of the computing hardware, automated program generators such as FFTW [Frigo and Johnson, 2005] or SPIRAL [Püschel *et al.*, 2005] proved to be successful in producing highly efficient code for computing fast transforms on given target platforms. A main ingredient in the automated search for efficient codes in systems like SPIRAL is the Kronecker product formalism. The appropriate representation of algorithms in terms of Kronecker products allows to discover many theoretically equivalent variants of an algorithm in order to search for the most efficient implementation on a certain platform.

Traditional concepts for speeding up computations are *vectorization* and *threading*. By vectorization we mean that a single instruction is performed on a vector of operands instead of a single operand. On the contrary, threading means performing several instructions independently in parallel. Both of these concepts can be found in the Kronecker product formalism as well as the following definitions show.

Definition A.6 (Parallel Kronecker Product). Let $A \in \mathbb{R}^{m \times n}$ be an arbitrary matrix. Then,

$$I_k \otimes A = \text{diag}(\underbrace{A, A, \dots, A}_{k \text{ times}}),$$

is called the *parallel Kronecker product*. //

Definition A.7 (Vector Kronecker Product). Let $A \in \mathbb{R}^{m \times n}$ be an arbitrary matrix. Then,

$$A \otimes I_k = \begin{pmatrix} a_{1,1}I_k & \dots & a_{1,n}I_k \\ \vdots & \ddots & \vdots \\ a_{m,1}I_k & \dots & a_{m,n}I_k \end{pmatrix}$$

is called the *vector Kronecker product*. //

These terms can be motivated by investigating the matrix vector product with the parallel and vector Kronecker product, respectively. In case of a parallel Kronecker Product $y = (I_k \otimes A)x$, the input vector x is split up into k parts where each of those parts gets independently (“in parallel”) multiplied. In case of the vector Kronecker product, the computation of $y = (A \otimes I_k)x$ is tantamount to computing the matrix vector product with A in vectorized form, i. e., with k -dimensional variables instead of scalars. Moreover, the parallel and the vector Kronecker product can be interpreted as “hidden” matrix-matrix multiplications. Consider a matrix $A \in \mathbb{R}^{n \times n}$ with $n = rc$

and a vector $x \in \mathbb{R}^n$. Then we observe the equivalences

$$\begin{aligned} y = (I_c \otimes A)x &\equiv y_{r \times c} = Ax_{r \times c} \\ y = (A \otimes I_r)x &\equiv y_{r \times c} = x_{r \times c} A^T \end{aligned}$$

The following example illustrates these representations as matrix products more concretely.

Example A.7. Let the matrix $A \in \mathbb{R}^{2 \times 2}$ and the vector $x \in \mathbb{R}^6$ be arbitrary. Then,

$$y = (I_3 \otimes A)x = \begin{pmatrix} A & 0 & 0 \\ 0 & A & 0 \\ 0 & 0 & A \end{pmatrix} \begin{pmatrix} x(1:2) \\ x(3:4) \\ x(5:6) \end{pmatrix} = \begin{pmatrix} Ax(1:2) \\ Ax(3:4) \\ Ax(5:6) \end{pmatrix},$$

and moreover, as a matrix matrix product we get

$$y_{2 \times 3} = (Ax(1:2), Ax(3:4), Ax(5:6)) = A(x(1:2), x(3:4), x(5:6)) = Ax_{2 \times 3}.$$

Analogously, we observe for

$$y = (A \otimes I_3)x = \begin{pmatrix} a_{1,1}I_3 & a_{1,2}I_3 \\ a_{2,1}I_3 & a_{2,2}I_3 \end{pmatrix} \begin{pmatrix} x(1:3) \\ x(4:6) \end{pmatrix} = \begin{pmatrix} a_{1,1}x(1:3) + a_{1,2}x(4:6) \\ a_{2,1}x(1:3) + a_{2,2}x(4:6) \end{pmatrix}$$

the matrix matrix product

$$\begin{aligned} y_{3 \times 2} &= (a_{1,1}x(1:3) + a_{1,2}x(4:6), a_{2,1}x(1:3) + a_{2,2}x(4:6)) \\ &= (x(1:3), x(4:6)) \begin{pmatrix} a_{1,1} & a_{2,1} \\ a_{1,2} & a_{2,2} \end{pmatrix} = x_{3 \times 2} A^T. \end{aligned}$$

//

In order to see the expressiveness of the Kronecker product formalism we investigate in the following a simply structured linear transform.

Definition A.8. Consider the following recursively defined matrices H_{2^k} ,

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad H_{2^k} = \begin{pmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{pmatrix} = H_2 \otimes H_{2^{k-1}} = \bigotimes_{i=1}^k H_2,$$

The matrices H_{2^k} are called *Walsh matrices* and are a special form of *Hadamard matrices*. In general, a matrix $H_n \in \{1, -1\}^{n \times n}$ is called Hadamard matrix if its columns are pairwise orthogonal.

//

Example A.8. The linear transform $y = H_{2^k}x$ based on the Walsh matrices is called *Fast Walsh-Hadamard Transform* (FWHT). The FWHT can be efficiently computed in subquadratic time, e. g., in the following two ways:

1. Recursive computation of $H_{2^k}x$:

$$H_{2^k}x = \begin{pmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \bar{x}_1 + \bar{x}_2 \\ \bar{x}_1 - \bar{x}_2 \end{pmatrix}, \quad \begin{matrix} \bar{x}_1 = H_{2^{k-1}} \cdot x_1 \\ \bar{x}_2 = H_{2^{k-1}} \cdot x_2 \end{matrix} \quad (\text{A.2})$$

2. Iterative computation of $H_{2^k}x$:

$$H_{2^k}x = \begin{pmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} H_{2^{k-1}}(x_1 + x_2) \\ H_{2^{k-1}}(x_1 - x_2) \end{pmatrix} \quad (\text{A.3})$$

Note that here “recursive computation” and “iterative computation” are supposed to be interpreted from a programmers point of view. In other words, the algorithm defined in (A.2) allows for a natural recursive implementation whereas the algorithm defined by (A.3) allows for a natural iterative implementation using two nested loops. Both of these computations require only $\Theta(k2^k)$ arithmetic operations as an immediate consequence of the Master theorem [Cormen *et al.*, 2001, p. 73].

//

While Example A.8 outlines how the matrix vector product with a Kronecker product of matrices can be computed efficiently, it does not yet reveal the full potential of the Kronecker product formalism as a “language” for describing algorithms. Observe, that by the mixed product property the matrix H_{2^k} can also be factorized as

$$H_{2^k} = (H_2 \otimes I_{2^{k-1}})(I_2 \otimes H_{2^{k-1}}). \quad (\text{A.4})$$

Thus, by the interpretation of the parallel and vector Kronecker product the computation of $(H_2 \otimes I_{2^{k-1}})(I_2 \otimes H_{2^{k-1}})x$ reads as follows. First, we compute the parallel Kronecker product $(\bar{x}_1, \bar{x}_2)^T = (I_2 \otimes H_{2^{k-1}})(x_1, x_2)^T$, i. e., we compute $\bar{x}_1 = H_{2^{k-1}}x_1$ and $\bar{x}_2 = H_{2^{k-1}}x_2$ independently in parallel. Second, we compute the vector Kronecker product $(H_2 \otimes I_{2^{k-1}})(\bar{x}_1, \bar{x}_2)^T$, i. e., we compute the matrix vector product with H_2 with 2^{k-1} -dimensional vectors instead of scalars. Therefore, we observe that (A.4) is a “description” of the recursion (A.2) in the Kronecker product formalism. Moreover, by the generalized mixed product property, we obtain the factorization

$$H_{2^k} = \prod_{i=1}^k (I_{2^{i-1}} \otimes H_2 \otimes I_{2^{k-i}}). \quad (\text{A.5})$$

The validity of this factorization can be immediately verified by

$$I_{2^{i-1}} \otimes H_2 \otimes I_{2^{k-i}} = \underbrace{I_2 \otimes I_2 \otimes \cdots \otimes I_2}_{i-1 \text{ times}} \otimes H_2 \otimes \underbrace{I_2 \otimes I_2 \otimes \cdots \otimes I_2}_{k-i \text{ times}}$$

As it can be easily seem, (A.5) “describes” the iterative algorithm (A.3): The first part, $H_2 \otimes I_{2^{k-i}}$ means that H_2 is applied to vectors of length 2^{k-i} and overall, this vector operation gets applied to 2^{i-1} vectors in parallel.

In general, for $k = k_1 + k_2 + \dots + k_p$ we obtain the factorization

$$H_{2^k} = \prod_{i=1}^p (I_{2^{k_1+k_2+\dots+k_{i-1}}} \otimes H_{2^{k_i}} \otimes I_{2^{k_{i+1}+\dots+k_p}}), \quad (\text{A.6})$$

which is, analogously to the previous factorization, verified by representing the respective identity matrices as Kronecker products of smaller identity matrices. The factorization (A.6) clearly subsumes the previous two special cases. By considering $p = 2$, $k_1 = 1$ and $k_2 = k - 1$ we get (A.4) and by $p = k$ and $k_i = 1$ for all i we get (A.5). Thus, the factorization (A.6) leads to a wide variety of—in terms of the result—equivalent potential implementations of the matrix vector product $H_{2^k}x$. For a more comprehensive discussion on how to find the best performing of these implementations see [Johnson and Püschel \[2000\]](#).

For the sake of simplicity we considered the FWHT as an introductory example to the way how the Kronecker product formalism is applied in the context of high performance computing. In more complicated fast transforms like the *Fast Fourier Transform* (FFT) also additional permutation matrices are needed in order to describe the different variants of a certain algorithm. However, the fruitful relationship between the Kronecker product formalism and fast linear transforms was not utilized for a long time. In particular [Cooley and Tukey \[1965\]](#) did not expose this framework in their seminal work. While early work [[Drubin, 1971](#)] outlined the connection between the Kronecker product formalism and the FFT, many seemingly independent algorithms for computing the FFT were developed. For instance, [Van Loan \[1992\]](#) elaborated in detail how the Kronecker product formalism can be used to unify the manifold of known approaches towards computing the FFT.

