POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Spaziale

Decision-making on robotic networks with hybrid performance metrics for planetary exploration applications

Relatore: **Prof. Franco Bernelli**
Correlatori: **Prof. Marco Pavone**
Prof. Nicole Viola

Tesi di Laurea di
Federico Rossi
M. 766727

Anno Accademico 2012-2013

To my grandfather, Mario Rossi

Acknowledgments

First and foremost, I would like to thank my advisor, Professor Marco Pavone from Stanford University. Marco has been an excellent advisor and mentor: I am deeply indebted to him for his guidance and his help in this thesis and beyond, for introducing me to the world of academia and for believing in my potential. I do look forward to pursuing my studies under Marco's guidance for the next four years.

My gratitude also goes to Professor Franco Bernelli from Politecnico di Milano, who believed in this work, agreed to be my internal advisor and guided me from Milan. I thank him for his assistance, his patience and his invaluable academic advice before and during this thesis.

I would like to acknowledge Professor Nicole Viola, who followed the progression of my work from Politecnico di Torino and agreed to act as my co-advisor: I am grateful to her for her advice and her kindness.

This thesis is the final chapter of a six-year journey at PoliMi: I would like to acknowledge professor Lastaria, who instilled the love of mathematics in me when I was a freshman; professor F. Auteri, who rekindled it; and professor P. Masarati, thanks to whom I learned to see and appreciate the elegant structure underlying linear dynamical systems.

My closest friends, my *mellyn*, were always there to patiently remind me that there is a wide world outside the Terman Library. Thank you for your nighttime phone calls, for the train conversations, the long emails and the dinners in Milan; thank you for granting me your friendship.

My parents Teresite and Giuseppe supported me through these years, always pushing me forward and supporting me in my decisions. To them I owe everything.

Finally, I thank Valentina for her love and support. Thanks to her, I never felt alone. This is just our first step together.

Contents

List of Figures

List of Tables

List of Notations

log	The *base 2* logarithm
$\mathcal{G}$	The maximal set of graphs with node set V
d	The number of spatial dimensions of the problem under consideration
E	The set of edges in a graph
G	An undirected graph
n	The number of nodes in a graph or, equivalently, agents in a network
N_i	The set of neighbors of node i
V	The node set of a graph
BC	Byte complexity
CC	Communication complexity
CS	Computer Science
GHS	Gallager, Humblet and Spira's MST algorithm
LEO	Low Earth Orbit
LTL	Linear Temporal Logic
MER	Mars Exploration Rover Mission (Spirit and Opportunity rovers)
MSL	Mars Science Laboratory (Curiosity rover)
MST	Minimum Spanning Tree
PLA	Parallel Linear Algebra
SAR	Search and Rescue

SPF Single points of failure

SSSB Small Solar System Body

TC Time complexity

TTR Time to recovery

UAV Unmanned Aerial Vehicle

Summary

This thesis is about distributed consensus on robotic networks for planetary exploration.

The advantages of *distributed* architectures for space exploration have long been studied; furthermore, multiagent architectures are extremely advantageous on small solar system bodies, whose low gravity and uncertain dynamic environment make traditional mobility paradigms unapplicable. Relativistic delays make *autonomy* paramount for all probes operating beyond Earth orbit. Yet no *energy-efficient* procedures for autonomous consensus on robotic networks exist: current algorithms are either optimized for ground-based applications or largely inefficient.

The purpose of this thesis is to design *efficient* algorithms to reach an agreement between cooperative stationary or slow-moving robotic agents. We explore metrics describing time performance, power consumption and robustness; we propose time-optimal and energy-optimal algorithms and show how optimality with respect to one parameter typically leads to very bad performance with respect to other metrics.

We then design a novel *hybrid* algorithm that scales from time-optimal to message-optimal behavior, trading time performance and robustness for energy efficiency, according to an user-defined tuning parameter. Worst-case performance of the algorithm is investigated analytically; real-world performance on a simplified space exploration scenario is explored through numerical simulations with satisfactory results.

Future research directions will include extension of our work to fast-moving robotic networks such as swarms of planetary hoppers, optimization with respect to legacy omnidirectional (broadcast) communication protocols and application to problems such as UAV deployment for patrolling and ATC conflict resolution.

Sommario

Questa tesi studia come prendere decisioni in modo efficiente e decentralizzato su reti robotiche per l'esplorazione di corpi del Sistema Solare.

Le architetture multiagente per l'esplorazione spaziale promettono di offrire maggiore robustezza e migliore efficienza rispetto ai paradigmi monolitici oggi dominanti. Un'architettura multiagente è inoltre particolarmente adatta all'esplorazione dei piccoli corpi del Sistema Solare. Comete, asteroidi e piccole lune presentano morfologie e composizioni geologiche complesse che rendono necessaria l'esplorazione di regioni estese; d'altra parte, la microgravitá rende impossibile o sconsigliabile l'applicazione di sistemi di mobilitá tradizionali. Piú agenti stazionari ancorati alla superficie permetterebbero di esplorare ampie regioni senza incorrere negli svantaggi inerenti in nuovi sistemi di mobilitá non ancora testati; una rete di agenti mobili, d'altra parte, permetterebbe di rilassare i requisiti di mobilitá richiesti ad ogni singolo veicolo, concedendo di testare con gradualitá tecnologie innovative e garantendo il raggiungimento degli obiettivi scientifici anche a fronte di guasti catastrofici di uno o piú agenti.

L'autonomia é necessaria nell'esplorazione spaziale: i ritardi dovuti alle distanze interplanetarie e la congestione del Deep Space Network non permettono di controllare direttamente i veicoli se non per periodi di tempo molto brevi.

La letteratura scientifica presenta molti algoritmi per prendere decisioni collaborative in modo decentralizzato: questi algoritmi, tuttavia, sono tipicamente ottimizzati per applicazioni terrestri e, in particolare, non tengono conto del consumo energetico dovuto alle comunicazioni tra agenti.

Lo scopo di questa tesi é presentare algoritmi che permettano ad agenti robotici di prendere decisioni in modo *efficiente* rispetto a metriche rilevanti per l'esplorazione robotica del Sistema Solare.

Dopo aver evidenziato i limiti delle soluzioni esistenti nella letteratura, presentiamo delle metriche che caratterizzano il comportamento temporale, il consumo energetico e la robustezza degli algoritmi di consenso distribuito. Esploriamo quindi i limiti fondamentali del problema del consenso rispetto a queste metriche e mostriamo come due algoritmi esistenti permettano di

minimizzare rispettivamente il tempo necessario e il numero di bytes scambiati (quindi il consumo energetico) per raggiungere una decisione in modo decentralizzato.

Questa soluzione, tuttavia, non é soddisfacente: prestazioni ottimali nel tempo comportano forti compromessi in termini di consumo energetico, mentre la minimizzazione del consumo energetico rende l'algoritmo lento e fragile rispetto a guasti dei canali di comunicazione.

Progettiamo quindi un algoritmo *ibrido* e *regolabile* che permette di ottenere compromessi tra tempo di convergenza, consumo energetico e robustezza. Le prestazioni dell'algoritmo sono esplorate analiticamente nel caso peggiore e su alcune semplici topologie di rete; simulazioni numeriche permettono quindi di valutare il comportamento dell'algoritmo in uno scenario realistico di esplorazione di un piccolo corpo del Sistema Solare. I risultati sono soddisfacenti.

Nel prossimo futuro, la ricerca si concentrerá sull'estensione dei nostri risultati a reti di agenti in rapido movimento e alla ricerca di algoritmi ottimali per reti broadcast. Le applicazioni del nostro algoritmo vanno oltre l'esplorazione spaziale e includono problemi di interesse aerospaziale come il controllo di gruppi di pattugliatori aerei senza pilota e la risoluzione di conflitti nel controllo del traffico aereo.

Chapter 1

Introduction

This thesis is about autonomous decision-making in robotic networks. Our goal is to find algorithms to reach consensus in a highly dynamical environment, with limited time, scarce energy and failures of communication equipment.

Our quest is motivated by exploration of small solar system bodies. We have visited all planets in the Solar System: we have landed on Mars, Venus and the Moon, we have pierced Jupiter's clouds and deployed a probe on the surface of Saturn moon Titan. Yet small bodies, which hold the answer to many fundamental questions about the origin of our Solar System, are virtually unexplored: only in recent years have we been able to gain insight into them and, to date, questions about them far outnumber the few answers we have gathered from flybys.

This chapter is dedicated to robotic exploration of small bodies with swarms of autonomous agents. In the next sections, we outline what planets, moons and asteroids have already been visited by mankind; we discuss the importance of small bodies to a better understanding of our Solar System; we show how a distributed architecture is instrumental to exploration of hostile, low-gravity objects and we (hopefully) convince the reader that distributed decision making is an key technology to better understand these bodies and, through them, our home among the stars.

1.1 Robotic exploration of the Solar System

Probes have flown by all planets in the Solar System. Mercury, Venus, Mars, Jupiter and Saturn have or have had manmade satellites; landers have touched down on Venus and Mars, a probe has entered gas giant Jupiter's atmosphere and we have landed a vehicle on another planet's moon, Titan [75].

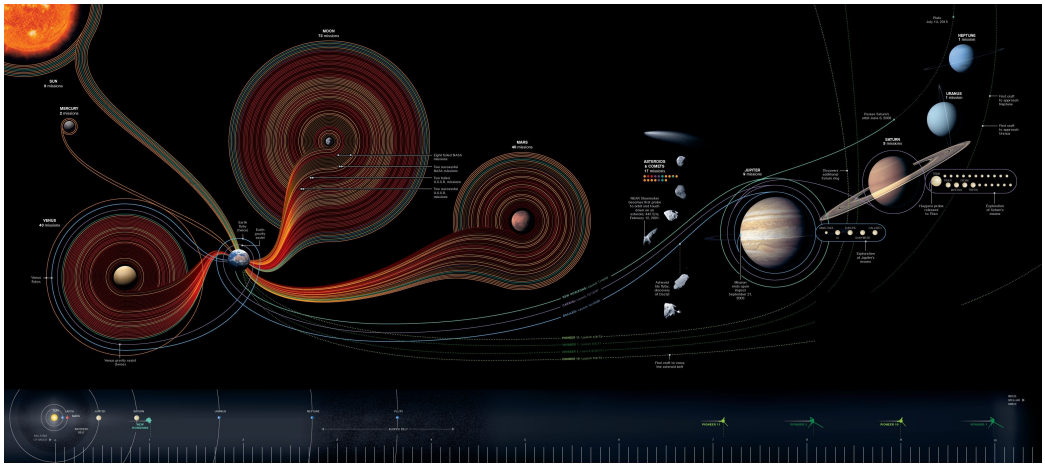


Figure 1.1: An artist's interpretation of Solar System exploration (Image credit: Sean McNaughton and Samuel Velasco, National Geographic Society)

The Moon The Earth's Moon was the first Solar System body targeted by U.S. and Russian efforts in the early years of the space age. The first man-made vehicle to escape Earth gravity was Soviet probe Luna 1, in January 1959. In September of the same year, Luna 2 was the first vehicle to crash land on the Moon; in October 1959, Luna 3 performed the first controlled Lunar flyby and imaged the dark side of our natural satellite.
The Moon would remain a prime target for robotic and manned exploration: it was the host to the first survivable landing by Luna 9 in 1966, the first rover, Soviet Lunokhod 1, in 1970 and the first robotic sample return mission, Luna 16, in the same year. As of 2012, 73 missions, manned or unmanned, have targeted our natural satellite. The Lunar Reconnaissance Orbiter spacecraft is currently in Lunar orbit, producing high-resolution maps of our natural satellite. [76, 78]

Mercury The first spacecraft to fly by Mercury was Mariner 10, which approached the planet three times in 1974. MESSENGER was the first vehicle to orbit the planet in 2011. Plans to carry a lander with the BepiColombo mission, currently in development, have been canceled. [33, 74, 99]

Venus The first Venus flyby was performed by U.S. Mariner 2 in 1962. The planet's atmosphere was first studied by the Soviet Venera 4 probe in 1967; in 1970, Venera 7 reached the surface of the planet with an intact scientific payload. Venera 9 became Venus's first artificial satellite in 1975; its lander was the first to send back photos from the surface of another planet. To date, more than 40 missions have directly targeted Venus; in addition, many

missions have studied the planet during gravity assists en route to other targets. ESA's Venus Express is currently in orbit around the planet. [35, 99]

Mars After many failed attempts, Soviet probes Mars 2 and 3, launched within weeks of each other, became the first Martian artificial satellites in November 1970. Mars 2's lander achieved the first landing on the Martian surface; the first *survivable* landing was performed by Mars 3 five days later. Since then, Mars has been a prime target for robotic exploration: the planet is currently host to two active U.S. rovers and three orbiters. [51, 98]

Jupiter and its moons The first Jupiter flyby was performed by Pioneer 10 in 1973 and the first close approach to Jupiter's Galilean satellites is due to Voyager 1 in 1979. In 1995, the Galileo spacecraft became Jupiter's first artificial satellite; its probe relayed data from the upper layers of the gas giant's atmosphere, whereas the orbiter performed several flybys of major Jupiter moons Io, Europa, Ganymede and Callisto. Jupiter and its moons are currently host to no man-made spacecraft. The Juno mission is slated to reach the gas giant in 2016. [67, 73, 75]

Saturn and its moons Pioneer 11 performed the first flyby of Saturn in 1979. Twenty-five years later, the Cassini spacecraft entered Saturn's orbit, where it currently resides. Cassini has performed a number of flybys of Saturn and its moons and relayed invaluable scientific data; its lander Huygens was the first man-made object to land on another planet's moon when it touched down on Titan in January 2005. [48, 62, 75]

Uranus and Neptune The Voyager 2 probe flew by Uranus and Neptune in 1986 and 1989 respectively. Follow-up missions to Uranus have been proposed but none are actively being planned at the time of writing. [82]

Small Solar System bodies To date, Small Solar System bodies have received significantly less attention than planets and large moons.
The first flyby of a comet was performed in 1985 by the ISEE3 spacecraft, originally designed to study the magnetosphere from an halo orbit, during an extension of its mission. The first spacecraft purpose-built to rendezvous with a comet was ESA's Giotto, who passed 596 km from the Halley comet in 1986.
The NEAR-Shoemaker probe first entered an asteroid's orbit in 2000 and, after collecting information about EROS 433, landed on it in 2001. The

probe was not designed to survive the landing and had no dedicated surface experiments on board; despite this, its instruments kept sending information from the asteroid surface for fourteen days after touchdown.

In 2006, Stardust returned the first samples from a comet tail to Earth; the first samples from an asteroid's surface were returned to our planet by Hayabusa in 2006. [34, 39, 60, 66, 77, 81]

1.2 On small solar system bodies

Small Solar System bodies hold the answer to many questions about the origin of our Solar System. Current models for the evolution of the Solar System (especially the Nice model, introduced in [46, 70, 113]) theorize large dynamic shifts in the position of SSSBs as massive planets migrated to their current orbits; analysis of the geochemical composition of small bodies would allow to confirm or refute such models [18].

In addition, SSSBs are a pristine sample of the composition of the early Solar System: while the heat produced by the core of larger planets and moons has causes significant shifts in their chemical and mineralogical composition, small bodies are largely unaffected by geochemical evolution. Beneath their regolith surface, SSSBs probably hold a snapshot of the early stages of the Solar System.

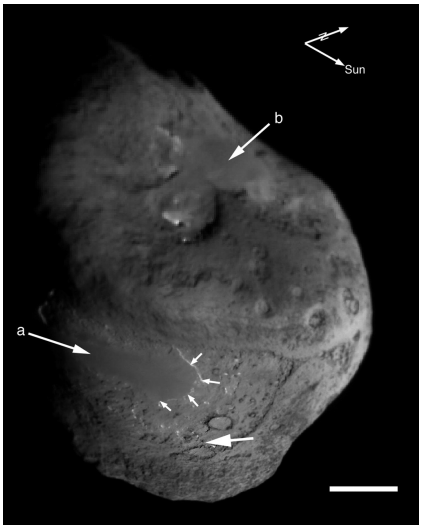
Despite numerous efforts, these asteroids, comets and irregular moons remain mysterious: while their orbital parameters are well understood from ground-based observations and the mass of select bodies has been estimated through flybys, little is known about their internal composition and optical observation is often hindered by a thick regolith surface layer.

Irregular Martian moon Phobos is a prime example of this: the moon has been the target of five distinct flybys, which allowed to determine the mass [4], surface composition [12, 44] and outer shape [109] of the object to a very high degree of accuracy. Yet the internal composition of Phobos, which holds the answer to the moon's origin, is still a matter of debate within the scientific community. Only a lander would be able to pierce the regolith surface and study the moon's geology and chemistry. We refer the reader to [72] for a more thorough discussion of the debate within the scientific community and the potential benefits of a mission to Phobos's surface.

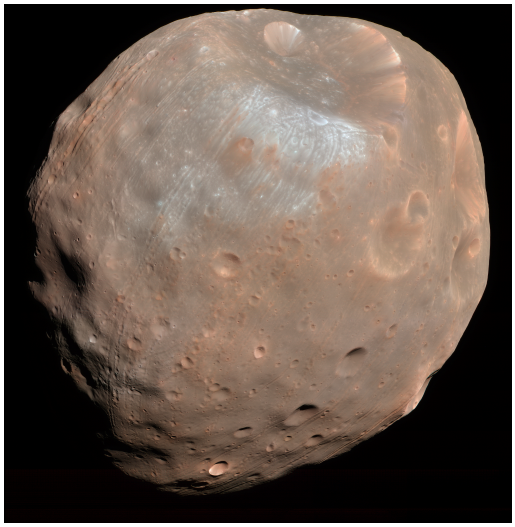
1.3 The advantages of an autonomous multi-agent architecture

Autonomous multi-agent architectures [54] have often been proposed for space exploration. Multi-agent systems present several advantages with respect to monolithic architectures: among these are the ability to optimize the mission architecture on the fly as science objectives change, higher adaptability to an uncertain environment and more graceful degradation in presence of failures. We refer the reader to Truszkowski's work [112] for further information.
Autonomy is necessary in robotic space exploration: limited opportunities for communication and relativistic delays require that agents be able to react to their environment, execute complex tasks and plan actions with limited or no human intervention.

Small Solar System Bodies offer additional compelling reasons to adopt a multiagent architecture. The varied geological landscape of SSSBs can not be thoroughly explored by a single, static lander: Castillo et al. [18] show that comet Tempel 1, shown in fig. 1.2a presents at least four distinct geological regions, whereas Phobos's surface, shown in fig. 1.2b can be partitioned in four to five spectral units.
On massive bodies, exploration of contiguous yet geologically diverse regions may be achieved with rovers; mobility in microgravity, however, is far more complex and less understood.
Castillo's paper surveys a number of possible strategies for mobility in microgravity; an hybrid spacecraft-rover architecture is further explored in [95] and [3]. Technologies for mobility in microgravity, however, are still in very early stages of development: a single agent would not be able to offer reliability guarantees and reconfigurability capabilities needed to successfully explore Small Solar System bodies. A network of *stationary* agents (e.g. penetrators) deployed on the surface of a small body, on the other hand, could easily perform exploration of its geophysical properties without the complexity and uncertainty inherent in microgravity mobility; even in presence of mobile agents, a distributed architecture would offer far greater reliability guarantees and allow for less conservative mission profiles, guaranteeing graceful performance degradation even in presence of catastrophic failures of one or more agents.



(a) Tempel 1's surface presents a varied topology, including large, smooth areas (indicated by arrows a and b), scarps and very rough terrain. ITS composite observation by Deep Impact, from [2].



(b) The surface of Phobos shows lateral variation in color properties, which suggest Phobos material may have different origins. HiRISE observation PSP_007769_9010 (IRB composite), via [18]. Image credit NASA/University of Arizona

Figure 1.2: Geophysical features of comet Tempel 1 and Martian moon Phobos

1.4 Challenges

In recent years, the scientific community has shown great interest in decision-making in multiagent systems. Models and algorithms have been proposed that can solve distributed optimization problems, study the collective behavior of cooperative and noncooperative agents, reproduce learning and teaching and negotiate decisions through voting and auctions [106, 121, 122].

Most studies in multiagent autonomy, however, focus on the high-level dynamics of knowledge and decision of a system of agents; little to no thought is paid to *how* agents should *communicate* to efficiently exchange information and gather the knowledge they need to make informed decisions.

The issue is not excessively pressing for software-based agents, which typically have access to an all-to-all network with efficient, reliable, transparently implemented message routing algorithms.

Ground-based robotic applications also often exploit TCP/IP based networks which offer transparent all-to-all message routing, typically relying on already-in-place infrastructures and well-known protocols.

Yet such architectures are expensive and inefficient when applied to multiagent systems for space exploration, which are typically subject to strong

constraints on available power and energy consumption. *Energy-efficient* decision-making algorithms that exploit the variable topology of a multiagent system can enable longer, more ambitious missions, leaving power and energy for more demanding scientific instruments and longer experiments.

This thesis is about the quest for these algorithms.

1.5 Contributions of this thesis

In this thesis, we focus on the *consensus* problem for robotic networks. Consensus is a key subroutine of virtually all distributed decision-making protocols: it allows all participating agents to *agree* on a common logical or numerical value computed as a function (which we call *consensus function*) of the agents' initial opinions. Consensus directly encompasses *distributed estimation*, *leader election*, *majority voting* and *mediation* among a set of policies; it can also be used as a subroutine to solve complex distributed optimization, deployment and task allocation problems.

Our work is laid out as follows:

Chapter 2 presents an overview of the state of the art of consensus in the Computer Science and the Control Systems communities with an historical perspective. After reviewing the existing literature, we show how metrics used in the Computer Science community are inadequate to the needs of robotic multiagent systems, whereas the dominant paradigm within the Control Systems community is only optimal for a limited range of simple applications and underexploits capabilities of modern robotic agents.

Chapter 3 rigorously presents the problem we wish to solve and motivates the mathematical hypotheses that will guide our research. We give a mathematically sound definition of the consensus problem and present a *model* of a cyber-physical robotic network, motivating our hypotheses regarding the agents' capabilities and the network topology. We present the concepts of *locally computable* and *hierarchically computable* consensus functions and give real-world examples of both. Finally, we introduce *time*, *message* and *byte* complexity, the three metrics that will guide our research, and we show how message and byte complexity relate to power consumption for telecommunications in robotic networks. We also discuss the issue of *robustness* in presence of link failures.

Chapter 4 is dedicated to *fundamental limitations* of decision-making on cyber-physical networks. We show a trivial lower bound on the *time complex-*

ity of the consensus problem and produce a matching time-optimal consensus algorithm, flooding. We then turn our attention to *message complexity*: we prove a lower bound on the message complexity of a wide class of *convex* consensus problems and show that an existing algorithm, GHS, solves problems in this class with message-optimal performance. We show that GHS also achieves optimal *byte complexity* if some simple hypotheses about the nature of the communication protocol are verified. We discuss the drawbacks of the two optimal algorithms: time-optimal flooding is resilient but sports very high byte complexity, while byte-optimal GHS is comparatively slow and very fragile in case of link failures.

Chapter 5 presents a novel *hybrid*, *tunable* algorithm that solves the consensus problem on certain slowly time-varying network topologies with time and byte performance intermediate between time-optimal flooding and byte-optimal GHS. Upper bounds on the time, message and byte complexity of the algorithm are rigorously proven; analytical performance of the hybrid algorithm on select network topologies is also presented at the end of the chapter.

Chapter 6 explores performance of GHS, flooding and our hybrid algorithm in a simplified model of a real-world space exploration scenario, soil sampling of a small solar system body with a network of penetrators. Numerical simulations show that the cost in terms of running time, messages and bytes exchanged of the three algorithms agrees with theoretical results. Our hybrid algorithm is proven to be a solid choice for distributed consensus problems with *hybrid* performance metrics, where optimality with respect to one metric does not justify subpar performance in all others.

Chapter 7 concludes our work by summarizing our findings and their relevance to space exploration. We also discuss future research directions and potential further applications of our results, which extend to airborne and ground-based robotic networks.

Chapter 2

State of the Art

This chapter strives to give a comprehensive review of scientific efforts in the field of distributed consensus. The relevant literature spans several fields, including graph theory, computer science, statistics and control systems engineering, and several decades: in an effort to organize it, after a brief historical introduction in Section 2.1, we present the state of the art in the two fields most relevant to our work, computer science and control engineering, in Sections 2.2 and 2.3 respectively. We highlight the main areas of research within each discipline and present a selection of landmark works from an historical perspective. Finally, Section 2.4 is dedicated to existing applications of centralized and distributed autonomy algorithms, which represent a superset of the consensus problem, to space exploration.

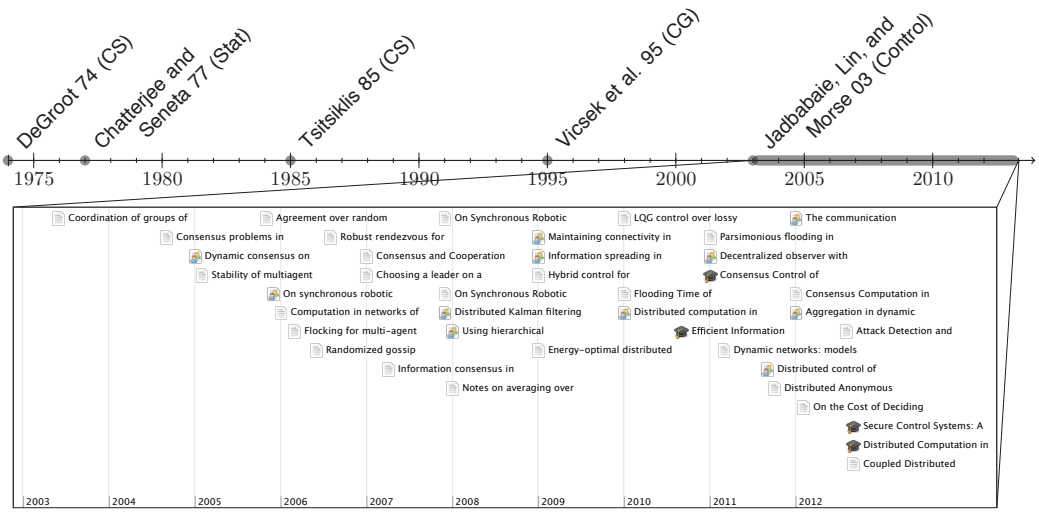


Figure 2.1: A representative cross-section of the literature

2.1 Introduction

The first rigorous treatment of the distributed consensus problem is due to Morris Degroot [27], a statistician studying decision-making under uncertainty, in 1974. Degroot considers a network of discrete-time synchronous agents exchanging opinions, each with an associated confidence: at each step, agents update their opinion with a *weighed average* of their neighbors' and theirs. Degroot identifies necessary as well as sufficient conditions on node weights for consensus to be achieved.

Chatterjee and Seneta [19] generalize Degroot's results and, crucially, explore the case in which node weights may change in time. They show a relation between *ergodicity* of the matrices P_k representing the nodes' weights[1] and convergence of the average-based consensus procedure: for convergence to be achieved, the matrix $U = \prod_{k=1}^{\infty} P_k$ must be $U = \mathbf{1}' \cdot \mathbf{p}$, where $\mathbf{1}'$ is a row vector of ones and $\mathbf{p}$ is a stochastic column vector.

J. N. Tsitsiklis's 1985 Ph.D. thesis [114] proposes a *model* for decentralized discrete-time asynchronous decision-making systems and explores *feasibility* and *complexity* of a wide range of distributed decision problems as a function of the communication pattern of the network, which determines the information available to the agents. Tsitsiklis also proposes a distributed *optimization* algorithm and characterizes the effect of communication delays on the system's convergence. Tsitsiklis's work, which explicitly refrains from focusing on specific applications and concentrates on the mathematical aspects of the consensus problem, represents a milestone in consensus theory: it represents the basis of most modern research on consensus within the control community.

After Tsitsiklis's work, interest in decentralized decision-making systems in the engineering community appeared to wane. At the same time, advances in parallel computing spurred interest in distributed computation in the Computer Science community.

2.2 Computer Science

2.2.1 Distributed algorithms on wired networks

J. Burns's 1980 report [17], which predates Tsitsiklis's thesis by five years, first proposes a formal model for computation on asynchronous networks of deterministic agents. Burns' model was widely adopted by the CS community

[1] At each time step k, the authors defines $P_k = [p_{1k}, p_{2k} \dots p_{nk}]'$, where elements of the column vector p_{ik} represent agent i's confidence in his neighbors' opinion at time k.

and is still the basis of modern models for distributed wired and wireless networks.

In 1983, Gallager, Humblet and Spira [40] publish a seminal paper proposing an algorithm for distributed minimum spanning tree construction: the GHS algorithm still is a cornerstone of many complex distributed computation problems, including breadth-first search, leader election and, by extension, consensus. In 1984, Santoro [105] proves GHS to be worst-case message-optimal (with $O(n \log n + |E|)$ messages per execution) for spanning tree and minimum spanning tree construction.

