

École polytechnique de Louvain

Data Fusion in Open Multi-Agent Systems for Decentralized Estimation

Author: **Sébastien COLLA**

Supervisor: **Julien HENDRICKX**

Readers: **Jean-Charles DELVENNE, Charles MONNOYER DE GAL-
LAND DE CARNIÈRES**

Academic year 2019–2020

Master [120] in Mathematical Engineering

Abstract

This master thesis designs new efficient multi-agent algorithms that operate in open systems (i.e. systems subject to arrivals and departures of agents) and that aggregate the data held by each arriving agent for allowing them to estimate some external quantities in a decentralized way.

Multi-Agent Systems are systems composed of independent, intelligent and interactive entities acting toward some common objective. Recognized for their advantages in solving large problems in a decentralized manner, their study generally considers a fixed composition of agents. However, arrivals and departures of agents may be unavoidable (such as computer failures in a network) and may impact the objective pursued by the agents. In that *Open Multi-Agent Systems* context, the agents may need to aggregate information from all the agents that have ever been in the system for estimating some external quantities but without using a growing memory. This is the case, for example, when each agent holds a measurement of a same random phenomenon for which each wants to estimate the distribution mean in a decentralized way. This latter problem is the focus of this master thesis. Its main challenge is to incorporate correctly the information from new agents without forgetting the information of the agents having left the system and without being too much impacted by noise.

This master thesis establishes theoretical and empirical performance limitations for this problem in *Open Multi-Agent Systems* and presents new algorithms that solve it efficiently. The best designed algorithm reaches the performance limitations (within 1%) in most cases.

Key Words: Multi-Agent Systems, Open Systems, Data Fusion, Decentralized Estimation, Gossip, Push-Sum.

Acknowledgments

First of all, I would like to thank my advisor Julien Hendrickx for his precious support throughout the realization of my master thesis. He always made himself available to answer my questions, providing useful advice, and giving me many ideas. He motivated me to always improve and further develop my work by his interest in the topic and his involvement in guiding me to produce this final rendering of my work.

I would also express my gratitude to Charles Monnoyer who works as a researcher with Julien Hendrickx on a related topic. He helped me and reassured me a lot by responding to any of my questions and providing useful tips. He has attended many of our meetings and has reflected on some problems with us. I am also grateful to him for his reading of my works and his helpful comments.

I would like to thank Carole for the proofreading of my work.

Finally, I am grateful to Alexia for her constant understanding, help and support during the writing of this master thesis. Our discussions about my work often helped me to improve my own understanding of the subject.

Contents

Abstract	i
Acknowledgments	iii
Introduction	1
State-of-the-art	5
1 The Max-consensus problem [AHH17]	5
2 Averaging consensus problem	6
3 Other works about open multi-agent systems	8
Contribution and overview	9
1 Problem statement and tools	11
1.1 Problem definition and general assumptions	11
1.1.1 Synthetic problem statement: the external averaging problem	11
1.1.2 The system, the agents and their goal	12
1.1.3 Performance evaluation	13
1.1.4 System events: the communications and the replacements	14
1.1.5 Problem statement limitations	16
1.2 The gossip averaging algorithm: a baseline algorithm	16
1.3 Simulation tool	17
2 Performance Limitations	21
2.1 Ideal algorithm	21
2.2 Upper bound on $\mathbb{E}_\epsilon[\mathcal{K}_i(t)]$	26
2.2.1 Time-dependent approach using a simplified restart definition	27
2.2.2 Global approach using the general restart definition	31
2.3 Comparison of the different lower bounds on performance	34
Appendices	
2.A Appendix for theoretical expressions in the performance lower bounds	37
3 Gossip averaging algorithm	39
3.1 Analysis of the gossip averaging algorithm	39
3.1.1 Descriptor evolution system	40
3.1.2 Steady state expected performance and convergence rate	41
3.2 Main issues of the gossip algorithm for the external averaging problem	42
3.3 Comparison between the gossip averaging algorithm and the best ideal algorithm	45
4 Symmetric Push-Sum algorithm	49
4.1 Algorithm description and framework	49
4.2 Setting the initial weight of the incoming agents	51
4.2.1 Convergence assumption	51
4.2.2 Optimal initial weights	55
4.3 Performance analysis	61
4.3.1 Time evolution of the expected performance	61

4.3.2	Expected performance after 500 seconds	63
4.4	Theoretical evolution of the system	65
4.5	Implementable version of the algorithm	72
5	Gossip averaging with difference tracking	77
5.1	Algorithm description	77
5.2	Theoretical evolution of the system	82
5.2.1	Descriptor evolution system	82
5.2.2	Validation	84
5.2.3	Steady state expected performance and convergence rate	84
5.3	Tuning of the incorporation rate β	89
5.3.1	Fixed incorporation rate β	89
5.3.2	Decreasing incorporation rate β	91
5.4	Relaxation of Assumption 5.1 (arrival with communication)	93
5.5	Performance analysis	96
5.5.1	Time evolution of the expected performance	96
5.5.2	Expected performance after 500 seconds	96
Appendices		
5.A	Appendix for the proof of Lemma 5.1	99
5.B	Appendix for the proof of Lemma 5.2	101
6	Symmetric Push-Sum algorithm with difference tracking	105
6.1	Algorithm description	105
6.2	Performance analysis	107
6.2.1	Time evolution of the expected performance	107
6.2.2	Expected performance after 500 seconds	109
7	Weighted gossip averaging algorithm using the knowledge sets	113
7.1	Assumption and motivation	113
7.2	Impossibility of recovering the best estimates using a weighted gossip with knowledge sets	114
7.3	Algorithm using the knowledge sets and the knowledge weights	115
7.3.1	Minimizing the resulting variance	115
7.3.2	Perspectives	119
Conclusion		121
Bibliography		127

Introduction

These days, people, computers and every thing are more connected to each other than ever. Artificial intelligence is more and more present in our lives through smartphones, cars, internet, industries or other. It is then logically that the idea of combining individual intelligent entities to collaborate and solve problems has become popular in the past decades. The distributed artificial intelligence already started as a branch of artificial intelligence forty years ago. It quickly became an independent research discipline in its own right, representing a combination of ideas from several disciplines [A02]. Multi-agent systems are one of the main examples of distributed intelligence [Woo09].