A.3. Perron-Frobenius Theory

[[Niederbrucker, 2013](#), Appendix B]

Definition A.9. We call a real matrix A *positive* ($A > 0$), if $a_{i,j} > 0$ for all i, j . Likewise, we call A *nonnegative* ($A \geq 0$) if $a_{i,j} \geq 0$ for all i, j . The same notation applies to vectors as well. //

The Perron-Frobenius theory is a particularly elegant piece of linear algebra which reveals, e. g., how the nonnegativity (positivity) of a matrix is reflected in its eigenvalues and eigenvectors. To indicate how well the Perron-Frobenius theory can be utilized in application contexts, we employ the convergence analysis of finite Markov chains as a running example throughout this appendix.

Example A.9 (Markov chain). A finite discrete-time Markov chain is a mathematical model consisting of a finite amount of states, where the current state can change at each discrete point of time. In particular, we consider so-called homogenous Markov chains, where at each point of time the probability for a state transition from state i to state j is given by $0 \leq p_{j,i} \leq 1$. Thus, the transition probabilities can be conveniently

represented by a square matrix P . In order to obtain a sound model, the *transition matrix* P has to be *column stochastic*, i.e., each column of P sums up to one (is a *stochastic vector*). In particular, this means that for every node the sum over its (outgoing) transition probabilities equals to one.

Consider we are given a stochastic vector s whose elements s_i represent the probability that the Markov chain is initially in state i . Then the probability distribution of being in a certain state after k time steps is given by the vector $P^k s$, since a column-stochastic matrix times a stochastic vector is a stochastic vector. The following three question are of tremendous interest: Does the Markov chain converge, i.e., does a *stationary distribution* $\pi = \lim_{k \rightarrow \infty} P^k s$ exist? How does the stationary distribution π look? Does π depend on the initial distribution s or not? As we will see, these questions can easily be answered by utilizing results from the Perron-Frobenius theory. //

Historically, the Perron-Frobenius theory originates in investigations from positive matrices [Perron, 1907; Frobenius, 1908], which were later extended to nonnegative matrices [Frobenius, 1912]. In the following, we give a brief overview of the most fundamental notions and results of the Perron-Frobenius theory. A more detailed exposition can be found, e.g., in the textbook of Meyer [2000, § 8].

The core of the Perron-Frobenius theory is what is today commonly known as Perron-Frobenius theorem. Several versions of this theorem exist which slightly differ in their assumptions and conclusions. First, we consider the Perron-Frobenius theorem for positive matrices.

Theorem A.10 (Perron-Frobenius [Meyer, 2000, p. 667]). *Let $A \in \mathbb{R}^{n \times n}$ be a positive matrix, then the following assertions hold.*

1. *There exists a positive real eigenvalue λ of A such that $\lambda > |\mu|$ for all other eigenvalues $\mu \neq \lambda$ of A .*
2. *The eigenvalue λ is a simple root of the characteristic polynomial of A .*
3. *There is a strictly positive left eigenvector $u > 0$ with $u^T A = \lambda u^T$ and a strictly positive right eigenvector v with $Av = \lambda v$ associated with λ .*
4. *Except for u and v all other eigenvectors of A contain at least one negative entry.*
5. *Let u and v be normalized such that $u^T v = 1$, then*

$$\lim_{k \rightarrow \infty} A^k / \lambda^k = v u^T. \quad (\text{A.7})$$

Example A.10 (Markov chain, continued). In case we assume a positive transition matrix P (which means that any state change can happen with a certain probability), Theorem A.10 straightforwardly delivers answers to all questions we raised in Example A.9. Concretely, since P is column stochastic, its left eigenvector is given by the constant vector $\mathbf{1}$ with corresponding eigenvalue one. Thus, the spectral radius

of P is one and $\lim_{k \rightarrow \infty} A^k = v\mathbf{1}^T$, i. e., the Markov chain converges. Moreover, since $v\mathbf{1}^T s = v$ for all stochastic vectors s , we see that the stationary distribution $\pi = v$ and that π is independent of the initial distribution.

In the following we will investigate up to which extent [Theorem A.10](#) can be generalized to nonnegative matrices. To illustrate the differences between matrices of a particular structure, we consider the following four nonnegative transition matrices P_1 – P_4 .

	P_1	P_2	P_3	P_4
Matrix	$\begin{pmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 0 & 1/2 \\ 1 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 1/2 \\ 0 & 1/2 \end{pmatrix}$
Induced graph				
Eigenvalues	$\{1, 0\}$	$\{1, -1/2\}$	$\{1, -1\}$	$\{1, 1/2\}$
Eigenvectors	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1/2 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

We can see that (some of) the conclusion of [Theorem A.10](#) do not hold in case of P_3 and P_4 . //

Nonnegative matrices can be represented as limits of positive matrices. Thus, for a nonnegative matrix $A \in \mathbb{R}^{n \times n}$ with spectral radius $\rho(A) \geq 0$, the real value $\lambda = \rho(A)$ is an eigenvalue of A and its corresponding eigenvector is nonnegative [[Meyer, 2000](#), p. 670]. Moreover, a variational characterization of the spectral radius $\rho(A)$ of a square matrix $A \geq 0$ is given by the so-called Collatz-Wielandt formula [[Meyer, 2000](#), pp. 669–670]

$$\rho(A) = \max_{\substack{x \geq 0 \\ x \neq 0}} \min_{\substack{1 \leq i \leq n \\ x_i \neq 0}} \frac{[Ax]_i}{x_i} = \min_{x > 0} \max_{1 \leq i \leq n} \frac{[Ax]_i}{x_i}.$$

From an application point of view, it seems to be reasonable that [Theorem A.10](#) can be extended to certain classes of nonnegative matrices. Consider, e. g., a Markov chain with nonnegative transition matrix (cf. [Example A.10](#)). In case that it is possible to reach from each state (after a finite amount of transitions) every other state, one would expect that the stationary distribution should show a nonzero (potentially small) probability for every state. In the following, we make these considerations precise, which leads to the notion of *irreducible matrices*. Interestingly, it is indeed only the pattern of nonzero values what matters in order to generalize the Perron-Frobenius theorem to nonnegative matrices and not the overall number of nonzero entries or their concrete values (cf. [Example A.10](#)).

In the following we give two equivalent characterizations of irreducible matrices. A matrix $A \in \mathbb{R}^{n \times n}$ is called *reducible*, if there exists a permutation matrix P such that

$$P^T A P = \begin{pmatrix} B & C \\ 0 & D \end{pmatrix},$$

with square matrices B and D . Consequently, a matrix is called *irreducible* if it is not reducible. A more insightful characterization can be given by utilizing notions from graph theory. Recall that a (directed) graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes is fully determined by its *adjacency matrix* $A \in \{0, 1\}^{n \times n}$, i. e., $A_{i,j} = 1$, if and only if there is a directed edge between node i and node j . Therefore, given a matrix $A \in \mathbb{R}^{n \times n}$, we can define its *induced graph* by the adjacency matrix $A^{\mathcal{G}}$ with $A_{i,j}^{\mathcal{G}} = 1$, if and only if $A_{i,j} \neq 0$ and $A_{i,j}^{\mathcal{G}} = 0$ otherwise. Now a matrix $A \in \mathbb{R}^{n \times n}$ is irreducible, if and only if its induced graph is strongly connected [Meyer, 2000, p. 671], i. e., there exists a directed path from every node to any other node. For nonnegative irreducible matrices Theorem A.10 can be generalized as follows.

Theorem A.11 (Perron-Frobenius [Meyer, 2000, p. 673]). *Let $A \in \mathbb{R}^{n \times n}$ be a nonnegative irreducible matrix and let $\lambda = \rho(A)$ denote its spectral radius. Then the following assertions hold.*

1. *The spectral radius $\lambda > 0$ and λ is an eigenvalue of A .*
2. *The eigenvalue λ is a simple root of the characteristic polynomial of A .*
3. *There is a strictly positive left eigenvector $u > 0$ with $u^T A = \lambda u^T$ and a strictly positive right eigenvector v with $Av = \lambda v$ associated with λ .*
4. *Except for u and v all other eigenvectors of A contain at least one negative entry.*
5. *If A has h eigenvalues on its spectral circle, then they are for $1 \leq i \leq h$ given by $\lambda_i = \lambda \omega^{(i-1)}$, where $\omega = e^{2\pi i/h}$ is a h -th root of unity.*

Thus, we observe two major difference between Theorem A.10 and Theorem A.11. First, there can exist several eigenvalues on the spectral circle in case of an irreducible matrix which are structured according to the last assertion in Theorem A.11. Moreover, the convergence property (A.7) in Theorem A.10 is missing, which can indeed not be generalized to irreducible matrices, as the following example illustrates.

Example A.11 (Markov chain, continued). Consider the matrix P_3 from Example A.10. The graph induced by P_3 is obviously strongly connected. Thus, P_3 is irreducible. On the other hand, the limit $\lim_{k \rightarrow \infty} P_3^k$ does not exist, since for $j \in \mathbb{N}$ we see that $P_3^{2j} = P_3$ and $P_3^{2j+1} = I_2$. Therefore, even though irreducible matrices preserve almost all properties of positive matrices, e. g., the positive eigenpair, they are not suited for deriving a convergence result of the kind we require for the analysis of Markov chains. //

As the previous example illustrates, the notion of irreducibility is not yet fully satisfactory in certain application contexts. The notion of *primitive matrices* plays an important role in the generalization of Theorem A.10 in its entirety to nonnegative matrices.

Definition A.10. We call a square nonnegative matrix $A \in \mathbb{R}^{n \times n}$ *primitive*, if there exists a constant $k \in \mathbb{N}$ such that A^k is a primitive matrix, i. e., $A^k > 0$. //

Corollary A.12 (see Meyer [2000, p.674]). *Theorem A.10 holds also in case we assume A to be primitive instead of strictly positive.*

It is noteworthy that this generalization which is very important for applications was already stated in the pioneering work of Perron [1907, p.262]. The question which is left open is how the notions of irreducibility and primitivity are related to each other. We refrain here from a detailed discussion and instead, we give a very simple criterion to detect the primitivity of an irreducible matrix.