In 1987, Awerbuch [7] manages to reduce the time complexity of the GHS algorithm from $O(n \log n)$ to $O(n)$ while maintaining message optimality: Awerbuch's improvement on GHS still remains the state of the art for distributed MST construction.
Awerbuch states that his algorithm is time-optimal and even suggests that it can solve the leader election problem optimally with respect to time and message complexity. He supports his first claim by arguing that that the worst-case diameter of a network, which gives a trivial lower bound on the time complexity of any MST algorithm, can be as high as $O(n)$; the second claim follows from the observation that, at the time of writing, all leader election algorithms are based on MST construction. In fact, Awerbuch's algorithm is *only* time-optimal for networks whose diameter is $O(n)$: his algorithm may require up to $O(n)$ rounds even on a fully connected network, whose diameter is trivially one. The second claim is empirical.
Awerbuch also shows that, if no *a priori* information is available, counting the number of agents is as hard as computing a sensitively decomposable function (whose definition we recall in Chapter 3) on a static, connected network.

In the same years, Korach et al. [57] publish a comprehensive study of lower bounds for a class of problems including leader election, spanning tree construction and MST construction, on *fully connected* networks. Their main contribution is twofold:

- a lower bound of $O(n \log n)$ messages for any problem that must use the edges of a *spanning subgraph* of the fully connected network. This includes any problem requiring all nodes to hear (directly or indirectly) from all others, which encompasses all sensibly defined consensus problems. The authors also produce a matching $O(n \log n)$ leader election algorithm.

- a lower bound of $|E| - 1$ messages for the same type of problems on a class of "almost complete" graphs that are not fully connected but have $\Theta(n(n-1)/2)$ edges.

The last result is crucial: it shows that, unless the network is guaranteed to be fully connected, the worst-case message complexity of consensus can be as high as $O(|E|)$. The GHS algorithm is therefore a message-optimal solution to consensus problems on "almost complete" networks.

Failures Failures during consensus have always been a major concern in the CS community[2]. Both edge and node failures are typically considered; node failures are typically modeled either as *stopping* or as *byzantine*. In the former case, a node suddenly stops working, possibly mid-message; in the latter, a node arbitrarily updates its state and sends arbitrary messages to its neighbors with potentially nefarious intents.

Gray [47] shows that, if arbitrary edge failures are allowed, no deterministic algorithm can solve the consensus problem. Randomized algorithms fare better: yet the same author shows that *any* synchronous algorithm converging in r rounds has probability of disagreement $p_d \geq 1/(r+1)$ on a *fully connected* graph.

Stopping node failures are more benign: Lynch [63, ch. 6] shows that a simple flooding algorithm can solve the consensus problem on a *fully connected* network with f node failures in $f+1$ rounds. The algorithm also achieves optimal time complexity in the absence of failures. Message and byte complexity are rather abysmal at $O((f+1)n^2)$ and $O((f+1)n^3 \log(n))$ respectively, although they can be improved on fully connected networks for very simple consensus functions.

Exponential information gathering algorithms, first proposed by Lamport, Shostak and Pease [61, 96], sport the same time and message complexity as flooding. The byte complexity is much worse at $O(n^{f+1} \log(n))$. The main advantage of EIG algorithms is their ability to solve the consensus problem in presence of Byzantine failures if communications are authenticated, i.e. if nodes are able to *sign* their tokens.

Non-authenticated Byzantine failures are beyond the scope of our work.

2.2.2 Wireless networks: challenges and opportunities

In recent years, the advent of wireless networks has spurred a new wave of studies focusing on the peculiarities of these structures, including a focus on time-varying topologies and new concern for energy optimality.

In 2006, Angluin et al. [5] first characterize the class of functions that can be computed by agents subject to sporadic pairwise interactions and with

[2] In fact, consensus in absence of failures is not considered a very interesting problem: Lynch states that *When there are no failures of system components, consensus problems are usually easy to solve, using a simple exchange of messages.* [63, ch. 5].

small on-board storage space. While the paradigm adopted by the authors is quite far from ours, their work is worth mentioning since it is one of the first signs of an interest in computation on *mobile* networks within the CS community.

In 2009, Khan et al. [56] propose an *energy-optimal* algorithm that builds *approximate* minimum spanning trees on random geometric graphs. The resulting tree can be cheaply rearranged under nodes insertions and deletions. The interest of Khan's work to our purposes is twofold. First, the authors explore tradeoffs between the energy needed to build a spanning tree, a one-off cost, and the quality of the resulting tree, which influences the cost of all subsequent computations that use the tree structure. Second, the authors concern themselves with *reconfiguration* of the network under node insertions and deletions, while most previous works dealt with failures by building a *robust* (and expensive) structure to passively respond to node failures.

In the same year, Choi et al. [20] publish an *energy-optimal* algorithm for minimum spanning trees on random geometric graph. Choi's algorithm cleverly exploits known properties of random geometric graphs (discussed by Penrose in [97]) to adapt the GHS algorithm to wireless applications.

In 2010, Kuhn, Lynch and Oshman [58] explore lower bounds on certain distributed problems, including token dissemination (which is a prerequisite for consensus), on time-varying network topologies. The authors assume the network to be adversarial and agents to be unaware of their neighbors until *after* they have sent them messages. They show that the token dissemination problem can be solved in $O(n + n^2/T)$ rounds with messages of size $O(\log n)$ if the network sports a stable spanning subgraph for T consecutive rounds. Kuhn's algorithm is shown to be optimal under certain restrictive assumptions about the nodes' (lack of) knowledge of the network.
The same model and connectivity properties are applied to a wider range of problems, including consensus, in Oshman's Ph.D. thesis [90]. No parameters other than time complexity are considered in these works: in fact, the communication complexity of the algorithms proposed is typically very bad, a direct result of the agents' lack of knowledge of the network topology.

More recently, Kuhn and Oshman [59] have published a partial survey of algorithms for distributed computation on dynamic networks. Their work focuses on adversarial graphs with unceasing dynamic behavior and explicitly avoids geographical networks: despite these limitations, their paper is among the most complete surveys of distributed models and algorithms for wireless networks to date.

Drucker, Kuhn and Oshman [30] study the problem of distributed task allocation on wireless networks: their work proposes time and byte efficient ($O(\log n)$ and $O(n \log n)$ respectively) algorithms for distributed task alloca-

tion in a *shared memory* setting, which models a fully connected, broadcast-enabled network.

2.2.3 Parallel linear algebra

Parallel linear algebra has long been an active area of research in computer science. While the algorithms employed in the field are very distant from those used for consensus, the field holds a significant interest for our purposes: PLA shares our concern with message and energy complexity and many of the metrics developed for distributed linear algebra carry over to decentralized consensus on hybrid networks.

Demmel's book on parallel numerical linear algebra [29] still is a crucial reference on the subject; recent publications by Ballard, Demmel and others [9, 10, 28] propose cost metrics for distributed algorithms that our work borrows from in Chapter 3.

2.3 Control Systems

Research on distributed consensus in the control community evolved quite independently of developments in CS. After Tsitsiklis's 1985 Ph.D. thesis, theoretical work on the topic saw very little significant development until the early 21st century.

In 2003, Jadbabaie et al. [53] published a seminal paper proposing a theoretical explanation to a flocking phenomenon observed by Vicsek [117] in numerical simulations. Jadbabaie and Vicsek consider *average-based* algorithms: at each time step (or instant), nodes update their state according to their neighbors'. Rigourously, in continuous networks, the system dynamics is described by the switched system

$$\dot{x}(t) = -A_t x(t)$$

where $x = [x_1^T x_2^T \dots x_N^T]^T$ is the collection of the agents' states and $A_t = [a_{t,ij}]$ is the adjacency matrix of the underlying proximity graph, whose elements a_{ij} are nonzero if and only if nodes i and j are neighbors at time t. Different choices of a_{ij} lead to different average-based consensus algorithms.
In discrete networks, the system dynamics is described by

$$x(t+1) = L(t)x(t)$$

where $L(t) = [l_{ij}(t)]$ is the Laplacian matrix of the proximity graph, defined as

$$l_{ij} = \begin{cases} -a_{ij} & \text{if} \quad i \neq j \\ \sum_{k \neq i} a_{ik} & \text{if} \quad i = j \end{cases}$$

Consensus is defined as asymptotic convergence of all agents' states to a common value.

Jadbabaie shows that, if the set $\{A_t\}$ obeys certain connectivity properties, then a discrete system will asymptotically achieve consensus. Specifically, if there exists an infinite sequence of continuous, nonempty, bounded time intervals $[t_i, t_{i+1})$ starting at $t_0 = 0$ such that the *union* of $\{A_j\}$, $j \in [t_i, t_{i+1})$ is connected for any i, then the system converges to a linear combination of the initial values.
Jadbabaie also shows that, if a leader node follows an independent trajectory, other nodes will converge to the leader's trajectory under the above assumptions. Moreover, continuous systems have the same behavior if a dwell time is introduced between transitions, so as to discourage chattering.

As a note of color, Bertsekas and Tsitsiklis show [14] that Jadbabaie's results are a special case of those in Tsitsiklis's 1985 thesis [114] and related works from the late Eighties [15, 115]. Yet Jadbabaie has the unquestionable merit of bringing distributed consensus back into focus in the control community: Olshevsky argues that

> [Jadbabaie's] paper has created an explosion of interest in averaging algorithms, and the subsequent literature expanded in a number of directions. It is impossible to give a complete account of the literature since [Jadbabaie, Lin and Morse's paper] in a reasonable amount of space (A. Olshevsky, [89])

Our review of the literature will not be comprehensive: we prefer to focus on a comparatively small selection of high quality works and paint a picture of the control community's approach to distributed consensus through these. The reader may also be interested in the review papers by Olfati Saber [85], which gives a good overview of findings in cooperative consensus, and Ren [102] which focuses on distributed estimation. Both papers were published in 2007: the introduction to Acikmese's work [1] gives a more recent, if succinct, overview of key results in distributed estimation.

2.3.1 Models and fundamental limitations

In 2003, Olfati Saber and Murray propose a *model* for networks of dynamic, acceleration-controlled agents and a protocol that implements Reynolds's rules of flocking [103], i.e. *collision avoidance*, *velocity matching* and *flock centering*.

In 2004, Olfati Saber et al. [87] propose a set of consensus algorithms for fixed and switching network topologies with or without time delays in *directed*

and undirected networks. The role of directed networks is especially important, since it allows to introduce a hierarchy in the otherwise peer-to-peer flocking model and generalizes Jadbabaie's [53] results on leader following. Olfati's paper also establishes a crucial connection between the algebraic connectivity of the network (represented by the second smallest eigenvalue of the Laplacian, also known as the Fiedler eigenvalue [37, 38]) and the convergence speed of an average-based consensus protocol. The same results are discussed in further detail in a technical report by the same author [84].

A 2005 paper by Spanos et al. [108] presents average-based consensus algorithm designed to work in presence of *arbitrary time delays* and *splitting and merging* of the network [108].

Also in 2005, Moreau [71] extends and generalizes Jadbabaie's results on sufficient conditions for consensus to a wide class of (possibly *nonlinear*) update schemes.

A 2006 paper by Boyd et al. [16] proposes *gossip* algorithms for averaging on dynamic networks. Boyd also proposes a novel *decentralized optimization* procedure to design the fastest possible gossip algorithm on the very network it will be executed on; his work also relates the convergence rate of gossip algorithms to the Fiedler eigenvalue of the network.

Two companion papers by Martinez et al. [64, 65] propose a rigorous *model* for cyber-physical networks which draws inspiration from work in the Computer Science community and especially from Lynch [63]. Martinez's model is then applied to rendezvous and deployment problems, discussed in Section 2.3.5.

Benezit et al. [13] study the convergence rate of *gossip-based* consensus algorithms: their paper proposes a geographic gossip algorithm that achieves linear message complexity, as opposed to the quadratic message complexity of nearest neighbor-based gossip. Their work is especially significant since it is among the first to exhibit a concern for the *message* complexity of average-based consensus algorithms.

In 2010, Olshevsky's Ph.D. thesis [89] offers what is probably the most exhaustive and comprehensive study of average-based consensus to date. Olshevsky proposes an averaging algorithm whose convergence time scales as $O(n^2)$ steps on a wide class of time-varying graph sequences on discrete networks with bounded delays. He also shows this algorithm to be *optimal* within a large class of linear and nonlinear update schemes. He then explores the effect of communication quantization and shows that the convergence rate of his algorithm is not affected if the message size is upper-bounded by $c \log n$. Finally, he turns his attention to algorithms that use a *constant* number of bits per link, which are beyond the scope of our work.

2.3.2 Specific network topologies

In 2005, Hatano and Mesbahi [49] first study stability and rate of convergence of a consensus protocol on *random* Erdős-Rényi [32] graphs. To the best of the author's knowledge, Hatano's paper is the first to adopt a stochastic description of the dynamic network exploited by the consensus protocol.

Epstein et al. [31] exploit existing hierarchical structures within the network to speed up average-based consensus: their technique revolves around the fact that smaller networks may have a larger Fiedler eigenvalue, which leads to an higher rate of convergence. Epstein's work is the first to introduce the idea of using multilevel hierarchies within a network, although the paper only concerns itself with exploiting an *existing* hierarchy and offers no insight in how to best partition a network to improve performance.

2.3.3 Distributed filtering

Distributed Kalman filtering was first studied for continuous systems by Olfati Saber in 2005 and 2007 [83, 88]. Olfati Saber proposes several distributed Kalman filters and compares their performance numerically. The second article also estimates the byte complexity (per round) of a distributed Kalman filter.

A 2005 paper by Spanos proposes an application of *robust* algorithms presented in [108] to multivariable consensus, namely Kalman filtering [107].

In 2011, Açıkmeşe [1] proposes a decentralized observer for *discrete-time* linear systems.

2.3.4 Security of cyber-physical networks

Pasqualetti and Bullo's research is devoted to byzantine failures in cyber-physical network: their work characterizes the vulnerabilities of hybrid networks [92], explores *fundamental limitations* of attack detection and identification and proposes *algorithms* to detect such failures [93, 94]. Results by this group are also summarized in Pasqualetti's 2012 Ph.D. thesis [91].

2.3.5 Rendezvous, deployment and tracking

In 2006, Cortes et al. [24] propose an improved version of an existing *circumcenter algorithm* for rendezvous of multiple agents (i.e. a specific application of consensus) and analytically prove its convergence under lax hypotheses on the time evolution of the network.

A paper by Martinez et al. [65] applies the model developed in [64] to rendezvous and deployment problems. The model allows for continuous

evolution of the network with asynchronous communication: the rendezvous and deployment algorithms, on the other hand, are studied in a synchronous, discrete-time setting.

The same group is responsible for a paper [41] that discusses the relationship between *continuous coverage control* and average-based consensus on select network topologies, which the authors term discrete Voronoi graphs.

Zavlanos et al. [123] first propose a *connectivity preserving* flocking algorithm: Zavlanos's procedure enforces connectivity conditions rather than relying on their being verified.

Sabattini et al. [104] also propose a connectivity preserving algorithm that works in presence of additional bounded control terms: they show how their algorithm can be used to achieve rendezvous and formation control.

Olfati Saber proposes a coupled distribution, estimation and control algorithm for mobile sensor networks in [86]: agents collaborate to estimate the position of a target, then use flocking to track its position and improve their estimate in a closed-loop process.

2.4 Autonomy and decentralization in space exploration

Space exploration is a prime application for autonomous decision-making: relativistic delays make it all but impossible for humans to directly control space probes beyond LEO and time on the high-power Deep Space Network, which guarantees communications with all probes outside Earth orbit, is an extremely scarce and expensive resource. Despite this, only in recent years has a shift from monolithic spacecraft requiring strong human interaction to comparatively autonomous systems started to manifest itself; decentralized architectures have been studied for many spaceborne applications but, at the time of writing, they have seen extremely limited use in flight missions.

2.4.1 Autonomy

Algorithms deployed on Mars rovers Spirit, Opportunity and Curiosity represent the state of the art in autonomous decision-making for planetary exploration [50, ch. 3]. Surprisingly, autonomy is mostly relegated to navigation tasks and very limited in time. During typical operations, a set of instructions is sent to the rovers at the beginning of every Martian day; the rover then proceeds to execute the instructions, which include collection of images and data, placement of scientific instrument and driving instructions [68], and reports back at the end of the sol. High-level planning is performed

offline daily by ground operators, helped by the dedicated Science Activity Planner (SAP) software.

Rovers are able to drive autonomously, generating a terrain map from stereoscopic images and performing path planning and obstacle avoidance between gateways defined by mission controllers. Fully autonomous driving, however, is used less than a quarter of the time on Spirit and Opportunity: mission controllers feel that autonomous planning is inherently less reliable than human-in-the-loop operations and visual odometry, required to precisely estimate the position of the rovers in presence of wheel slip, is computationally intensive and significantly slows down roving. Table 2.1 shows the usage of different driving modes on MER rovers as of August 2005, eighteen months after landing: fully autonomous driving was used 27% of the time on Spirit and only 21% of the time on Opportunity.

Driving Mode	**Terrain Assessment**	**Path selection**	**Visual Odometry**	***Spirit***		***Opportunity***	
Directed Driving	no	no	no	451 m	9 %	1973 m	33%
VisOdom	no	no	YES	410 m	8 %	561 m	9%
Blind Goto Waypoint	no	YES	no	2196 m	46 %	1911 m	32%
VisOdom Goto Waypoint	no	YES	YES	379 m	7 %	121 m	2%
Guarded Motion	YES	no	no	36 m	1 %	117 m	1%
Guarded VisOdom	YES	no	YES	0 m	0 %	0 m	0%
AutoNav	YES	YES	no	1315 m	27 %	1262 m	21%
AutoNav with VisOdom	YES	YES	YES	3 m	0 %	0 m	0%
				4798 m	100 %	5947 m	100%

Table 2.1: MER driving mode usage as of 15 August 2005, counting 573 sols for Spirit and 555 sols for Opportunity. [50, Ch. 3, Table 1]

In 2009, the AEGIS software was deployed on Opportunity [36]. AEGIS allows autonomous identification of targets of scientific interest: after identifying promising targets via low-resolution, wide-angle cameras, the rover autonomously images them with narrow-field, high-resolution instruments and only sends the resulting images back to Earth, with significant time and bandwidth savings. AEGIS remains ancillary to manually planned science operations: it has seen comparatively little use since 2011 [79].

The Curiosity rover is a direct descendant of Spirit and Opportunity: while its scientific package and mechanical subsystems are significantly improved with respect to the MER mission, the high-level autonomy architecture remains the same [80]. In fact Curiosity, remained stationary and performed pre-planned experiments for over one month during the 2013 Mars opposition, which made daily direct communications impossible. The AEGIS software holds much promise on MSL-Curiosity, where it may be used to autonomously target the ChemCam spectrometer as well as narrow-field cameras [118]: software deployment on MSL is planned for 2013.

Many theoretical analyses and ground-based studies on autonomous planning for single-agent and multi-agent architectures have been proposed in recent years: notable example include NASA's CLARAty robotic framework [119] and LAAS's framework [52], while Zilberstein's work [124] gives an excellent overview of mathematical methods for *single-agent* planning under uncertainty. We refer the reader to the introduction to [43] for a comprehensive review of further *theoretical* work on autonomy and planning for planetary exploration.

2.4.2 Decentralization and multiagent architectures

To date, no unmanned planetary exploration missions counting more than two agents have been launched. Many two-agent missions have been deployed: typical architectures include an orbiter and a lander experiencing limited interactions to relay telecommunications to Earth.

Thanks to its scientific interest and relative ease of access, Mars has become a testbed for multiagent interaction between vehicles independently developed from different space agencies at different dates. The planet is currently host to two U.S. rovers, Curiosity and Opportunity, and three orbiters, NASA's Odyssey and Mars Reconnaissance Orbiter and ESA's Mars Express. All three orbiters are routinely used to relay information from both rovers to Earth, allowing for operations even in absence of a direct link between the Deep Space Network and the ground vehicles. However, no direct interaction other than as a telecommunication relay is currently used or planned.

The GRAIL mission sported an innovative two-agent architecture: near-twin spacecraft were flown on identical Moon orbits and the probes' relative position was measured to micrometer precision to obtain an accurate reconstruction of our satellite's gravity field. The measurement, however, was fully passive: no interaction between the two spacecraft was required [125].

The A-Train also offers a fascinating example of multiagent (albeit not autonomous) space exploration: six Earth observation satellites (GCOM-W1, Aqua, CALIPSO, CloudSat, PARASOL, and Aura) from different agencies observe our planet from an identical polar orbit with very small differences in anomaly, offering multiple measurements of different geophysical quantities at virtually the same time and location. Satellites within the A-Train, however, are managed independently and do not interact with each other in flight: scientific data is merged after collection, on the ground.

Several ground-based studies have been carried out about the *feasibility* of multiagent *autonomous* architectures for space exploration. A notable example is NASA's biomimetic ANTS architecture: ANTS consists of a set of thousands of miniaturized, autonomous, self-similar, reconfigurable, address-

able components that can configure to form large structures with no external intervention. Proposed missions include exploration of the Moon, asteroids and Saturn's rings. ANTS is currently a technology development platform: while nonminiaturized prototypes are currently being tested, a full-scale deployment for space exploration is not foreseen for the next two decades [25, 26].

Also of note is MIT's SPHERES program. SPHERES is a robotic testbed for high-risk formation flight, autonomous docking, rendezvous and reconfiguration algorithms: it consists of a cluster of small probes, currently located within the International Space Station, used to test and validate control algorithms for autonomous spacecraft. A promising application of control algorithms developed on SPHERES is space-based interferometry, which would exploit arrays of precisely spaced probes to improve the resolution of extra atmospheric radar observations of deep space by several orders of magnitude by increasing the radar array baseline from meters to kilometers. At this time, SPHERES is a technology development program: no flight missions outside the ISS are being actively planned [69].

2.5 Conclusion

Most aspects of decentralized decision-making on *fixed*, wired networks have been studied in the CS literature.
Yet results in the field do not translate well to robotic networks. Most existing algorithms are incapable of dealing with link or node failures; those that do use inefficient redundancy rather than reconfiguring the network on the fly. Furthermore, while message and byte complexity are typically explored, emphasis is typically placed on *time* complexity: Lynch states that

> The time measure is the more important measure in practice, not only for synchronous distributed algorithms but for all distributed algorithms. The communication complexity is mainly significant if it causes enough congestion to slow down processing. [63, par. 2.6]

This contrasts deeply with robotic space exploration applications, where time complexity, often measured in fractions of a second, is typically very secondary with respect to energy consumption.

In recent years, efforts have been made to develop algorithms for wireless mobile networks. Yet the hypotheses underlying these efforts are typically incompatible with our robotic networks: leading authors such as Kuhn and Oshman typically assume agents to be completely unaware of their neighbors'

identity and of the overall number of agents participating in a computation. These hypotheses are very relevant to cellular networks and wireless networks of computers, where an user ID may be easily spoofed and the number of agents is unknown a priori; on the other hand, they are far too restrictive for robotic networks, where the number of nodes is upper-bounded and agents can be trusted to have a unique identity.

In the control community, consensus is a synonym for average-based consensus. Average-based algorithms have many advantages: they mimic the natural flocking behavior of birds and schools of fish, are extremely robust to variations in the network topology and require exchange of very simple pieces of information among nodes. If agents are only aware of their neighbors' relative positions[3], average-based algorithms are an excellent solution to the consensus problem.

On the other hand, average-based algorithms are extremely inefficient: they converge asymptotically, whereas CS algorithms typically reach a solution after a small, finite number of messages, and they send messages on all available communication channels at each time step. Modern robotic agents are able to sense, elaborate and communicate complex information: average-based algorithm underexploit their capabilities and the resulting performance suffers greatly.

Autonomous multi-agent architectures have seen extremely limited use in space exploration. In principle, hardware is not a limitating factor: conversely, distributed architectures should afford significantly higher reliability than monolithic ones. One of the reasons for this lack of adoption may therefore be the the relative immaturity of *software* architectures for multi-agent coordination and decision-making: existing paradigms translate very poorly to the constraints and opportunities typical of robotic space exploration.

In the next chapters, we will strive to bridge this gap by proposing CS-inspired algorithms optimized to reach decisions on *robotic* networks for space exploration.

[3]This could be the case if agents are only equipped with cameras or ranging sensors and unable to otherwise communicate.

Chapter 3

Problem statement

In Chapter 1, we motivated the importance of decentralized consensus for space exploration applications; after a review of the state of the art in Chapter 2, we are ready to formalize considerations outlined in previous chapters and rigorously define the consensus problem we wish to solve.

After introducing some definitions in Section 3.1, we formalize and motivate our mathematical hypotheses in Section 3.2; we introduce the metrics used to assess the complexity of consensus problems and algorithms in Section 3.3 . Finally, we formally define the problem we strive to solve in Section 3.4.

3.1 Preliminaries

3.1.1 Consensus, convex consensus, sensitively decomposable and locally computable functions

Consensus Our work borrows Nancy Lynch's definition of consensus [63, par. 5.1]. Agents are supposed to have an initial *opinion* on the value they should agree on (which can be a scalar, a vector or a logic value) and exchange messages to reach an agreement. An algorithm solves the consensus problem if the three following properties hold.

- **Agreement**: No two processes decide on different values.
- **Validity**:
    1. If all processes start with the same value k_0, then k_0 is the only possible decision value.
    2. If all processes start with the same value k and all messages are delivered, then k is the only possible decision value.

- **Termination**: All processes decide in a *finite* number of steps.

Convex consensus We restrict our attention to *convex consensus* problems: here, the consensus value $\bar{k}$ is further restricted to belong to the convex hull of the agents' initial values k_i $i = 1, \ldots, n$.

In other words, $\bar{k}$ can be *represented* as a convex combination of the initial values k_i's, i.e.:

$$\bar{k} := \sum_{i=1}^{n} c_i \, k_i, \qquad \text{where } c_i \in [0,\, 1], \text{ and } \sum_{i=1}^{n} c_i = 1,$$

where the weights c_i, $i = 1, \ldots, n$, are problem-dependent. The vector of weights $[c_i]_i$ *parameterizes* the convex consensus problem.

The above definition is redundant for boolean consensus problems. For scalar and vector values, it encompasses most reasonable consensus problems, including

- Computation of $\max_i k_i$, e.g., for leader election. This problem can be represented with the weight choice (assuming there exists a unique maximum): $c_i = 1$ if $k_i = \max_j(k_j)$ and $c_i = 0$ otherwise.

- Computation of of $\min_i k_i$. This problem can be represented with the weight choice (assuming there exists a unique minimum): $c_i = 1$ if $k_i = \min_j(k_j)$ and $c_i = 0$ otherwise.

- Average consensus, which can be employed to solve problems as diverse as distributed sensing and filtering [83], formation control [53], rendezvous [24] and coverage control [41]. This problem can be represented with the weight choice: $c_i = 1/n$.

- Weighed average consensus, which can be employed for data fusion when information about the confidence of several measurements is available. This problem can be represented with the weight choice: $c_i = 1/[\sigma_i \cdot \sum_j (1/\sigma_j)]$, where σ_i is the uncertainty of each measurement.

- Mode This problem can be represented with the weight choice: $c_i = 1/n_{mo}$ if $k_i = \text{Mode}(k_j)$ and $c_i = 0$ otherwise. n_{mo} is the number of occurrences of the mode value.

- Median. This problem can be represented with the weight choice: $c_i = 1/n_{me}$ if $k_i = \text{Median}_j(k_j)$ and $c_i = 0$ otherwise. n_{me} is the number of occurrences of the median value.

- Any *logical* operation whose outcome lies within the hull of the nodes' initial opinions of a policy to follow. If policies are mutually exclusive, this problem can be represented with the weight choice (assuming only one agent proposes the selected policy) $c_i = 1$ if i is the selected policy, c_i=0 otherwise. If the problem admits a notion of *mediation* between different policies, c_i can assume values between 0 and 1.

Sensitively decomposable, hierarchically computable and locally computable functions In [7], Awerbuch introduces the concept of *sensitively decomposable* functions:

> such functions are sensitive to every input, but the influence of a set of arguments can be represented by a string whose size is not much bigger than the size of a string needed to represent just one argument. Examples of such functions are maximum, sum, parity, majority, OR, AND.

Awerbuch's definition, however, does not capture whether the consensus function can be computed on a *subset* of opinions. We therefore introduce the concept of *hierarchically computable* and *locally computable* functions.

A hierarchically computable function obeys the following property: given the value of a function on a number of *disjoint* sets, it is possible to compute the function of the *disjoint union* of these sets and store the result in a string of the same order of magnitude as the size of the string needed to represent an argument. Examples of such functions are average and weighed average: given a subset of nodes, their contribution to the consensus value can be represented by their (weighed) average and the associated weight. Majority voting on a *limited* number of options also is a hierarchically computable function: it is sufficient to store the number of votes obtained by each option. The name is inspired by the observation that hierarchically computable functions can be computed with messages of small size on a *hierarchical* structure such as a tree.

A locally computable function, on the other hand, obeys the following property: given the value of the function on a number of *potentially overlapping* sets, it is possible to compute the function of the *union* of these sets and store the result in a string not much bigger than the size of the string needed to represent one argument. In other words,

$$S = \cup_i S_i; \qquad f(S) = f(f(S_1), f(S_2), \ldots);$$

Examples of such functions are maximum and minimum. Note that mean, weighed mean, median, mode and majority are *not* locally computable.

We restrict our *complexity* analysis to hierarchically computable functions. We do not, on the other hand, specifically exclude locally computable functions: we do, however, discuss the impact of local computability on the byte complexity of a consensus algorithm.

3.1.2 Cyber-physical networks, graphs and automata

In this work, we model an asynchronous robotic network with n agents as a connected, *undirected* graph $G = (V, E)$, where the node set $V = \{1, \dots, n\}$ corresponds to the n agents, and the edge set $E \subset V \times V$ is a set of *unordered* node pairs modeling the availability of a communication channel. We will henceforth refer to nodes and agents interchangeably. Two nodes i and j are neighbors if $(i, j) \in E$. The neighborhood set of node $i \in V$, N_i, is the set of nodes $j \in V$ neighbors of node i.