Think for example of a system of robots that need to communicate with each other in order to play football (potentially against another team) as depicted in Figure 1. Robots should not be controlled by one centralized entity but can each take their own decisions on the field. Therefore, in order to play correctly, these soccer robots need to reach an agreement about the strategy to adopt, control the formation they have on the field, synchronize them between each other in order to execute passes correctly and ideally also observe the moves of the opposing team to collectively estimate what they are going to do and try to optimize their own strategy from that. This robot system example combines most of classical problems that are studied in multi-agents systems (consensus, decentralized control, decentralized optimization,...) and that will be presented in this work.

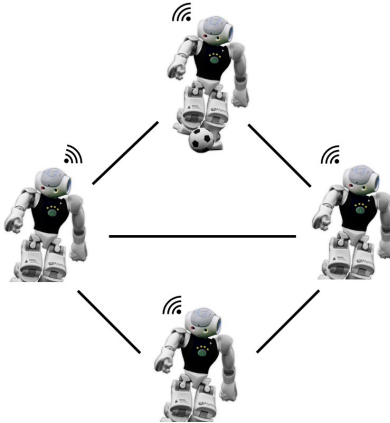


Figure 1: An example of multi-agent system: the soccer robots system

More formally, a multi-agent system (MAS) is a system that contains several autonomous entities (called agents) that can take their own decisions and actions, and that interact with each other in order to achieve a common objective, that is often beyond the individual capabilities or knowledge of each entity. The amusing example of soccer multi-robots system exists for real [DCX12] and competitions are even organized. Other (more useful) examples of systems of robots that have to achieve specific tasks requiring coordination and collaboration [Ota06] exist or can be imagined. As explained in [DBB14], the applications of swarm of robots are promising when the tasks they are carrying are dangerous, require different activities to be performed concurrently or when they are redundant (e.g. mine clearance or search and rescue). Other situations, where it is technically impossible to set up the necessary infrastructure to control the robots in a centralized way or where the absence of a single point of failure is desired, can also be handled by swarm of robots, such as for planetary or underwater exploration.

Multi-robots systems are not the only examples of multi-agent systems. Another example of multi-agent system is the sensor network where the agents try to collectively estimate a parameter or a function from the data they collect. Multi-agent approaches for this estimation problem are often more efficient (in terms of energy and communication) than centralized estimation schemes [RN04]. One other well-known example of multi-agent systems is the ad-hoc network which is a network that does not rely on any pre-existing infrastructure such as routers and where devices can communicate directly with each other without intermediary routers. It can be used for examples with cars [CHW11]. Multi-agent systems can also model social networks, opinion dynamics [VMH18] [Van+16] [Ye+18b] [Ye+18a] and animals behaviors (as flocks of birds) [Leo13]. This short list of examples shows that there are various fields of application for multi-agent systems. Different types of applications and their challenges are detailed in [DKJ18] and [Cao+13] presents an overview of the classes of problems that can be solved using multi-agent systems and their respective applications.

As shown in Figure 2, a multi-agent system is generally represented by a graph where the set of nodes $\mathcal{N}$ is the set of agents and the presence of an edge between two agents depicts the existence of a communication channel between the corresponding agents. This abstraction allows using graph theory results in the analysis of multi-agent systems.

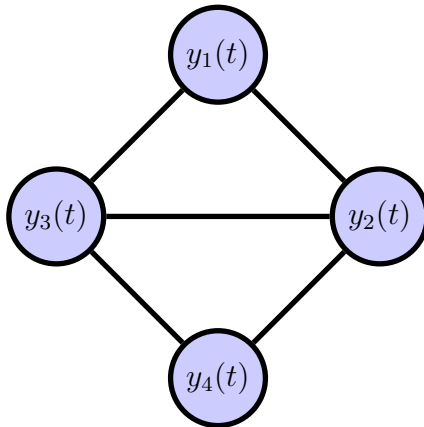


Figure 2: Representation of the multi-agent system of soccer robots from Figure 1. Each agent (robot) holds a time-varying value $y_i(t)$ that represents its own approximation of the solution to a given problem.

Problem solving with multi-agent systems is not an easy task: since each agent is an independent entity that often evolves into an environment subject to some randomness, predictions and analyses of such systems may be difficult and often rely on stochastic approaches to estimate certain expected behaviors. This challenge is worth the effort because in return, multi-agent systems provide many advantages related to the decentralized and collaborative resolution of a problem.

Firstly, multi-agent systems do not work with a centralized master node that collects and manages all the information contained in the system. It is said that multi-agent systems work in a decentralized manner. This approach can be useful to solve difficult problems that a single agent could not be able to solve or problems where data cannot be stored on a single agent (computer), either for privacy or memory reasons. In other situations, the communication constraints do not allow all agents to communicate with a same central entity, or only intermittently (due to cost of communication, distance between nodes or infrastructure constraints). In these situations, the decentralized approach is essential. In some systems where decisions need to be taken in real time, without latency, as for the soccer robots systems, a central entity often constitutes a bottleneck in the execution speed of the decisions since it has to manage many communications and computations on its own. Moreover, in decentralized algorithms, the nodes do not need to know the entire structure of the system, which is very desirable for large systems.

Furthermore, multi-agent systems are easily scalable: agents can cooperate just as well in small or large systems without large modifications, which is not the case of centralized solutions for which the

master node often constitutes a bottleneck when the system needs to be scaled up.

The absence of centralized master node may also allow the system to be in a sense more robust to failures. Indeed, when a node fails in multi-agent systems, it has undesirable effects on the balance of the system as it will be explained later but a single failure is rarely fatal to the system. Even if this worst case scenario happens, each remaining agent can still access to its own approximation of the solution and the entire process can restart without the broken agent. In centralized settings, the solution is only computed at the central master node and can potentially be sent to others thereafter but if the central node fails, all the system fails without having any access. Also since all agents of a multi-agent system are identical, a defective agent can be replaced pretty easily in comparison with a centralized node for which a replacement may be difficult and expensive.

As shown in representation of Figure 2, in multi-agent system, each agent $i \in \mathcal{N}$ holds its own approximation of the solution¹ that evolves with time. The solutions of all agents are stacked in a vector $y(t)$, and under some assumption on the interactions network, it converges to a final desirable vector y^* when time goes to infinity.