Lemma A.13 (see Meyer [2000, p.678]). *Let A be a nonnegative irreducible matrix with at least one positive diagonal element, then A is primitive.*

Example A.12 (Markov chain, continued). By the considerations above, the transition matrices given in Example A.10 can be easily categorized (and analyzed). In particular, P_1 is positive, P_2 is primitive, P_3 is irreducible and P_4 is (only) nonnegative. Accordingly, either Theorem A.10 (P_1, P_2) or Theorem A.11 (P_3) or neither of them (P_4) applies. //

Summarizing, we have seen that the Perron-Frobenius theorem provides several insights into the structure of the eigenvalues and eigenvectors of positive and nonnegative matrices. A particularly interesting variant of the theorem is the one for primitive matrices (cf. Corollary A.12), which allows for the same conclusion as the variant for positive matrices. The generalization of the Perron-Frobenius theorem to classes of nonnegative matrices requires assumptions on the pattern of nonzero entries, which lead to the notion of irreducible matrices. In many applications this property, i. e., the strong connectivity of the induced graph, is given in a natural way and its verification is fairly straightforward. Moreover, the primitivity of an irreducible matrix is simply verified by the existence of a nonzero diagonal element.

A.4. Graph Structures: Definitions and Properties

The technical contributions in this thesis are mostly of a generic (“existential”) nature and consequently, they are not affected by the properties of particular graph structures. Hence, we always considered network graphs $(\mathcal{V}, \mathcal{E})$ which are reasonably well connected such that the computation of an all-to-all aggregation is in principle possible (cf. Section 3.3.4). However, in practice the particular properties of the structure of the communication network heavily influence the convergence speed of the respective algorithm.

A.4.1. Connectivity Measures

Due to their simplicity and local scope, the distributed algorithms we consider in this thesis converge the faster the better the underlying graph structure is connected. In the following, we review some connectivity measures which can be used to quantify up to which extent a certain graph is well-connected.

In [Section A.3](#) we used the convergence analysis of Markov chains as a straightforward application of the Perron-Frobenius theorem. It is well known that there is a natural correspondence between Markov chains and (certain) average computations. In particular, [Boyd et al. \[2006\]](#) showed for a class of aggregation algorithms that it takes $\mathcal{O}(\frac{\log n + \log \varepsilon^{-1}}{1 - \lambda_2})$ iterations until all local estimates are ε -accurate. The same bound is known in the theory of Markov chains for the so-called ε -mixing time, i. e., the time until all states are—up to ε —equally likely. The quantity $1 - \lambda_2$ is called *spectral gap*, where λ_2 denotes the second largest eigenvalue (in magnitude) of the respective iteration matrix. For instance, we saw in [Section A.3](#) for convergent Markov chains that the dominating eigenvalue $\lambda_1 = 1$ and $\lambda_1 > \lambda_2$. However, $1 - \lambda_2$ can still be arbitrary small. In particular, the worse a graph is connected the smaller is its spectral gap. Besides the spectral gap, the following two graph connectivity measures are commonly considered.

Definition A.11 (Vertex expansion). Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an arbitrary graph and let $\partial\mathcal{S} = \{i \in \mathcal{V} \setminus \mathcal{S} \mid \mathcal{N}_i \cap \mathcal{S} \neq \emptyset\}$ be the so-called boundary of $\mathcal{S}$. Then the *vertex expansion* χ of $\mathcal{G}$ is defined as

$$\chi = \min_{\mathcal{S} \subseteq \mathcal{V}, 0 < |\mathcal{S}| \leq n/2} \frac{|\partial\mathcal{S}|}{|\mathcal{S}|} \in (0, 1].$$

//

Definition A.12 (Conductance). Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an arbitrary graph, $\mathcal{E}(\mathcal{S}, \mathcal{V} \setminus \mathcal{S}) = \{(i, j) \in \mathcal{E} \mid i \in \mathcal{S}, j \in \mathcal{V} \setminus \mathcal{S}\}$, and $\text{vol}(\mathcal{S}) = \sum_{i \in \mathcal{S}} |\mathcal{N}_i|$ be the volume of a set $\mathcal{S}$. Then the *conductance* ϕ of $\mathcal{G}$ is defined as

$$\phi = \min_{\mathcal{S} \subseteq \mathcal{V}, 0 < \text{vol}(\mathcal{S}) < \text{vol}(\mathcal{V})/2} \frac{|\mathcal{E}(\mathcal{S}, \mathcal{V} \setminus \mathcal{S})|}{\text{vol}(\mathcal{S})} \in (0, 1].$$

//

Note that there is a simple relationship between the vertex expansion and the conductance of a graph (see, e. g., [Giakkoupis and Sauerwald \[2012\]](#))

$$(\delta/\Delta)\phi \leq \chi \leq \Delta\phi, \quad \delta = \min_{i \in \mathcal{V}} |\mathcal{N}_i|, \quad \Delta = \max_{i \in \mathcal{V}} |\mathcal{N}_i|.$$

Therefore, for regular graphs vertex expansion and conductance are qualitatively the same. However, for general graphs there are substantial difference between these two

notions. Moreover, the conductance of a regular graph is related to its spectral gap by the so-called Cheeger inequality [Hoory *et al.*, 2006]

$$\frac{1 - \lambda_2}{2} \leq \phi \leq \sqrt{2(1 - \lambda_2)}.$$

Obviously, for all defined measures $(1 - \lambda_2, \phi, \chi)$ it holds that a bad connectivity is indicated by values close to zero. Hence, we call a graph well-connected with respect to one of these connectivity measures if the respective measures can be bounded away from zero. If such a bound can be found, then it is reasonable to treat, e.g., the spectral gap as a constant in complexity bounds.

Vertex expansion, conductance, and at a large scale also the spectral gap are difficult to compute in practice. As a rule of thumb whether a graph is well-connected (i.e., whether epidemic processes converge quickly) one can consider the *diameter* as a simple connectivity measure, which is defined as the maximal shortest distance between any two nodes. However, there exist graphs with short diameter but otherwise bad connectivity properties, e.g., so-called dumbbell graphs which are defined as two complete graphs connected by a single edge.

A.4.2. Graph Structures

In the following, we define and discuss the graph structures we use in our experiments.

Complete Graph $C(n)$: As “perfectly” connected graph, complete graphs are particularly interesting as a benchmark. Obviously, the superior connectivity of complete graphs is also reflected in the aforementioned connectivity measures. However, the large node degree also leads to inefficiencies in link-based fault tolerance mechanisms (cf. Section 4.2).

Hypercube $HC(d)$: A d -dimensional hypercube has $n = 2^d$ nodes and the neighborhoods are defined as $\mathcal{N}_i = \{i \oplus 2^k \mid 0 \leq k < d\}$, where the operator $\oplus$ denotes the binary XOR operator. The particularly interesting property of hypercubes in our context is that they have small node degrees as well as a small diameter (both $d = \log_2 n$).

Erdős-Rényi $ER(n, p)$: In an Erdős-Rényi graph $ER(n, p)$ each edge exists with probability p . This means, an Erdős-Rényi graph is created by flipping a coin for each node with probabilities $\{p, 1 - p\}$, in order to decide whether the particular edge exists.

Random geometric $G(n, d)$: A random geometric graph $G(n, d)$ is created by first picking coordinates from the unit square (uniformly at random) for each node. Thereafter, there exists an edge between two nodes if and only if the Euclidean distance between the coordinates of the nodes is smaller than d .

Square Lattice $L(n)$: As an example for a topology with a bad connectivity we consider square lattices with $n = m^2$ nodes. Hence, each node is assigned coordinates (x, y) and its neighbors have coordinates $(x \pm 1, y)$ and $(x, y \pm 1)$ (if these nodes exist). The inferior connectivity can be easily seen from its large diameter of $2(\sqrt{n} - 1)$.

Listings

List of Figures

1.1. Aggregation as “dual” of rumor spreading	5
2.1. Parallel vs. distributed all-to-all aggregation	18
2.2. Increased difficulty of aggregation over rumor spreading	22
3.1. The (un)expected convergence of the power iteration	31
4.1. Flow-recovery graph for two connected nodes.	60
4.2. History-recovery graph for two connected nodes.	67
4.3. Slow-down caused by silent message loss on different topologies	72
4.4. Recovery overhead: Push-Flow vs. LiMoSense	73
4.5. Edge in an asynchronicity graph based on $\tau_{\min} = 1$ and $\tau_{\max} = 4$	74
4.6. Flows in a bus network after convergence.	78
4.7. (In)Accuracy caused by link-based fault tolerance mechanisms	79
4.8. Restarts caused by permanent failure handling	80
4.9. Accuracy of the Push-Cancel-Flow algorithm	82
4.10. Efficient permanent failure handling by Push-Cancel-Flow	83
4.11. Overhead caused by bit flips	84
5.1. Convergence behavior of Push-Sum	89
5.2. Performance of simple parameter estimation (random geometric graph)	100
5.3. Performance of simple parameter estimation (hypercube)	101
5.4. Performance of simple parameter estimation (square lattice)	102
5.5. Performance of simple parameter estimation (fully connected)	103
5.6. Performance of bound-and-float algorithm (random geometric graph)	104
5.7. Performance of bound-and-float algorithm (hypercube)	105
5.8. Performance of bound-and-float algorithm (square lattice)	106
5.9. Performance of bound-and-float algorithm (fully connected)	107
6.1. Visualization of the error threshold phenomenon	114
6.2. Performance of XMVP($d_H^{\max}$) using different levels of sparsification $d_H^{\max}$	119
6.3. Overall runtime for computing the dominating eigenvector of $W = QF$	126
6.4. Speedups of advanced methods over simple power iteration	127

List of Tables

2.1. Summary of our assumptions on parallel and distributed systems. . . .	13
4.1. Description of failures we aim to handle	51
4.2. Parameters of the graphs used in the experiments (cf. Figure 4.3). . . .	73
5.1. Graph structures for accuracy estimation experiments	98