Each node is internally modeled as a I/O automaton, which is essentially a labeled state transition system commonly used to model reactive systems (a more formal definition can be found in [63, ch. 8]). All nodes are identical except, possibly, for a unique identifier (UID). The following key assumptions characterize the time evolution of each node in G:

- Fairness [63, ch. 8]: the order in which transitions happen and messages are delivered is not fixed a priori. However, any enabled transition will *eventually* happen;

- Non-blocking [63, ch. 8]: every transition is activated within l time units of being enabled and every message is delivered within d time units of being dispatched.

We will sometimes refer to our robotic network as a *cyber-physical* network: the term points to the coexistence of a physical, hardware component and a cybernetic, logical component in the system.

Spanning trees and minimum spanning trees In the following we will frequently deal with spanning trees and minimum spanning trees. We remind the reader that a *spanning tree* on a graph $G = (V, E)$ is an *acyclic*, *connected* subgraph $T = (V, E')$ of G such that all nodes in G belong to T and $E' \in E$. If edges of G have a weight, we define a *minimum spanning tree* as a spanning tree $M = (V, E'')$ of G such that the sum of the weights of edges $e'' \in E''$ belonging to M is minimal. It can be shown that, if the weights of edges $e \in E$ belonging to G are unique, G has an unique minimum spanning tree.

3.1.3 Complexity and asymptotic notation

In this work, we focus our attention on the running time, number of messages exchanged and number of bits exchanged by a class of consensus algorithms on robotic networks. Borrowing from the CS literature, we define these three quantities as *time*, *message* and *byte complexity* (or, interchangeably, *cost*). Section 3.3 below gives a rigorous definition of the metrics employed to assess these quantities.

In order to evaluate the complexity of an algorithm and its rate of growth as the number of agents increases, we make ample use of Θ, O, and Ω notation.

Asymptotic notation Asymptotic notation, presented in [22], is a powerful tool to evaluate and compare the rate of growth of complexity as a function of the input size (in our case, the number of agents). It concentrates on the highest-order contributor to the complexity of an algorithm, ignoring slower-growing terms and constant factors.

Asymptotic notation is widely used in the CS community: an asymptotically more efficient algorithm is typically the best choice for all inputs but very small ones, which usually bear no interest to a computer scientist.

We, on the other hand, must be careful to always verify the validity of this approach: while a fifty element array may be trivially small in CS, a 20- or 50-agent *robotic* network is quite large by current standards.
This does not mean that asymptotic notation should be scrapped altogether: an algorithm with a complexity of $8n \log n$ is significantly more efficient than one requiring n^2 operations for a 50-element network, despite the former's high constant factor; as the number of agents increases[1], the performance gap only widens. Yet we will take care not to neglect lower-order terms in our analysis, so as to verify the validity of asymptotic notation whenever we employ it.

Upper bounds: O and o notation Asymptotic upper bounds on the rate of growth of a function are typically expressed with O and o notation.

The notation $O(g(n))$ denotes the set of functions $f(n)$ that are *asymptotically smaller* than $g(n)$: rigorously,

$$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

[1]For instance, NASA's ANTS mission architecture concept may see up to a thousand agents interacting to prospect Kuiper belt asteroids [25].

Figure 3.1a shows an example of O notation.

The notation $o(g(n))$ is used to express upper bounds that are known not be tight. Rigourously, the set of functions $f(n) = o(g(n))$ obeys

$$o(g(n)) = \{f(n) : \text{for all positive constants } c \text{ there exists } n_0 > 0 \text{ such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0\}$$

Lower bounds: Ω and ω notation Ω and ω notations are used to express lower bounds on the rate of growth of a function. ω notation is used when a bound is known not to be tight, whereas Ω is used in all other case. Rigorously, $\Omega(g(n))$ and $\omega(g(n))$ denote the following sets:

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}$$

$$\omega(g(n)) = \{f(n) : \text{for all positive constants } c \text{ there exists } n_0 > 0 \text{ such that } 0 \leq cg(n) < f(n) \text{ for all } n \geq n_0\}$$

Figure 3.1b shows an example of a function $f(n) = \Omega(g(n))$.

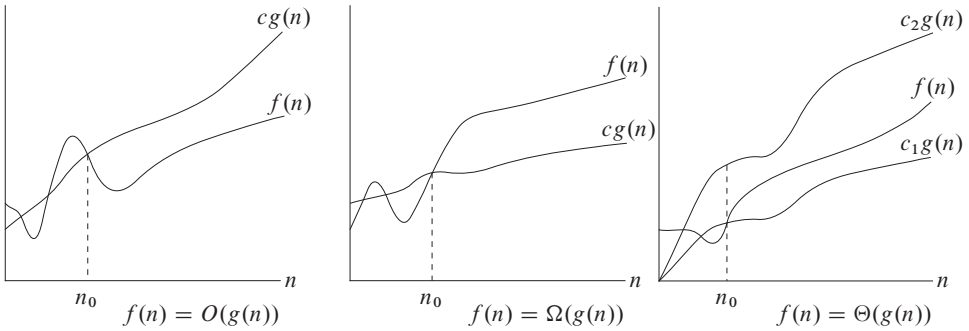
Θ notation When a function's asymptotic upper and lower bounds coincide, Θ notation is employed. Rigourously,

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}$$

Figure 3.1c shows an example of a function $f(n) = \Theta(g(n))$.

Worst-case and average-case complexity The complexity of an *execution* of an algorithm is easy to compute: one can simply time the execution or count the number of messages exchanged. Evaluating the complexity of an *algorithm*, on the other hand, requires additional hypotheses on its input values: in our case, we should specify the initial network topology, its evolution in time and the agents' initial values.

The two main approaches found in the literature are *worst-case* and *average-case* analysis.
In the former, the complexity is defined as the worst possible cost required by a successful execution of the algorithm for any input within a given set.
In the latter case, the algorithm's performance is evaluated on an average-case input. The definition of "average case" is strongly problem dependent:



(a) O notation (b) Ω notation (c) Θ notation

Figure 3.1: Graphic examples of the Θ, O, and Ω notations (from Cormen's Introduction to Algorithms [22, fig. 3.1])

it typically follows from considerations about known statistical properties of the input and/or on randomization occurring within the algorithm.

In this work, we mainly deal with analytical worst-case complexity. Average case complexity on select randomized network topologies is numerically evaluated in Chapter 6.

3.1.4 Random geometric graphs

Throughout this work, we consider a number of network topologies. We dedicate special attention to random geometric graphs, which accurately model many real-world wireless sensor networks (see e.g. [55]) and capture the uncertainty inherent in many robotic networks with non fully deterministic deployment mechanisms.

Nodes in a random geometric networks are uniformly distributed within a d-cube of size one, where d is the number of dimensions of the problem. For robotics applications, $d \in \{2, 3\}$. Nodes are neighbors if their Euclidean distance is lower than a prescribed radius $\bar{r}(n)$, typically decreasing in n.

Random geometric networks have long been studied; for our purposes, it is sufficient to highlight a few useful statistical properties.

- Connectivity. It can be shown that the number of isolated clusters in a random geometric graph presents two sharp thresholds. Below the first threshold, $O(n)$ nodes have no neighbors; above the second threshold, the graph is almost surely connected; in between, the graph has one giant component containing $O(n)$ nodes. It can be shown that on a 2D random geometric graph the second (connectivity) threshold,

which especially interests us, is

$$r_c \approx \left(\frac{\ln n}{\pi n}\right)^{1/2}$$

This surprising property can be generalized: Goel et. al. [45] show that all *monotone increasing* properties of the graph[2] have sharp thresholds.

- Neighbors: it is easy to see that, given the uniform distributions of nodes, the expected degree of each node $E[D]$, i.e. is the average number of neighbors, is $V_d(r(n)) \cdot (n-1)$, where $V_d(r(n))$ is the volume of the d-sphere of radius $r(n)$. For $d = 2$,

$$E[D] = \pi r(n)^2 \cdot (n-1)$$

We refer the reader to Penrose [97] for a comprehensive treatment of the subject.

3.2 Hypotheses

Chapter 2 shows how how slight changes in the problem's hypotheses can lead to largely different solutions and complexities.To this end, in this section we lay out and motivate our hypotheses, showing their relevance to modern multiagent systems for space exploration.

Knowledge of the order of magnitude of the number of agents In [7], Awerbuch shows that, if no *a priori* information is available, counting the number of agents is as hard as computing a sensitively decomposable function on a static, connected network. Awerbuch's improved GHS algorithm solves both problems optimally under certain assumptions.

In this work, we do not assume that agents know the *exact* number of participants in the network. We do, on the other hand, assume that they know the *order of magnitude* of the number of agents $O(n)$. This allows us to formulate simple termination criteria that do not require an expensive counting stage in Ch. 5.

The hypothesis is very relevant to robotic networks, especially in space-borne applications: the number of deployed agents is well known a priori, yet a small number of agents may fail to activate due to hostile environmental conditions or uncertainties intrinsic in the deployment phase.

[2] A property $\mathcal{P}$ of a graph is monotone increasing if, whenever a subgraph H of G has $\mathcal{P}$, then G has $\mathcal{P}$ too.

In practice, we require the maximum error in the number of agents to be strictly smaller than $\underline{t}$, a parameter defined by the designer. The relevance of this parameter will be made clear in Chapter 5.

Identical agents with unique UIDs We model our agents as *identical* nodes running the same logic code. Each agents, however, sports a unique User ID, typically a natural number.

A hierarchical structure is intrinsically less robust than a "flat" one: while a flat structure is able to tolerate the loss of any agent, a hierarchical architecture may have to undergo significant reconfigurations following the loss of one (local or global) leader.

Depending on the mission, it may be advantageous to deploy agents with different *physical* capabilities. On the other hand, it is sensible to refrain from embedding a hardwired hierarchical structure or markedly different *software* proficiency in the agents: the uncertainty of the environment, especially in the deployment phase, suggests giving every node the same capabilities, waiting until deployment is over before configuring them in a hierarchical or semi-hierarchical structure.

Asynchronous network with asynchronous wakeup In a *synchronous* network, all nodes evolve at once following a global clock: at each round nodes simultaneously receive messages from their neighbors, update their internal state, then place messages in their neighbors' inboxes before proceeding together to the next round.

Asynchronous networks, on the other hand, do not offer such timing guarantees. We assume communication links to be FIFO[3]: a partial ordering between messages exchanged between two nodes is maintained. On the other hand, no *global* ordering of messages is maintained and agents do not have access to an exogenous global clock.

Synchronous networks are a special case of asynchronous networks: results on the latter directly translate to the former. On the other hand, synchronous algorithms typically do not work on asynchronous networks out of the box: significant effort has been spent on design of *synchronizers* that distributely generate a global clock and make it available to all nodes. We refer the reader to Raynal's work [100] for details.

In this work, we model our network as an *asynchronous* network with *asynchronous* node wakeup. Real networks are asynchronous: all synchronous algorithms assume (more or less explicitly) the existence of a shared clock, which must be enforced in design.

[3]First In, First Out, a natural assumption for *direct* wired or wireless TCP-like links

For terrestrial applications it is trivial to obtain very high precision time measurements thanks to the GPS constellation of navigation satellites. Yet, even barring security and availability considerations, a shared time it is not enough to guarantee synchronism unless an hard upper bound on the transition time of each agent and the delivery time across each communication link are known: this may lead to very conservative implementations with extremely high real-world time complexity.
For space borne applications, guaranteeing availability of a shared clock is a formidable task *per se*: it is more sensible to assume that no shared time is available and, should the need arise, explicitly devise protocols to this end.

A consequence of the hypothesis above is that nodes will wake up asynchronously: having no global time available, they will come online and start running as they are available.

Unfortunately our algorithms do require nodes to wake up once they are contacted by a neighbor: to this end, we introduce a *dwell time* at the beginning of any distributed algorithm to guarantee that all agents are ready to react to exogenous inputs when the first message is sent. This dwell time does not help enforce synchronism among the agent: it simply guarantees that no agent will ignore external inputs only to come online at a later time.

In synchronous networks, time complexity is often expressed in terms of the number of rounds. This approach does not apply to asynchronous networks: on the other hand, the non-blocking property presented in sec 3.1.2 can be used to impose upper bounds on the time complexity of a process. In the following we will sometimes confuse the reader by referring to the combined maximum time complexity of a transition and a message $(l + d)$ as a *round*, even though our work is set in an *asynchronous* setting unless otherwise specified.

Unknown network topology, neighbor discovery protocol Our agents have no global knowledge of the network topology. This is rather sensible: if agents are dynamically deployed, the network topology is unknown, even to a global observer, until deployment is over; furthermore, if all agents shared knowledge of the entire network topology, it would be comparatively easy to elect a leader based on a previously agreed-upon rule and with no inter-agent communication, making the problem far less interesting.

On the other hand, agents do know the number of their neighbors and their distance: if the number and identity of neighbors is unknown, all non-broadcast-based protocols, which are the focus of this work, are inapplicable.

For wired networks, knowledge of local neighbors is trivial: an agent should be well aware of the number of outgoing physical communication channels. In wireless networks, on the other hand, such knowledge is not

guaranteed *a priori*.

Throughout this work, we assume our network to be equipped with a low-power network discovery protocol that allows agents to precisely determine the number and the distance of its neighbors upon startup. Such a protocol can be implemented in several ways: for spaceborne applications, where atmospheric attenuation is typically not an issue, the power of an agent's broadcast as it is received by another could be used as a very precise proxy for the distance among the two. Neighbor discovery is even easier for terrestrial applications: location-aware agents could broadcast their position and locally compute neighbors' distance as they receive their broadcasts.

Directional communication In this work we assume agents to use an unicast protocol to communicate with their peers: broadcast communications are not forbidden but, when an agent communicates with m neighbors, the associated communication cost is m times higher than if a single agent had been contacted.

One of the main concerns of this work is to devise *energy-efficient* algorithms for cyber-physical networks: we therefore believe it is safe to assume that hardware solutions for directional communication will be included in the design of any multiagent system where energy consumption is a major concern, such as those used in space exploration applications.

Directional communication can be achieved with mechanically or electronically steerable antennas. Mechanical steering is simple and effective, but it has significant drawbacks, including reliability, comparatively high maintenance requirements and potentially slow actuation. Mechanical solutions are also strongly discouraged in space applications because of their weight and inherently low reliability. Electronic steering, which can be achieved via beamforming, overcomes most of these limitations: phased array antennas achieve double-digit gains with no moving parts in a light and compact package. Beamforming has long been studied [6, 116] and used [11] in commercial terrestrial applications; phased array antennas are starting to see adoption in space applications , starting with the MESSENGER mission to Mercury [120].

TCP-like communication protocol We assume the inter-agent protocol to be TCP-like. In particular, we require the protocol to have the following two properties:

- No incomplete or incorrect messages are delivered across a nonfaulty communication link.
- Each successfully sent message is acknowledged.

A TCP-like protocol allows agents to automatically detect link failures and ensures that no partial or incorrect messages are delivered. This is coherent with our hypotheses of reliable FIFO communication channels with potential stopping failures, introduced in the next paragraph.

Link failures We allow inter-agent *links* to experience *stopping failures*. Links may go offline but not come back: messages across the link at the time of failure are dropped. Agents on both sides of the link are notified of the failure when they try to send a message: in practice, agents note that a link is down when their messages are not acknowledged for a given, low number of times.

We do not allow new links to be added to the network during execution.

Throughout this work, we assume that the network graph stays connected despite link failures. Links can not come back online after they fail: once disconnected, a network never achieves connectivity again and it is trivial to observe that no algorithm can solve the consensus problem on a disconnected network.

Our hypotheses are coherent with a set of *stationary* agents, e.g. planetary penetrators, that experience link failures because of network equipment degradation, electronic or mechanical failures or nontransient environmental conditions. Our results can be applied to *slow-moving* agents too. Current planetary exploration rovers have top speeds in the order of hundreds of meters per *day*: their low speed makes such agents essentially stationary for our purposes.

Many link failure modes, especially *electrical* failures, are accurately modeled by a Poisson process parametrized with parameter λ. In Chapter 5 we discuss the effect of a link's mean time to failure (which is trivial to compute, given λs for all potential failure modes) on the tuning parameter of the hybrid algorithm we propose.

Note that an *agent* stopping failure can be modeled as the simultaneous failure of all links comprising that node.

Message size Most of the computer science literature on consensus considers message size as a constraint: messages are often not allowed to be larger than a given amount, typically $O(\log n)$ or just enough to store a constant number of User IDs.

We see no reason to artificially limit the size of a message: while sending large (e.g. $O(n)$) messages may be impractical as a network gets very large, this is less of a concern for small to medium networks counting up to thousands of agents usually found in robotics applications.

We do not, however, disregard message size completely: to this end, we take message size into account when computing *byte complexity*, introduced in Section 3.3.

Edge weights Each edge in the graph representing the cyber-physical network has an associated weight, representing the *distance* between the end nodes.

We assume edge weights to be *unique*. This hypothesis is merely technical: while multiple nodes may indeed be at the same distance, it is trivial to assign unique weights to congruent edges by appending the end nodes' IDs to their distance. Lexicographic ordering can then be used to rank edges in a strict total order.

3.3 Performance metrics

Let P be a problem to be solved by the nodes in G; more formally, P represents the task of computing a *computable* function of the initial values of the I/O automata in the network G. Let $\mathcal{A}$ be the set of algorithms implementable on the I/O automata in G, $\mathcal{G}$ be a set of graphs with node set $V = \{1, \ldots, n\}$, $\mathcal{K}$ be the set of initial conditions for the I/O automata (independent of algorithm), and $\mathcal{F}(a, k)$ be the set of fair executions for $a \in \mathcal{A}$ and $k \in \mathcal{K}$.

The following definitions naturally capture the notions of time complexity, communication complexity, and are widely used in the theory of distributed algorithms [63, ch. 8].

3.3.1 Time complexity of a problem

Time complexity is defined as the infimum worst-case (over initial values and fair executions) completion time of an algorithm. Rigorously, the time complexity for a given problem P with respect to the class of graphs $\mathcal{G}$ is

$$\mathrm{TC}(P, \mathcal{G}) = \inf_{a \in \mathcal{A}} \sup_{G \in \mathcal{G}} \sup_{k \in \mathcal{K}} \sup_{\alpha \in \mathcal{F}(a,k)} T(a, k, \alpha, G),$$

where $T(a, k, \alpha, G)$ is the first time when all nodes have computed the correct value for problem P and have stopped

3.3.2 Communication complexity of a problem

Communication complexity is defined as the infimum worst-case (over initial values and fair executions) number of messages exchanged by an algorithm

before its completion. Rigorously, the communication complexity for a given problem P with respect to the class of graphs $\mathcal{G}$ is

$$\mathrm{CC}(P, \mathcal{G}) = \inf_{a \in \mathcal{A}} \sup_{G \in \mathcal{G}} \sup_{k \in \mathcal{K}} \sup_{\alpha \in \mathcal{F}(a,k)} M(a, k, \alpha, G),$$

where $M(a, k, \alpha, G)$ is the number of messages exchanged between the initial time and $T(a, k, \alpha)$.

The type of message exchanged depends on the algorithm. In average-based consensus algorithms, nodes typically exchange their state, a real number. In algorithms such as GHS, a wide range of logical commands establishing hierarchical relationships, informing neighbors about the progress of the algorithm and requiring them to perform edge searches is exchanged: we refer the reader to [40] and [7] for details. The key point is that a message "counts the same" regardless of its type and size.

3.3.3 Byte complexity of a problem

In many communication protocols, message size plays an important role in the energy consumption of a problem. Communication complexity, however, fails to capture this effect.

The byte complexity of a *message* is proportional to its size in bytes. The byte complexity of a *problem* is defined as the infimum worst-case (over initial values and fair executions) overall size of all messages exchanged by any algorithm before its completion. Rigorously, the byte complexity for a given problem P with respect to the class of graphs $\mathcal{G}$ is

$$\mathrm{BC}(P, \mathcal{G}) = \inf_{a \in \mathcal{A}} \sup_{G \in \mathcal{G}} \sup_{k \in \mathcal{K}} \sup_{\alpha \in \mathcal{F}(a,k)} B(a, k, \alpha, G),$$

where $B(a, k, \alpha, G)$ is the overall size (in bytes) of all messages exchanged between the initial time and $T(a, k, \alpha)$.

3.3.4 Complexity of an algorithm

The definitions of time, communication and byte complexity of an algorithm follow directly from the definitions of the same quantities for a problem.

The time complexity of an algorithm is the worst-case (over initial values and fair executions) completion time of the algorithm; the communication complexity is the worst-case number of messages exchanged by the algorithm before its completion; the byte complexity is the worst-case overall size of messages exchanged by the algorithm before its completion.

Rigorously,

$$\mathrm{TC}(P, \mathcal{G}, a) = \sup_{G \in \mathcal{G}} \sup_{k \in \mathcal{K}} \sup_{\alpha \in \mathcal{F}(a,k)} T(a, k, \alpha, G)$$
$$\mathrm{CC}(P, \mathcal{G}, a) = \sup_{G \in \mathcal{G}} \sup_{k \in \mathcal{K}} \sup_{\alpha \in \mathcal{F}(a,k)} M(a, k, \alpha, G)$$
$$\mathrm{BC}(P, \mathcal{G}, a) = \sup_{G \in \mathcal{G}} \sup_{k \in \mathcal{K}} \sup_{\alpha \in \mathcal{F}(a,k)} B(a, k, \alpha, G)$$

3.3.5 Discussion of complexity measures

Power consumption is a limiting factor in many modern distributed systems, including multiagent robotic systems; mobility (if applicable), communication and thermal control are typically the three major contributors to power depletion on robotic platforms for space exploration. Yet most of the research on distributed algorithms for robotic networks focuses on time complexity, with little attention to the *energy* required for their execution.

This thesis strives to explore the "energy complexity" of the distributed consensus problem. Communication complexity is a reasonable proxy for the energy cost of a problem in settings where

- the energy cost of a message is independent of the receiver's distance (although the neighborhood of the sender can be a function of the range of the communication equipment);
- the cost of a message is independent of the payload size;
- the cost of sending the same piece of information to k agents is k times the cost of a single message (which is in general not true for broadcast communication models);

Independence of the cost on the payload size holds true for protocols in which constant-length or short messages are exchanged: in the second case, the cost of the message is dominated by the fixed cost to establish the connection, handshake, exchange connection parameters and frame the payload.

Byte complexity, on the other hand, is an appropriate proxy for the energy cost of a problem in settings where the message cost depends linearly on the message *size* and the number of receivers but is independent of the receiver's distance. Linear dependence of the cost on payload size holds true for lightweight protocols whose handshakes, headers and acknowledgements are small with respect to the actual payload.

Throughout this thesis, we will study both message complexity and byte complexity; we will favor the latter in presence of large messages.

3.3.6 Hybrid metrics

Thus far, research has strived to produce algorithms that optimize *one* of the metrics above, typically favoring time complexity. Yet real-world application often call for optimization of *hybrid* metrics.

Consider for instance a low-power wireless sensor network observing a transient phenomenon. Examples could include a network of penetrators observing the geophysical evolution of a SSSB after an artificially induced shock or a temperature change as they cross the body's terminator. Here, low power consumption is paramount, yet the algorithm has to be faster than the phenomenon under observation to correctly sample it. A relevant hybrid metric would minimize energy consumption (which we model via byte complexity for simplicity) under the constraint that time complexity be lower than a given amount $\overline{\text{TC}}$. Rigourously, we seek

$$\text{argmin}_a[\text{BC}(P, \mathcal{G}, a) \text{ s.t. } (\text{TC}(P, \mathcal{G}, a) < \overline{\text{TC}})]$$

Other non space-based applications may call for different metrics. Consider a *search and rescue* application. Examples may include a swarm of drones looking for a survivor at sea or a low-power sensor network deployed over an avalanche. Here, time complexity is paramount; yet, if the agents are battery-powered, the batteries' capacity gives an hard upper bound on the energy consumption. We therefore seek a time-optimal algorithm under the constraint that energy consumption be lower than a given amount $\overline{\text{EC}}$. Let us assume once again byte complexity to be an accurate proxy for energy complexity: rigorously, we require

$$\text{argmin}_a[\text{TC}(P, \mathcal{G}, a) \text{ s.t. } (\text{BC}(P, \mathcal{G}, a) < \overline{\text{BC}})]$$

3.3.7 Robustness

The robustness of an algorithm relates to the number and type of failures it can withstand while still being able to successfully solve the problem it addresses. To this end, we define two metrics for robustness. As previously stated, we address *link* failures throughout this work.

Single points of failure The number of single points of failure, or SPF, is defined as the number of links whose failure requires significant reconfiguration of the algorithm and temporarily or permanently stops nominal execution.

Time to recovery The time to recovery, or TTR, is the time required for the algorithm to reconfigure after a major failure and resume computation. If reconfiguration is impossible, the TTR may be infinite.

3.4 Problem statement

We are finally ready to state the two main problems we wish to solve in this thesis.

The first problem concerns optimality with respect to time complexity, message complexity and byte complexity *separately*;

> **Problem 1 (Convex consensus with maximal set of graphs):** — Let $\mathcal{G}$ be the set of all graphs with node set V. Find the order of growth for $\mathrm{TC}(P, \mathcal{G})$, $\mathrm{CC}(P, \mathcal{G})$ and $\mathrm{BC}(P, \mathcal{G})$ where P is the convex consensus problem with a hierarchically computable consensus function. Find, is possible, an algorithm that achieves the order of growth of $\mathrm{TC}(P, \mathcal{G})$, an algorithm that achieves the order of growth of $\mathrm{CC}(P, \mathcal{G})$ and an algorithm that achieves the order of growth of $\mathrm{BC}(P, \mathcal{G})$ (i.e., a time-optimal algorithm, a message-optimal algorithm and a byte-optimal algorithm, not necessarily coincident).

In the second problem, we seek an algorithm that can satisfy tradeoffs between time complexity and energy consumption, achieving optimal byte complexity or optimal time complexity depending on a parameter and transitioning smoothly between the two.

> **Problem 2 (Parametrized convex consensus algorithm):** — Let $\mathcal{G}$ be the set of all graphs with node set V. Find a parametrized algorithm $a(\tau)$, $\tau \in [0, 1]$ that solves the convex consensus problem P with optimal order of growth of $TC(P, \mathcal{G}, a) = TC(P, \mathcal{G})$ for $\tau = 0$, optimal order of growth of $BC(P, \mathcal{G}, a) = BC(P, \mathcal{G})$ for $\tau = 1$, and orders of growth $TC(P, \mathcal{G}, a(\tau)) < TC(P, \mathcal{G}, a(\tau = 1))$ and $BC(P, \mathcal{G}, a(\tau)) < BC(P, \mathcal{G}, a(\tau = 0))$ for $\tau \in (0, 1)$.

3.5 Conclusion

This chapter presents and motivates the mathematical hypotheses that will guide our work towards a solution of Problems 1 and 2 in Chapters 4 and 5

respectively. After presenting a formal definition of the consensus problem and introducing the notions of convex consensus, hierarchically computable and locally computable consensus function, we show how to describe a cyber-physical network of stationary or slow-moving robotic agents as a graph of I/O automata with constrained time evolution. We then list and motivate our hypotheses regarding the network:

- Each agent knows the order of magnitude of the overall number of nodes;
- Agents are identical except for a unique User ID;
- Agents share no global clock and evolve asynchronously;
- Agents are aware of their neighbors;
- Communications are directional and exploit a TCP-like protocol;
- Link failures are modeled by a Poisson process;
- Edge weights are proportional to the inter-agent distance;

Each hypothesis is relevant to the space exploration applications we consider.

We introduce three performance metrics: time, message and byte complexity. The first metric is the only performance parameter typically considered in current consensus algorithms, while message and byte complexity relate to the energy required to solve a problem on a cyber-physical network.

We also quantify robustness of the algorithm to link failures through through two metrics, the number of single points of failure and the time to recovery.

Finally, we formally state the two problems we wish to solve. First, we want to identify attainable lower bounds for the time, message and byte complexity of the consensus problem; we also wish to find algorithms that achieve these bounds.

However space exploration applications often require good message or byte complexity with constraints on the maximum time complexity; moreover, as we will show in the following, low message complexity often relates to lack of robustness. We therefore wish to identify an *hybrid* algorithm that achieves time-optimal *or* byte-optimal behavior and seamlessly move from the former to the latter according to a tuning parameter.

Chapter 4

Fundamental limitations

In this Chapter, we explore the first problem outlined in Section 3.4: we look for *lower bounds* on the rate of growth of the time, message and byte complexity of the consensus problem as a function of the number of agents and *algorithms* that achieve these lower bounds.

4.1 A lower bound on time complexity

4.1.1 A lower bound on the time complexity of consensus

We define the *distance* between two nodes as the length of the shortest path connecting these nodes. The *diameter* of the network is the maximal distance between any given pair of nodes belonging to the network.

Consensus problems with hierarchically computable consensus functions require that nodes hear from all others, directly or indirectly, before making a decision. A trivial lower bound on the time complexity of consensus problems is therefore given by the diameter of the graph representing the network: information from a node on one end of the diameter of the graph requires at least $\mathrm{Diam}(G)$ rounds to reach the node on the other end of the diameter.