For example, in consensus problem, the agents aim to achieve a consensus on a common value, starting each with a different initial value. Therefore, as shown in Figure 3, the agents start with different initial values $y_i(0)$ and (pairwise) communications allow the different $y_i(t)$ values to converge toward a common one. Consequently, the final vector y^* have identical components ($y^* \in \text{span}\{\mathbf{1}\}$).

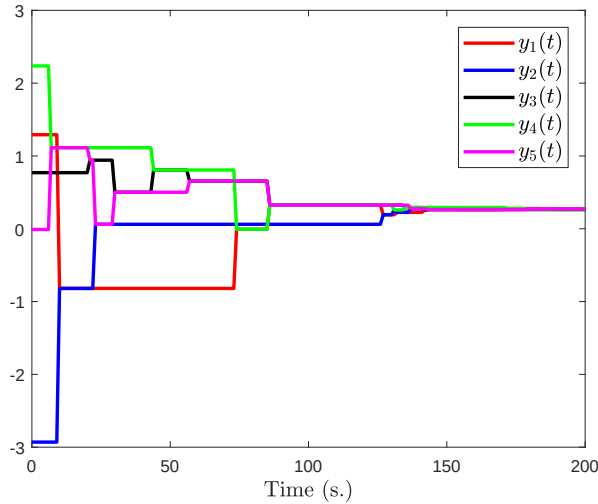


Figure 3: Example of averaging consensus with 5 agents using the gossip averaging algorithm (presented in the state-of-the-art and in Chapter 3). Each communication involves two (random) agents that average and update their own y_i value.

In decentralized optimization, the final solution vector of the system is $y^* = \text{argmin} \sum_i f_i(y)$ where f_i are local objective functions available only at node i . Alternatively, convergence can also happen to a desirable set or trajectory $y^*(t)$ as it is often the case in decentralized control.

In all these scenarios (consensus, decentralized optimization, decentralized control,...), the agents communicate through a (potentially) time-varying network, meaning that new communication channels can appear and others can disappear, but the system has a fixed composition, meaning that the same agents are present from the beginning of the process to the end. However, the reality is often different since in many applications, we observe repeated arrivals and departures of agents: people join and leave social network, nodes in communication ad-hoc network get connected and disconnected, sensor or robots are replaced and may fail... That is why *open multi-agent systems* are introduced. In these open systems, the arrivals and departures have time-scales similar to those of the studied phenomenon, and therefore

¹Depending on the problem, $y_i(t)$ could be scalars or vectors. For simplicity of notation, we assume that $y_i(t) \in \mathbb{R}$.

there may be a new arrival or departure before the effect of the last one has been fully incorporated. For our soccer robots system, it means that the robots may fail and be replaced during the course of a match and not only between two games. A broken robot leaves with some information that the system may miss or on the contrary, remaining robots may continue to be influenced by some outdated information coming from this broken gone robot. A new robot enters into the system without any knowledge about it and brings its personal information to the system. The situation is represented in Figure 4.

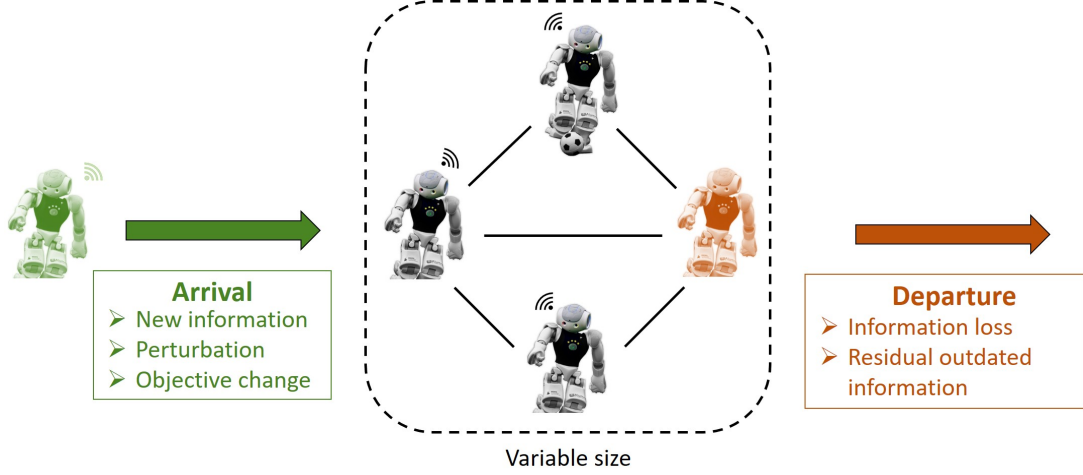


Figure 4: Representation of an Open Multi-Agent System depicting a soccer robots system, with the implications of an agent entering or leaving the system.

In general, arrivals bring new information that may perturb the system and also change the objective pursued by the agents. Similarly, departures imply a loss of information and possibly a break of the balance of the system. In some situations, the information left in the system by previous agents may become outdated and influence the system into the wrong direction. This is the case for the consensus problems about a value which is intrinsic to the system (i.e. that depends only on the agents currently in it). This kind of problems is detailed below in the state-of-the-art.

Repeated arrivals and departures lead to significant differences in the analysis and design of open multi-agent systems and pose several challenges. First the arrival and departure events imply a change of the system size that makes the analysis of the evolution of the system more challenging. Secondly, continuous perturbations due to departures and arrivals lead open systems to never asymptotically converge to a specific state as it was the case in closed system (see Figure 3). The *open* systems may rather approach some form of steady state expected behavior as explained in [HM17] and in this master thesis. Therefore, many algorithms designed for *closed* multi-agents systems are not working correctly anymore in an *open* situation. Design of algorithms for open multi-agent systems that are robust to repeated arrivals and departures are nonetheless important to extend the applications of multi-agent systems in cases where frequent arrivals and departures are inherent or beneficial, and to make systems more robust to failures. For example, it seems better (practically and economically) to have basic agents (e.g. robots or servers) that may often fail but with algorithms that can manage these failures correctly than very sophisticated and expensive agents (e.g. robots or servers) that (almost) never fail.