List of Algorithms

2.1. Skeleton of a parallel all-to-all aggregation (for $n = 2^d$)	16
2.2. Skeleton of a distributed aggregation	16
3.1. Generic linear iteration-based algorithm	34
3.2. Push-Sum	36
3.3. An implementable truly asynchronous variant of Algorithm 3.2	38
3.4. Schema of an aggregation-based distributed computation	46
3.5. SUMMA for computing $C = AB$ at node (i, j)	47
4.1. Push-Flow	57
4.2. Generic shape of flow-based algorithms	59
4.3. Generic shape of history-based algorithms	64
4.4. LiMoSense	65
4.5. Push-Cancel-Flow	81
5.1. Bound-and-Float algorithm for a k -dimensional aggregation	96
6.1. Sparse matrix vector multiplication $y = Wx$ (XMVP($d_H^{\max}$))	118
6.2. FMMP based on (6.9)	124
6.3. GPU/OpenCL ready implementation of (6.9)	125

Bibliography

- Adachi, T., Shida, N., Miura, K., Sumimoto, S., Uno, A., Kurokawa, M., Shoji, F., and Yokokawa, M. (2013). “The design of ultra scalable MPI collective communication on the K computer”. In: *Computer Science - Research and Development* **28**(2-3), pp. 147–155 (see pp. [13](#), [15](#)).
DOI: [10.1007/s00450-012-0211-7](#)
- Ajima, Y., Sumimoto, S., and Shimizu, T. (2009). “Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers”. In: *Computer* **42**(11), pp. 36–40 (see p. [13](#)).
DOI: [10.1109/MC.2009.370](#)
- Alexandrov, A., Ionescu, M. F., Schauser, K. E., and Scheiman, C. (1995). “LogGP: Incorporating Long Messages into the LogP Model—One Step Closer Towards a Realistic Model for Parallel Computation”. In: *Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures*. SPAA 1995, pp. 95–105 (see p. [20](#)).
DOI: [10.1145/215399.215427](#)
- Almeida, P. S., Baquero, C., Farach-Colton, M., Jesus, P., and Mosteiro, M. A. (2011). “Fault-Tolerant Aggregation: Flow-Updating Meets Mass-Distribution”. In: *Proceedings of the 15th International Conference on Principles of Distributed Systems*, pp. 513–527 (see p. [52](#)).
DOI: [10.1007/978-3-642-25873-2_35](#)
- Anfinson, C. and Luk, F. (1988). “A Linear Algebraic Model of Algorithmic-Based Fault Tolerance”. In: *Proceedings of the International Conference on Systolic Arrays, 1988*, pp. 483–493 (see p. [54](#)).
DOI: [10.1109/ARRAYS.1988.18085](#)
- Avin, C. and Elsässer, R. (2013). “Faster Rumor Spreading: Breaking the $\log n$ Barrier”. In: *Distributed Computing — 27th International Symposium*. DISC 2013, pp. 209–223 (see p. [21](#)).
DOI: [10.1007/978-3-642-41527-2_15](#)
- Aysal, T., Yildiz, M., Sarwate, A., and Scaglione, A. (2009). “Broadcast Gossip Algorithms for Consensus”. In: *IEEE Transactions on Signal Processing* **57**(7), pp. 2748–2761 (see p. [40](#)).
DOI: [10.1109/TSP.2009.2016247](#)
- Bahi, J., Contassot-Vivier, S., Couturier, R., and Vernier, F. (2005). “A decentralized convergence detection algorithm for asynchronous parallel iterative algorithms”.

- In: *IEEE Transactions on Parallel and Distributed Systems* **16**(1), pp. 4–13 (see p. 87).
DOI: [10.1109/TPDS.2005.2](https://doi.org/10.1109/TPDS.2005.2)
- Bai, Y., Gansterer, W.N., and Ward, R.C. (2004). “Block tridiagonalization of “effectively” sparse symmetric matrices”. In: *ACM Transactions on Mathematical Software* **30**(3), pp. 326–352 (see pp. 117, 119).
DOI: [10.1145/1024074.1024078](https://doi.org/10.1145/1024074.1024078)
- Barabási, A.-L. and Albert, R. (1999). “Emergence of Scaling in Random Networks”. In: *Science* **286**(5439), pp. 509–512 (see p. 20).
DOI: [10.1126/science.286.5439.509](https://doi.org/10.1126/science.286.5439.509)
- Benezit, F., Blondel, V., Thiran, P., Tsitsiklis, J., and Vetterli, M. (2010). “Weighted Gossip: Distributed Averaging using non-doubly stochastic matrices”. In: *2010 IEEE International Symposium on Information Theory Proceedings*, pp. 1753–1757 (see pp. 26, 27, 36, 40, 41).
DOI: [10.1109/ISIT.2010.5513273](https://doi.org/10.1109/ISIT.2010.5513273)
- Benezit, F., Dimakis, A., Thiran, P., and Vetterli, M. (2010). “Order-Optimal Consensus Through Randomized Path Averaging”. In: *IEEE Transactions on Information Theory* **56**(10), pp. 5150–5167 (see pp. 46, 48, 140).
DOI: [10.1109/TIT.2010.2060050](https://doi.org/10.1109/TIT.2010.2060050)
- Bertsekas, D.P. and Tsitsiklis, J.N. (1989). “Convergence Rate and Termination of Asynchronous Iterative Algorithms”. In: *Proceedings of the 3rd International Conference on Supercomputing*. ICS 1989, pp. 461–470 (see p. 87).
DOI: [10.1145/318789.318894](https://doi.org/10.1145/318789.318894)
- Boyd, S., Ghosh, A., Prabhakar, B., and Shah, D. (2006). “Randomized Gossip Algorithms”. In: *IEEE Transactions on Information Theory* **52**(6), pp. 2508–2530 (see pp. 17, 26, 33, 37, 41, 42, 45, 48, 71, 140, 159).
DOI: [10.1109/TIT.2006.874516](https://doi.org/10.1109/TIT.2006.874516)
- Breshears, C. (2009). *The Art of Concurrency*. O’Reilly Media (see p. 126).
ISBN: [978-0-596-52153-0](https://doi.org/10.1007/978-0-596-52153-0)
- Brin, S. and Page, L. (1998). “The Anatomy of a Large-Scale Hypertextual Web Search Engine”. In: *Computer Networks* **30**(1–7), pp. 107–117 (see p. 29).
DOI: [10.1016/S0169-7552\(98\)00110-X](https://doi.org/10.1016/S0169-7552(98)00110-X)
- Cappello, F., Geist, A., Gropp, B., Kale, L., Kramer, B., and Snir, M. (2009). “Toward Exascale Resilience”. In: *International Journal of High Performance Computing Applications* **23**(4), pp. 374–388 (see pp. 4, 53).
DOI: [10.1177/1094342009347767](https://doi.org/10.1177/1094342009347767)
- Charr, J.-C., Couturier, R., and Laiymani, D. (2010). “A decentralized and fault tolerant convergence detection algorithm for asynchronous iterative algorithms”. English. In: *The Journal of Supercomputing* **53**(2), pp. 269–292 (see p. 87).
DOI: [10.1007/s11227-009-0293-6](https://doi.org/10.1007/s11227-009-0293-6)
- Chen, J. and Pandurangan, G. (2012). “Almost-Optimal Gossip-Based Aggregate

- Computation". In: *SIAM Journal on Computing* **41**(3), pp. 455–483 (see pp. [5](#), [21](#), [22](#)).
- DOI: [10.1137/100793104](#)
- Chen, Y., Tron, R., Terzis, A., and Vidal, R. (2010). "Corrective consensus: Converging to the exact average". In: *Proceedings of the 49th IEEE Conference on Decision and Control*. CDC 2010, pp. 1221–1228 (see p. [53](#)).
- DOI: [10.1109/CDC.2010.5717925](#)
- Chen, Z. (2011). "Algorithm-Based Recovery for Iterative Methods without Checkpointing". In: *Proceedings of the 20th International Symposium on High Performance Distributed Computing*. HPDC '11, pp. 73–84 (see p. [54](#)).
- DOI: [10.1145/1996130.1996142](#)
- Chen, Z. (2013). "Online-ABFT: An Online Algorithm Based Fault Tolerance Scheme for Soft Error Detection in Iterative Methods". In: *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPOPP 2013, pp. 167–176 (see p. [54](#)).
- DOI: [10.1145/2442516.2442533](#)
- Chen, Z. and Dongarra, J. J. (2008). "Algorithm-Based Fault Tolerance for Fail-Stop Failures". In: *IEEE Transactions on Parallel and Distributed Systems* **19**(12), pp. 1628–1641 (see p. [54](#)).
- DOI: [10.1109/TPDS.2008.58](#)
- Chierichetti, F., Lattanzi, S., and Panconesi, A. (2010). "Almost tight bounds for rumour spreading with conductance". In: *Proceedings of the 42nd ACM Symposium on Theory of Computing*. STOC 2010, pp. 399–408 (see p. [21](#)).
- DOI: [10.1145/1806689.1806745](#)
- Chierichetti, F., Lattanzi, S., and Panconesi, A. (2011). "Rumor spreading in social networks". In: *Theoretical Computer Science* **412**(24), pp. 2602–2610 (see p. [20](#)).
- DOI: [10.1016/j.tcs.2010.11.001](#)
- Cooley, J. W. and Tukey, J. W. (1965). "An Algorithm for the Machine Calculation of Complex Fourier Series". In: *Mathematics of Computation* **19**(90), pp. 297–301 (see pp. [150](#), [154](#)).
- DOI: [10.1090/S0025-5718-1965-0178586-1](#)
- Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. (2001). *Introduction to Algorithms*. 2nd ed. McGraw-Hill Higher Education (see pp. [117](#), [121](#), [153](#)).
- ISBN: [0-07-013151-1](#)
- Cullum, J. K. and Willoughby, R. A. (2002). *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*. Society for Industrial and Applied Mathematics (see p. [31](#)).
- DOI: [10.1137/1.9780898719192](#)
- Davies, T. and Chen, Z. (2013). "Correcting soft errors online in LU factorization". In: *The 22nd International Symposium on High-Performance Parallel and Distributed Computing*. HPDC 2013, pp. 167–178 (see p. [54](#)).
- DOI: [10.1145/2462902.2462920](#)