The lower bound is reachable under certain hypotheses: a simple *flooding* algorithm achieves optimal time complexity in absence of failures.

4.1.2 A time-optimal flooding algorithm

In a flooding algorithm, every node sends all information it has ever received to each of its neighbors until convergence. A simple optimization is possible: a node can only send information *once* after it learns it, significantly reducing the message and byte complexity.

Flooding is time-optimal on any connected network.

Lemma 4.1.1 (Time complexity of the simple flooding algorithm)**.** *In absence of failures, nodes at distance d from a given node i learn i's opinion at round d.*

Proof. By induction on the number of rounds.

Basis At Round 1, information from node i is sent to each of its neighbors.

Inductive step Let us assume nodes at distance r from node i have learned about node i's opinion at round r. At round $r+1$, these nodes send their neighbors all information they have received at round r. Nodes at distance $r+1$ from i therefore learn about its opinion at round $r+1$, if they exist. □

From Lemma 4.1.1 it follows that all nodes receive information from all other nodes within $\mathrm{Diam}(G)$ rounds.

Termination If nodes know the *exact* number of agents participating in the computation and the function being computed is not locally computable, termination is trivial: a node stops once it has received n pieces of information. If, however, n is unknown to the agents or the function of interest is locally computable, making it hard to count the number of pieces of information received, termination is less trivial. If the diameter of the network is known, nodes can terminate after $\mathrm{Diam}(G)$ rounds (equivalently, $\mathrm{Diam}(G)(l+d)$ time units in an asynchronous setting). If the diameter is unknown but the function is not locally computable and the network is *synchronous*, a simple termination criterion can be devised: once a round passes without an agent receiving any new pieces of information, the agent can be sure that it has collected information from every other node.

Lemma 4.1.2 (Termination of flooding on synchronous networks)**.** *Consider a synchronous network of agents executing a flooding algorithm reaching consensus on a non locally computable function. If an agent receives no new pieces of information during one round, then the agent has heard from all other nodes in the network.*

Proof. By contradiction. Let us assume that a node j receives no information at round r but receives information from node i *for the first time* at round $r+1$. By Lemma 4.1.1, node i is at distance $r+1$ from node j. Then there exists a node k at distance r from node j; by the same Lemma, node j hears from node k at round r. □

In an asynchronous setting, the *Alpha* synchronizer presented by Lynch in [63, par. 16.4] can be used to obtain the same result with no asymptotic increase in the time complexity.

If nodes have access to an upper bound $\bar{n}$ of the number of agents, they can wait $\bar{n}$ rounds ($\bar{n}(l+d)$ time units on an asynchronous networks) before terminating. Note that agents are able to correctly compute the consensus value after $\text{Diam}(G)$ rounds: they just don't know it.

Message and byte complexity Each edge is crossed by information about each node at most once: the byte complexity is therefore $O(|E|nb)$, where b is the size of information held by each node. If nodes attach their ID to the information they send, typically $b = \Omega(\log n)$ since the UID requires at least $\log n$ bytes.

At each time step, at most $|E|$ messages are sent: the message complexity of flooding is therefore $O(\text{Diam}(G)|E|)$. Note that, if the function of interest is locally computable, this bound on the message complexity allows to establish a tighter lower bound on byte complexity at $O(\text{Diam}(G)|E|b)$: no message is larger than b bytes.

In an asynchronous setting, implementation of the *Alpha* synchronizer requires exactly $2 \cdot \text{Diam}(G)|E|$ additional messages of constant size.

Failures Gray [47] shows that, if arbitrary edge failures are allowed, no algorithm can solve the consensus problem. On the other hand, if permanent edge failures are allowed that leave the network *connected*, the flooding algorithm outlined above converges: the proofs presented above do not make any specific assumptions about the shape of the network. Note that the convergence time is upper-bounded by the diameter of the network *after* failures occur.

4.2 A lower bound on communication complexity

4.2.1 Dense networks

In [57], Korach et al. show that any problem that requires use of a *spanning subgraph* of the network requires use of up to $|E| - 1$ edges (and therefore $\Omega(|E| - 1)$ messages) on a certain class of *almost complete* graphs. It is easy to see that any consensus algorithm whose consensus function depends on *all* nodes' initial values needs to use a spanning subgraph of the network: in absence of a spanning subgraph, information can't travel from a given node to all other nodes.

It follows that the order of growth of the communication complexity of consensus on the *maximal* set of graphs is lower-bounded by $|E|$.

Lemma 4.2.1. *Let $\mathcal{G}$ be the set of all graphs with node set V, then $CC(P, \mathcal{G}) \in \Omega(|E|)$.*

Proof. Consider an "almost complete" network with $\Theta(n^2)$ edges but fewer than $n(n-1)/2$ edges, i.e. not fully connected. Then Korach [57] shows that, in order to solve the consensus problem on this network, messages must be sent across $|E| - 1$ edges. Therefore there exist $G \in \mathcal{G}$ s.t. $CC(P, G) = \Omega(|E|)$. If follows that $CC(P, \mathcal{G}) = \Omega(|E|)$. □

A message-optimal algorithm on dense networks Gallager, Humblet and Spira's algorithm builds a rooted minimum spanning tree on an arbitrary network with $O(n \log n + |E|)$ messages. Once a rooted spanning tree is in place, the consensus problem is trivially solved: the root can collect information from all other nodes via a broadcast followed by a convergecast on the tree, with an overall communication complexity of $2n$. If the consensus function is hierarchically computable, the byte complexity of this phase is $2nb$.

GHS is therefore message-optimal on almost complete networks, where $|E| = \Omega(n \log n)$.

4.2.2 Sparse networks

In the previous section, we show that there exist network topologies where $\Omega(|E|)$ messages are required to solve the consensus problem.

Furthermore, if $|E| = \Omega(n \log n)$, the GHS algorithm offers an asymptotically message-optimal solution to the consensus problem.

In the following, we show that there exist network topologies with $O(n)$ edges where any *asynchronous* algorithm solving a *convex* consensus problem requires $\Omega(n \log n)$ messages.

The proof technique leverages the results in Lynch [63], which shows that leader election on a ring of size n requires at least $\Omega(n \log(n))$ messages. The proof assumes that the network size is a power of two: extension to natural numbers is straightforward.

The proof for the lower bound requires two preliminary lemmas.

Lemma 4.2.2 (I/O limitations)**.** *Consider an infinite set S of I/O automata running a convex consensus algorithm. Then all but at most one of them can send a message without first receiving any messages.*

Proof. By contradiction. Let us assume that there exist agents $\exists p, q \in S$ that can't send a message without first receiving any.

Let us consider the single-agent network shown in fig. 4.1*a*. p can not send any message, since it receives none. Yet it must reach consensus on a convex combination of the values of nodes in the ring, i.e. its initial value.

Let us now consider the network shown in fig. 4.1*b*. With the same reasoning as above, q reaches consensus on its initial value.

Let us now arrange p and q them in a ring, as shown in fig. 4.1c. Neither p nor q can send messages without first receiving a message; the ring is therefore silent. The execution of the algorithm on this network is therefore indistinguishable from the previous two cases for both p and q: p decides on its initial value and so does q. These two values are, in general, different: we have thus reached a contradiction. □

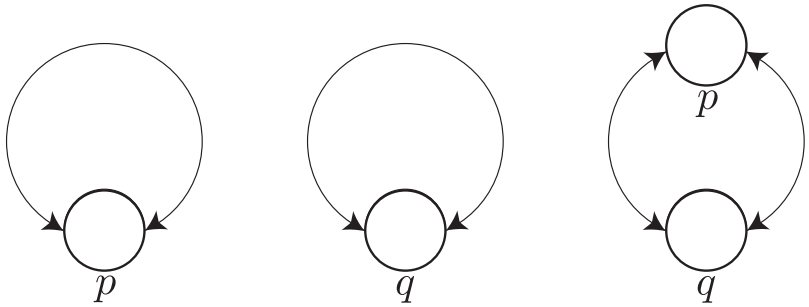


Figure 4.1: All but one of the automata must be able to send a message before receiving any.

Lemma 4.2.3 (Communication complexity on a line)**.** *For any r, there exist infinite pairwise disjoint lines of I/O automata $L \in \mathcal{L}_r$ such that $L \in \mathcal{L}_r \rightarrow |L| = 2^r$ with communication complexity $CC(L) \geq r \cdot 2^{r-2}$.*

Proof. We prove the claim by induction on r.

Basis $r = 0$: the claim is trivial.

$r = 1$: Lemma 4.2.2 shows that at least one of the two participating nodes can send a message before receiving any. Therefore there exists one execution in which at least one message is sent.

Inductive step We want to show that, provided we have a set $\mathcal{L}_{r-1}$ with $|L| = 2^{r-1}$ and $C(L) \geq (r-1) \cdot 2^{(r-3)}\, \forall L \in \mathcal{L}_{r-1}$, we can build a set $\mathcal{L}_r$ with analogous properties.

Let us call $n = 2^r$. The communication cost is thus $\geq n/4 \cdot \log(n)$.

We proceed by contradiction. Let us consider four lines A, B, C, D belonging to $\mathcal{L}_{r-1}$ and four executions α, β, γ, δ of a consensus algorithm

on these lines. By the inductive hypothesis, the communication cost of each execution is higher than $(n/8) \cdot \log(n/2) = (n/8) \cdot (\log(n) - 1)$

Let us now arrange A and B in a line, as shown in fig. 4.2, and consider an execution of the convex consensus algorithm in which all messages going through the junction are delayed until A and B reach a silent state. It is easy

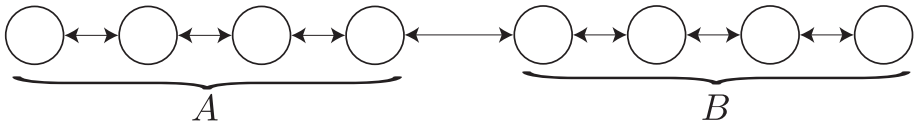


Figure 4.2: Junction of lines A and B

to see that, in this scenario, A and B follow executions α and β respectively, with an overall communication cost of $2 \cdot n/8 \cdot (\log(n) - 1)$. The delayed messages are then delivered.

If more than $n/4 - 1$ messages are generated at this point, we reach the contradiction that we sought. Let us therefore assume that at most $n/4 - 1$ messages are generated. This means that information about the junction does not reach either the midpoint of A or the midpoint of B and does not propagate beyond either. All nodes before the midpoint of A and after the midpoint of B are unaware of (and therefore unaffected by) the junction. Yet the nodes reach an agreement.

The same holds if we join B and C, C and D or D and A.

Let us now join A, B, C and D in a ring, as shown in fig. 4.3.

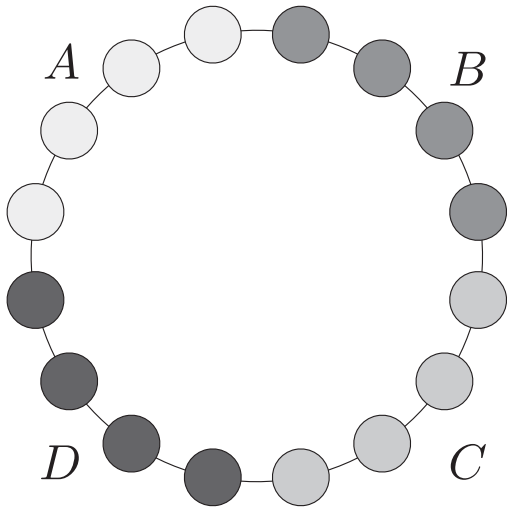


Figure 4.3: Four lines of automata arranged in a ring

Let us consider an asynchronous execution of the consensus algorithm in which all messages at the junctions are delayed until the four segments have run through α, β, γ and δ. Then the delayed messages are delivered. We wish to show that, for an appropriate choice of the initial data, the consensus space reduces to the empty set.

We showed that information carried by the delayed messages across the junction of two segments does not travel beyond the midpoint of each involved segment. Therefore the network follows the same evolution as the disjoint segments AB, BC, CD and DA and each node agrees on the same value as in the disjoint segments scenario.

We now call k_A the set of initial values of nodes in A, k_B the set of initial values of nodes in B and so on. Let us choose initial values so that $k_{Ai} < k_{Bj} < k_{Cl} < k_{Dm}\, \forall i, j, l, m$, as shown in fig. 4.4. Now, $\bar{k}$ must lie in

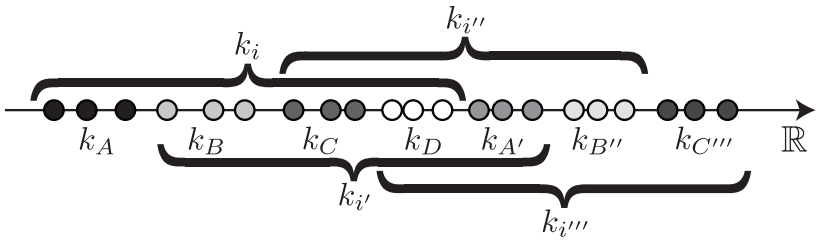


Figure 4.4: Distribution of initial values

$\text{Hull}(k_i)$. For the aforementioned initial values, it must lie in at most two among $\text{Hull}(k_A, k_B)$, $\text{Hull}(k_B, k_C)$ and $\text{Hull}(k_C, k_D)$, since the intersection of all three is the empty set.

Let us assume that $\bar{k}$ lies in $\text{Hull}(k_A, k_B) \cap \text{Hull}(k_B, k_C)$. Now, let us choose a new set of initial values $k_{A'}$ for nodes in A such that $k_{A'i} \geq K_{Dj}\, \forall i, j$. Nodes belonging to C are unaware of the initial values of nodes in A: they therefore choose the same $\bar{k}$. All other nodes have to agree. Yet $\bar{k}$ can not belong to the part of $\text{Hull}(k_A, k_B)$ that is not a part of $\text{Hull}(k_B, k_C)$: therefore $\bar{k} \in \text{Hull}(k_B, k_C)$.

Let us now substitute k_B with $k_{B''}$ such that $k_{B''i} \geq K_{A'j}\, \forall i, j$. Nodes in D are unaware of this change: they will pick the same $\bar{k}$ as before. Yet $\text{Hull}(k_B, k_C) \setminus \text{Hull}(k_C)$ does not belong to the hull of the new values k_i'' anymore: therefore $\bar{k} \in \text{Hull}(k_C)$.

Finally, let us substitute k_C with $k_{C'''}$ such that $k_{C'''i} \geq k_{B''j} \forall i, j$. Nodes in A are unaware of this substitution and pick $\bar{k}$ again. Yet $\text{Hull}(k_C)$ does not belong to $\text{Hull}(\ k_{i'''})$ anymore: the consensus algorithm fails and a contradiction is obtained.

If the initial $\bar{k}$ belongs to $\text{Hull}(k_B, k_C) \cap \text{Hull}(k_C, k_D)$, the same proof

holds substituting progressively smaller initial values in D, C and B respectively. □

The next lemma present a lower bound for the growth order of communication complexity.

Lemma 4.2.4 (Lower bound for communication complexity)**.** *Let $\mathcal{G}$ be the set of all graphs with node set V, then $CC(P, \mathcal{G}) \in \Omega(n \log n)$.*

Proof. Consider an execution of the algorithm on a ring in which all messages passing through a given communication channel are delayed until the rest of the network becomes silent. Lemma 4.2.3 shows that at least $n/4 \log(n)$ messages are required to reach consensus on this network. □

We are now in a position to characterize the growth order for communication complexity.

Lemma 4.2.5 (Lower bound on the order of growth of communication complexity)**.** *Let $\mathcal{G}$ be the set of all graphs with node set V. The communication complexity of the convex consensus problem is $\Omega(n \log n + |E|)$, i.e., $CC(P, \mathcal{G}) \in \Omega(n \log n + |E|)$.*

Proof. Lemma 4.2.1 shows that $CC(P, (G)) = \Omega(|E|)$; Lemma 4.2.4 shows that $CC(P, (G)) = \Omega(n \log n)$. The claim follows. □

Theorem 4.2.6 (Order of growth for communication complexity)**.** *Let $\mathcal{G}$ be the set of all graphs with node set V. The communication complexity of the convex consensus problem is $\Theta(n \log n + |E|)$, i.e., $CC(P, \mathcal{G}) \in \Theta(n \log n + |E|)$.*

Proof. We show that there exists an algorithm $a \in \mathcal{A}$ such that

$$\sup_{G \in \mathcal{G}} \sup_{k \in \mathcal{K}} \sup_{\alpha \in \mathcal{F}(a,k)} M(a, k, \alpha, G) = \mathrm{CC}(P, \mathcal{G})$$

The GHS algorithm achieves a communication complexity of $O(n \log n + |E|)$ [40].Therefore the GHS algorithm achieves the lower bound on communication complexity. □

Time complexity The time complexity of the GHS algorithm is $O(n \log n)$ [40]. An improved version of the GHS algorithm, proposed by Awerbuch, achieves $O(n)$ time complexity [7]. Once a rooted tree has been established, the root can contact all nodes in $O(n)$ rounds using the tree, receive their replies in $O(n)$ more rounds, compute the consensus function and inform all nodes in further $O(n)$ rounds.

Byte complexity The GHS algorithm exchanges messages of size $O(\log n)$: messages contain at most a constant number of UIDs. The byte complexity of GHS is therefore $O[(n \log n + |E|) \log n]$. If the consensus function is hierarchically computable, convergecasting information to the leader requires n messages with payload size b, where b is the number of bytes required to store one agent's opinion.

One-time complexity and recurring complexity The GHS algorithm requires $O(n \log n + |E|)$ messages to *build* a rooted minimum spanning tree on the network. Once a tree has been established, further rounds of consensus can be completed with just $2n$ messages each.

Resilience The GHS algorithm exploits a tree to convergecast information to a leader. Trees are inherently fragile: a *single* failure of a branch of the tree is enough to disrupt the algorithm, even on a fully connected network, and a spanning tree has $n - 1$ single points of failure. Strategies can be devised to reconnect the tree without rebuilding it from scratch: we discuss one such strategy in Chapter 5. The time to rebuild a tree after a failure, however, may be significant: the strategy we propose requires up to n time steps before normal operations can resume.

4.3 A lower bound on byte complexity

In Section 4.2 we show that the communication complexity of the *convex consensus* problem is $\mathrm{CC}(P, \mathcal{G}) \in \Theta(n \log n + |E|)$. It follows that the byte complexity of convex consensus has at least the same order of growth, corresponding to *constant* message size.

$$\mathrm{BC}(P, \mathcal{G}) \in \Omega(n \log n + |E|)$$

In wireless communication protocols it is natural to append *unique sender and receiver identifier* to all messages: this ensures that a message is not misdelivered or misidentified in case multiple agents are physically aligned. The unique receiver identifier may be the receiver's UID or a *local* identifier; in both cases, it must lie in the $[1, n]$ range, since agents may have up to $n - 1$ neighbors. Storing a value in $[1, n]$ requires at least $\log n$ bits.

As a result, if the communication protocol requires specifying the sender or the receiver's ID, the size of any message is $\Omega(\log n)$. The following Lemma follows:

Lemma 4.3.1 (Lower bound on the order of growth of byte complexity)**.** *Let $\mathcal{G}$ be the set of all graphs with node set V. If the communication protocol requires that all messages carry a sender or a receiver ID, the byte complexity of the convex consensus problem is $\Omega[(n \log n + |E|) \log n]$, i.e., $BC(P, \mathcal{G}) \in \Omega[(n \log n + |E|) \log n]$.*

Proof. By Lemma 4.2.5, the *communication* complexity of any convex consensus algorithm is $O(n \log n + |E|)$. Furthermore, $\log n$ bits are required to store a unique ID in the $[1, n]$ range. The claim follows. □

Next we show that the bound presented in Lemma 4.3.1 is tight.

Theorem 4.3.2 (Order of growth for byte complexity)**.** *Let $\mathcal{G}$ be the set of all graphs with node set V. If the communication protocol requires that all messages carry a receiver ID, the byte complexity of the convex consensus problem is $\Theta[(n \log n + |E|) \log n]$, i.e., $BC(P, \mathcal{G}) \in \Theta[(n \log n + |E|) \log n]$.*

Proof. We show that there exists an algorithm $a \in \mathcal{A}$ such that

$$\sup_{G \in \mathcal{G}} \sup_{k \in \mathcal{K}} \sup_{\alpha \in \mathcal{F}(a,k)} B(a, k, \alpha, G) = \mathrm{CC}(P, \mathcal{G})$$

The GHS algorithm achieves a byte complexity of $O[(n \log n + |E|) \log n]$ (sec. 4.2).Therefore the GHS algorithm reaches the lower bound on byte complexity. □

Table 4.1: Time, message and byte complexity of a time-optimal and a message-byte-optimal algorithm

	Time	Message	Byte
Flooding	$O(\mathrm{Diam}(G))$	$O(\mathrm{Diam}(G)\|E\|)$	$O(\|E\|nb)$
GHS	$O(n \log n)$	$O(n \log n + \|E\|)$	$O[(n \log n + \|E\|) \log n]$

4.4 Conclusion

In this chapter, we prove three lower bounds on the time, message and byte complexity of convex consensus on multi-agent networks. The lower bound on time is unconditional; the lower bound on the number of messages requires the consensus function to be *convex* (as discussed in Chapter 3) and the bound on byte complexity holds under the further assumption that agents attach their or their destination's unique ID to messages.

These bounds are tight: we show that a simple *flooding* algorithm achieves optimal time complexity, while the well-known GHS MST construction algorithm is message and byte-optimal.
The time-optimal flooding algorithm is also resilient to failures: on the other hand, its message and byte complexity are extremely high.
The message-optimal GHS algorithm has comparatively good time complexity. Its resilience, on the other hand, is questionable: it sports $n - 1$ single points of failure, even on a fully connected network.

These results are dissatisfying: typical space exploration applications often require a combination of low energy complexity and high resilience, possibly coupled with requirements about the convergence time if a time-varying quantity is being observed. Neither flooding nor GHS are therefore generally suitable to reach agreement on networks of robotic rovers, hoppers or penetrators.

It is natural to wonder whether it is possible to find a compromise between these conflicting metrics, trading time complexity for message complexity and message complexity for resilience. The next chapter introduces an algorithm that achieves these goals.

Chapter 5

An hybrid algorithm for distributed consensus in presence of sporadic failures

In Chapter 4, we discussed fundamental limitations of distributed consensus on cyber-physical networks, introduced a *message-optimal* and a *time-optimal* algorithm and explored the role of link failures. In Chapter 3, we presented the importance of *hybrid* metrics in space exploration applications.

Here we introduce a tunable algorithm that seamlessly moves from time-optimal to message-optimal behavior according to a user-defined parameter. The algorithm allows to trade time complexity for energy complexity and energy complexity for resilience; its worst-case performance is in no case worse than the byte complexity of flooding and the time complexity of MST; its average-case complexity on random geometric graphs, explored through numerical simulation in the next chapter, is significantly better than either.

In Sections 5.1 and 5.2 we discuss the inspiration for the algorithm and sketch its high-level structure; in Section 5.3 we explore the details of each phase and prove its correctness, while we prove performance properties in Section 5.4. Section 5.5 gives a physical interpretation for the algorithm's structure and discusses ways of choosing the tuning parameter; finally, in Section 5.6 we explore analytical performance of the algorithm on select network topologies.

5.1 Inspiration

The structure of the algorithm is inspired by the *Gamma* synchronizer proposed by B. Awerbuch [8]. The purpose of the *Gamma* synchronizer is to

establish logical time in an asynchronous network: to this end, each node delays a transition until it is sure that all other nodes have caught up. More formally, each node outputs an acknowledgement (*ok*) after performing a transition and waits for an authorization (*go*) before proceeding any further. This behavior can be implemented in many ways, e.g. by explicitly contacting all neighbors or by exploiting an existing hierarchical tree structure.

Awerbuch's *Gamma* algorithm assumes the existence of a forest of m trees (also called clusters). Once a node in a tree has performed a transition and is sure that all children have done the same, it sends *ok* to its father. When a root learns that descendants have performed the transition, it sends *ok* to neighbor trees: to this end, it asks all nodes in the tree to forward the *ok* message to neighbors outside the tree.

When a node receives a message from another tree, it forwards it up the tree until it reaches the root. Once the root is sure that all neighbor clusters are *ok*, it sends *go* to nodes in the tree, who proceed to the next transition.

We refer the reader to Awerbuch [8] and Lynch [63] for details.

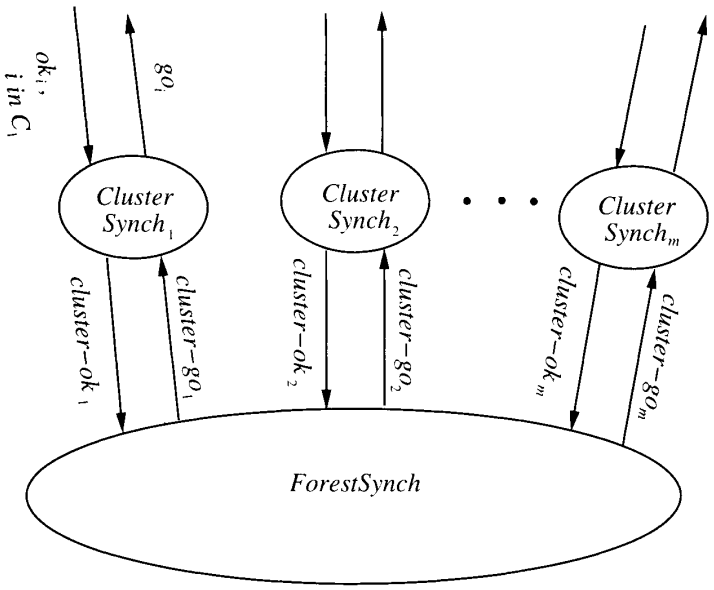


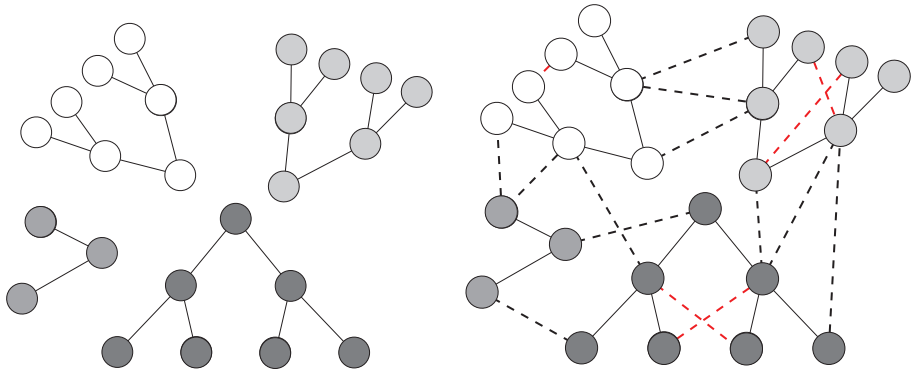
Figure 5.1: Gamma *synchronizer implementation with I-O automata (from [63, fig. 16.6])*

5.2 The high-level structure

Our algorithm has a similar structure.

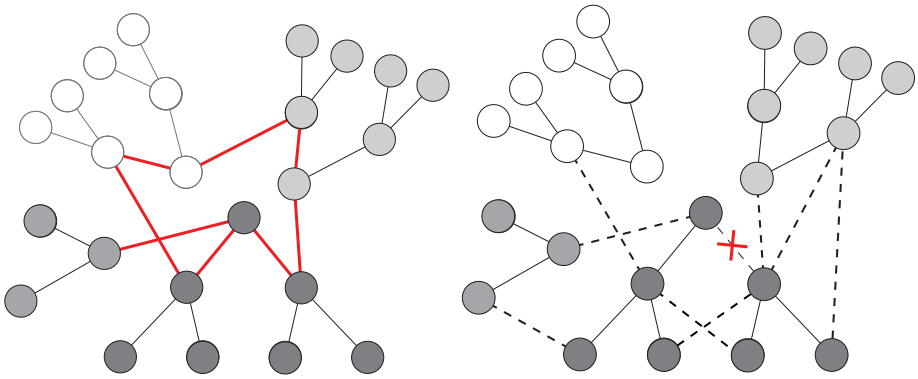
First, it builds a forest of *minimum weight* trees (shown in fig. 5.2a) of height $O(n/m)$. The value of m is the algorithm's tuning parameter.
Once a tree is in place, the root collects information from its descendants.
Each tree then establishes a certain number of connections with neighbor clusters, as shown in fig. 5.2b. Clusters *flood* information across these connections until they have heard from all others, as shown in fig. 5.2c.

If a link failure breaks one of the trees (as in fig. 5.2d), the two halves evaluate their size and, if it is below a certain threshold, they join another cluster. They then inform neighbor clusters, who update their inter-cluster routing tables.



(a) Phase 1 and 2: forest building

(b) Phase 3: establishment of inter-cluster links



(c) Phase 4: inter-cluster flooding

(d) Phase F: recovery from failure

Figure 5.2: Schematic representation of the algorithm behavior

5.3 The details

5.3.1 Phase 1: tree building

In Phase 1, all nodes run a modified version of the GHS algorithm [40].

The GHS algorithm builds a minimum spanning tree in stages: it grows a forest of minimum weight trees by incrementally merging clusters until they span the entire network. At each stage, every node in a cluster identifies its minimum weight outgoing edge and relays this information to the root: the root identifies the *cluster*'s minimum weight outgoing edge and adds it to the tree structure, rejoining the cluster across it. The algorithm terminates when the root is unable to identify a minimum weight outgoing edge because all nodes belong to the same cluster: it then informs all descendants, who stop.