This master thesis aims to develop efficient algorithms to solve a specific problem in this context of open multi-agent systems and to analyze the potential limitations due to arrivals and departures. Before describing the problem in question and the overview of this master thesis, the state-of-the-art in open multi-agent system is briefly reviewed.

State-of-the-art

We describe here different typical problems in multi-agent systems, their applications, the existing algorithms that solve these problems and their adaptation in the context of open multi-agent systems. The focus is set on the main challenges that arise due to arrivals and departures of agents and the recent emerging solutions. Many multi-agent problems have not been considered yet in an open multi-agent systems context and are therefore not presented here.

In a multi-agent system, a problem can be described as an objective that each agent aims to achieve. There may be different algorithms that solve the same problem, i.e. fulfills the same objective. Two main problems (objectives) are detailed in this state-of-the-art: the max-consensus problem and the averaging consensus problem. For both of them, the problem statement and some existing algorithms that solve the problem in *closed* systems are first presented. In *open* system, the statement of these problems need to be clarified about the agents that have left the system: « are they still part of the objective? » Depending on the answer to this question, each problem (max-consensus and averaging consensus) can be decomposed into two distinct problems for which closed system algorithms and their potential adaptations are discussed.

For the following, the agents are supposed to have constant bounded memory, identical algorithms and no identifier they can use. $\mathcal{N}(t)$ denotes the set of agents in the system at time t ($\mathcal{N}$ can be used for closed systems), $n(t) = |\mathcal{N}(t)|$ is the size of the system and z_i is the initial value held by agent i . The interactions are supposed to be asynchronous, pairwise, random and symmetric which means briefly that a communication involves only a random pair of agents at once. We suppose that all pair of agents $i, j \in \mathcal{N}(t)$ can communicate with each other (i.e. the communication graph is complete), even if more complex communication schemes can be considered [TJ08] [HM05].

1 The Max-consensus problem [AHH17]

1.1 In closed system

In closed system, the goal of each agent $i \in \mathcal{N}$ for the max-consensus problem is to find the maximum of all the initial values of the agents: $y_i(t) \rightarrow \max_{k \in \mathcal{N}} z_k$. It can be used for leader election (which is useful for self-organization, e.g. by setting the speed of a group of robots to the speed of the slowest one), in maximum likelihood for estimating system properties [Luc+15] or it can be part of another decentralized algorithm. A simple algorithm that solves this problem floods the system with the maximum value:

Max-consensus algorithm: (1)

Initialization for all $i \in \mathcal{N}$:	$y_i(0) = z_i,$
Interaction between a pair of agents $i, j \in \mathcal{N}$:	$y_i^+ = y_j^+ = \max(y_i, y_j).$

By applying this algorithm in closed system, agents eventually converge to the maximum of their initial values since at each communication between two agents, they only keep the maximum of their y values. The maximum initial value among all agents in the system ($\max_{k \in \mathcal{N}} z_k$) is always kept and is progressively spreading through the system with communications.

1.2 In open systems

In open systems, the problem splits into intrinsic and external cases.

- The intrinsic objective is related to the agents currently present in the system, i.e. $y_i(t) = \max \{z_k \mid k \in \mathcal{N}(t)\}$. It is used in coordination and self-organization. In this situation the information of leaving agents should be removed from the system. For this objective, the algorithm (1) may keep as maximum estimate the value of an agent that has left the system. Therefore, it should be adapted to erase outdated information.
- The external objective is related to all agents having ever been in the system at some time (until current time t), i.e $y_i(t) = \max \{z_k \mid k \in \cup_{s \leq t} \mathcal{N}(s)\}$. This is used, for example, to estimate the external value of which z_k are measures. The algorithm (1) works well for this external objective.

To be able to erase outdated information and achieve the intrinsic objective, two different solutions that adapt the algorithm (1) have been studied in [AHH17].

The first solution is to introduce information age and to erase information for which the age exceeds a threshold. The choice of the threshold is crucial: if it is too low, it leads to spurious reset and if it is too high, it leads to slow reaction. The optimal threshold depends on the size of the system. A theorem states that, with a proper threshold, if arrivals and departures stop, there is a time $T(\epsilon)$ after which agents hold correct solution with a probability $1 - \epsilon$ (for any ϵ).

The second solution allows agents to send a last *goodbye* message before leaving the system. The adaptation is not so easy but relies on auxiliary variables describing *trust level*. This solution reacts quickly to leaving agents. It can be proved that if arrivals and departures stop, the algorithm converges to the correct solution with probability 1.

Both solutions guarantee asymptotic correctness but present a transient disruption during which there is no performance guarantee, and therefore performance may be very poor if the arrivals and departures are frequent. A challenge for the future is to design algorithms that give good performance when agents keep arriving and leaving without necessarily having to give the exact solution if the system becomes closed. It should be a trade-off between disruption magnitude and speed of re-convergence.

2 Averaging consensus problem

2.1 In closed systems

In closed system, the averaging consensus problem is to ensure that each agent converges to a common value that is the average of the initial values of the agents: $y_i(t) \rightarrow \frac{1}{n} \sum_{i \in \mathcal{N}} z_i$. This problem is motivated by the importance of the average in many problems: coordination, synchronization, statistics, decentralized optimization [NOR18], decentralized control and others (see [WBA05] for a survey). Different consensus algorithms have been designed to solve this problem in closed system. Two of them are presented below: the gossip averaging algorithm and the push-sum algorithm.

The *gossip averaging algorithm* [Boy+06] is one of the simplest consensus algorithms that converges to the average of initial values of the agents. Each agent $i \in \mathcal{N}$ holds a value x_i initialized to its initial measurement z_i . In the gossip algorithm, agents perform pairwise and symmetric interactions to update their x_i values via pairwise averaging. During this algorithm, the estimate of the average is given by $y_i(t) = x_i(t)$ (for each agent i), and hence both notations x and y can be used indifferently in the context of the gossip algorithm (which is not the case for the push-sum algorithm described below). The algorithm can be summarized as follows:

Gossip averaging algorithm:

Initialization for all $i \in \mathcal{N}$:	$x_i(0) = z_i,$
Interaction between a pair of agents $i, j \in \mathcal{N}$:	$x_i^+ = x_j^+ = \frac{x_i + x_j}{2},$
Average estimate of agent $i \in \mathcal{N}$:	$y_i(t) = x_i(t).$

After each communication, agents get *closer* to an agreement and the sum of x_i values is preserved: $\sum_{i \in \mathcal{N}(t)} x_i^+ = \sum_i x_i = \sum_i z_i$. Hence, when consensus occurs, the agreement point corresponds to the average of initial values, as shown in Figure 3. It has been proved that the algorithm converges (exponentially) to the average of the initial values with probability 1. Indeed, the expected variance of the estimate vector x , that measures the disagreement between agents in the system, decreases exponentially: $\mathbb{E}[\text{Var}(x(t))] = (1 - \frac{1}{n})^t \mathbb{E}[\text{Var}(x(0))]$.