- DeGroot, M. H. (1974). “Reaching a Consensus”. In: *Journal of the American Statistical Association* **69**(345), pp. 118–121 (see p. 26).
DOI: [10.1080/01621459.1974.10480137](https://doi.org/10.1080/01621459.1974.10480137)
- Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. (1987). “Epidemic algorithms for replicated database maintenance”. In: *Proceedings of the 6th annual ACM Symposium on Principles of distributed computing*. PODC 1987, pp. 1–12 (see p. 19).
DOI: [10.1145/41840.41841](https://doi.org/10.1145/41840.41841)
- Dhillon, B. S. (1987). *Reliability in computer system design*. Ablex Publishing Corporation (see p. 54).
ISBN: [0-89391-412-6](https://www.isbn-international.org/product/0-89391-412-6)
- Dimakis, A., Sarwate, A., and Wainwright, M. (2008). “Geographic Gossip: Efficient Averaging for Sensor Networks”. In: *IEEE Transactions on Signal Processing* **56**(3), pp. 1205–1216 (see p. 46).
DOI: [10.1109/TSP.2007.908946](https://doi.org/10.1109/TSP.2007.908946)
- Dimakis, A., Kar, S., Moura, J. M. F., Rabbat, M., and Scaglione, A. (2010). “Gossip Algorithms for Distributed Signal Processing”. In: *Proceedings of the IEEE* **98**(11), pp. 1847–1864 (see pp. 26, 33, 53).
DOI: [10.1109/JPROC.2010.2052531](https://doi.org/10.1109/JPROC.2010.2052531)
- Doerr, B., Fouz, M., and Friedrich, T. (2011). “Social Networks Spread Rumors in Sublogarithmic Time”. In: *Proceedings of the 43rd ACM Symposium on Theory of Computing*. STOC 2011, pp. 21–30 (see p. 21).
DOI: [10.1145/1993636.1993640](https://doi.org/10.1145/1993636.1993640)
- Doerr, B., Friedrich, T., and Sauerwald, T. (2009). “Quasirandom Rumor Spreading: Expanders, Push vs. Pull, and Robustness”. In: *Automata, Languages and Programming, 36th International Colloquium*. ICALP 2009, pp. 366–377 (see p. 20).
DOI: [10.1007/978-3-642-02927-1_31](https://doi.org/10.1007/978-3-642-02927-1_31)
- Dongarra, J. J., Beckman, P., *et al.* (2011). “The International Exascale Software Project roadmap”. In: *International Journal of High Performance Computing Applications* **25**(1), pp. 3–60 (see p. 3, 4, 53).
DOI: [10.1177/1094342010391989](https://doi.org/10.1177/1094342010391989)
- Dongarra, J. J., Luszczek, P., and Petit, A. (2003). “The LINPACK Benchmark: past, present and future”. In: *Concurrency and Computation: Practice and Experience* **15**(9), pp. 803–820 (see p. 3).
DOI: [10.1002/cpe.728](https://doi.org/10.1002/cpe.728)
- Dongarra, J. J. *et al.* (2002). “Basic Linear Algebra Subprograms Technical (Blast) Forum Standard (1)”. In: *International Journal of High Performance Computing Applications* **16**(1), pp. 1–111 (see p. 1).
DOI: [10.1177/10943420020160020101](https://doi.org/10.1177/10943420020160020101)
- Dongarra, J. J. *et al.* (2002). “Basic Linear Algebra Subprograms Technical (Blast) Forum Standard (2)”. In: *International Journal of High Performance Computing Applications* **16**(2), pp. 115–199 (see p. 1).
DOI: [10.1177/10943420020160020101](https://doi.org/10.1177/10943420020160020101)

- Drake, J. (1993). “Rates of spontaneous mutation among RNA viruses.” In: *Proceedings of The National Academy of Sciences* **90**(9), pp. 4171–4175 (see p. 115).
DOI: [10.1073/pnas.90.9.4171](https://doi.org/10.1073/pnas.90.9.4171)
- Dress, A.W.M. and Rumschitzki, D.S. (1988). “Evolution on sequence space and tensor products of representation spaces”. In: *Acta Applicandae Mathematicae* **11**(2), pp. 103–115 (see p. 120).
DOI: [10.1007/BF00047282](https://doi.org/10.1007/BF00047282)
- Drineas, P., Drinea, E., and Huggins, P.S. (2003). “An Experimental Evaluation of a Monte-Carlo Algorithm for Singular Value Decomposition”. In: *Advances in Informatics*. Vol. 2563. Lecture Notes in Computer Science, pp. 279–296 (see p. 122).
DOI: [10.1007/3-540-38076-0_19](https://doi.org/10.1007/3-540-38076-0_19)
- Drubin, M. (1971). “Kronecker Product Factorization of the FFT Matrix”. In: *IEEE Transactions on Computers* **C-20**(5), pp. 590–593 (see p. 154).
DOI: [10.1109/T-C.1971.223306](https://doi.org/10.1109/T-C.1971.223306)
- Du, P., Luszczek, P., and Dongarra, J. (2012). “High Performance Dense Linear System Solver with Resilience to Multiple Soft Errors”. In: *Procedia Computer Science* **9**, pp. 216–225 (see p. 55).
DOI: [10.1016/j.procs.2012.04.023](https://doi.org/10.1016/j.procs.2012.04.023)
- Du, P., Luszczek, P., Tomov, S., and Dongarra, J. (2011). “Soft error resilient QR factorization for hybrid system with GPGPU”. In: *Proceedings of the Second Workshop on Scalable Algorithms for Large-Scale Systems*. Scala 2011, pp. 11–14 (see p. 55).
DOI: [10.1145/2133173.2133179](https://doi.org/10.1145/2133173.2133179)
- Eigen, M. (1971). “Selforganization of matter and the evolution of biological macromolecules”. In: *Naturwissenschaften* **58**(10), pp. 465–523 (see pp. 111–113).
DOI: [10.1007/BF00623322](https://doi.org/10.1007/BF00623322)
- Eigen, M. (2002). “Error catastrophe and antiviral strategy”. In: *Proceedings of the National Academy of Sciences of the United States of America* **99**(21), pp. 13374–13376 (see p. 115).
DOI: [10.1073/pnas.212514799](https://doi.org/10.1073/pnas.212514799)
- Eigen, M. and Schuster, P. (1977). “A principle of natural self-organization”. In: *Naturwissenschaften* **64**(11), pp. 541–565 (see p. 112).
DOI: [10.1007/BF00450633](https://doi.org/10.1007/BF00450633)
- Elsässer, R. and Sauerwald, T. (2008). “The Power of Memory in Randomized Broadcasting”. In: *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA 2008, pp. 218–227 (see p. 21).
URL: <http://doi.acm.org/10.1145/1347082.1347107>
- Elsässer, R. and Sauerwald, T. (2010). “Discrete Load Balancing is (Almost) As Easy As Continuous Load Balancing”. In: *Proceedings of the 29th Symposium on Principles of Distributed Computing*. PODC 2010, pp. 346–354 (see p. 28).
DOI: [10.1145/1835698.1835780](https://doi.org/10.1145/1835698.1835780)
- Elstrodt, J. (2011). *Maß- und Integrationstheorie*. 7th ed. Springer-Lehrbuch (see p. 40).
ISBN: [978-3-642-17905-1](https://doi.org/10.1007/978-3-642-17905-1)

- Erseghe, T., Zennaro, D., Dall’Anese, E., and Vangelista, L. (2011). “Fast Consensus by the Alternating Direction Multipliers Method”. In: *IEEE Transactions on Signal Processing* **59**(11), pp. 5523–5537 (see p. 27).
DOI: [10.1109/TSP.2011.2162831](https://doi.org/10.1109/TSP.2011.2162831)
- Eyal, I., Keidar, I., and Rom, R. (2011). “LiMoSense – Live Monitoring in Dynamic Sensor Networks”. In: *Proceedings of the 7th International Symposium on Algorithms for Sensor Systems, Wireless Ad Hoc Networks and Autonomous Mobile Entities*. ALGOSENSORS 2011, pp. 72–85 (see pp. 52, 55, 64–66, 141).
DOI: [10.1007/978-3-642-28209-6_7](https://doi.org/10.1007/978-3-642-28209-6_7)
- Feige, U., Peleg, D., Raghavan, P., and Upfal, E. (1990). “Randomized broadcast in networks”. In: *Random Structures & Algorithms* **1**(4), pp. 447–460 (see p. 20).
DOI: [10.1002/rsa.3240010406](https://doi.org/10.1002/rsa.3240010406)
- Fiala, D., Mueller, F., Engelmann, C., Riesen, R., Ferreira, K., and Brightwell, R. (2012). “Detection and correction of silent data corruption for large-scale high-performance computing”. In: *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pp. 1–12 (see p. 54).
DOI: [10.1109/SC.2012.49](https://doi.org/10.1109/SC.2012.49)
- Friedrich, T., Gairing, M., and Sauerwald, T. (2012). “Quasirandom Load Balancing”. In: *SIAM Journal on Computing* **41**(4), pp. 747–771 (see p. 28).
DOI: [10.1137/100799216](https://doi.org/10.1137/100799216)
- Frieze, A. and Molloy, M. (1994). “Broadcasting in random graphs”. In: *Discrete Applied Mathematics* **54**(1), pp. 77–79 (see p. 20).
DOI: [10.1016/0166-218X\(94\)00033-6](https://doi.org/10.1016/0166-218X(94)00033-6)
- Frigo, M. and Johnson, S. (2005). “The Design and Implementation of FFTW3”. In: *Proceedings of the IEEE* **93**(2), pp. 216–231 (see p. 151).
DOI: [10.1109/JPROC.2004.840301](https://doi.org/10.1109/JPROC.2004.840301)
- Frobenius, G. (1908). “Über Matrizen aus positiven Elementen”. In: *Sitzungsberichte der Preussischen Akademie der Wissenschaften* **1908**, pp. 471–476 (see p. 155).
URL: <http://archive.org/stream/sitzungsberichte1908deutsch>
- Frobenius, G. (1912). “Über Matrizen aus nicht negativen Elementen”. In: *Sitzungsberichte der Preussischen Akademie der Wissenschaften* **1912**, pp. 456–477 (see p. 155).
ISBN: [978-3-11-197350-0](https://doi.org/10.1007/978-3-11-197350-0)
- Gansterer, W.N., Niederbrucker, G., Straková, H., and Schulze Grotthoff, S. (2011). “Robust Distributed Orthogonalization Based on Randomized Aggregation”. In: *Proceedings of the Second Workshop on Scalable Algorithms for Large-Scale Systems*. ScalA 2011, pp. 7–10 (see pp. 52, 55, 57).
DOI: [10.1145/2133173.2133177](https://doi.org/10.1145/2133173.2133177)
- Gansterer, W.N., Niederbrucker, G., Straková, H., and Schulze Grotthoff, S. (2013). “Scalable and Fault Tolerant Orthogonalization Based on Randomized Distributed Data Aggregation”. In: *Journal of Computational Science* **4**(6), pp. 480–488 (see pp. 46, 51, 52, 55, 57).
DOI: [10.1016/j.jocs.2013.01.006](https://doi.org/10.1016/j.jocs.2013.01.006)