We modify GHS's stopping criterion. At each stage, the root keeps track of the number of nodes in its cluster: when the cluster size exceeds $\lfloor n/m \rfloor$, the root stops the tree-building phase and informs its descendants.

At this point, other smaller groups may try and join the cluster: they are allowed to do so immediately, at which point they inherit the cluster's identity and they are notified that the tree-building phase is complete.

When a node discovers that Phase 1 is over, it contacts all neighbors, excluding its father and children, to inquire whether they are done. When all have replied, it switches to Phase 2.

Correctness

Lemma 5.3.1 (Minimum weight tree structure)**.** *At the end of Phase 1, each cluster is a tree and only contains edges belonging to the graph's minimum spanning tree.*

Proof. The claims follows from correctness of Awerbuch's algorithm, which eventually produces a minimum spanning tree, and from the observation that the algorithm never removes edges. Therefore a) at no point in time does any of the cluster contain a cycle and b) at any given time, only edges belonging to the graph's MST are present in any of the clusters. □

Lemma 5.3.2 (Cluster size)**.** *At the end of Phase 1, all clusters contain at least* $\lfloor n/m \rfloor$ *nodes.*

Proof. The algorithm only terminates if either the size of the cluster is larger than $\lfloor n/m \rfloor$ or there are no outgoing edges left. In the latter case, the cluster includes all nodes: its size is n. □

Lemma 5.3.3 (Participation)**.** *All nodes eventually join a cluster.*

Proof. Once a node wakes up, it declares itself root of an unary tree and executes the algorithm until either the size of the cluster is greater than or equal to $\lfloor n/m \rfloor$ or there are no outgoing edges left.

Nodes wake up either spontaneously or when they receive a message from another node. At the end of Phase 1, each node contacts all of its neighbors (except its father and children, who are known to be awake). The network is connected: it follows that, as long as at one agent wakes up spontaneously, all agents are eventually contacted and therefore wake up. □

Lemma 5.3.4 (Termination of Ph. 1)**.** *All nodes are eventually informed of the end of Phase I.*

Proof. The algorithm terminates when the size of a cluster is greater than or equal to $\lfloor n/m \rfloor$. When this happens, the root of the cluster informs all its offspring.

Until then, each cluster at least doubles it size at each stage unless all nodes belong to the same cluster [7, 40]. Cluster size therefore monotonically increases until it exceeds $\lfloor n/m \rfloor$. □

Cluster size and height Trees thus obtained are guaranteed to be strictly larger than $\lfloor n/m \rfloor$. Yet this is not sufficient: we wish to bound the number of clusters and the height of each tree. Despite being a good heuristic, the stopping criterion does not offer worst-case guarantees: certain network topologies and weight distributions can give rise to a single tree spanning the whole network.

Example 5.3.5 (Phase 1 may give rise to a single spanning tree)**.** *Let us consider a* fully connected *network $G(V, E)$. The weight w_{ij} of the edge connecting nodes i and j is*

$$w_{ij} = \begin{cases} 1 & \text{if } j = i + 1 \\ 1 + \varepsilon & \text{otherwise} \end{cases}$$

with $\varepsilon > 0$. It is trivial to see that an edge belongs to the MST of the graph if and only if its weight is one; the resulting spanning tree has diameter $n - 1$, whereas the original network has diameter one.

5.3.2 Phase 2

Overview In Phase 2, we bound the height of each tree by splitting overgrown clusters.

This phase of the algorithm starts at the leaves of each tree. From there, it counts the number of descendants of each agent moving up towards the root; when an agent discovers that it has more than $\lfloor n/m \rfloor$ offspring, it declares itself a root and cuts the connection with its father. This may leave the last tree, containing the original root, with too few nodes: the root can therefore undo one cut to guarantee that all clusters contain at least a minimum number of nodes.

Detailed structure Before executing Phase 2 of the algorithm, each node waits to be sure that all neighbors (excluding his father and children) are in Phase 2.

Leaves (childless nodes) then send a message to their fathers. The algorithm proceeds recursively from here: once a node has heard from all of its children and made sure that all neighbors are in Phase 2, it computes the number of its offspring by adding its children's offspring to the number of its own children. It then sends this information to its father.

If a node learns that it has more than $\lfloor n/m \rfloor$ offspring, it cuts the connection with its father after letting him know and tentatively declares itself a root (but waits to notify its offspring). The estranged father makes a local note of this. Information about the cut which removed the least number of children (i.e. the new root UID, the estranged father's UID and the number of removed children) is relayed towards the root during the counting process.

The procedure eventually reaches the cluster's original root. If the number of remaining offspring is higher than a lower bound $\underline{t} < n/m$, $\underline{t} = \Theta(n/m)$, the root switches to Phase 3 and informs its offspring. These, in turn, inform removed children that the cuts survive and switch to Phase 3. Removed children (now bona fide roots) do the same with their offspring. Each child records the UID of its tree's root, which is used as the cluster's identifier.

If the root is unhappy with the size of its mutilated tree, it asks its offspring to undo the cut that removed the least number of children (identified by the UIDs of father and child), then switches to Phase 3. The estranged father asks the removed child to rejoin him and switches to Phase 3; the child notifies its offspring; all other nodes behave as in the previous case.

Correctness

Lemma 5.3.6 (Cluster height)**.** *At the end of Phase 2, trees all have height lower than* $(\lfloor n/m \rfloor + \underline{t}) + 2 = \Theta(n/m)$.

Proof. It is easy to see that the procedure outlined in the previous paragraphs correctly counts the number of descendants of each node as long as cuts are

not mended. When a node's offspring exceed $\lfloor n/m \rfloor$, the node cuts the connection with its father: all children of a node therefore have fewer than $\lfloor n/m \rfloor$ offspring. The height of a tree is upper-bounded by the number of nodes in the tree: the height of any tree before cuts are mended is therefore lower than $\lfloor n/m \rfloor + 1$.

A cut can only be mended if the root agent determines that it has fewer than $\underline{t}$ offspring. The mending procedure joins a tree counting fewer than $\underline{t} + 1$ nodes to a tree of height lower than $\lfloor n/m \rfloor + 1$: the resulting tree is no taller than $\underline{t} + \lfloor n/m \rfloor + 2$ □

Lemma 5.3.7 (Number of clusters)**.** *At the end of Phase 2, there are at most* $n/\underline{t} = O(m)$ *clusters.*

Proof. The splitting procedure guarantees that all trees but one per original cluster are larger than $\lfloor n/m \rfloor$. If the remaining tree is smaller than $\underline{t}$, it rejoins another cluster, resulting in a tree larger than $n/m > \underline{t}$. Therefore no tree can be smaller than $\underline{t}$. It follows that, at the end of Phase 2, there are no more than $(n/\underline{t})$ trees.
Furthermore, $\underline{t} = \Theta(n/m)$: therefore the number of trees is $O(n/\underline{t}) = O(m)$. □

Lemma 5.3.8 (Termination of Ph. 2)**.** *All nodes are eventually notified of the end of Phase 2*

Proof. In Phase 2, each node sends a message to its father, either to inform it of the number of children or to cut the connection, as soon as it has heard from all children and all neighbors are in Phase 2. All nodes eventually enter Phase 2 by Lemma 5.3.4: the algorithm therefore behaves like a convergecast [63, par. 15.3] and reaches the root.

Then each father, starting from the root, contacts all children to a) announce the Cluster ID, b) confirm that a cut survives or c) ask to undo it. All three messages cause the child to switch to Phase 3. For termination purposes, the algorithm is therefore a simple broadcast and terminates by [63, par 15.3]. □

5.3.3 Phase 3

In Phase 3, nodes explore inter-cluster connections and build internal routing tables.

Overview When a node switches to Phase 3, it contacts all neighbors except for its father and children, inquiring about their cluster ID. Upon

reception of an inquiry, a node replies as soon as it enters Phase 3 (and is therefore sure of its cluster ID).

Information is then covergecast on the tree, starting from the leaves: each node informs the father of which clusters it is connected to (either directly or through its children) and how many connections per cluster are available. When the root receives the information, it switches to Phase 4.

Detailed structure Each node waits until it has heard from all children (if any) and all neighbors before informing its father.

Nodes maintain two local routing tables: one (the *neighbor* routing table), relates non-tree neighbors and their cluster, whereas the other (the *children* routing table) records which clusters each child is connected to (directly or indirectly) and how many connections are available per cluster. When informing its father, a node only denotes which clusters it is connected to and how many connections are available, making no distinction between direct and children-mediated connections.

Correctness

Lemma 5.3.9 (Termination of Ph. 3)**.** *Each node is eventually informed of the correct number of neighbor clusters connected either to it or to its offspring;*

Proof. Consider an execution α of the algorithm where a) no node contacts its neighbors until all nodes are in Phase 3 and b) the convergecast of routing information does not start until *all* nodes have learned their direct neighbors' ClusterIDs.

It is easy to see that any execution of Phase 3 of the algorithm is *similar* to execution α: two rounds after the last node enters Phase 3, the *neighbor* routing table of each node is identical to α's in any execution; moreover, $r+2$ rounds after the last node enters Phase 3, the *children* routing table of any node closer than r to the farthest leave among their offspring is identical to α's in any execution.

The correctness of execution α is easy to verify: discovery of neighbors' Cluster IDs is trivial if all nodes are in Phase 3 and correctness of children routing tables follows from correctness of the convergecast algorithm [63, par. 15.3]. □

5.3.4 Phase 4

In Phase 4, cluster roots communicate with each other through the connections discovered in the previous stage.

Overview Conceptually, this phase of the algorithm is simply flooding across clusters. Each root sends a message containing its cluster's *information* to each of its neighbor trees through the connections built in Phase 3. Each message is replicated a few times as a protection against link failures. When a root learns new information, it forwards it *once* to its neighbor clusters (sender excluded) via the same mechanism.

Detailed structure The root of each tree generates a message for each of its neighbor clusters with its cluster's piece of information. Each message is replicated k times, where k is a user-defined parameter. The root then sends as many copies of each message as is possible through its direct connections, stored in its local *neighbor* routing table. Unsent copies of the message are distributed to children proportionally to the number of links available, stored in the *children* routing table.

Children do the same: when required to forward a message to a cluster, they send as many copies as possible through their direct connections and divide the rest among their own children according to the number of links available.

When a node receives a message for its cluster, it checks whether it has already received this information, either from a non-cluster neighbor or from a child. If this is not the case, it just forwards the message up the tree; otherwise, it discards the information. The first time the root hears new information, it broadcasts it to neighbor clusters via the same mechanism as above, generating and forwarding k copies of a new message with the new information and the origin cluster's ID.

Roots include the number of their children in their cluster's information. When a root has heard from $n - \underline{t} + 1$ nodes, it terminates: after forwarding new information one last time, it computes the consensus value and informs it offspring.

Correctness

Lemma 5.3.10 (Diffusion of information)**.** *In absence of failures, all clusters eventually hear from each other*

Proof. Let us abstract Phase 4 by building a network G_c containing one node for each of the existing clusters in G. Nodes in G_c are connected if the corresponding clusters are "neighbors", as discovered in Phase 3.

Now, thanks to the correctness of the routing tables (Lemma 5.3.9) Phase 4 of the algorithm reduces to flooding on G_c, whose correctness follows from [63, par. 4.1].

□

Lemma 5.3.11 (Termination of Phase 4)**.** *Phase 4 of the algorithm eventually terminates.*

Proof. By Lemma 5.3.10, the roots of all cluster eventually hear from each other. By Lemma 5.3.7, no tree is smaller than $\underline{t}$. Messages from one cluster to another carry the number of nodes in the cluster at the time of dispatch: It follows that, once a tree has heard from $n - \underline{t} + 1$ nodes, it must have heard from all clusters. □

5.3.5 Phase F (recovery from in-tree failure)

Overview A failure within one of the clusters formed in Phase 1 and 2 splits the cluster in two. If either of the two halves is too small, it initiates a search for its minimum weight outgoing edge and rejoins the cluster across it; a splitting procedure guarantees that tree height stays bounded. After the failure, all nodes in the affected cluster contact their neighbors to update their routing tables.

Detailed structure Upon being notified of a severed connection with a child, a node notifies its root. Conversely, upon being notified of a severed connection with its father, a node declares itself a root. If either root has fewer than $\underline{t}$ offspring, it sends them a *unique* cut identifier and initiates a search for the cluster's minimum weight outgoing edge. If the number of offspring is high enough, the root just sends offspring the cut identifier, which includes the old cluster ID, the IDs of the two nodes immediately upstream and downstream of the cut and a *local* timestamp[1].

The presence of a failure complicates the search for a minimum weight outgoing edge: nodes downstream of the cut, who receive a new Cluster ID, may mistakenly accept a connection with a node in the same cluster if they do not know about the cut yet. To avoid this, nodes disclose the unique cut identifier when looking for the minimum weight outgoing edge: if a node sports the old Cluster ID but does not hold the cut identifier, it delays the reply until it is informed of the cut by its father.

Once a small cluster finds its minimum weight outgoing edge, it rejoins the cluster on the other side. A splitting procedure, akin to the one outlined in Phase 2, is then initiated to maintain tree height bounded. The procedure is initiated by the node rejoining the cluster and proceeds up to the root: nodes outside this path see no change in the number of their offspring and are therefore unaffected.

[1] No global time is assumed to exist. On the other hand, neighbors are assumed to share a local partial ordering of time, e.g. based on the number of messages they exchanged.

As nodes learn their final Cluster ID, either at the end of the splitting procedure or because the post-failure cluster is large enough, they inform all non-cluster neighbors. Neighbors, in turn, update their routing tables and inform their fathers, as in Phase 3. When an unaffected node is contacted by a non-cluster neighbor, it does not immediately notify its father, since this may lead to many expensive incremental updates: the node waits to hear from all offspring who were connected to the affected cluster (recorded in the children routing table) before sending an update. This way, routing tables are updated in a convergecast.

When a root learns about a variation in the topology of neighbor clusters (either because a new cluster is formed or because the number of connections to an existing cluster decreases) it crafts a message with all information it holds and sends it to the updated clusters as in Phase 4. The roots of clusters born or modified because of the cut collect information from their children and craft messages for all their neighbors, too.

Correctness

Lemma 5.3.12 (Cluster height)**.** *At the end of Phase F, trees all have height lower than* $(\lfloor n/m \rfloor + \underline{t} + 2) = \Theta(n/m)$.

Proof. The proof is identical to Lemma 5.3.6's and follows from correctness of the splitting procedure. □

Lemma 5.3.13 (Number of clusters)**.** *At the end of Phase F, there are at most* $n/\underline{t} = O(m)$ *clusters.*

Proof. The proof is identical to Lemma 5.3.7 and follows from the lower bound imposed by the splitting procedure on the size of each tree. □

Lemma 5.3.14. *All nodes in the tree affected by the cut eventually learn about the cut*

Proof. Nodes below the failure are informed of the cut via a simple broadcast. The node above the cut informs its father, who does the same until the message reaches the root. The root then proceeds with a broadcast. The lemma therefore follows from the correctness of broadcast. □

Lemma 5.3.15 (Inter-cluster connections)**.** *Each node is eventually informed of the correct number of neighbor clusters connected to it either directly or through its offspring.*

Proof. Before a failure occurs, routing tables are correct by Lemma 5.3.9. When a failure occurs, nodes in the affected cluster are informed by Lemma 5.3.14. Once informed, nodes in the affected cluster contact all their neighbors at once. These, in turn, update their routing tables with a convergecast. Nodes formerly belonging to the broken cluster rebuild their routing tables ex novo: the correctness of the procedure follows from Lemma 5.3.9. □

5.3.6 Phase OF (recovery from out-of-tree failure)

When a link outside a tree fails, nodes on the two sides of the failure update their routing tables and notify their fathers, who do the same until the information reaches the root.

Note that up to $k-1$ simultaneous, adversarial failures can occur while the algorithm updates its routing tables without disrupting cluster flooding: routing within the trees distributes messages proportionally to the number of available links so that, even if a node is unable to forward a message because it is no more connected to a cluster, other nodes in different branches will be able to do so.

Correctness

Lemma 5.3.16 (Termination of Ph. OF)**.** *At the end of Phase OF, all nodes' routing tables are correct.*

Proof. The proof is identical to that of Lemma 5.3.9. □

Lemma 5.3.17 (Resilience to inter-cluster failures)**.** *Phase 4 of the algorithm correctly terminates even in presence of* $k-1$ *simultaneous adversarial failures.*

Proof. Phase 4's routing strategy ensures that, if two clusters are connected by at least k edges, any message among the two clusters will be sent across k *distinct* edges. Up to $k-1$ failures of inter-cluster links can therefore occur without invalidating the similarity between Phase 4 and flooding outlined in Lemma 5.3.10. □

5.4 Complexity analysis

The overall time and message complexity of the algorithm are reported in table 5.1.

Phase 1

Time complexity The GHS algorithm proceeds in stages, each requiring at most $O(n)$ rounds. At each phase, the size of the smaller cluster at least doubles. Phase 1 terminates when every cluster is larger than n/m: its time complexity is therefore upper-bounded by $(n \log n/m)$.

Message complexity and size At each stage, communications within the clusters require $O(n)$ messages. Furthermore, $2n$ test-accept messages are sent at each stage: each node accepts exactly one connection. Each edge is also rejected once during the algorithm. The overall message complexity of the algorithm is therefore $O(n \log n/m + |E|)$. Messages carry one cluster ID at most: their size is upper-bounded by $\log n$.

Phase 2

Message complexity and size Each non root node sends exactly one message to its father, either to notify it of the number of its children or to sever the connection. It receives exactly one message to notify that Phase 2 is over, authorize a cut or revert it. The message complexity is therefore upper bounded by $2n$.

Each upstream message contains the number of offspring of a node. Downstream messages contain a cluster ID (typically the root node's UID) and may also identify a cut to be restored by carrying two extra UIDs. Both the number of offspring and the nodes' UIDs can be represented in any reasonable encoding format with $\log n$ bytes. The message size in Phase 2 is therefore upper-bounded by $3 \log n$.

Time complexity The algorithm proceeds from the leaves of the trees formed in Phase 1 to their roots and vice versa, akin to a convergecast followed by a broadcast. The time complexity is therefore upper-bounded by twice the height of the trees formed in Phase 1, which is itself upper bounded by $2n$.

Phase 3

Message complexity and size Each non-tree edge is crossed by two messages: an inquiry about the cluster ID and a reply. Each edge belonging to a tree is charged with one convergecast messages. The overall message complexity is therefore upper bounded by $2(|E| - n)$ (inter-cluster) $+ n$ (intra-cluster).

Inquiries on non-tree edges have constant size and replies, which carry a cluster's ID, have size $\log n$. Messages relayed over the tree carry the number of connections with each neighbor cluster: their size is therefore upper bounded by $m \log n$. The byte complexity of Phase 3 is $O(2(|E| - n) \log n + nm \log n)$.

Time complexity Once all nodes are in Phase 3, nodes discover their neighbors' clusters in two rounds at most: in the first round inquiries are sent on all non root-channels, in the second round replies are collected. The subsequent convergecast requires as many steps as the height of the tree, which is upper bounded by $\lfloor n/m \rfloor + \underline{t}$. The overall time complexity is therefore $O(\lfloor n/m \rfloor + \underline{t} + 2) = O(n/m)$.

Phase 4

Time complexity Let us abstract Phase 4 by building an artificial network G_c composed of $O(m)$ nodes, each corresponding to one of the existing clusters in G and labeled accordingly. Nodes in G_c are connected if at least one edge exists between the corresponding clusters in G. Let us also define a *stage* time complexity as the time required for information to travel from the root of one cluster to the root of its neighbor.

Phase 4 is simply flooding on G_c: the algorithm is therefore guaranteed to terminate in $\text{Diam}(G_c)$ stages. Note that $\text{Diam}(G_c) = O(\text{Diam}(G))$ and $\text{Diam}(G_c) = O(m)$

Now, the time complexity of one stage is upper bounded by $2(\lfloor n/m \rfloor + \underline{t}) + 1$: in each stage, information travels away from the root across the cluster, then hops from a cluster to the next and is finally convergecast to the root.

The overall time complexity of Phase 4 is therefore $\text{Diam}(G_c)(2(\lfloor n/m \rfloor + \underline{t}) + 1) = O(\text{Diam}(G)(n/m))$.

Message complexity and size In absence of failures, each of the $O(n)$ edges belonging to a tree is crossed by information about one cluster at most twice: once when the cluster learns about the information, once when information is relayed to neighbors. The overall byte complexity of *intra-cluster* messages in Phase 4 is therefore upper-bounded by $(2mn \log n)$: information about each of the m clusters is stored in $\log n$ bits.

Each of the $k|E_c|$ inter-cluster connections is also crossed by information about each cluster once: clusters send new information once after they receive it. The associated byte complexity is $(k|E_c|m \log n)$. The overall byte complexity is therefore $O(m(n + k|E_c|) \log n)$. Note that, if nodes are

reaching consensus on a *locally computable* function of the initial states[2], the size of inter-cluster messages is reduced by a factor of m: information about several clusters can be relayed in $\log n$ bits even in absence of a hierarchical structure.

Messages can be as small as $\log n$ if they carry information about a single cluster; larger message may contain multiple cluster's opinions of the consensus value. The worst-case message complexity is upper-bounded by $(2mn + k|E_c|m)$.

Phase F (recovery from in-tree failure)

Time complexity All nodes within a tree are informed of a link failure within $2(\lfloor n/m \rfloor + \underline{t})$ rounds of the failure. The node downstream of the failure broadcasts the information to its offspring directly, whereas the upstream node informs the root who, in turn, broadcasts information to other nodes.

If a tree is found to be too small, a search for the minimum weight outgoing edge is initiated. Any node can be rejected by $O(\underline{t})$ other nodes in the same group at most; furthermore, the first reply may be delayed by as much as $(\lfloor n/m \rfloor + \underline{t})$ as nodes are informed of the cut.

The time complexity of splitting is upper-bounded by twice the height of the tree, as in Phase 2. In Phase F, the maximum height of a tree is $(\lfloor n/m \rfloor + 2\underline{t})$: before the failure, no tree can be taller than $(\lfloor n/m \rfloor + \underline{t})$ and only trees smaller than $\underline{t}$ perform a MWOE search.

Once the tree has been reformed, it updates its neighbors about its cluster ID and rebuilds internal routing tables. Neighbors update their own routing tables, too. As in Phase 3, the time complexity is upper bounded by $\lfloor n/m \rfloor + \underline{t} + 2 = O(n/m)$.

Message complexity and size The number of messages required to inform all nodes in a broken cluster of a failure is $O(n)$: one message is charged to each node in the cluster, and cluster size (as opposed to cluster height) has no nontrivial upper bound. The corresponding byte complexity is $O(n \log n)$: messages carry a unique Cut ID containing two node IDs and one cluster ID.

If a MWOE search is initiated, each of the $O(\underline{t})$ nodes is rejected by at most $\underline{t} - 2$ siblings and accepted by one neighbor: $O(\underline{t}^2)$ messages are exchanged. The subsequent convergecast requires $O(\underline{t})$ messages.

The splitting procedure requires up to $2n$ messages, i.e. twice the size of a cluster.

[2]Max and min are examples of locally computable function; for a rigorous definition, we refer the reader to Section 3.1.1.

Finally, exploring connections with neighbor clusters can require up to $2|E|$ messages (which dominates the message complexity of Phase F) and updating routing tables in the broken node and its neighbors requires up to n messages with a convergecast.

Messages informing nodes of a failure and exploring neighbor clusters carry a cluster ID and a unique cut identifier, which includes two node IDs, a cluster ID and a timestamp. Nodes unaffected by the failure must update their routing tables by adding or removing information about *three* clusters at most: the original cluster may disappear and its two halves may join two existing trees. The size of all these messages is therefore $O(\log n)$.

On the other hand, clusters containing nodes affected by the failure must update their routing tables thoroughly: connections to many clusters may have been lost in the cut and, if nodes join an existing tree, their ancestors must be notified of newly available connections. Up to n messages of size $m \log n$ may therefore be sent.

If nodes are performing multiple consensus rounds or tracking a time-varying quantity, no further messages are required: once the routing tables have been restored, the newly formed clusters just wait until the next round of consensus. If, on the other hand, consensus on a single, static value is to be performed, neighbor clusters have to update new or mutilated clusters, who may have lost messages because of the failure: the corresponding message complexity is the same as one stage of Phase 4 of the algorithm.

Phase OF (recovery from out-of-tree failure)

Time complexity When an inter-cluster link failure occurs, nodes on both sides of the failure update their routing table and inform their fathers, who do the same until the information reaches the root. The associated time complexity is upper-bounded by the height of a tree, $\lfloor n/m \rfloor + \underline{t} = O(n/m)$. Note that cluster flooding (Phase 4) does not stop while Phase OF is executed unless more than $k-1$ failures occur while routing tables are being updated.

Message complexity and size Each node along the path between the nodes next to the failure and their roots send exactly one message to its father. The overall message complexity is therefore upper bounded by $2(\lfloor n/m \rfloor + \underline{t}) = O(2n/m)$. Each message carries updated information about one cluster: message size is therefore $O(\log n)$.

Table 5.1: Time, message and byte complexity of our hybrid algorithm

	Time	Message	Byte
Phase 1	$O(n \log(n/m))$	$O(\|E\|)$	$O(\|E\| \log n)$
Phase 2	$O(2n)$	$O(2n)$	$O(2n \log n)$
Phase 3	$O(n/m)$	$O(2\|E\|)$	$O(2\|E\| \log n + nm \log n)$
Phase 4	$O(\mathrm{Diam}(G_c)n/m)$	$O(2mn + k\|E_c\|m)$	$O(m(n + k\|E_c\|) \log n)$
Phase F	$O(n/m)$	$O(\|E\|)$	$O(\|E\| \log n + nm \log n)$
Phase OF	$O(n/m)$	$O(2n/m)$	$O(2n/m \log n)$

5.5 Physical insight and tuning parameters

5.5.1 Node clustering and selective redundancy

In our applications, link weight is typically proportional to the cube of the distance between nodes to represent the transmission power required for error-free communication. The algorithm builds a forest of *minimum weight* trees: qualitatively, nodes are clustered according to their proximity[3].

Clusters then build redundant long-range links with neighbor trees. If a link was not included in a tree in Phase 1, it weighs more than any link within the tree: Phase 1 iteratively expands a cluster by annexing the *minimum weight* outgoing edge. Most inter-cluster links created in Phase 3 therefore have a longer range than intra-cluster links: the sole exception are links deleted in Phase 2, which used to belong to a tree structure.

The algorithm links nearby nodes with a lean, nonredundant tree structure; redundancy (up to k times) is reserved to long range, inter-cluster links. Our algorithm's approach trades energy for resilience, consistently with the hypothesis that long-range links are more susceptible to failure and more expensive to replace than short-range ones.

5.5.2 Error isolation

Dividing the network in $O(m)$ trees with robust intra-cluster connections allows to isolate in-tree failures: modulo some (unfortunately expensive) rewiring of inter-cluster connections, only nodes belonging to a tree need to be informed of a failure.

A crucual advantage of controlling tree height is the ability to quickly reconfigure the network topology following a failure. The time complexity

[3]Rigourously, this is not always true: the exact composition of the clusters depends nontrivially on the order in which nodes wake up, and Phase 2 splitting is concerned with tree *height* as opposed to tree *weight*.

of Phase F is $O(n/m)$; if a single spanning tree had been employed, up to $O(n)$ time steps would have been required to inform all nodes of a failure and safely resume a search for the minimum weight outgoing edge.

5.5.3 Reducing single points of failure

A spanning tree has $n-1$ single points of failure: if any link belonging to the tree fails, an expensive reconfiguration is required. In our algorithm, on the other hand, $n-m$ edges belong to tree structures: the number of single points of failure therefore decreases.

5.5.4 Failure frequency

Our algorithm requires $O(m/n)$ time steps to recover from a failure. An in-tree failure stops Phase 4, since the tree structure is broken, whereas up to $k-1$ inter-cluster failures can be tolerated before routing fails. This gives insight in a potential driver for the choice of the tuning parameter m: if the expected time to fail of a short-range link is $O(\tau)$ and the MTTF of a long-range link is $O(\tilde{\tau})$, it is advisable to choose m and k so that $n/m \ll \tau$ and $n/(mk) \ll \tilde{\tau}$.

5.5.5 Consensus on time-varying parameters

Our algorithm can be used to perform multiple decisions in sequence, a crucial capability when observing a time-varying quantity. Once the basic structure of the algorithm is in place, tree roots can collect information from their offspring at regular intervals[4] and issue messages to neighbor clusters.

If the clusters share a global time, inter-cluster messages can be timestamped and synchronized: roots can then compute the consensus value at a predetermined sequence of points in time.

Even in our asynchronous setting, the algorithm guarantees a certain degree of synchronization. It is easy to show that, if tree roots collect new opinions from their offspring via a broadcast-convergecast just after they have reached consensus on the previous reading, our algorithm behaves *similarly* to the *Gamma* synchronizer it takes inspiration from. The synchronizer guarantees sequencing of readings: collection of a new set of readings or opinions only begins once every agent has broadcast its previous reading or manifested its opinion.

[4] A broadcast-convergecast protocol with a time complexity of $O(n/m)$, a message complexity of $O(n-m)$ and a byte complexity of $O((n-m)\log n)$ is trivial to implement.