The *push-sum algorithm* is another algorithm that solves the averaging consensus problem. It has been designed to handle asymmetric interactions in closed multi-agent systems and has been studied for example in [KDG03] or [GH19]. The latter paper also studies the effect of transmission failures on the algorithm. A communication is said to be asymmetric if an agent $i \in \mathcal{N}$ sends information to $j \in \mathcal{N}$ without receiving anything from j in return. If the gossip averaging algorithm is performed with asymmetric communications ($x_j^+ = \frac{x_i + x_j}{2}$ but $x_i^+ = x_i$), the sum of the x values is no more preserved and the algorithm is not guaranteed to converge to the exact average. The idea of the push-sum algorithm is that each agent records (through interactions) a linear combination of agents values (as for the gossip

algorithm) but it also keeps track of its relative importance in the system. Therefore, two variables are required for each agent i : a value $x_i(t)$ and a weight $w_i(t)$. The algorithm is the following:

Push-sum algorithm: (2)

$$\begin{aligned}
&\text{Initialization for all } i \in \mathcal{N}: & x_i(0) = z_i & \quad \text{and} \quad w_i(0) = 1. \\
&\text{Interaction from agent } i \in \mathcal{N} \text{ to } j \in \mathcal{N}: & x_i^+ = \frac{x_i}{2} & \quad \text{and} \quad x_j^+ = \frac{x_i + x_j}{2}, \\
& & w_i^+ = \frac{w_i}{2} & \quad \text{and} \quad w_j^+ = \frac{w_i + w_j}{2}. \\
&\text{Average estimate of agent } i \in \mathcal{N}: & & y_i(t) = \frac{x_i(t)}{w_i(t)}.
\end{aligned}$$

We observe that the sum of x_i and the sum of w_i are both preserved. In closed multi-agent system, the estimates of the agents $y_i(t)$ are guaranteed to converge exponentially to the average of the agents initial values: $\lim_{t \rightarrow \infty} y_i(t) = \frac{1}{n} \sum_{i \in \mathcal{N}} z_i$ for all i with probability 1. According to [GH19], when transmission failures are possible (i.e. some messages can be lost), the agents estimates $y_i(t)$ still converge exponentially to a common value with probability 1, but this might not be the exact average of the initial measurements.

2.2 In open systems

In open system, the averaging problem also splits into intrinsic and external averaging problem. [HM16] and [HM17] have studied the behavior of an *open* multi-agent system subject to the averaging gossip algorithm, independently of the objective for which the algorithm is used (intrinsic or external). Since arrivals and departures systematically change the size of the system (and then the dimension of vector $x(t)$), the analysis follows size-independent descriptors (such as the expected (squared) mean over all the agent in the system). This analysis of the gossip algorithm behavior in open multi-agent systems will be largely used in this master thesis and is therefore presented in a dedicated section (see Section 3.1).

The algorithms and the performance limitations of the intrinsic averaging problems are discussed in the following. Thereafter, the external averaging problem is briefly presented. This has not been studied yet in the literature and is the subject of this master thesis.

Intrinsic averaging problem aims to compute (or estimate) the average of the initial values of the agents present in the system at time t : $y_i(t) \rightarrow \frac{1}{n(t)} \sum_{k \in \mathcal{N}(t)} z_k$. One might wonder if the averaging gossip algorithm works to solve this problem. To converge to the average of the system, the sum of estimates x_i (recall that $y_i(t) = x_i(t)$ for the gossip algorithm) should be preserved, i.e. be equal to the sum of the initial values z_i of the agent in the system: $\sum_{i \in \mathcal{N}(t)} x_i = \sum_{i \in \mathcal{N}(t)} z_i$. This property remains satisfied after gossip interactions and arrivals of new agents (since the x value of new agent $n+1$ is initialized to $x_{n+1} = z_{n+1}$) but not for departures since information (i.e. a part of different z_i values) goes out of the system. As it has been studied by Baptiste Brasseur in its Master Thesis [Bra18], a last message strategy can be implemented to try to correct this apparent issue with the gossip algorithm. Assume agent d leaves the system and sends a last message to r before leaving. Then if r updates its estimate with: $x_r^+ = x_r + (x_d - z_d)$, the sum of x_i would be equal to the sum of z_i after the departure of agent d . This strategy improves the performance of the system when arrivals and departures stop, since it allows converging to the exact average, but surprisingly it achieves worst results as long as the system remains open (comparing to the uncorrected gossip strategy). Hence, this correction is counterproductive for the systems where convergence is not always possible between departures (i.e. for large systems or for systems subject to frequent departures).

It is obvious that there exists fundamental limitations on the performance that can be achieved by any algorithm for this problem since the system cannot adapt instantaneously to arrivals and departures. These limitations are studied in [Mon18] and [MH20]. A lower bound on the expected mean squared error of the estimates x_i can be derived by considering an ideal (and optimal) algorithm for which each agent i has access to improved knowledge: the communication model and all the information it has gathered through all interactions (direct or indirect) with other agents up to time t . The performance of this optimal algorithm can be analyzed using infection models which allow to model the spread of

information. This approach gives a lower bound on the performance of any algorithm for which nodes can access to any improved knowledge (distribution of agent value, departure and arrival rate,...). These limitations are useful to see the area of possible improvement for this problem.