- Georgiou, C., Gilbert, S., Guerraoui, R., and Kowalski, D. R. (2013). “Asynchronous Gossip”. In: *Journal of the ACM* **60**(2), 11:1–11:42 (see pp. 37, 73).
DOI: [10.1145/2450142.2450147](https://doi.org/10.1145/2450142.2450147)
- Giakkoupis, G. (2011). “Tight bounds for rumor spreading in graphs of a given conductance”. In: *Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science*. STACS 2011, pp. 57–68 (see p. 20).
DOI: [10.4230/LIPIcs.STACS.2011.57](https://doi.org/10.4230/LIPIcs.STACS.2011.57)
- Giakkoupis, G. (2014). “Tight Bounds for Rumor Spreading with Vertex Expansion”. In: *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA 2014, pp. 801–815 (see p. 21).
DOI: [10.1137/1.9781611973402.59](https://doi.org/10.1137/1.9781611973402.59)
- Giakkoupis, G. and Sauerwald, T. (2012). “Rumor Spreading and Vertex Expansion”. In: *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA 2012, pp. 1623–1641 (see pp. 21, 159).
DOI: [10.1137/1.9781611973099.129](https://doi.org/10.1137/1.9781611973099.129)
- Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations*. 3rd ed. The Johns Hopkins University Press (see pp. 123, 143, 145).
ISBN: 0-8018-5413-X
- Golub, G. H. and Varga, R. S. (1961). “Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order Richardson iterative methods”. English. In: *Numerische Mathematik* **3**(1), pp. 157–168 (see p. 45).
DOI: [10.1007/BF01386014](https://doi.org/10.1007/BF01386014)
- Hadjicostis, C., Dominguez-Garcia, A., and Vaidya, N. (2012). “Resilient average consensus in the presence of heterogeneous packet dropping links”. In: *51st Annual Conference on Decision and Control*. CDC 2012, pp. 106–111 (see pp. 33, 38, 52, 53, 55, 66).
DOI: [10.1109/CDC.2012.6426666](https://doi.org/10.1109/CDC.2012.6426666)
- Haeupler, B. (2013). “Simple, Fast and Deterministic Gossip and Rumor Spreading”. In: *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA 2013, pp. 705–716 (see p. 19).
DOI: [10.1137/1.9781611973105.51](https://doi.org/10.1137/1.9781611973105.51)
- Hamming, R. W. (1950). “Error Detecting and Error Correcting Codes”. In: *Bell System Technical Journal* **29**(2), pp. 147–160 (see p. 128).
DOI: [10.1002/j.1538-7305.1950.tb00463.x](https://doi.org/10.1002/j.1538-7305.1950.tb00463.x)
- Hoory, S., Linial, N., and Wigderson, A. (2006). “Expander graphs and their applications”. In: *Bulletin of the American Mathematical Society* **43**, pp. 439–561 (see p. 160).
DOI: [10.1090/S0273-0979-06-01126-8](https://doi.org/10.1090/S0273-0979-06-01126-8)
- Huang, K.-H. and Abraham, J. (1984). “Algorithm-Based Fault Tolerance for Matrix Operations”. In: *IEEE Transactions on Computers* **C-33**(6), pp. 518–528 (see p. 54).
DOI: [10.1109/TC.1984.1676475](https://doi.org/10.1109/TC.1984.1676475)
- Jesus, P., Baquero, C., and Almeida, P. S. (2009). “Fault-Tolerant Aggregation by

- Flow Updating”. In: *Proceedings of the International Conference on Distributed Applications and Interoperable Systems*. DAIS 2009, pp. 73–86 (see p. 52).
DOI: [10.1007/978-3-642-02164-0_6](https://doi.org/10.1007/978-3-642-02164-0_6)
- Jesus, P., Baquero, C., and Almeida, P.S. (2010). “Fault-Tolerant Aggregation for Dynamic Networks”. In: *Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems*. SRDS 2010, pp. 37–43 (see p. 52).
DOI: [10.1109/SRDS.2010.13](https://doi.org/10.1109/SRDS.2010.13)
- Johnson, J. and Püschel, M. (2000). “In Search of the Optimal Walsh-Hadamard Transform”. In: *Proceedings of the 25th International Conference on Acoustics, Speech, and Signal Processing, Volume 6*. ICASSP 2000, pp. 3347–3350 (see p. 154).
DOI: [10.1109/ICASSP.2000.860117](https://doi.org/10.1109/ICASSP.2000.860117)
- Johnsson, S. and Ho, C.-T. (1989). “Optimum broadcasting and personalized communication in hypercubes”. In: *IEEE Transactions on Computers* **38**(9), pp. 1249–1268 (see p. 15).
DOI: [10.1109/12.29465](https://doi.org/10.1109/12.29465)
- Kar, S. and Moura, J.M.F. (2008). “Sensor Networks With Random Links: Topology Design for Distributed Consensus”. In: *IEEE Transactions on Signal Processing* **56**(7), pp. 3315–3326 (see p. 46).
DOI: [10.1109/TSP.2008.920143](https://doi.org/10.1109/TSP.2008.920143)
- Kar, S. and Moura, J.M.F. (2009). “Distributed Consensus Algorithms in Sensor Networks With Imperfect Communication: Link Failures and Channel Noise”. In: *IEEE Transactions on Signal Processing* **57**(1), pp. 355–369 (see p. 53).
DOI: [10.1109/TSP.2008.2007111](https://doi.org/10.1109/TSP.2008.2007111)
- Kar, S. and Moura, J.M.F. (2010). “Distributed Consensus Algorithms in Sensor Networks: Quantized Data and Random Link Failures”. In: *IEEE Transactions on Signal Processing* **58**(3), pp. 1383–1400 (see p. 53).
DOI: [10.1109/TSP.2009.2036046](https://doi.org/10.1109/TSP.2009.2036046)
- Kar, S. and Moura, J.M.F. (2013). “Consensus + Innovations Distributed Inference over Networks: Cooperation and sensing in networked systems”. In: *IEEE Signal Processing Magazine* **30**(3), pp. 99–109 (see p. 26).
DOI: [10.1109/MSP.2012.2235193](https://doi.org/10.1109/MSP.2012.2235193)
- Karp, R., Schindelhauer, C., Shenker, S., and Vocking, B. (2000). “Randomized rumor spreading”. In: *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*. FOCS 2000, pp. 565–574 (see pp. 19–22).
DOI: [10.1109/SFCS.2000.892324](https://doi.org/10.1109/SFCS.2000.892324)
- Kempe, D., Dobra, A., and Gehrke, J. (2003). “Gossip-Based Computation of Aggregate Information”. In: *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*. FOCS 2003, pp. 482–491 (see pp. 17, 27, 35, 36).
DOI: [10.1109/SFCS.2003.1238221](https://doi.org/10.1109/SFCS.2003.1238221)
- Kempe, D. and McSherry, F. (2004). “A decentralized algorithm for spectral analysis”.

- In: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. STOC 2004, pp. 561–568 (see p. 87).
DOI: [10.1145/1007352.1007438](https://doi.org/10.1145/1007352.1007438)
- Luk, F. T. and Park, H. (1988). “Fault-tolerant matrix triangularizations on systolic arrays”. In: *IEEE Transactions on Computers* **37**(11), pp. 1434–1438 (see p. 54).
DOI: [10.1109/12.8712](https://doi.org/10.1109/12.8712)
- Mascagni, M. and Karaivanova, A. (2002). “A Parallel Quasi-Monte Carlo Method for Computing Extremal Eigenvalues”. In: *Monte Carlo and Quasi-Monte Carlo Methods 2000*, pp. 369–380 (see p. 122).
DOI: [10.1007/978-3-642-56046-0_25](https://doi.org/10.1007/978-3-642-56046-0_25)
- Meyer, C. D. (2000). *Matrix analysis and applied linear algebra*. Society for Industrial and Applied Mathematics (see pp. 155–158).
ISBN: 0-89871-454-0
- Mosk-Aoyama, D. and Shah, D. (2006). “Computing Separable Functions via Gossip”. In: *Proceedings of the 25th annual ACM symposium on Principles of distributed computing*. PODC 2006, pp. 113–122 (see pp. 23, 27, 53).
DOI: [10.1145/1146381.1146401](https://doi.org/10.1145/1146381.1146401)
- Mota, J., Xavier, J., Aguiar, P., and Püschel, M. (2012). “ADMM for consensus on colored networks”. In: *Proceedings of the 51st Annual Conference on Decision and Control*. CDC 2012, pp. 5116–5121 (see p. 27).
DOI: [10.1109/CDC.2012.6426225](https://doi.org/10.1109/CDC.2012.6426225)
- Muthukrishnan, S., Ghosh, B., and Schultz, M. H. (1998). “First- and Second-Order Diffusive Methods for Rapid, Coarse, Distributed Load Balancing”. In: *Theory of Computing Systems* **31**(4), pp. 331–354 (see p. 45).
DOI: [10.1007/s002240000092](https://doi.org/10.1007/s002240000092)
- Niederbrucker, G. and Gansterer, W. N. (2014). *Robust Distributed Aggregation*. (under revision, IEEE Transactions on Signal Processing) (see pp. 40, 51, 55, 58).
- Niederbrucker, G. (2013). “Structured Preconditioners for Matrix-free Iterative Methods”. Master’s Thesis. Vienna University of Technology, Faculty of Mathematics and Geoinformation, Institute of Analysis and Scientific Computing (see pp. 111, 123, 128, 130, 131, 133, 137, 145, 146, 154).
URL: <http://permalink.obvsg.at/AC11105284>
- Niederbrucker, G. and Gansterer, W. N. (2011a). “A Fast Solver for Modeling the Evolution of Virus Populations”. In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. SC 2011, 74:1–74:11 (see pp. 111, 124, 132, 137).
DOI: [10.1145/2063384.2063483](https://doi.org/10.1145/2063384.2063483)
- Niederbrucker, G. and Gansterer, W. N. (2011). “Efficient Solution of Evolution Models for Virus Populations”. In: *Procedia Computer Science* **4**. Proceedings of the International Conference on Computational Science, ICCS 2011, pp. 126–135 (see pp. 116, 126, 137).
DOI: [10.1016/j.procs.2011.04.014](https://doi.org/10.1016/j.procs.2011.04.014)
- Niederbrucker, G. and Gansterer, W. N. (2013). “Robust Gossip-Based Aggregation:

- A Practical Point of View”. In: *Proceedings of the 2013 Meeting on Algorithm Engineering & Experiments*. ALENEX 2013, pp. 133–147 (see pp. 50, 76, 84).
 URL: <http://knowledgecenter.siam.org/0238-000028/>
- Niederbrucker, G., Straková, H., and Gansterer, W. N. (2012). “Improving Fault Tolerance and Accuracy of Distributed Reduction Algorithms”. In: *Proceedings of the Third Workshop on Scalable algorithms for Large-Scale Systems*. ScalA 2012, pp. 7–10 (see pp. 53, 76, 79, 81).
 DOI: [10.1109/SC.Companion.2012.89](https://doi.org/10.1109/SC.Companion.2012.89)
- Nowak, M. and Schuster, P. (1989). “Error thresholds of replication in finite populations mutation frequencies and the onset of muller’s ratchet”. In: *Journal of Theoretical Biology* **137**(4), pp. 375–395 (see pp. 115, 133, 137).
 DOI: [10.1016/S0022-5193\(89\)80036-0](https://doi.org/10.1016/S0022-5193(89)80036-0)
- Olfati-Saber, R., Fax, J. A., and Murray, R. M. (2007). “Consensus and Cooperation in Networked Multi-Agent Systems”. In: *Proceedings of the IEEE* **95**(1), pp. 215–233 (see pp. 26, 33, 38, 53).
 DOI: [10.1109/JPROC.2006.887293](https://doi.org/10.1109/JPROC.2006.887293)
- Perron, O. (1907). “Zur Theorie der Matrices”. In: *Mathematische Annalen* **64**(2), pp. 248–263 (see pp. 155, 158).
 DOI: [10.1007/BF01449896](https://doi.org/10.1007/BF01449896)
- Pittel, B. (1987). “On Spreading a Rumor”. In: *SIAM Journal on Applied Mathematics* **47**(1), pp. 213–223 (see p. 20).
 DOI: [10.1137/0147013](https://doi.org/10.1137/0147013)
- Pješivac-Grbović, J., Angskun, T., Bosilca, G., Fagg, G. E., Gabriel, E., and Dongarra, J. J. (2007). “Performance analysis of MPI collective operations”. In: *Cluster Computing* **10**(2), pp. 127–143 (see p. 15).
 DOI: [10.1007/s10586-007-0012-0](https://doi.org/10.1007/s10586-007-0012-0)
- Plank, J., Kai, L., and Puening, M. (1998). “Diskless checkpointing”. In: *IEEE Transactions on Parallel and Distributed Systems* **9**(10), pp. 972–986 (see p. 54).
 DOI: [10.1109/71.730527](https://doi.org/10.1109/71.730527)
- Püschel, M. *et al.* (2005). “SPIRAL: Code Generation for DSP Transforms”. In: *Proceedings of the IEEE* **93**(2), pp. 232–275 (see pp. 124, 151).
 DOI: [10.1109/JPROC.2004.840306](https://doi.org/10.1109/JPROC.2004.840306)
- Rabbat, M. and Nowak, R. (2004). “Distributed Optimization in Sensor Networks”. In: *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*. IPSN 2004, pp. 20–27 (see p. 27).
 DOI: [10.1145/984622.984626](https://doi.org/10.1145/984622.984626)
- Rabenseifner, R. (1999). “Automatic Profiling of MPI Applications with Hardware Performance Counters”. In: *Proceedings of the 6th European PVM/MPI Users’ Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pp. 35–42 (see p. 14).
 DOI: [10.1007/3-540-48158-3_5](https://doi.org/10.1007/3-540-48158-3_5)

- Rumschitzki, D. S. (1987). “Spectral properties of Eigen evolution matrices”. In: *Journal of Mathematical Biology* **24**(6), pp. 667–680 (see p. 120).
DOI: [10.1007/BF00275509](https://doi.org/10.1007/BF00275509)
- Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. 2nd ed. Society for Industrial and Applied Mathematics (see pp. 29, 31, 44).
ISBN: [0-89871-534-2](https://www.isbn-international.org/product/0-89871-534-2)
- Sanders, P., Speck, J., and Träff, J. L. (2009). “Two-tree Algorithms for Full Bandwidth Broadcast, Reduction and Scan”. In: *Parallel Computing* **35**(12), pp. 581–594 (see p. 15).
DOI: [10.1016/j.parco.2009.09.001](https://doi.org/10.1016/j.parco.2009.09.001)
- Sauerwald, T. and Sun, H. (2012). “Tight Bounds for Randomized Load Balancing on Arbitrary Network Topologies”. In: *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science*. FOCS 2012, pp. 341–350 (see p. 28).
DOI: [10.1109/FOCS.2012.86](https://doi.org/10.1109/FOCS.2012.86)
- Savari, S. A. and Bertsekas, D. P. (1996). “Finite Termination of Asynchronous Iterative Algorithms”. In: *Parallel Comput.* **22**(1), pp. 39–56 (see p. 87).
DOI: [10.1016/0167-8191\(95\)00059-3](https://doi.org/10.1016/0167-8191(95)00059-3)
- Sayed, A. H. (2013). “Diffusion adaptation over networks”. In: *E-Reference Signal Processing. (to appear)* (see p. 27).
URL: <http://arxiv.org/abs/1205.4220>
- Schuster, P. (2006). “Prediction of RNA secondary structures: from theory to models and real molecules”. In: *Reports on Progress in Physics* **69**(5), pp. 1419–1477 (see p. 112).
DOI: [10.1088/0034-4885/69/5/R04](https://doi.org/10.1088/0034-4885/69/5/R04)
- Schuster, P. (2008). *The mathematics of Darwinian systems*. Appendix A4 to the book “From Strange Simplicity to Complex Familiarity: A Treatise on Matter, Information, Life and Thought” by Manfred Eigen. Oxford University Press, pp. 667–700 (see pp. 112, 113, 115, 130, 133).
DOI: [10.1093/acprof:oso/9780198570219.001.0001](https://doi.org/10.1093/acprof:oso/9780198570219.001.0001)
- Schuster, P. (2011). “Mathematical modeling of evolution. Solved and open problems”. In: *Theory in Biosciences* **130**(1), pp. 71–89 (see pp. 112, 113, 115, 126, 130, 131).
DOI: [10.1007/s12064-010-0110-z](https://doi.org/10.1007/s12064-010-0110-z)
- Silvester, J. R. (2000). “Determinants of Block Matrices”. In: *Mathematical Gazette* **84**(501), pp. 460–467 (see p. 69).
URL: <http://www.jstor.org/stable/3620776>
- Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J. (1998). *MPI-The Complete Reference, Volume 1: The MPI Core*. 2nd ed. MIT Press (see p. 12).
ISBN: [0-262-69215-5](https://www.isbn-international.org/product/0-262-69215-5)
- Straková, H. (2013). “Truly Distributed Approaches to Orthogonalization and Orthogonal Iteration on the Basis of Gossip Algorithms”. PhD thesis. University of Vienna, Faculty of Computer Science (see pp. 2, 46, 48, 55, 75, 139).
URL: <http://ubdata.univie.ac.at/AC11020522>
- Straková, H., Gansterer, W. N., and Zemen, T. (2011). “Distributed QR Factorization

- Based on Randomized Algorithms”. In: *Parallel Processing and Applied Mathematics, 9th International Conference, PPAM 2011*, pp. 235–244 (see p. 46).
DOI: [10.1007/978-3-642-31464-3_24](https://doi.org/10.1007/978-3-642-31464-3_24)
- Straková, H., Niederbrucker, G., and Gansterer, W. N. (2013). “Fault Tolerance Properties of Gossip-Based Distributed Orthogonal Iteration Methods”. In: *Procedia Computer Science* **18**, pp. 189–198 (see p. 46).
DOI: [10.1016/j.procs.2013.05.182](https://doi.org/10.1016/j.procs.2013.05.182)
- Swetina, J. and Schuster, P. (1982). “Self-replication with errors: A model for polynucleotide replication”. In: *Biophysical Chemistry* **16**(4), pp. 329–345 (see pp. 115, 116, 133, 137).
DOI: [10.1016/0301-4622\(82\)87037-3](https://doi.org/10.1016/0301-4622(82)87037-3)
- Thakur, R., Rabenseifner, R., and Gropp, W. (2005). “Optimization of Collective Communication Operations in MPICH”. In: *International Journal of High Performance Computing Applications* **19**(1), pp. 49–66 (see p. 15).
DOI: [10.1177/1094342005051521](https://doi.org/10.1177/1094342005051521)
- Thompson, C. J. and McBride, J. L. (1974). “On Eigen’s theory of the self-organization of matter and the evolution of biological macromolecules”. In: *Mathematical Biosciences* **21**(1–2), pp. 127–142 (see p. 113).
DOI: [10.1016/0025-5564\(74\)90110-2](https://doi.org/10.1016/0025-5564(74)90110-2)
- Tolentino, M. and Cameron, K. (2012). “The Optimist, the Pessimist, and the Global Race to Exascale in 20 Megawatts”. In: *Computer* **45**(1), pp. 95–97 (see p. 3).
DOI: [10.1109/MC.2012.34](https://doi.org/10.1109/MC.2012.34)
- Trefethen, L. N. and Bau, D. (1997). *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics (see pp. 17, 30, 122, 123).
ISBN: 0-89871-361-7
- Tsianos, K., Lawlor, S., and Rabbat, M. (2012). “Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning”. In: *50th Annual Allerton Conference on Communication, Control, and Computing*. Allerton 2012, pp. 1543–1550 (see p. 36).
DOI: [10.1109/Allerton.2012.6483403](https://doi.org/10.1109/Allerton.2012.6483403)
- Tsitsiklis, J. N. (1984). “Problems in decentralized decision making and computation”. PhD thesis. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (see p. 26).
URL: <http://hdl.handle.net/1721.1/15254>
- Van De Geijn, R. A. and Watts, J. (1997). “SUMMA: scalable universal matrix multiplication algorithm”. In: *Concurrency: Practice and Experience* **9**(4), pp. 255–274 (see p. 47).
DOI: [10.1002/\(SICI\)1096-9128\(199704\)9:4<255::AID-CPE250>3.0.CO;2-2](https://doi.org/10.1002/(SICI)1096-9128(199704)9:4<255::AID-CPE250>3.0.CO;2-2)
- Van Loan, C. F. (1992). *Computational Frameworks for the Fast Fourier Transform*. Society for Industrial and Applied Mathematics (see p. 154).
DOI: [10.1137/1.9781611970999](https://doi.org/10.1137/1.9781611970999)