To guarantee reliable tracking, the consensus algorithm should be much faster than the phenomenon under observation : rigorously, if the characteristic time of the observed phenomenon is $\bar{\tau}$, we require $[\mathrm{Diam}(G_c)n/m] \ll \bar{\tau}$.

5.6 Analytical performance on select network topologies

In this section, we analytically evaluate the performance of our algorithm on select network topologies.

As stated in previous chapters, performance of any asynchronous consensus algorithm is strongly influenced by the network topology, the nodes' wakeup sequence and the distribution of edge weights. The purpose of this section is not to assess our algorithm's performance in real-world applications but rather to demonstrate its mechanisms on simple, intuitive network topologies. We therefore introduce four simplifying assumptions:

- The network evolves *synchronously*. Recall that synchronous realizations of an algorithm are admissible in an asynchronous setting.
- Nodes all wake up at the first round.
- Edge weights are assigned according to a lexicographic (dictionary) ordering: the weight of each edge is determined by the UIDs of its end nodes, lowest first.
- Nodes are numbered from 0 to n. In a star structure, node 0 is the core of the star. In a line structure, node i ($i \in [2, n-1]$) has neighbors $i-1$ and $i+1$.

5.6.1 Star

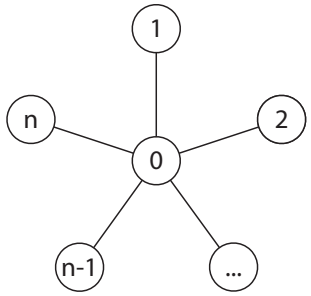


Figure 5.3: Star network topology

A star (represented in figure 5.3) is a tree structure with diameter 2. It is easy to see that, if $\underline{t} > 2$, Phase 1 and 2 of the algorithm leave all nodes in the same tree, which contains all edges in the graph:

- During Round 1, all nodes except from the core (node 0) connect to it, since they only have it as a neighbor. The core connects to node 1. n small ($O(\log n)$) messages are exchanged
- In Round 2, nodes 0 and 1 acknowledge each other. Node 1 declares itself a leader; node 0 silently acknowledges node 1 as its leader and father and informs all remaining neighbors. $n-2$ small messages are exchanged.
- In round 3, nodes 2 to n join node 1's group and enter Phase 2. They immediately start the splitting procedure.
- The splitting procedure reaches Node 1 at Round 5 after $n-1$ small messages. The tree remains as it is.
- The root realizes that all nodes are in the same cluster. It initiates Phase 3 anyway, so as to collect information from its children.
- Nodes 2 to n enter Phase 3 in Round 7 and immediately contact Node 0. Node 1 receives a function of all other agents' information through node 0 in Round 9. $2n-2$ small messages message are exchanged in rounds 6 through 9.
- Node 0 is informed that the procedure is over in Round 10. Nodes 2 to n are informed in Round 11. $n-1$ small messages are sent.

The overall time complexity is 11 rounds. The message complexity is $6n-6$ small messages.

As a comparison, a flooding algorithm requires two rounds and $2n$ small messages during the first round, followed by $n-1$ large ($O(n \log n)$) messages sent by node 0 to all neighbors during the second round.

The GHS algorithm requires $3n-3$ small messages and four rounds to establish the tree and verify termination; two rounds and (2n-2) small messages are required to relay information to the root and $n-1$ small messages are sent over two rounds to inform all nodes of the decision value. Foregoing the unneeded splitting procedure makes GHS faster than our hybrid algorithm in this specific configuration.

Any link failure is bound to disconnect the network: Phases F and OF are therefore not relevant in this case.

5.6.2 Line

A line, represented in figure 5.4, is a tree structure with diameter n. Let us assume for simplicity that n/m is an integer s.

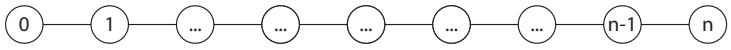


Figure 5.4: Line network topology

- In Round 1, nodes 1 to n connect to their left neighbor. Node 0 connects to node 1. n small messages are exchanged.
- In Round 2, node 1 declares itself a leader and informs node 2. Node 0 silently acknowledges node 1 as a leader. One small message is sent.
- Rounds 3 through n are devoted to informing nodes 3 to n to switch to Phase 2. $n-2$ small messages are exchanged.
- In Round n, node n finally switches to Phase 2 and start the splitting procedure, which reaches node 1 at Round $2n-2$, after $n-2$ messages.
- Node 1's acknowledgement reaches node 0 in round $2n-1$ and node n in round $3n-4$. $n-1$ small messages are sent. The line is now partitioned in m segments, each of size s, with a leader at its left end (see fig. 5.5).

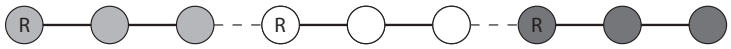


Figure 5.5: Hybrid algorithm: clusters line partition

- Each segment discovers connections to their left and right neighbor clusters in $2s+2$ time steps. Overall, $2n$ small messages are exchanged. At Round 3n+2s-2 routing tables are established and cluster roots know their children's opinions of the consensus value.
- Cluster roots prepare messages containing their cluster's opinion and send it to neighbor segments. Each root (except for the leaders of the first and last cluster) sends exactly one message directly to its left neighbor and one message to the cluster to its right through its children. Both messages reach the neighbor's root after s rounds. In the process, $2n$ small messages are exchanged.

- The flooding process goes on for $m-1$ flooding stages, each requiring s rounds. Note that $ms = n$. Each cluster learns about one or two clusters from one neighbor at each flooding stage and, unless it is an end cluster, forwards information to its other neighbor at the next stage. Overall, each of the m clusters learns about $m-1$ other clusters exactly once with an expense of s small messages each. Overall, $m(m-1)s = n(m-1)$ small ($O(\log n)$) messages are exchanged.

- Once every root has learned from all other clusters, it informs its offspring. This last step requires s rounds and $n-m$ messages.

Overall, the algorithm has a time complexity of $4n+3s-m-2$ rounds and message complexity of $n(m+6)-m+4$ small messages.

A flooding algorithm requires n rounds on the same network. Each node learns about each other node's opinion once and forwards it to a neighbor: overall, n^2 small messages are exchanged.

The GHS algorithm requires $2n$ rounds and $2n-1$ small messages to establish a tree. A further $2n$ rounds and $2n-2$ small messages are required to collect information from the leaves.

Similarly to the previous case, any link failure is bound to disconnect the network: Phases F and OF are not relevant in this case.

5.6.3 Ring

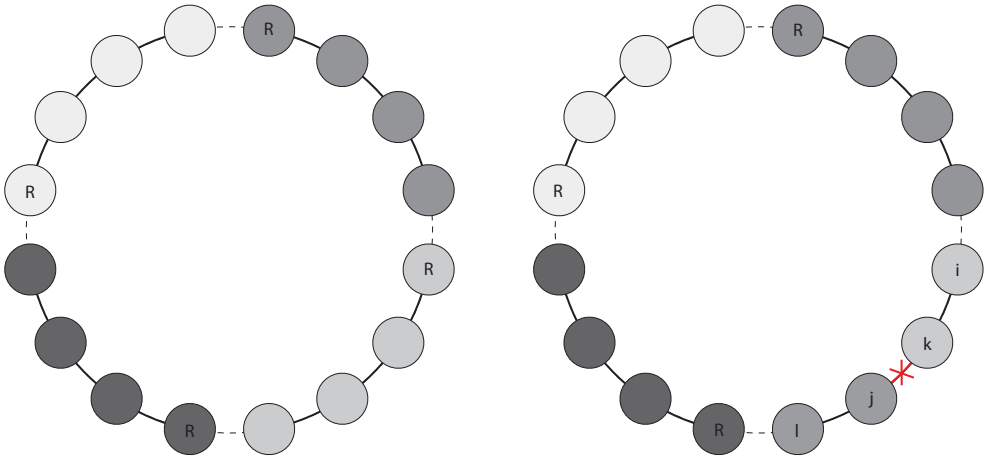
In absence of failures, our algorithm behaves almost identically on a ring, shown in fig. 5.6a, and on a line: nodes arrange in a set of segments connected at the tips exactly as in the previous example, then use flooding to communicate with each other. $\lceil m/2 \rceil$ flooding stages are required to complete flooding; each stage lasts s rounds.

Let us now assume a link failure occurs midway through a segment, as shown in figure 5.6b, at round r. Let us also assume that $s/2 < \underline{t}$.

- Node j immediately asks its offspring to look for the minimum weight outgoing edge; node k informs node i, who learns of the cut at round $r + (s/2) - 1$ after $(s/2) - 1$ small messages.

- Node l receives word of the cut at round $r + (s/2) - 1$, after $(s/2) - 1$ messages: it exchanges two messages with the cluster to its left and receives a reply at round $r+(s/2)+1$. Node j learns about the cluster's sole outgoing edge at round $r + s$ with $s/2 - 1$ messages.

- Node j asks node l to join the cluster; node l complies at round $r + 3/2s - 1$, after a chain of $s/2 - 1$ messages.



(a) Ring partition in clusters (b) Link failure within a cluster

Figure 5.6: Ring network topology

- Node l's neighbor is the root of the adjacent cluster: it immediately authorizes the merging. Node l rejoins the cluster to its left at round $r + 3/2s + 1$ after an exchange of two messages.
- Node l's sibling join the cluster to their left in the next $s/2 - 1$ rounds with $s/2 - 1$ messages. Node j is the last to join the cluster at round $r + 2s$.
- In the meantime, node i asks its offspring to look for an outgoing edge and contacts the cluster above it at round $r + s/2$. The cluster above replies at the next round. Two inter-cluster messages are exchanged.
- After $s - 2$ messages, at round $r + 3/2s - 2$ node i's offspring report that they have found no outgoing edges . Node i therefore joins the cluster above it.
- The cluster above node i starts a splitting procedure along the path between i and the cluster root at the top of the ring. After $2s$ rounds and as many messages, node i joins the cluster. Node i's offspring join the same cluster in the next $s/2 - 1$ rounds with $s/2 - 1$ messages. Node k is the last to join the cluster at round $r + 6s - 3$.
- No new clusters appear in routing tables and no new connections are established: the flooding procedure therefore continues. The leaders of the trees who harbored nodes i, j, k, l and their siblings collect information from all offspring and craft new messages to their neighbors.

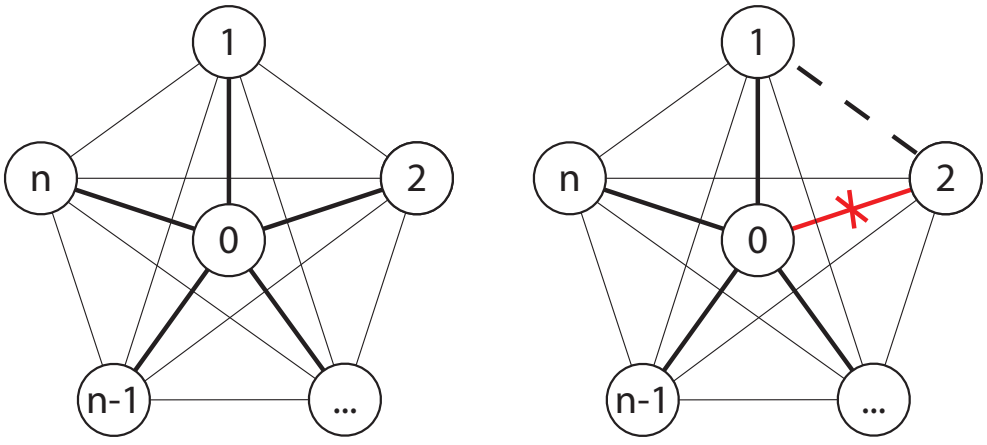
The reconfiguration procedure terminates after $6s - 3$ rounds; $11/2s + 1$ messages are exchanged.

A flooding algorithm does not require any reconfiguration after a failure. On the other hand, its message complexity on this network topology is $O(n^2)$ regardless of failures.

The GHS algorithm is not designed to reconfigure after a link failure. On the other hand, any algorithm that tries to reconnect the two halves of a tree after a failure requires $O(n)$ rounds and $O(n)$ messages just to inform all nodes of the cut and ask them to look for the new minimum weight outgoing edge. Our algorithm, on the other hand, achieves a time complexity of $O(n/m)$ and a message (and byte) complexity of $O(n/m)$ on this lightweight network topology.

5.6.4 Fully connected network

In absence of link failures and with lexicographic edge ordering, our algorithm behaves identically on a star and on a fully connected network, shown in fig. 5.7a. Note that, however, a different choice of edge weights (such as the one



(a) Tree structure after Phase 1 and 2 (b) Link failure within the tree structure

Figure 5.7: Fully connected network topology

presented in Example 5.3.5) may lead to the formation of a very tall tree in Phase 1, which would then be subdivided in compact clusters in Phase 2.

Let us walk through recovery from a failure of the link between agents 0 and 2 in a fully connected network with lexicographic edge ordering, as shown in fig. 5.7b. The failure occurs at round r.

- At round r, nodes 0 and 2 are notified of the failure. Agent 1, the cluster leader, is informed by Agent 0 at round $r + 1$ with one small message.

- Agent 2's cluster is too small, with a node count of one. Node 2 therefore starts looking for its minimum weight outgoing edge: it contacts node 1, who receives the message at round $r + 1$.

- Node 1 is now aware of the failure: it therefore promptly accepts node 2's request. Node 2 is notified at round $r + 2$, with one message.

- In parallel, node 1 informs all its offspring of the link failure. Node 0 receives the information at round $r + 2$ and nodes 3 to n are informed at round $r + 3$. Overall, $n - 2$ small messages are exchanged.

- Node 2 initiates a splitting procedure at round $r + 3$. The procedure reaches node 1 at round $r + 4$ and stops. At round $r + 5$, node 2 joins node 1's cluster for good.

5.7 Conclusion

This chapters presents an *hybrid* algorithm whose behavior smoothly moves from time-optimal flooding to message and byte-optimal GHS according to a user-defined parameter. The algorithm partitions the network in a set of hierarchically organized clusters: clusters then communicate with each other with a "flat", leaderless communication pattern.

Theoretical analysis shows that the worst-case time and byte performance of our algorithm is intermediate with respect to GHS and flooding.

Time complexity The time complexity of our algorithm is dominated by Phase 1 and Phase 4.

- Phase 1, which only needs to be executed *once*, sports time complexity lower than GHS by $O(n \log m)$.

- The time complexity of Phase 4 is worse than flooding's by a factor of (n/m). It is also upper-bounded by $O(n)$, since $\text{Diam}(G_c) = O(m)$.

- Recovery from a intra-tree failure can be achieved in $O(n/m)$ time steps. The same failure recovery protocol requires $\Omega(n)$ rounds on the spanning tree built by GHS: all nodes must be informed of a failure before new edges are added to ensure that no cycles are created.

Message complexity Our algorithm's message complexity is unspectacular: it is dominated by Phase 4's $O(2mn + k|E_c|m)$, which can be even higher than flooding's. Our algorithm is therefore unsuitable whenever message complexity, as opposed to byte complexity, is a good proxy for energy cost.

The higher message complexity, however, is compensated by significantly smaller message size: our hybrid algorithm never exchanges messages larger than $m \log n$, while flooding uses messages as large as $n \log n$. This helps our algorithm achieve *byte* complexity significantly lower than flooding's.

Byte complexity The byte complexity of our algorithm is dominated by the cost of Phase 4, with $O(m(n + k|E_c|) \log n)$ bytes exchanged.

- Flooding can require as many as $O[n|E| \log n]$ bytes: our algorithm's byte complexity is lower than flooding's by a factor of $n/(mk)$ at least.
- our algorithm's worst-case byte complexity is at least $m/\log n$ times higher than GHS, which requires $O[(n \log n + |E|) \log n]$ bytes.
- The *recurring* byte cost of consensus on GHS, once a tree has been established, is $O(2n \log n)$: our algorithm, on the other hand, requires $O(m(n + k|E_c|) \log n)$ bytes for *each* agreement, over m times more than consensus, even after a structure has been established.

Our hybrid algorithms solves the second problem presented in Chapter 3: it can be tuned to emulate flooding as well as GHS and it predictably scales from the former's behavior to the latter, degrading time performance and robustness (as measured both by the time required to recover from a failure and by the number of single points of failure) to reduce byte complexity and vice versa.

The algorithm is ideal to satisfy *hybrid* metrics relevant to space exploration applications, where optimal *nominal* energy consumption typically doesn't justify low robustness and bad time performance while, conversely, extremely high energy consumption can not be accommodated even when it affords an high degree of robustness and near-instantaneous recovery from failures.

In this chapter, we analyzed the *worst-case* complexity of our hybrid algorithm; it is now natural to wonder whether its advantages with respect to GHS and flooding also hold true in real-world applications. In the next chapter, we strive to answer this question by evaluating performance of the procedure on a real-world space exploration scenario.

Chapter 6

Numerical investigation of a SSSB sampling scenario

In Chapter 5 we propose an *hybrid* algorithm that is able to trade time complexity for byte complexity and byte complexity for resilience; we prove its worst-case performance properties on *generic* network topologies, propose criteria for selection of the tuning parameter and run through the workings of the algorithm on select simple network topologies.

This chapter is dedicated to exploration of the algorithm's performance in a real-world space exploration scenario. We show that the algorithm does gracefully transition from a flooding-like behavior to a fully hierarchical conduct, achieving time and byte complexities bounded by GHS and flooding respectively. The hybrid algorithm's consistently performs faster than GHS and exchanges significantly fewer bits than flooding; different choices of the tuning parameter lead to behaviors closer to the former or the latter, respectively.

6.1 Sampling of Small Solar System Bodies

Consider the following scenario: a significant number of low-mass, low-cost battery-powered penetrators are deployed on a small Solar System body, e.g. Phobos. The penetrators land in different regions, anchor themselves to the ground and start collecting measurements about its composition. For simplicity, let us consider a single, scalar measurement, e.g. ground hardness or temperature: extension of our results to multiple measurements and vector values is trivial.

Individual measurements are of little interest; furthermore, they may be noisy and influenced by local conditions (e.g. the presence of rocks). What interests scientists is a *global* measure of the body's *average* composition

or temperature over geologically uniform areas. It is therefore pointless to broadcast readings from every probe back to Earth, wasting precious onboard power and time on the Deep Space Network: a better approach would be for agents to filter data and send back the result through *one* agent. The same holds true even in presence of an orbiting mothership: the complex and poorly understood gravity field of SSSBs makes it unadvisable for an orbiter to approach the body too closely, at least during the initial phases of an exploration mission (see e.g. Tricarico's work on the dynamic environment of Vesta in [111]).

In principle, a hierarchical structure may be established prior to deployment of the agents on the SSSB. The irregular gravity field of SSSBs, however, is generally poorly understood[1]: the dynamics of the deployment phase are very uncertain and, even in absence of agent failures, any predetermined hierarchical structure may reveal itself to be far from optimal in the light of the agents' actual landing positions. It is therefore best for agents to establish a communication pattern *after* touchdown.

Once agents have collected their individual measurements, they need to agree on two issues: *what* value to send back to Earth (or to the orbiting mothership) and *which agent* should send it.

Both problems are easily cast in our consensus framework. The value of the physical quantity to report to Earth can be estimated via a weighed average, a *hierarchically computable* function. Leader election can be performed by picking the agent located in the most favorable position with respect to the receiver (if such information is available) or simply the agent with the largest unique ID: both options require distributedly computing a maximum, a *locally computable* function.

We explore this scenario via numerical simulations on random geometric graphs, presented in Chapter 3. Random geometric graphs are a good model for many real-world distributed robotic systems: in the absence of further information about the deployment mechanism, it is natural to assume that each agent will independently assume a random position in a bidimensional plane. The agents' communication radius is selected to be slightly higher than the connectivity threshold discussed in Section 3.1.4, so as to ensure connectivity of almost all networks under examination (a basic requirement for consensus) while simulating a comparatively lean topology.

We assume agents to be aware of their neighbors' number and location, as discussed in Chapter 3; for simplicity, we consider no failures during execution of the consensus protocol. We consider a variable number of agents, ranging

[1] In fact, study of the internal composition of SSSBs and its influence on the body's gravity field is a strong scientific driver for the exploration of asteroids, comets and small moons.

from 10 to 750: while near-term distributed space missions may only involve comparatively few agents, future architecture such as NASA's ANTS foresee deployment of hundreds or thousands of autonomous vehicles.

We evaluate the execution time, message complexity and byte complexity of flooding, GHS and our hybrid algorithm applied to this consensus problem. The aforementioned metrics are discussed in Chapter 3: byte complexity and message complexity are proxies for the energy required by wireless telecommunications, while time complexity may be a concern if a *time-varying* quantity (e.g. the rate of change of temperature as agents move across the body's terminator) is measured.

The example we consider concerns static planetary penetrators. However we remark that our algorithm does not necessarily require agents to be static: in practical applications it is sufficient for them to move much more slowly than the characteristic execution time of the algorithm. The real-world time performance of our agreement procedure is measured in seconds at most, whereas current rovers exploring Mars have a top speed measured in hundreds of meters per *day*: our results are very applicable to swarms of *slowly moving* vehicles, including current planetary rovers.

6.2 Simulation methodology

The algorithm was implemented with an object-oriented software architecture. Agents are created by a central supervisor, which also generates the network topology before each execution; agents are informed of their immediate neighbors by the supervisor before waking up.
At this time, the software architecture is *synchronous*, although our algorithm is capable of asynchronous operation: the central supervisor acts as a global clock for all agents.

Agents communicate through a *mailman* process: the mailman oversees message collection and delivery and enforces timing constraints. It is also able to selectively or probabilistically drop messages, although this feature is not currently used.

Visualization is handled through an external open-source application, Gephi (available at [42]): the supervisor and mailman interface with a dedicated process which relays agents' IDs, positions and neighbors as well as messages exchanged to a Gephi streaming server running on a local or remote machine.

The modular architecture is specifically designed so as to facilitate future deployment on hardware agents: we discuss a potential implementation in Chapter 7.

Figure 6.1 shows a scheme of the software architecture.

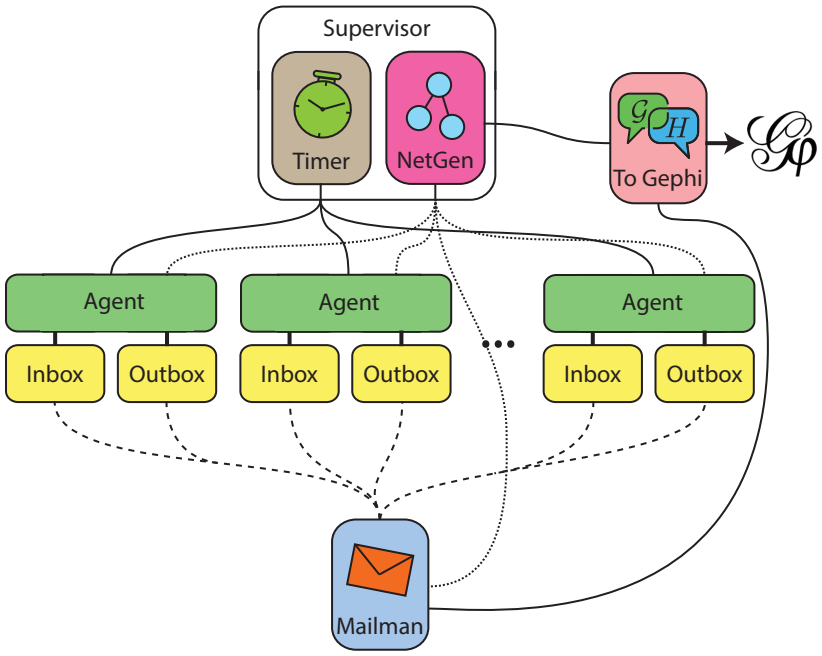


Figure 6.1: Simulation software architecture

Our hybrid algorithm, GHS and flooding are executed on random geometric graphs counting 10 to 750 agents, in increments of 10. For each number of agents, 100 executions on randomly generated networks are considered. The hybrid algorithm, GHS and flooding are run on the same networks[2] to ensure consistency of results. The hybrid algorithm is executed with four different values for m, decreasing from $m = n/10$ to $m = 3$ through $m = n/\log n$ and $m = \log n$. We evaluate three performance parameters:

- Time complexity, measured by the number of rounds to completion.
- Message complexity, measured by the number of messages exchanged before the algorithm converges.
- Byte complexity, measured by the overall size of messages exchanged before the algorithm converges. Small messages such as the ones exchanged by GHS contribute to byte complexity with a constant value; large messages such as those used by flooding algorithm contribute proportionally to the number of agent or cluster values they carry.

[2]Networks are randomly generated. The random number generator's seed is univocally determined by the date (but not the time) of execution, the number of agents and the execution number. All simulations were run on the same day.

All three algorithms terminate when all nodes *stop* after computing the consensus value: nodes are required to be aware that they hold the consensus value.

Source code for all simulations is available at the following URL:
`http://stanford.edu/~frossi2/HybridCons/MSCThesis_code.zip` .

6.3 Results

6.3.1 Time complexity

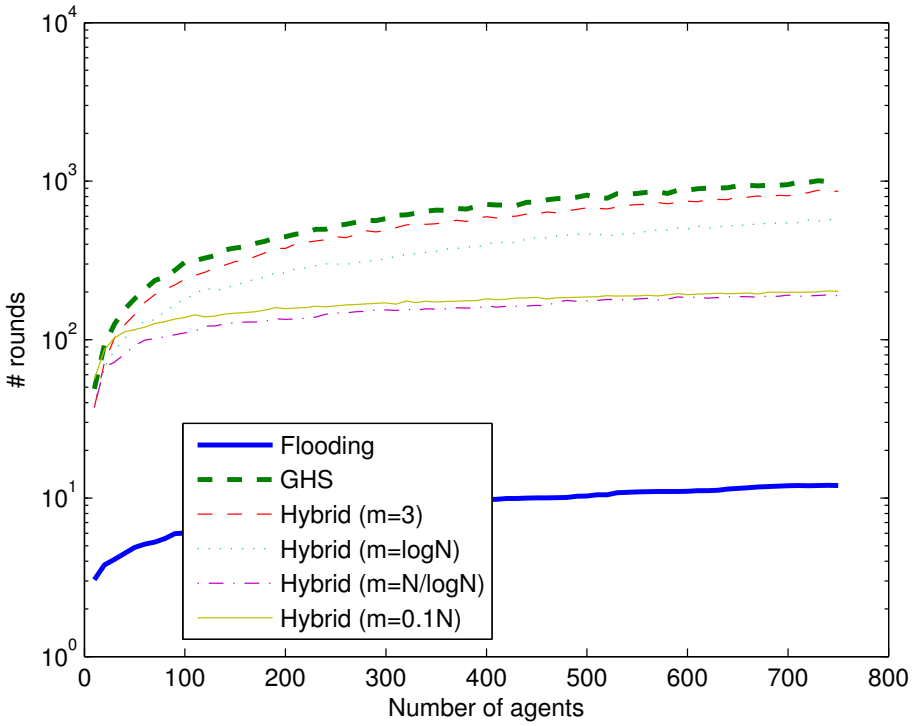


Figure 6.2: Rounds to completion of our hybrid algorithm compared to GHS and flooding

Figure 6.2 shows the number of rounds different algorithms require by the three algorithms to reach consensus. As expected, flooding achieves the best time complexity and GHS is by far the worst. Our algorithm gracefully scales from the former's behavior to the latter: as the tuning parameter m decreases from $n/10$ to $n/\log n$, $\log n$ and finally 3, tree size, proportional to n/m, increases and so does the number of rounds required to reach consensus.

Intuitively, two effects come into play:

- As m increases and the maximum size of trees decreases, generally smaller trees are created in Phase 1, which has very high time complexity.

- The recurring time complexity of Phase 4 grows with $\mathrm{Diam}(G)n/m$:

performance degradation with respect to optimal time behavior is upper-bounded by n/m.

Surprisingly, the choice of $m = n/\log n$ consistently yields results closer to flooding than $m = n/10$. In fact, for low values of n, $n/\log n$ is *lower* than 10: the difference between the two methods is significantly less marked and well within experimental error for higher numbers of agents, as shown in fig. 6.5a.

Besides this small anomaly, results are extremely consistent, even for small number of agents: clear trends are already evident in simulations involving as few as 50 nodes.