External averaging problem can be used to perform decentralized estimation which consists in estimating a value defined *outside* of the system, and that is thus independent of the system state. Supposing that the initial values of the agents z_i (for $i \in \mathcal{N}(t)$) are random samples independently and identically drawn from an unknown distribution of mean μ and variance σ^2 , the purpose of agents is to find an estimate of the distribution mean μ : $y_i(t) \rightarrow \mu$ for all $i \in \mathcal{N}(t)$. Given the measurements of μ of all the agents that have ever been in the system, the best linear unbiased estimator of μ is the external average of the system, i.e. the average of the measurement z_i of all the agents that have ever been in the system up to time t :

$$y_i(t) \rightarrow \frac{1}{|\cup_{s \leq t} \mathcal{N}(s)|} \sum_{k \in \cup_{s \leq t} \mathcal{N}(s)} z_k.$$

Computing (or estimating) it by aggregating the data held by each agent that has ever been in the system therefore allows to estimate efficiently the distribution mean μ .

The gossip averaging algorithm can be used to try to solve this problem but as we will see in Chapter 3, it does not perform very well, especially for small systems which forget more quickly the information of the former agents.

This problem has not been addressed yet in the literature and is the focus of this master thesis. The contributions and the overview of this document are presented right after this state-of-the-art section.

3 Other works about open multi-agent systems

Some other problems, as the *dynamic* average consensus algorithm [Kia+19], have already been studied for open multi-agent systems but are less relevant for the problem addressed in this master thesis since they formalize ideas from a control theory perspective that we do not describe here. The dynamic average consensus problem is explained in detail in [Kia+19] for closed system and is studied for open multi-agent system (in its intrinsic version) in [FF18]. The dynamic average consensus problem is an extension of the classical averaging consensus problem presented above, where the reference (initial) value of each agent i is a time-varying signal (often denoted $u_i(t)$ instead of z_i since it is mainly used for control applications). Let $u_i(t)$ be the time-varying reference signal of each agent $i \in \mathcal{N}(t)$. In dynamic average consensus problem, agents aim to track the average of the reference signal of all agents of the system:

$$u_{\text{avg}}(t) = \frac{1}{n} \sum_{i \in \mathcal{N}(t)} u_i(t).$$

This problem has various applications in decentralized control as explained in [Kia+19]. Motivation to study the problem in open system in [FF18] is mainly modern power grids, where intelligent devices with communication and computing capabilities can arrive or depart at any time. In these applications, the devices aim to estimate in a distributed way the current and future power consumption of the network to which they belong: this information is then used locally to optimize the energy consumption of the devices (see [FF18] for details and references).

[SSF19] studies another dynamic consensus in open multi-agent system, about the median of agents reference signals. As said at the beginning of this state-of-the-art section, many others problems have never been addressed yet in an open multi-agent systems context.

Contribution and overview

This master thesis analyzes the external averaging problem in open multi-agent system, briefly presented above, in order to estimate the distribution mean μ of the measurements held by each agent. The main contributions of this work are the development of performance limitations and of new efficient algorithms for solving this problem. The best designed algorithm reaches the performance limitations (within 1%) in most cases.

We restrict to the case where only *replacements* occur in the system, meaning that each departure of agent is instantaneously followed by the arrival of a new agent in the system. This restriction allows the system to keep a constant size. This allows us to focus on the main other challenges in solving this problem in open systems: (i) to incorporate information from new agents without having the estimates affected too much by noise, (ii) to make sure to exploit information from agents before they leave, (iii) to avoid forgetting information from agents having left in the past, and (iv) to avoid any error accumulation resulting from imbalances created by agent departures.

Chapter 1 defines properly the problem, its assumptions and the metric used to evaluate the quality of an algorithm which is the expected mean squared error. Chapter 2 studies the performance limitations for this problem by considering an ideal algorithm. The theoretical expected mean squared error of this algorithm has been lower bounded using two different ways. This lower bounds are not tight and the performance of this best ideal algorithm is rather computed empirically in order to have a comparison bound for our new designed algorithms.

Chapter 3 presents in detail another comparison point which is the gossip averaging algorithm that has already been presented in the state-of-the-art. The analysis of an open multi-agent system subject to the gossip averaging algorithm pursued in [HM17] is presented and used to deduce theoretical performance of the algorithm. The gossip averaging will serve as baseline to beat for new algorithms. The chapter also presents the two main issues of this algorithm for solving the external averaging problem in open system. The first issue is that too much importance is attributed to the new agents that arrive in the system, resulting in a progressive forgetting of the former agents. The second issue is related to the disruption caused by the arrivals of new agents which makes it impossible for the algorithm to perform well in many situations.

Chapter 4 presents an algorithm that aims to resolve the first issue of the gossip algorithm. It is a (symmetric) push-sum based algorithm that allows giving a weight to each agent. Decreasing correctly the initial weight of new incoming agents allows the algorithm to perfectly solve this first issue of the gossip and to reach the performance limitation in many cases. Reducing progressively the importance of the new agents also allows reducing their disruption on the system, which partially solves the second issue of the gossip averaging algorithm. The performance of this new algorithm is analyzed in detailed through simulations results. Some theoretical descriptions are also explored, without leading to performance guarantees.

To be implementable in practice, the symmetric push-sum algorithm requires the agents to know (or to estimate) the size of the system and the frequency of the replacements in the system in comparison with the communication.

Chapter 5 designs a new gossip-based algorithm that aims to reduce the perturbation of the estimates of the agents due to the arrival of a new agent in the system. This allows solving the second issue of the gossip averaging algorithm. The idea of this algorithm relies on the *difference tracking* in which the new agents, in order not to disturb the system abruptly, pretend to have the value of another agent already in the system and retain the *difference* with their true value to add it progressively into the system and spread out the arrival of new information. The chapter analyzes theoretically this gossip averaging with difference tracking, under the assumption that an agent directly communicates when it arrives into the system. This theoretical analysis gives some guarantees about the performance of the algorithm under this assumption and allows tuning correctly the rate at which the *difference* held by each agent should be incorporated into their estimates. The simulation results show that this algorithm reduces well the disruption of the system due to arrival of new agents. It complements the symmetric push-sum algorithm

by reaching closely to the performance limitation in the only situations where the symmetric push-sum algorithm does not.