Van Loan, C.F. (2000). “The ubiquitous Kronecker product”. In: *Journal of Computational and Applied Mathematics* **123**, pp. 85–100 (see pp. [120](#), [151](#)).

DOI: [10.1016/S0377-0427\(00\)00393-9](https://doi.org/10.1016/S0377-0427(00)00393-9)

Wu, P., Ding, C., Chen, L., Gao, F., Davies, T., Karlsson, C., and Chen, Z. (2011). “Fault tolerant matrix-matrix multiplication: correcting soft errors on-line”. In: *Proceedings of the Second Workshop on Scalable Algorithms for Large-Scale Systems*. ScalA 2011, pp. 25–28 (see p. [54](#)).

DOI: [10.1145/2133173.2133185](https://doi.org/10.1145/2133173.2133185)

Index

- $[A]_{i,j}$, 143
- Γ_k , 114
- χ , 97
- $\chi(i)_j$, 98
- $\det(A)$, 143
- $\otimes$, 145
- $\mathcal{E}$, 12
- $\mathcal{G}$, 12
- $\mathcal{N}_i$, 12
- $\mathcal{V}$, 12
- $i:j$, 144
- $\text{rank}(A)$, 143
- $\rho(A)$, 144
- $\sigma(A)$, 144
- $i:k:j$, 144
- $C(n)$, 160
- $\text{ER}(n, p)$, 160
- $G(n, d)$, 160
- $\text{HC}(d)$, 160
- $L(n)$, 161
- $\text{tr}(A)$, 143
- ε -accurate, 16
- $x_{r \times c}$, 144
- Accuracy, 17
- Adjacency matrix, 157
- Aggregate function, 14
- Aggregation, 5
 - address-oblivious, 21
 - all-to-all, 14
 - continuous, 22
 - discrete, 22
 - hypercube, 15, 18
- Algorithm
 - address-oblivious, 19
 - consensus, 26
 - gossip, 26
 - push-based, 49
- All-to-all aggregation, 14
- Asymptotic bounds
 - $\mathcal{O}(g(n))$, 145
 - $\Omega(g(n))$, 145
 - $\Theta(g(n))$, 145
- Asynchronicity, 36, 73
- Average consensus, 26
- Bernoulli distribution, 58
- Bit flips, 84
- BLAS, 1
- Block matrix, 143
- Broadcast, 47
- Chain length, 111
- Cheeger inequality, 160
- Collatz-Wielandt formula, 156
- Colon notation, 144
- Communication, 13, 34
 - asynchronous, 73
 - pull, 19
 - push, 19
- Communication matrix, 40
- Complete graph, 160
 - $C(n)$, 160
- Conductance, 159
- Connectivity measures, 159
 - conductance, 159
 - spectral gap, 159

- vertex expansion, 159
- Consensus algorithm, 26
- Convergence
 - acceleration, 44
 - in expectation, 40
 - in probability, 40
 - second moment, 40
- Determinant, 143
 - 2×2 block, 69
 - $\det(A)$, 143
- Diameter, 160
- Eigendecomposition, 145
- Eigenvalue, 144
- Eigenvector
 - left, 144
 - right, 144
- Epidemic algorithms, 18
- Erdős-Rényi graph, 160
 - $ER(n, p)$, 160
- Error class, 114, 130
 - Γ_k , 114
 - $\Gamma_{k,i}$, 114
- Error rate, 111
- Error threshold phenomenon, 115
- Exascale, 3
- Fast Fourier transform, 154
 - FFT, 154
- Fast Walsh-Hadamard Transform, 152
 - FWHT, 152
 - FWHT, 120
- Fitness landscape, 113
 - linear peak, 114
 - single peak, 114
- Floating-point arithmetic, 17, 30, 76
- Flop/s, 3
- Flows, 52, 55
 - interpretation, 57
 - skew-symmetry, 52, 56
 - two-sided recovery, 58
- Global control, 14
- Gossip algorithm, 26
- Graph, 12
 - asynchronicity, 74
 - connectivity, 159
 - diameter, 160
 - flow-recovery, 59
 - history-recovery, 66
 - strongly connected, 38
 - well-connected, 160
- Graph structures
 - complete graph, 160
 - Erdős-Rényi graph, 160
 - hypercube, 160
 - random geometric graph, 160
 - square lattice, 161
- Hadarmard matrix, 152
- Hamming distance, 112, 128
- Hamming distance-based
 - matrix, 129
 - vector, 129
- Histories, 52
 - interpretation, 64
 - one-sided recovery, 65
- Hypercube, 15, 160
 - $HC(d)$, 160
 - aggregation, 18
- i.i.d., 26
- Keeper, 60, 66
- Kronecker product, 145
 - parallel, 151
 - vector, 151
- Krylov subspace methods, 31
- Lanczos type methods, 31
- Linear iteration, 25
- LINPACK benchmark, 3
- Load balancing, 28
 - continuous, 28
 - discrete, 28
- Local knowledge, 13
- Markov chain, 154

-
- stationary distribution, 28, 155
 - transition matrix, 155
 - Mass conservation, 49, 56, 86
 - Master sequence, 112
 - Matrix, 143
 - block, 143
 - irreducible, 157
 - nonnegative, 154
 - positive, 154
 - primitive, 40, 157
 - reducible, 156
 - similar, 145
 - spectral radius, 144
 - spectrum, 144
 - transpose, 143
 - vectorization, 144
 - Mixed product property, 148
 - MPI, 12
 - Mutation matrix, 113
 - Neighborhood, 3, 12
 - Node, 12
 - One-sided recovery, 52
 - Orthogonality, 30
 - PageRank, 29
 - Perfect shuffle, 146
 - Perron-Frobenius
 - theorem, 41, 155, 157
 - theory, 41, 154
 - Power iteration, 28
 - Push-based algorithm, 49
 - Push-distribution, 40, 62, 68, 75
 - Quasispecies, 112
 - Random geometric graph, 160
 - $G(n, d)$, 160
 - Random phone call model, 19
 - Rank, 143
 - $\text{rank}(A)$, 143
 - Rayleigh quotient, 30
 - Recovery
 - bounds, 71
 - one-sided, 52, 65
 - two-sided, 52, 58
 - Rumor spreading, 4, 19
 - Sample mean, 92
 - Scheme
 - first-order, 45
 - second-order, 45
 - sum-weight, 32, 40
 - Self-citation policy, 9
 - Sensor network, 3
 - Similarity transformation, 145
 - Spectral gap, 71, 159
 - Spectral radius, 144
 - Spectrum, 144
 - Square lattice, 161
 - $L(n)$, 161
 - Stationary distribution, 155
 - Stochastic vector, 155
 - Submatrix, 144
 - Sum-weight scheme, 32, 40
 - SUMMA, 47
 - System
 - distributed, 2
 - distributed memory, 2, 11
 - shared memory, 2
 - Tensor product, 145
 - Top 500 list of supercomputers, 3
 - Tianhe-2, 3
 - Topology, 13, 38
 - Trace, 143
 - $\text{tr}(A)$, 143
 - Transition matrix, 155
 - Two-sided recovery, 52
 - Uniform error model, 113
 - Vertex expansion, 159
 - Walsh matrix, 152

Curriculum Vitae

Gerhard Niederbrucker born 19.10.1986 in Steyr, Austria

Marital status: married, one child

Citizenship: Austria

Address: Comeniusgasse 8/19, 1170 Vienna, Austria



Education

9/2001–6/2006 **College for Data Processing and Organisation Leonding**

7/2006–1/2007 **Austrian Armed Forces**, Military Service

10/2006–11/2010 **Vienna University of Technology**, Faculty of Computer Science

- **BSc** in Software & Information Engineering
Thesis: *Formal Verification of Cryptographic Protocols* (in German)
- **Dipl.-Ing. (MSc)** in Computational Intelligence (with distinction)
Thesis: *A Numeric Semantics for UML Class Diagrams: Methods and Tools*

10/2006–10/2013 **Vienna University of Technology**, Faculty of Mathematics

- **BSc** in Mathematics in Science and Technology
Thesis: *Stationary Distributions in Darwinian Systems*
- **Dipl.-Ing. (MSc)** in Technical Mathematics
Thesis: *Structured Preconditioners for Matrix-free Iterative Methods*

10/2011–2014 **University of Vienna**, Faculty of Computer Science

- **Dr. techn. (PhD)** (*expected*) in Computer Science
Thesis: *Towards Truly Distributed Computing: Uniting Theory, Algorithms and Practice*

Professional Experience (excluding internships)

12/2010–5/2014 **University of Vienna** Position: **Research Associate**

12/2004–4/2006 **dvo Software GmbH** Position: **Software Engineer**
(part-time, during summer full-time)