6.3.2 Byte complexity

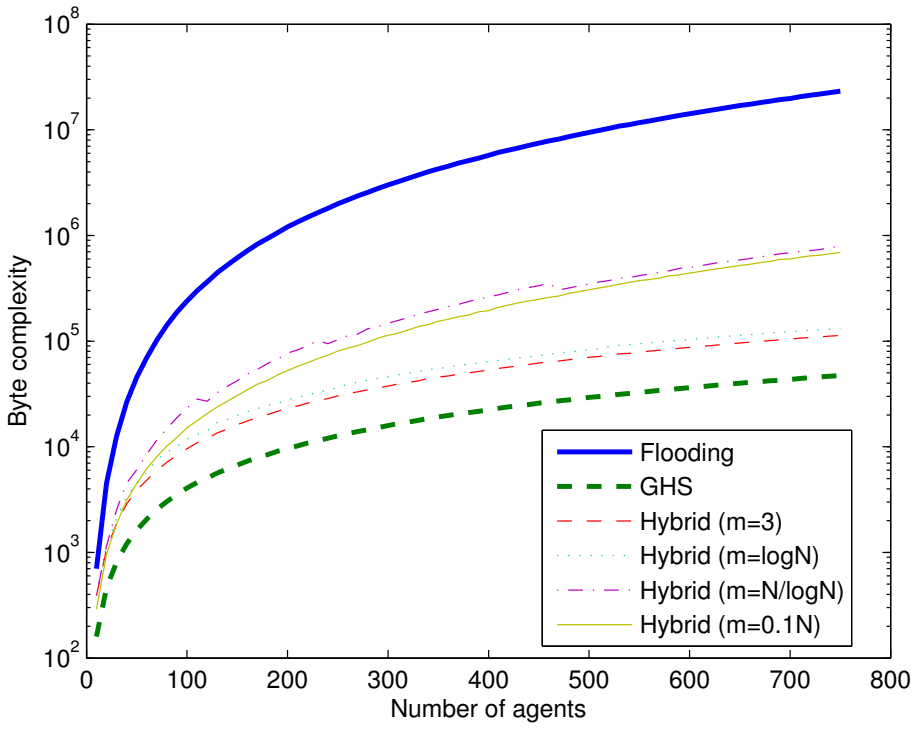


Figure 6.3: Bytes exchanged by our hybrid algorithm, GHS and flooding

Figure 6.3 reports the overall number of bytes exchanged by GHS, flooding and our hybrid algorithm for four different values of m. As expected, GHS and all instances of our hybrid algorithm perform significantly better than flooding; as the size of clusters decreases, the hybrid algorithm's behavior

approaches flooding's and the byte complexity increases. Routing information on lean tree structures, which guarantee no duplication, contributes to a large reduction in time complexity: the overall amount of information being flooded among the clusters is also upper bounded by the number of clusters, governed by m.

Once again, simulations with $m = n/\log n$ consistently yields results closer to flooding than $m = n/10$: we believe the reason for this phenomenon is the same as in the previous paragraph.

Trends are extremely clear even for very small (sub-50) numbers of agents.

6.3.3 Message complexity

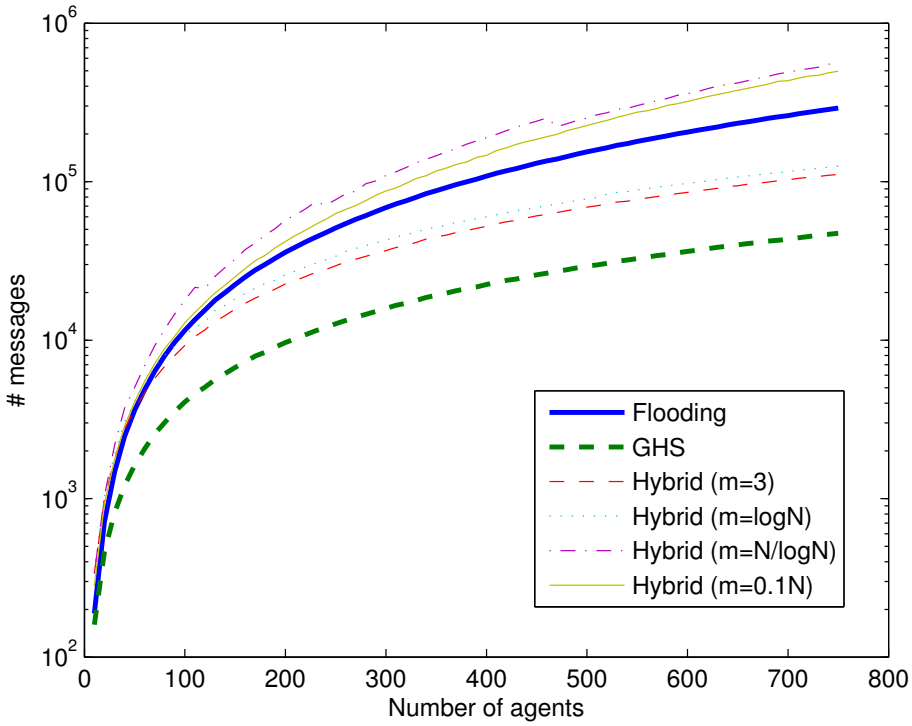
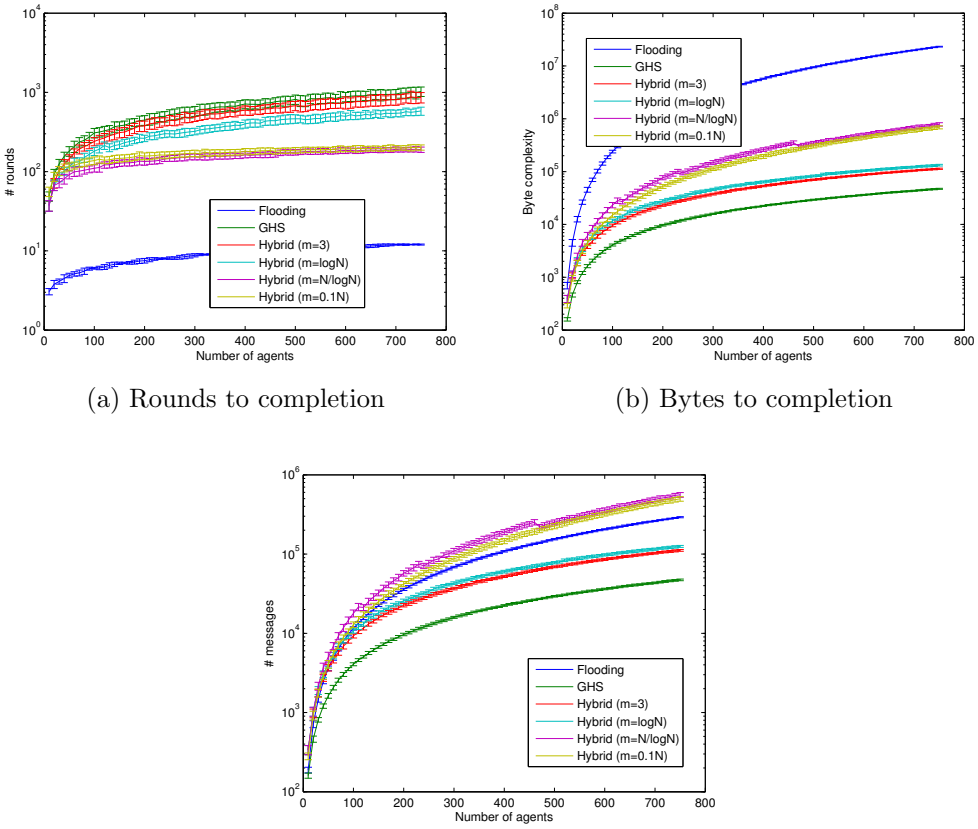


Figure 6.4: Messages exchanged by our hybrid algorithm compared to GHS and flooding

The message complexity of our flooding algorithm is strongly dependent on the selection of the tuning parameter. Unsurprisingly, no choice of m yields better performance than the message-optimal GHS algorithm; it is more interesting to note that, for high m, the number of messages exchanged

by our hybrid algorithm significantly exceeds even that of notoriously inefficient flooding. Yet, while the message complexity of our algorithm is far from stellar, the size of exchanged messages is much smaller than in flooding, as discussed in section 6.3.2: our algorithm is a better choice whenever a parsimonious communication protocol with low overhead is employed.



(a) Rounds to completion

(b) Bytes to completion

(c) Messages to completion

Figure 6.5: Variance (1σ) of numerical results

6.3.4 Tradeoffs between time and byte complexity

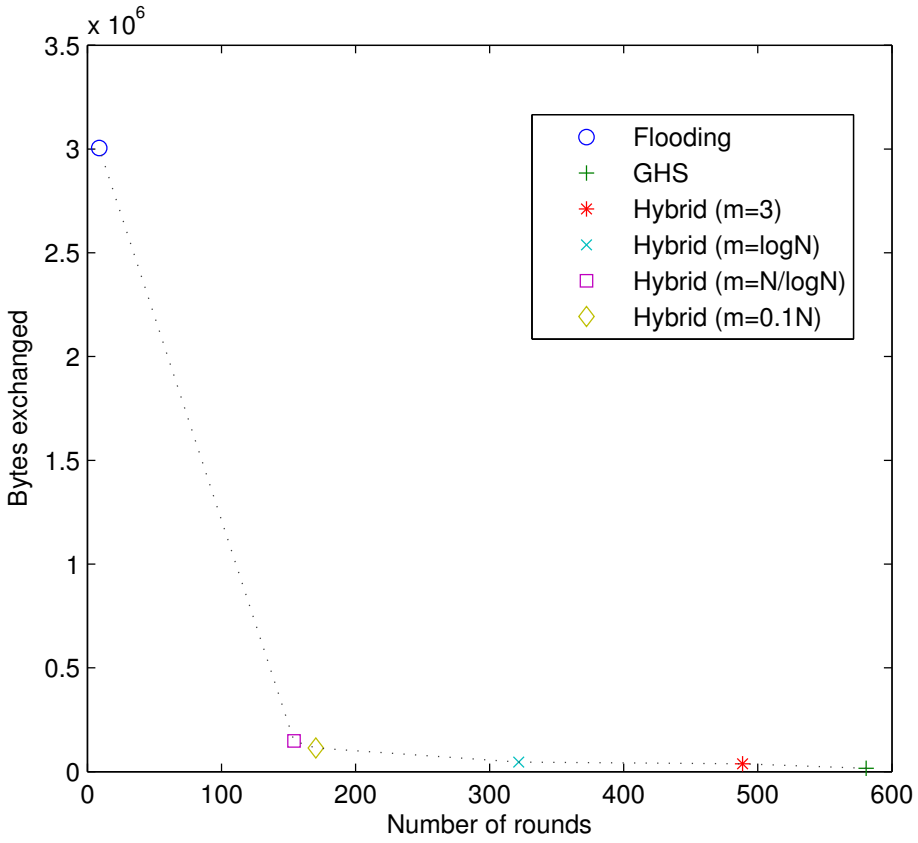


Figure 6.6: Pareto front formed by executions of our hybrid algorithm for several values of m compared to time-optimal flooding and byte-optimal GHS. $n = 300$.

Figure 6.6 shows a Pareto front formed by executions of our algorithm, GHS and flooding for $n = 300$. While we do *not* claim optimality of our algorithm with respect to hybrid metrics, the results highlight the smooth scaling between optimal time and optimal byte complexity offered by our hybrid algorithm: our protocol offers mission designers the opportunity to intuitively trade time complexity and robustness for byte complexity and thus energy efficiency according to mission requirements.

6.3.5 Recurring complexity

The simulations presented up to now refer to *one-shot* consensus: penetrators build a semi-hierarchical (or, in the case of GHS, fully hierarchical) struc-

ture from scratch, then use it to exchange information until they reach an agreement.

In absence of failures or other significant changes in the network topology, however, the same structure can be used for several consensus rounds, with evident savings in terms of time and power consumption.

We estimate the *recurring* cost of consensus as follows:

- Flooding does not build a routing structure within the network: each new round of consensus incurs in the same time and energy cost, irrespective of previous executions.

- Once GHS has built a spanning tree, subsequent consensus rounds are easily performed: the root contacts all nodes through a broadcast across the spanning tree, then collects values through a convergecast and informs all nodes of the consensus value via a second broadcast. The message complexity of the procedure is $3(n-1)$: each edge of the tree is crossed by three messages. Each message has size $O(\log n)$. The time complexity is three times the height of the spanning tree, which can be evaluated numerically.

- Our hybrid algorithm uses different message types to build the hierarchical *structure* and to exchange *information* among nodes: it is therefore easy to numerically assess the recurring message and byte complexity of consensus once a structure has been built by only counting messages belonging to the second class.
  Assessing time complexity is less trivial: clusters enter recurring Phase 4 at different times and progression of inter-cluster communication may be hindered by a single slow cluster. Despite these limitations, we consider the *average* amount of time spent by nodes in Phase 4 as an estimator of the recurring time complexity of our hybrid algorithm: while imperfect, this estimator gives a simple, synthetic assessment of the time complexity of recurring operations and its value agrees well with theoretical worst-case results.

Recurring time complexity

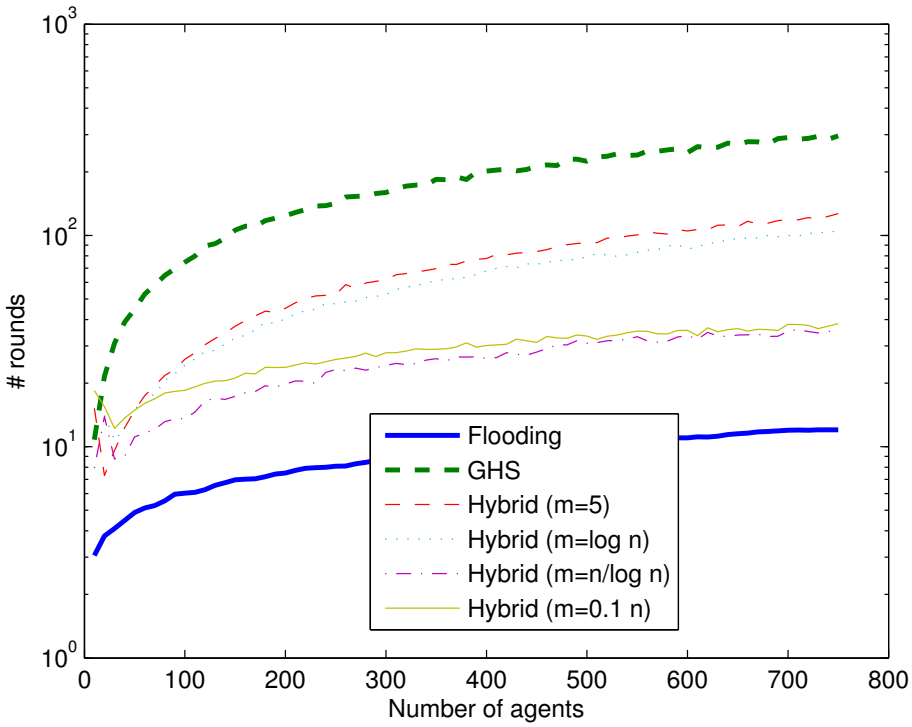


Figure 6.7: Recurring *rounds to completion of our hybrid algorithm compared to GHS and flooding*

Figure 6.7 shows the *recurring* number of rounds required by our algorithm, GHS and flooding to solve the consensus problem. While flooding remains time-optimal, the performance gap with hierarchical and semi-hierarchical algorithms decreases considerably: for high n, the recurring running time of our hybrid algorithm is as much as one order of magnitude lower than the one-shot running time. The same trends observed in 6.2 are evident here: decreasing m yields worse time performance.

Recurring byte complexity

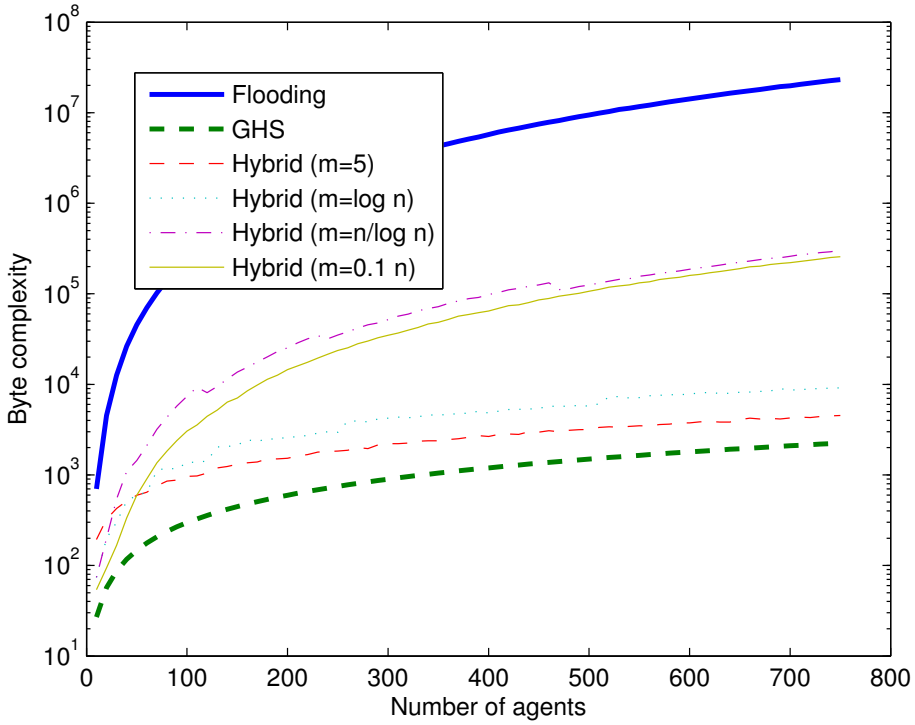


Figure 6.8: Recurring *overall bytes exchanged by our hybrid algorithm compared to GHS and flooding*

Figure 6.8 reports the *recurring* number of bytes exchanged by our algorithm, GHS and flooding to solve the consensus problem. Algorithms with strong hierarchical structures, namely GHS and our algorithm with low m, exhibit excellent performance, up to an order of magnitude better than in the one-shot case. On the other hand, algorithms with mainly nonhierarchical structures (e.g. flooding and our hybrid algorithm with high m) see little benefit in reusing an existing hierarchy.

Recurring message complexity

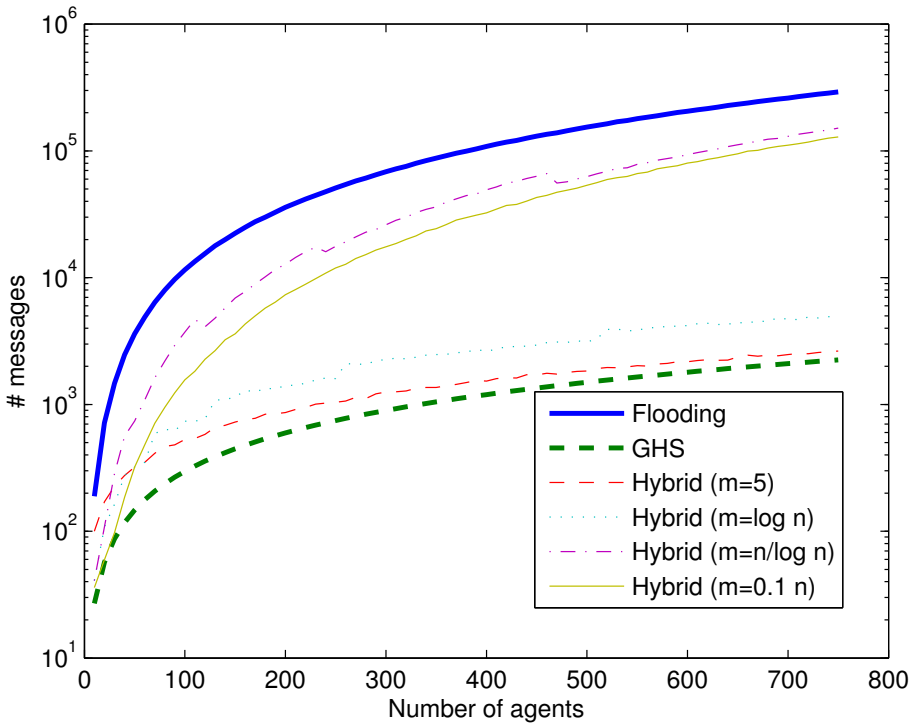


Figure 6.9: Recurring *number of messages exchanged by our hybrid algorithm compared to GHS and flooding*

We show the recurring message complexity of GHS, flooding and our algorithm in figure 6.9. Surprisingly, flooding has the worst *recurring* message performance: our hybrid algorithm may use more messages than flooding during *set up* but, once a structure has been established, it performs considerably better than flooding for *any* value of m even if message size is not taken into account.

6.4 Conclusion

Flooding exhibits time-optimal performance within the class of algorithms under consideration: the number of messages and byte exchanged, on the other hand, is extremely high. This makes flooding generally unsuitable for space exploration applications, where power consumption is a concern: only when time performance is paramount (which is typically never the case in space exploration) should a flooding algorithm be considered.

On the other end of the spectrum, GHS is message and byte-optimal among the algorithms under study; its power consumption is the lowest in its class, no matter which metric is used. The price to pay for this is an execution time consistently higher than all other algorithms and one to two orders of magnitude away from the theoretical optimum. Furthermore, GHS is *fragile*: a single failure in one of $n - 1$ separate points can bring the algorithm to a halt and as many as $O(n)$ time steps may be required to resume operations. GHS is the algorithm of choice if power consumption is paramount and reliability is not a concern: while the first condition is typical of space exploration applications (except when time-varying quantities are being estimated), the second is typically not verified.

Our hybrid algorithm smoothly moves from time-optimal to byte-optimal behavior, with results in between GHS and flooding: its performance can easily be tuned to best suit constraints imposed by the specific consensus problem under consideration and by environmental conditions. As we expected, the number of *messages* exchanged by the hybrid algorithm, can be worse than flooding's for certain tuning parameters: our algorithm is generally unsuitable for applications where the "cost" of one message is independent of the size of its payload.

As long as this is not the case, our algorithm offers designers great flexibility: it can be optimized to maximize a given *hybrid* performance metric over the set of possible tunings, offering satisfactory time performance and robustness without compromising power consumption. It is trivial, for instance, to tune the algorithm so as to guarantee meeting an "hard" time constraint (e.g. to ensure that sampling of a temperature gradient is performed at a sufficiently high rate) while minimizing energy complexity or to choose the most energy-efficient architecture that can withstand a given link failure rate.

As long as the network topology does not change significantly, the hierarchical structure built by our hybrid algorithm and GHS can be reused for multiple decisions: mostly hierarchical procedures (i.e. GHS and our hybrid algorithm for low values of m) significantly benefit from reuse of the existing structure, while mostly nonhierarchical algorithms such as flooding and

our hybrid algorithm for high m see little appreciable benefit. On the other hand, mostly hierarchical structures are more fragile: they are more likely to be frequently and significantly disrupted by intra-cluster failures, requiring significant power and a nontrivial amount of time to recover.

These tradeoffs are the same as we observed in Chapter 4. Now, however, the designer is not faced with a binary choice: our algorithm can be optimally tuned to reach an adequate compromise between execution time, energy consumption and resilience to failures.

Chapter 7

Conclusions and future research directions

7.1 Conclusions

In Chapter 1, we showed how distributed consensus on robotic networks is a key technology required to explore small Solar System bodies and, through them, better understand the origin of our Solar System.

Unfortunately, while consensus has long been studied in the Computer Science and Control Theory communities, current paradigms are unsuitable for robotic networks: existing algorithms either concern themselves with time complexity alone, are designed for mobile computer networks or grossly underexploit the communication capabilities of modern robotic agents.

This thesis strives to fill the gap between the existing body of knowledge on consensus theory and modern robotic space exploration applications. We rigorously explore fundamental limitations of consensus on *robotic* networks of *stationary* or *slow-moving* agents and show how existing algorithms can achieve optimal performance with respect to time and energy-based metrics. Results are not satisfying: optimal performance with respect to one parameter typically causes a strong degradation of other metrics. We therefore design an *hybrid* algorithm that autonomously builds a semi-hierarchical structure to achieve intermediate performance between time optimality and energy optimality. The algorithm can be tuned: we offer insight in how tuning affects execution time, energy consumption and robustness and show that extreme values of the tuning parameter allow to recover either time-optimal or energy-optimal behavior. Finally, we explore performance of the algorithm on a real-world ground sampling scenario on a small solar system body: as expected, the algorithm smoothly trades execution time for energy consumption, moving on a Pareto front (although our algorithm is *not*

proven to be optimal with respect to hybrid metrics) from time-optimal to message-optimal behavior.

While our simulations concern a simple data fusion application, our algorithm is designed to work with any *hierarchically computable* function, including leader election, majority voting and mediation between different policies. The procedure is comparatively easy to implement and tuning is extremely intuitive: system-level tradeoffs can readily translate in a single numerical parameter.

It is our hope that our hybrid algorithm will enable mission designer to devise bolder space missions, embracing massively distributed and highly autonomous architectures for the exploration of small Solar System bodies: through these, we may finally be able to answer fundamental questions about the origin of our Solar System, our home among the stars.

7.2 Future research directions

Our theoretical lower bounds and our hybrid algorithm are just steps towards a better understanding of the consensus problem on robotic networks. Future research direction include an extension of our results to quickly time-varying networks, which are representative of moving agents such as fast hoppers, and especially to networks whose topology depends on the information exchanged between the agents; lower bounds for *broadcast-based* communication protocols commonly used in legacy and low-cost robotic applications; handling of malicious failures for LEO and ground-based applications; and *complex* decision-making on hybrid networks via linear temporal logic.

Applications of our work go well beyond space exploration performed by stationary and slow-moving agents: potential future applications include consensus on swarms of fast-moving agents such as NASA's Tet Walker, airborne patrolling, distributed conflict resolution in air traffic control management and sensor fusion for demanding ground-based application such as Search and Rescue.

7.2.1 Application to moving networks

In typical rendezvous, deployment and formation flying applications, the time evolution of the agents' position, and therefore the network topology, depends on the information exchanged between the agents: rovers and probes perform closed-loop control of their position and velocity based on their and their neighbor's physical and logical state.

Despite significant research on the issue, the closed-loop interaction between the *logical* flow of information among agents and the evolution of the underlying *physical* network is not sufficiently understood. Algorithms have been proposed that guarantee connectivity maintenance while performing rendezvous and deployment tasks. These algorithms, however, sport the typical drawbacks of average-based protocols: namely, convergence is asymptotic and the procedure has very demanding requirements in terms of inter-agent communication.

Our hybrid algorithm does work on slowly moving networks: if the link failure frequency is slower than n/m, the algorithm manages to rearrange the network between changes in topology. New links generated when agents come within range of one another are simply added to inter-cluster routing tables with no interruption in the consensus procedure.
The algorithm, however, does not preserve minimum weight clusters. As new links appear, it may be convenient to rearrange nodes in a more efficient configuration; our procedure, on the other hand, simply preserves existing trees as long as no intra-cluster link fails.

Future research will focus on design of algorithms optimized for networks of *fast, actively moving* agents, maintaining minimum-weight (local) clusters through reactive and preemptive action. Reactive action involves maintaining *minimum weight* clusters as new links appear and the weight (i.e. the distance) of existing links varies; preemptive action exploits the agents' knowledge of the algorithm and their neighbors' state to *anticipate* link failures caused by agents receding from one another, thus minimizing downtime.

In certain sensing applications, agents have no means to control their own position: the time evolution of the network is rapid yet uncontrollable and unpredictable by the nodes. As a motivating example, consider be a swarm of weather balloons deployed in Venus's atmosphere: the aggressive and poorly-understood dynamic environment would cause an unpredictable evolution of the network topology, with agents would have no way to control. We will address fundamental performance limitations on a simplified yet expressive probabilistic model of these networks, extending results by Clementi et al. [21] on edge-Markovian evolving graphs.

7.2.2 Broadcast communication protocols

Metrics introduced in Chapter 3, fundamental limitations presented in Chapter 4 and upper bounds on complexity discussed in Chapters 5 and 6 are tailored to *directional* communication protocols. In Chapter 3 we show how this hypothesis is very relevant for modern space-based applications; yet many low-cost ground-based and airborne robotic networks still use off-the-shelf

omnidirectional telecommunication equipment.

Future research will address the issue of optimality on these networks, where the cost of sending a message to *all* neighbors is no higher than the cost of communicating with a single node: we will explore fundamental performance limitations and seek optimal and hybrid algorithms with *broadcast-based* performance parameters.

7.2.3 Distributed tuning

Our hybrid algorithm's tuning parameter is selected during the design phase, prior to execution. On the other hand, the algorithm's very goal is to reach decisions efficiently on networks of agents: it is natural to imagine agents using it to agree on a change of the tuning parameter, possibly in reaction to new environmental conditions or to an external input perceived by certain nodes. Research will focus on design of routines that allow for fast, incremental post-retuning reconfiguration without undergoing a complete rebuild of the network.

7.2.4 Handling of byzantine failures

Byzantine failures, in which one or more agents may act maliciously to disrupt the network's global behavior, are not a significant concern in space exploration: the telecommunication power required to communicate with a probe beyond Earth orbit puts malicious endeavors well outside the reach of any entity unable to secure access to the Deep Space Network. On the other hand, byzantine failures on cyber-physical networks are a major issue in critical ground infrastructures such as power distribution grids, ground-based telecommunication networks and large industrial plants. We refer the reader to Pasqualetti's work [91] for further information. Even LEO satellites are not completely immune to malicious failures: at least two instances of malicious entities gaining unrestricted, unauthorized access to LEO satellites have been recorded in recent years [101, p. 215].

Significant effort has been poured into studying fundamental limitations of failure detection, identification and mitigation on networks: current studies, however, focus on detection on *fixed* cyber-physical networks where communication and computational resources are not a limiting factor. We hope to extend and adapt results by Pasqualetti and Bullo [94] to robotic network and to design *energy-efficient* failure detection, identification and mitigation algorithms.

7.2.5 Complex decision-making via LTL

This thesis concentrates on how to build efficient communication structures to achieve consensus on robotic networks. Yet the actual function agents should agree on is almost secondary: throughout this work, we merely require it to be *hierarchically computable*. In Chapter 3, we list some hierarchically computable functions and their applications, which include majority voting and mediation among different options.

Future research will explore the possibility of performing complex decision making on robotic networks via Linear Temporal Logic: the expressivity of LTL allows to naturally and compactly represent complex logic issues and should lend itself to the semi-hierarchical decision-making inherent in our architecture.

7.2.6 Earth-based applications

Space exploration is a prime application of our research; yet many of the concerns that motivate our work are shared by ground-based applications.