Logically, Chapter 6 combines both algorithms in order to obtain an algorithm that responds to all the challenges of this external averaging problem and resolves both issues of the gossip averaging algorithm. The symmetric push-sum with difference tracking is therefore the most efficient algorithm that has been developed in this master thesis and it reaches very closely the performance limitations. For giving an idea, for a system with 50 agents with 1 replacement every 100 communications on average, it improves the expected mean squared error of the gossip averaging algorithm of about 26% (on a maximum of 27% achieved by the best ideal algorithm) and this improvement is even more significant when the replacements are more frequent: with 1 replacement every 5 communications (on average), the improvement is about 45.6% (on a maximum of 45.8% achieved by the best ideal algorithm).

Finally, Chapter 7 explores another category of algorithms that use more memory and computations but which may be relevant when the measurement held by each agent is a highly multi-dimensional data (which is heavy to store). This has been explored in the hope that an agent could calculate the best ideal estimate from Chapter 2. This has not succeeded and the over memory and computation costs are therefore not always worth it but the development of the algorithm remains interesting and opens up other perspectives.

Chapter 1

Problem statement and tools

This chapter first defines the problem of the external average evaluation in open multi-agent system that this master thesis analyzes and aims to solve. It also presents the assumptions that are taken. Thereafter, two tools that are used in the remainder of this document are introduced: the gossip averaging algorithm (for which an in-depth analysis is provided in Chapter 3) and the simulation tool.

1.1 Problem definition and general assumptions

This section presents the problem in a synthetic way in the first subsection. In a second time, the different parts of the problem and the assumptions that have been made are detailed and discussed in dedicated subsections.

1.1.1 Synthetic problem statement: the external averaging problem

Let us begin with a very synthetic description of the problem, before analyzing some aspects in detail in further subsections.

The system is an open multi-agent system, subject to replacements of agents during its operation. The agents present in the system at time t are described by the set of indices $\mathcal{N}(t)$. The size of the system, denoted $n = |\mathcal{N}(t)|$, is constant thanks to the replacement assumption: the departure of an agent from the system is immediately followed by the arrival of a new agent.

The agents are supposed to be identical, to have a bounded memory, to be capable of local computations and capable of communicating with each other, without having access to any kind of identifier. Each agent $i \in \mathcal{N}(t)$ holds an initial value z_i (also called initial measurement) independently drawn from an identical distribution of constant mean μ and variance σ^2 . The mean μ is also called the value of interest. Each agent i is supposed to execute the same algorithm in order to achieve a common desired goal for which it has its own solution estimate $y_i(t)$.

The goal of each agent is to estimate *as accurately as possible* the mean distribution μ :

$$y_i(t) \rightarrow \mu, \quad \text{for all } i \in \mathcal{N}(t).$$

Since the value of interest μ is constant, the best way of estimating it is to compute the external average of the system. The external average of the system is the average of the initial values z_k of all agents k that have ever been in the system until time t . The goal of each agent $i \in \mathcal{N}(t)$ can then be summarized as follows:

$$y_i(t) \rightarrow \mu \approx \frac{1}{|\cup_{s \leq t} \mathcal{N}(s)|} \sum_{k \in \cup_{s \leq t} \mathcal{N}(s)} z_k, \quad \text{for all } i \in \mathcal{N}(t). \quad (\text{External averaging problem})$$

The metric used to quantify the accuracy and the success of an algorithm to achieve this goal is the mean squared error criterion (MSE) computed between the agents estimates $y_i(t)$ and the value of interest μ . It is defined as:

$$\text{MSE}(t, \omega) = \frac{1}{n} \sum_{i \in \mathcal{N}(t, \omega)} (y_i(t, \omega) - \mu)^2.$$

It depends on the realization $\omega \in \Omega$ of the randomness into the system (the realization of the initial values of the agents and of the sequence of communication and replacement events that occur in the system). The expected mean square error $\mathbb{E}[\text{MSE}(t, \omega)]$ is used to evaluate the expected performance of an algorithm.

The communications occur in the system according to a Poisson clock of global rate $n\lambda_c$. Whenever a communication occurs, two randomly uniformly and independently selected agents $i, j \in \mathcal{N}(t)$ (possibly twice the same agent) exchange information with each other. In a nutshell, communications are said to be asynchronous, pairwise, symmetric and random.

The replacements occur in the system according to a Poisson clock of global rate $n\lambda_r$. The value of λ_r therefore the individual replacement rate. A replacement corresponds to the departure of one randomly and uniformly chosen agent from the system, instantaneously followed by the arrival of a new agent into the system.

The following subsections comment and provide some details about the different aspects of the problem.

1.1.2 The system, the agents and their goal

Let us consider an open multi-agent system whose composition evolves with time. Integers indices are used to label the agents: $\mathcal{N}(t) \subset \mathbb{N}$ denotes the set of agents present in the system at time t . These indices are used as identifiers for the sake of our analysis but in general, agents are not supposed to have access to any kind of identifiers between them. The cardinality of $\mathcal{N}(t)$ corresponds to the size of the system $n(t)$, i.e. the number of agents present at time t . We restrict the problem to the case where the system keeps a constant size $n(t) = n$, meaning that each departure of an agent is instantaneously followed by the arrival of a new agent in the system, which is called a *replacement*. This simplification allows avoiding the complications related to the system size change without sacrificing the challenges associated with the arrivals and departures of agents in the system.

Before arriving into the system, each agent $i \in \mathcal{N}(t)$ holds a value z_i independently drawn from an identical distribution of constant mean μ and variance σ^2 . For example, the z_i values can be considered as noisy measurements of a value of interest μ and thus follow a normal distribution: $z_i \sim \text{Normal}(\mu, \sigma^2)$. This distribution is supposed to be unknown to the agents and should therefore not been used for solving the problem. The agents have a bounded memory, are capable of local computations and capable of communicating with each other.

The purpose of the agents is to interact with each other so that each one estimates *as accurately as possible* the value of the distribution mean μ . Let $y_i(t)$ denotes the estimate of agent i , this purpose can be written as:

$$y_i(t) \rightarrow \mu, \quad \text{for all } i \in \mathcal{N}(t).$$

Since the value of μ is constant, a good way for each agent to pursue this objective is to average as many different measurements z_k of μ as it can and hence ideally, to average the initial measurements z_k of all agents k that have ever been in the system until time t . This average is called the external average of the system, denoted $\bar{z}_{\text{ext}}$:

$$\bar{z}_{\text{ext}}(t) = \frac{1}{|\cup_{s \leq t} \mathcal{N}(s)|} \sum_{k \in \cup_{s \leq t} \mathcal{N}(s)} z_k.$$

Using similar arguments as in Proposition 2.1 from Chapter 2, one can prove that, given i.i.d measurements z_k , the external average $\bar{z}_{\text{ext}}$ corresponds to the best estimator of μ among all the unbiased linear estimator¹ of μ . For our upcoming algorithmic design, we will stay in this framework of linear unbiased estimates. Therefore, estimating the value of interest μ , which is the true objective used for the performance evaluation described below, amounts to compute (or estimate) the external average of the system.