Our research group is actively exploring an application of our research to deployment, coordination and connectivity maintenance of swarms of UAVs patrolling the Galapagos natural reserve. The project, in collaboration with Universidad San Francisco de Quito, aims at discouraging illegal shark finning in the Galapagos natural reserve with low-cost, purpose-built autonomous patrolling planes; challenges include guaranteeing continuous communications with an existing ground station with unfavorable topography, autonomously deploying planes to maximize continuous coverage and reacting to observations and potential threats with no supervisor input.

Further potential applications of our research include automated conflict resolution in aircraft traffic control [110] and autonomous data filtering on low-power, quick deployment sensor networks for disaster response.

7.2.7 A reference hardware implementation

The object-oriented architecture of the simulator presented in Chapter 6 was specifically designed to facilitate deployment on real-world decentralized hardware.

Stanford's Autonomous Systems Laboratory is equipped with sixteen Pololu m3pi autonomous rovers controlled by a central PC-based dispatcher via off-the-shelf WiFi hardware. The rovers' position is estimated by a Vicom motion capture system interfacing with the dispatcher computer.

Implementation of our hybrid algorithm on this platform is underway; given

the centralized nature of the existing WiFi communication protocol, the selected architecture uses a centralized mailman protocol, hosted on the current dispatcher, to handle inter-agent communication. The agents' logic code is executed by the agents themselves; on the other hand, message delivery is transparently mediated by the dispatcher, which also keep agents up-to-date about their position and heading. Figure 7.1 shows a schematic representation of the selected architecture, while figure 7.2 shows the current setup of the robotic agents.

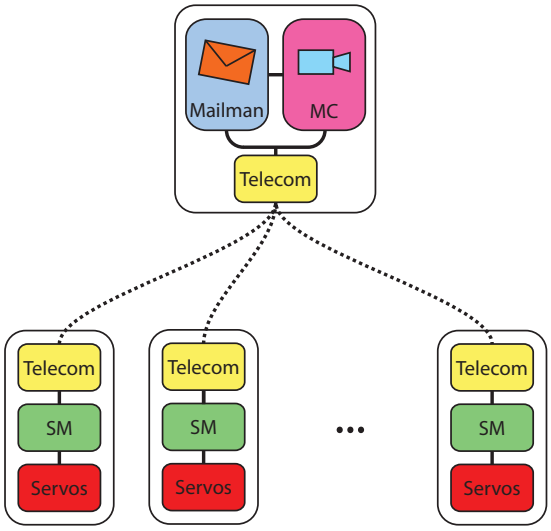


Figure 7.1: Software architecture of a distributed *implementation of our hybrid algorithm on ASL's robotic testbed*

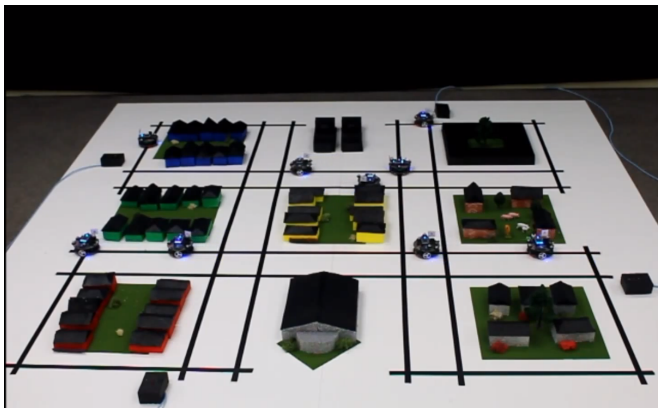


Figure 7.2: Stanford University Autonomous Systems Lab's multiagent robotic platform

Bibliography

[1] B. Acikmese and M. Mandic. "Decentralized observer with a consensus filter for distributed discrete-time linear systems". In: *American Control Conference (ACC), 2011*. IEEE. 2011, pp. 4723–4730.

[2] M. F. A'Hearn et al. "Deep Impact: Excavating Comet Tempel 1". In: *Science* 310.5746 (2005), pp. 258–264. DOI: 10.1126/science.1118923. URL: http://www.sciencemag.org/content/310/5746/258.abstract.

[3] Ross Allen et al. "Internally-Actuated Rovers for All-Access Surface Mobility: Theory and Experimentation". In: *Proc. IEEE Conf. on Robotics and Automation*. 2013.

[4] T. P. Andert et al. "Precise mass determination and the nature of Phobos". In: *Geophysical Research Letters* 37 (2010).

[5] D. Angluin et al. "Computation in networks of passively mobile finite-state sensors". In: *Distributed Computing* 18.4 (2006), pp. 235–253.

[6] Thierry Asté et al. "Downlink beamforming for cellular mobile communications (gsm system)". In: *Annales Des Télécommunications* 53.11-12 (1998), pp. 435–448. DOI: 10.1007/BF02998590. URL: http://dx.doi.org/10.1007/BF02998590.

[7] B. Awerbuch. "Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems". In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM. 1987, pp. 230–240.

[8] Baruch Awerbuch. "Complexity of network synchronization". In: *Journal of the ACM (JACM)* 32.4 (1985), pp. 804–823.

[9] Grey Ballard et al. "Graph Expansion and Communication Costs of Fast Matrix Multiplication". In: *Journal of the ACM* (2012).

[10] G. Ballard et al. "Minimizing communication in numerical linear algebra". In: *SIAM Journal on Matrix Analysis and Applications* 32.3 (2011), pp. 866–901.

[11] *Beamforming Boosts the Range and Capacity of WiMAX Networks - White Paper*. 2008. URL: http://www.fujitsu.com/downloads/MICRO/fma/formpdf/WiMAXbeamform.pdf.

[12] Jeffrey F. Bell, Fraser Fanale, and Dale P. Cruikshank. "Chemical and physical properties of the Martian satellites". In: *Resources of Near Earth Space* (1993), pp. 887–901.

[13] Florence Benezit et al. "Reaching consensus about gossip: convergence times and costs". In: *Information Theory and Applications* (2008).

[14] Dimitri P Bertsekas and John N Tsitsiklis. "Comments on "Coordination of groups of mobile autonomous agents using nearest neighbor rules"". In: *Automatic Control, IEEE Transactions on* 52.5 (2007), pp. 968–969.

[15] Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*. Available: http://hdl.handle.net/1721.1/371. 1989.

[16] S. Boyd et al. "Randomized gossip algorithms". In: *Information Theory, IEEE Transactions on* 52.6 (June 2006), pp. 2508–2530. ISSN: 0018-9448. DOI: 10.1109/TIT.2006.874516.

[17] James E Burns. *A formal model for message passing systems*. Computer Science Department, Indiana University, 1980.

[18] Julie C Castillo-Rogez et al. "Expected science return of spatially-extended in-situ exploration at small Solar system bodies". In: *2012 IEEE Aerospace Conference*. IEEE. 2012, pp. 1–15.

[19] S. Chatterjee and E. Seneta. "Towards Consensus: Some Convergence Theorems on Repeated Averaging". English. In: *Journal of Applied Probability* 14.1 (1977), pages. ISSN: 00219002. URL: http://www.jstor.org/stable/3213262.

[20] Yongwook Choi et al. "Energy-optimal distributed algorithms for minimum spanning trees". In: *Selected Areas in Communications, IEEE Journal on* 27.7 (2009), pp. 1297–1304.

[21] A.E.F. Clementi et al. "Flooding Time of Edge-Markovian Evolving Graphs". In: *SIAM journal on discrete mathematics* 24.4 (2010), pp. 1694–1712.

[22] T.H. Cormen et al. *Introduction to algorithms*. 3rd edition. MIT press, 2009.

[23] Alejandro Cornejo, Seth Gilbert, and Calvin Newport. "Aggregation in dynamic networks". In: *Proceedings of the 2012 ACM symposium on Principles of distributed computing*. PODC '12. Madeira, Portugal: ACM, 2012, pp. 195–204. ISBN: 978-1-4503-1450-3. DOI: 10.1145/2332432.2332468. URL: http://doi.acm.org/10.1145/2332432.2332468.

[24] J. Cortes, S. Martinez, and F. Bullo. "Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions". In: *Automatic Control, IEEE Transactions on* 51.8 (Aug. 2006), pp. 1289–1298. ISSN: 0018-9286. DOI: 10.1109/TAC.2006.878713.

[25] Steven A Curtis et al. "ANTS for human exploration and development of space". In: *Aerospace Conference, 2003. Proceedings. 2003 IEEE*. Vol. 1. IEEE. 2003, pp. 1–261.

[26] S. Curtis et al. "Tetrahedral Robotics for Space Exploration". In: *Aerospace and Electronic Systems Magazine, IEEE* 22.6 (2007), pp. 22–30. ISSN: 0885-8985. DOI: 10.1109/MAES.2007.384077.

[27] Morris H. Degroot. "Reaching a Consensus". In: *Journal of the American Statistical Association* 69.345 (1974), pp. 118–121. DOI: 10.1080/01621459.1974.10480137. eprint: http://www.tandfonline.com/doi/pdf/10.1080/01621459.1974.10480137. URL: http://www.tandfonline.com/doi/abs/10.1080/01621459.1974.10480137.

[28] James Demmel et al. *Perfect strong scaling using no additional energy*. Tech. rep. UCB/EECS-2012-126. EECS Department, University of California, Berkeley, Feb. 2011. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-13.html.

[29] J.W. Demmel, M.T. Heath, and H.A. Van Der Vorst. *Parallel numerical linear algebra*. Computer Science Division (EECS), University of California, 1993.

[30] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. "The communication complexity of distributed task allocation". In: *Proceedings of the 2012 ACM symposium on Principles of distributed computing*. ACM. 2012, pp. 67–76.

[31] M. Epstein et al. "Using hierarchical decomposition to speed up average consensus". In: *Proceedings of the 17th IFAC World Congress, 2008*. 2008, pp. 612–618.

[32] P. Erdős and A. Rényi. "On the evolution of random graphs". In: *Publications of the Mathematical Institute of the Hungarian Academy of Sciences* (1961), pp. 17–61.

[33] ESA. *BepiColombo*. URL: http://sci.esa.int/bepicolombo (visited on 06/22/2013).

[34] ESA. *Giotto*. Mar. 2006. URL: http://sci.esa.int/giotto/ (visited on 06/23/2013).

[35] ESA. *Venus Express*. URL: http://sci.esa.int/venus-express/ (visited on 06/22/2013).

[36] Tara A Estlin et al. "AEGIS Automated Science Targeting for the MER Opportunity Rover". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 3.3 (2012), p. 50.

[37] Miroslav Fiedler. "Algebraic connectivity of graphs". In: *Czechoslovak Mathematical Journal* 23.2 (1973), pp. 298–305.

[38] Miroslav Fiedler. "Laplacian of graphs and algebraic connectivity". In: *Combinatorics and graph theory* 25 (1989), pp. 57–70.

[39] A. Fujiwara et al. "Hayabusa at Asteroid Itokawa (Special Issue)". In: *Science* 312.5778 (2006), pp. 1327–1353.

[40] R.G. Gallager, P.A. Humblet, and P.M. Spira. "A distributed algorithm for minimum-weight spanning trees". In: *ACM Transactions on Programming Languages and systems (TOPLAS)* 5.1 (1983), pp. 66–77.

[41] Chunkai Gao, Jorge Cortés, and Francesco Bullo. "Notes on averaging over acyclic digraphs and discrete coverage control". In: *Automatica* 44.8 (2008), pp. 2120–2127. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2007.12.017.

[42] Gephi.org. *Gephi - The Open Graph Viz Platform*. URL: https://gephi.org/ (visited on 06/22/2013).

[43] Giovanni Giardini. "Multi-Agent Intelligent System for Planetary Space Exploration". PhD thesis. Politecnico di Milano - Dipartimento di Ingegneria Aerospaziale, 2007.

[44] M. Giuranna et al. "Compositional Interpretation of PFS/MEx and TES/MGS Thermal Infrared Spectra of Phobos". In: *European Planetary Science Congress Abstracts 2010-2011*. Vol. 5. 2010.

[45] Ashish Goel, Sanatan Rai, and Bhaskar Krishnamachari. "Sharp thresholds for monotone properties in random geometric graphs". In: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. STOC '04. Chicago, IL, USA: ACM, 2004, pp. 580–586. ISBN: 1-58113-852-0. DOI: 10.1145/1007352.1007441. URL: http://doi.acm.org/10.1145/1007352.1007441.

[46] Rodney Gomes et al. "Origin of the cataclysmic Late Heavy Bombardment period of the terrestrial planets". In: *Nature* 435.7041 (2005), pp. 466–469.

[47] J. N. Gray. "Notes on data base operating systems". In: *Operating Systems: An Advanced Course*. Ed. by R. Bayer, R. M. Graham, and G. Seegmüller. Vol. 60. Lecture Notes in Computer Science. New York: Springer-Verlag, 1978. Chap. 3.F, p. 465.

[48] David M Harland. *Mission to Saturn: Cassini and the Huygens Probe*. Springer, 2002.

[49] Y. Hatano and M. Mesbahi. "Agreement over random networks". In: *Automatic Control, IEEE Transactions on* 50.11 (Nov. 2005), pp. 1867–1872. ISSN: 0018-9286. DOI: 10.1109/TAC.2005.858670.

[50] Ayanna M Howard and Edward W Tunstel. *Intelligence for space robotics*. TSI Press, 2006.

[51] Scott Hubbard. *Exploring Mars: Chronicles from a decade of discovery*. University of Arizona Press, 2012.

[52] Félix Ingrand et al. "Decisional autonomy of planetary rovers". In: *Journal of Field Robotics* 24.7 (2007), pp. 559–580.

[53] A. Jadbabaie, Jie Lin, and A.S. Morse. "Coordination of groups of mobile autonomous agents using nearest neighbor rules". In: *Automatic Control, IEEE Transactions on* 48.6 (June 2003), pp. 988–1001. ISSN: 0018-9286. DOI: 10.1109/TAC.2003.812781.

[54] Nicholas R. Jennings. "On agent-based software engineering". In: *Artificial Intelligence* 117.2 (2000), pp. 277–296. ISSN: 0004-3702. DOI: http://dx.doi.org/10.1016/S0004-3702(99)00107-1.

[55] Xingde Jia. "Wireless networks and random geometric graphs". In: *Parallel Architectures, Algorithms and Networks, 2004. Proceedings. 7th International Symposium on*. 2004, pp. 575–579. DOI: 10.1109/ISPAN.2004.1300540.

[56] Maleq Khan, Gopal Pandurangan, and VS Anil Kumar. "Distributed algorithms for constructing approximate minimum spanning trees in wireless sensor networks". In: *Parallel and Distributed Systems, IEEE Transactions on* 20.1 (2009), pp. 124–139.

[57] Ephraim Korach, Shlomo Moran, and Shmuel Zaks. "Tight lower and upper bounds for some distributed algorithms for a complete network of processors". In: *Proceedings of the third annual ACM symposium on Principles of distributed computing*. ACM. 1984, pp. 199–207.

[58] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. "Distributed computation in dynamic networks". In: *Proceedings of the 42nd ACM symposium on Theory of computing*. STOC '10. Cambridge, Massachusetts, USA: ACM, 2010, pp. 513–522. ISBN: 978-1-4503-0050-6. DOI: 10.1145/1806689.1806760. URL: http://doi.acm.org/10.1145/1806689.1806760.

[59] Fabian Kuhn and Rotem Oshman. "Dynamic networks: models and algorithms". In: *SIGACT News* 42.1 (Mar. 2011), pp. 82–96. ISSN: 0163-5700. DOI: 10.1145/1959045.1959064. URL: http://doi.acm.org/10.1145/1959045.1959064.

[60] John Hopkins University Applied Physics Laboratory. *Discovery is NEAR*. URL: http://near.jhuapl.edu/index.html (visited on 06/23/2013).

[61] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine generals problem". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3 (1982), pp. 382–401.

[62] Ralph Lorenz and Jacqueline Mitton. *Lifting Titan's Veil: Exploring the Giant Moon of Saturn*. Cambridge University Press, 2002.

[63] Nancy A. Lynch. *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996. ISBN: 1558603484.

[64] S. Martinez et al. "On Synchronous Robotic Networks; Part I: Models, Tasks and Complexity". In: *Automatic Control, IEEE Transactions on* 52.12 (Dec. 2007), pp. 2199–2213. ISSN: 0018-9286. DOI: 10.1109/TAC.2007.908301.

[65] S. Martinez et al. "On Synchronous Robotic Networks; Part II: Time Complexity of Rendezvous and Deployment Algorithms". In: *Automatic Control, IEEE Transactions on* 52.12 (Dec. 2007), pp. 2214–2226. ISSN: 0018-9286. DOI: 10.1109/TAC.2007.908304.

[66] Howard E. McCurdy. *Low-Cost Innovation in Spaceflight - The Near Earth Asteroid Rendezvous (NEAR) Shoemaker Mission*. Monographs in Aerospace History 36. NASA History Division, 2005.

[67] Michael Meltzer. *Mission to Jupiter: A History of the Galileo Project*. NASA STI/Recon Technical Report SP-2007-4231. NASA History Division, 2007.

[68] Andrew H Mishkin et al. "Working the Martian night shift-the MER surface operations process". In: *Robotics & Automation Magazine, IEEE* 13.2 (2006), pp. 46–53.

[69] MIT Space Systems Laboratory. *SPHERES*. URL: http://ssl.mit.edu/spheres/ (visited on 06/23/2013).

[70] A Morbidelli et al. "Chaotic capture of Jupiter's Trojan asteroids in the early Solar System". In: *Nature* 435.7041 (2005), pp. 462–465.

[71] L. Moreau. "Stability of multiagent systems with time-dependent communication links". In: *Automatic Control, IEEE Transactions on* 50.2 (Feb. 2005), pp. 169–182. ISSN: 0018-9286. DOI: 10.1109/TAC.2004.841888.

[72] Scott L. Murchie et al. *The Scientific Rationale for Robotic Exploration of Phobos and Deimos*. White Paper. John Hopkins University Applied Physics Laboratory, 2009.

[73] NASA. *Juno - Unlocking Jupiter's Mysteries*. URL: http://www.nasa.gov/mission_pages/juno/main/index.html (visited on 06/21/2013).

[74] NASA. *MESSENGER - Mission to Mercury*. URL: http://www.nasa.gov/mission_pages/messenger/main/index.html (visited on 06/22/2013).

[75] NASA. *Solar System Exploration*. 2012. URL: http://solarsystem.nasa.gov/index.cfm (visited on 06/22/2013).

[76] NASA Goddard Space Flight Center. *Lunar Reconnaissance Orbiter*. URL: http://lro.gsfc.nasa.gov/ (visited on 06/21/2013).

[77] NASA Goddard Space Flight Center. *NASA's HEASARC: Observatories*. May 2010. URL: http://heasarc.gsfc.nasa.gov/docs/heasarc/missions/isee3.html (visited on 06/23/2013).

[78] NASA Goddard Space Flight Center. *Soviet Missions to the Moon*. URL: http://nssdc.gsfc.nasa.gov/planetary/lunar/lunarussr.html (visited on 06/22/2013).

[79] NASA Jet Propulsion Laboratory. *Autonomous Exploration for Gathering Increased Science*. URL: http://aegis.jpl.nasa.gov/ (visited on 06/23/2013).

[80] NASA Jet Propulsion Laboratory. *Mars Science Laboratory Curiosity Rover - Mobility*. 2013. URL: http://mars.jpl.nasa.gov/msl/mission/technology/insituexploration/planetarymobility/ (visited on 06/23/2013).

[81] NASA Jet Propulsion Laboratory. *Stardust: NASA's Comet Sample Return mission*. 2013. URL: http://stardust.jpl.nasa.gov/home/index.html (visited on 06/23/2013).

[82] NASA Jet Propulsion Laboratory. *Voyager - The Interstellar Mission*. URL: http://voyager.jpl.nasa.gov/ (visited on 06/22/2013).

[83] R. Olfati-Saber. "Distributed Kalman filtering for sensor networks". In: *Decision and Control, 2007 46th IEEE Conference on*. Dec. 2007, pp. 5492–5498. DOI: 10.1109/CDC.2007.4434303.

[84] R. Olfati-Saber. "Flocking for multi-agent dynamic systems: algorithms and theory". In: *Automatic Control, IEEE Transactions on* 51.3 (Mar. 2004), pp. 401–420. ISSN: 0018-9286. DOI: 10.1109/TAC.2005.864190.

[85] R. Olfati-Saber, J.A. Fax, and R.M. Murray. "Consensus and Cooperation in Networked Multi-Agent Systems". In: *Proceedings of the IEEE* 95.1 (Jan. 2007), pp. 215–233. ISSN: 0018-9219. DOI: 10.1109/JPROC.2006.887293.

[86] R. Olfati-Saber and P. Jalalkamali. "Coupled Distributed Estimation and Control for Mobile Sensor Networks". In: *Automatic Control, IEEE Transactions on* 57.9 (Sept. 2012).

[87] R. Olfati-Saber and R.M. Murray. "Consensus problems in networks of agents with switching topology and time-delays". In: *Automatic Control, IEEE Transactions on* 49.9 (Sept. 2004), pp. 1520–1533. ISSN: 0018-9286. DOI: 10.1109/TAC.2004.834113.

[88] Reza Olfati-Saber. "Distributed Kalman filter with embedded consensus filters". In: *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*. IEEE. 2005, pp. 8179–8184.

[89] Alexander Olshevsky. "Efficient Information Aggregation Strategies for Distributed Control and Signal Processing". PhD thesis. MIT - Department of Electrical Engineering and Computer Science, Sept. 2010.

[90] Rotem Oshman. "Distributed Computation in Wireless and Dynamic Networks". PhD thesis. MIT - Computer Science and Artificial Intelligence Laboratory, Sept. 2012.

[91] F. Pasqualetti. "Secure Control Systems: A Control-Theoretic Approach to Cyber-Physical Security". PhD thesis. Mechanical Engineering Department, University of California at Santa Barbara, Sept. 2012.

[92] F. Pasqualetti, A. Bicchi, and F. Bullo. "A graph-theoretical characterization of power network vulnerabilities". In: *American Control Conference (ACC) 2011*. San Francisco, CA, USA, June 2011, pp. 3918–3923.

[93] F. Pasqualetti, A. Bicchi, and F. Bullo. "Consensus Computation in Unreliable Networks: A System Theoretic Approach". In: *Automatic Control, IEEE Transactions on* 57.1 (2012), pp. 90–104.

[94] F. Pasqualetti, F. Dorfler, and F. Bullo. "Attack Detection and Identification in Cyber-Physical Systems". In: *Automatic Control, IEEE Transactions on* (Aug. 2012). Submitted.

[95] Marco Pavone et al. "Spacecraft/Rover Hybrids for the Exploration of Small Solar System Bodies". In: *IEEE Proceedings*. Vol. 2425. 2013.

[96] Marshall Pease, Robert Shostak, and Leslie Lamport. "Reaching agreement in the presence of faults". In: *Journal of the ACM (JACM)* 27.2 (1980), pp. 228–234.

[97] Mathew Penrose. *Random Geometric Graphs*. Department of Mathematical Sciences, Durham University: Oxford University Press, 2003. URL: http://www.oxfordscholarship.com/10.1093/acprof:oso/9780198506263.001.0001/acprof-9780198506263.

[98] V. G. Perminov. *The Difficult Road to Mars*. Monographs in Aerospace History 15. NASA History Division, July 1999.

[99] Planetary Society. *Missions to Venus and Mercury*. URL: http://www.planetary.org/explore/space-topics/space-missions/missions-to-venus-mercury.html (visited on 06/22/2013).

[100] Michel Raynal. *Networks and distributed computation: concepts, tools, and algorithms*. MIT Press, 1988.

[101] William A. Reinsch et al. *2011 Report to Congress*. Tech. rep. U.S.-China Economic and Security Review Commission, Nov. 2011.

[102] Wei Ren, R.W. Beard, and E.M. Atkins. "Information consensus in multivehicle cooperative control". In: *Control Systems, IEEE* 27.2 (Apr. 2007), pp. 71–82. ISSN: 1066-033X. DOI: 10.1109/MCS.2007.338264.

[103] Craig W Reynolds. "Flocks, herds and schools: A distributed behavioral model". In: *ACM SIGGRAPH Computer Graphics*. Vol. 21. 4. ACM. 1987, pp. 25–34.

[104] L. Sabattini, N. Chopra, and C. Secchi. "Distributed control of multirobot systems with global connectivity maintenance". In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. Sept. 2011, pp. 2321–2326. DOI: 10.1109/IROS.2011.6094818.

[105] Nicola Santoro. "On the message complexity of distributed problems". English. In: *International Journal of Computer & Information Sciences* 13.3 (1984), pp. 131–147. ISSN: 0091-7036. DOI: 10.1007/BF00979869. URL: http://dx.doi.org/10.1007/BF00979869.

[106] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2009.

[107] Demetri P Spanos, Reza Olfati-Saber, and Richard M Murray. "Distributed sensor fusion using dynamic consensus". In: *IFAC World Congress*. 2005.

[108] D.P. Spanos, R. Olfati-Saber, and R.M. Murray. "Dynamic consensus on mobile networks". In: *The 16th IFAC World Congress, Prague, Czech*. 2005.

[109] Peter Thomas. *Optical shape models of 9 planetary moons and asteroids, derived from spacecraft imaging*. June 2000. URL: http://sbn.psi.edu/pds/resource/oshape.html (visited on 05/16/2012).

[110] Claire Tomlin, George J Pappas, and Shankar Sastry. "Conflict resolution for air traffic management: A study in multiagent hybrid systems". In: *Automatic Control, IEEE Transactions on* 43.4 (1998), pp. 509–521.

[111] P. Tricarico and M.V. Sykes. "The dynamical environment of Dawn at Vesta". In: *Planetary and Space Science* 58.12 (2010), pp. 1516–1525. ISSN: 0032-0633. DOI: http://dx.doi.org/10.1016/j.pss.2010.07.017. URL: http://www.sciencedirect.com/science/article/pii/S0032063310002199.

[112] W.F. Truszkowski et al. "Autonomous and autonomic systems: a paradigm for future space exploration missions". In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 36.3 (2006), pp. 279–291. ISSN: 1094-6977. DOI: 10.1109/TSMCC.2006.871600.

[113] Kleomeris Tsiganis et al. "Origin of the orbital architecture of the giant planets of the Solar System". In: *Nature* 435.7041 (2005), pp. 459–461.

[114] John N Tsitsiklis. "Problems in decentralized decision making and computation". PhD thesis. Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science., 1985. URL: http://hdl.handle.net/1721.1/15254.

[115] John Tsitsiklis, Dimitri Bertsekas, and Michael Athans. "Distributed asynchronous deterministic and stochastic gradient optimization algorithms". In: *Automatic Control, IEEE Transactions on* 31.9 (1986), pp. 803–812.

[116] George Tsoulos. "Adaptive Antennas and MIMO Systems for Mobile Communications". In: *Adaptive Antenna Array*. Ed. by Sathish Chandran. Signals and communication technology. Springer Berlin Heidelberg, 2004, pp. 3–26. ISBN: 978-3-642-05775-5. DOI: 10.1007/978-3-662-05592-2_1. URL: http://dx.doi.org/10.1007/978-3-662-05592-2_1.

[117] Tamás Vicsek et al. "Novel Type of Phase Transition in a System of Self-Driven Particles". In: *Phys. Rev. Lett.* 75 (6 Aug. 1995), pp. 1226–1229. DOI: 10.1103/PhysRevLett.75.1226. URL: http://link.aps.org/doi/10.1103/PhysRevLett.75.1226.

[118] Richard Volpe. "Rover functional autonomy development for the mars mobile science laboratory". In: *Proceedings of the 2003 IEEE Aerospace Conference*. Vol. 2. 2003, pp. 643–652.

[119] Richard Volpe et al. "The CLARAty architecture for robotic autonomy". In: *Aerospace Conference, 2001, IEEE Proceedings.* Vol. 1. IEEE. 2001, pp. 1–121.

[120] R.E. Wallis and Sheng Cheng. "Phased-array antenna system for the MESSENGER deep space mission". In: *Aerospace Conference, 2001, IEEE Proceedings.* Vol. 1. 2001, 1/41–1/49 vol.1. DOI: 10.1109/AERO.2001.931694.

[121] Mathijs de Weerdt and Brad Clement. "Introduction to planning in multiagent systems". In: *Multiagent and Grid Systems* 5.4 (2009), pp. 345–355.

[122] Michael Wooldridge. *An introduction to multiagent systems*. Wiley, 2008.

[123] M.M. Zavlanos et al. "Hybrid control for connectivity preserving flocking". In: *Automatic Control, IEEE Transactions on* 54.12 (2009), pp. 2869–2875.

[124] Shlomo Zilberstein et al. "Decision-Theoretic Control of Planetary Rovers". In: *Advances in Plan-Based Control of Robotic Agents*. Ed. by Michael Beetz et al. Vol. 2466. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 270–289. ISBN: 978-3-540-00168-3. DOI: 10.1007/3-540-37724-7_16. URL: http://dx.doi.org/10.1007/3-540-37724-7_16.

[125] Maria T. Zuber et al. "Gravity Recovery and Interior Laboratory (GRAIL): Mapping the Lunar Interior from Crust to Core". In: *Space Science Reviews* (2013), pp. 1–22. ISSN: 0038-6308. DOI: 10.1007/s11214-012-9952-7. URL: http://dx.doi.org/10.1007/s11214-012-9952-7.