¹An estimator y of μ is linear (with respect to the measurements z_k) if $y = \sum_k c_k z_k$ and it is also unbiased if $\sum_k c_k = 1$.

This implicit objective is largely used for reasonings about the design of new algorithms and is therefore considered as integrant part of the objective of each agent $i \in \mathcal{N}(t)$ that can be resumed as follows:

$$\mathbf{Goal:} \quad y_i(t) \rightarrow \mu \approx \frac{1}{|\cup_{s \leq t} \mathcal{N}(s)|} \sum_{k \in \cup_{s \leq t} \mathcal{N}(s)} z_k, \quad \text{for all } i \in \mathcal{N}(t). \quad (\text{External averaging problem})$$

To achieve this goal, the agents, which are supposed to be identical and interchangeable should therefore execute an identical algorithm. This ensures a fully decentralized solution. To quantify the accuracy and the success of an algorithm, one need to choose a metric. It is what we called the performance evaluation.

1.1.3 Performance evaluation

To evaluate the performance of an algorithm at time t , for a given realization of the randomness $\omega \in \Omega$, the estimate $y_i(t, \omega)$ of each agent $i \in \mathcal{N}(t, \omega)$ is compared to the true distribution mean μ of the measurements z . The metric used is the mean squared error (MSE). The mean is taken over the different agents that are in the system:

$$\text{MSE}(t, \omega) = \frac{1}{n} \sum_{i \in \mathcal{N}(t, \omega)} (y_i(t, \omega) - \mu)^2.$$

The mean squared error is a classical performance measure of an estimator that has the advantage to be easily decomposed into squared bias and variance. The bias represents the error of the entire system while the variance rather measures the degree of agreement (or disagreement) between the different agents in the system. Their formal definitions are:

$$\text{Bias}(t, \omega) = \bar{y}(t, \omega) - \mu \quad \text{and} \quad \text{Var}(t, \omega) = \frac{1}{n} \sum_{i \in \mathcal{N}(t, \omega)} (y_i(t, \omega) - \bar{y}(t, \omega))^2 \quad (1.1)$$

Where $\bar{y}(t, \omega)$ denotes the average of estimates values in the system at time t for realization $\omega \in \Omega$: $\bar{y}(t, \omega) = \frac{1}{n} \sum_{i \in \mathcal{N}(t, \omega)} y_i(t, \omega)$. One can easily prove that (see [Ava19])

$$\text{MSE}(t, \omega) = \text{Bias}^2(t, \omega) + \text{Var}(t, \omega). \quad (1.2)$$

This decomposition provides an easy interpretation of the performance of an algorithm. It allows understanding if performance is dominated by the error of the group that is biased or by individual divergences that imply a large variance.

For the same system configuration $(n, \lambda_c, \lambda_r)$, the evolution (with time) of the MSE value can be very different from a realization $\omega \in \Omega$ to another. Furthermore, for a same realization $\omega \in \Omega$, the MSE can evolve very abruptly due to replacements of agents. Measuring the expected value of the MSE gives a valid idea of the expected performance of an algorithm in a given system configuration $(n, \lambda_c, \lambda_r)$ and resolves these issues related to the randomness of the problem.

$$\mathbb{E}[\text{MSE}(t, \omega)] = \mathbb{E} \left[\frac{1}{n} \sum_{i \in \mathcal{N}(t, \omega)} (y_i(t, \omega) - \mu)^2 \right]$$

This expected MSE can be estimated by considering a large sample of realizations (denoted S) over which the MSE is averaged.

$$\mathbb{E}[\text{MSE}(t, \omega)] \approx \overline{\text{MSE}(t, \omega)} = \frac{1}{|S|} \frac{1}{n} \sum_{\omega \in S} \sum_{i \in \mathcal{N}(t, \omega)} (y_i(t, \omega) - \mu)^2,$$

The expected squared bias $\mathbb{E}[\text{Bias}(t, \omega)^2]$ and the expected variance $\mathbb{E}[\text{Var}(t, \omega)]$ can be defined and computed in a similar way.

A realization of the randomness $\omega \in \Omega$ can be decomposed into realization of two independent vectors:

- A realization of the events sequence ϵ chosen among all the possible ones E ($\epsilon \in E$). It includes the order and the type of events (communication or replacement) and also agents that are involved in the different events.
- A realization of the agents' initial measurements ζ chosen among all the possible ones Z ($\zeta \in Z$).

The sample space Ω can be expressed as a cross product of both sample spaces E and Z : $\Omega = E \times Z$. Therefore, the expectation over all realizations ω (denoted simply with $\mathbb{E}$ or sometimes with $\mathbb{E}_\omega$) can be decomposed into two successive expectations: one over the realizations of events sequence ϵ (denoted $\mathbb{E}_\epsilon$) and another over the realization of initial measurements ζ (denoted $\mathbb{E}_\zeta$). For example, we have that

$$\mathbb{E}[\text{MSE}(t, \omega)] = \mathbb{E}_\omega[\text{MSE}(t, \omega)] = \mathbb{E}_\epsilon[\mathbb{E}_\zeta[\text{MSE}(t, \omega)]].$$

It can be used to simplify the expression of the expected mean squared error. Indeed, since the set of agents in the system $\mathcal{N}(t)$ only depends on the sequence of events (and not on the initial measurements of the agents), it holds that

$$\mathbb{E}[\text{MSE}(t, \omega)] = \mathbb{E}_\epsilon[\mathbb{E}_\zeta[\text{MSE}(t, \omega)]] = \mathbb{E}_\epsilon \left[\frac{1}{n} \sum_{i \in \mathcal{N}(t, \omega)} \mathbb{E}_\zeta \left[(y_i(t, \omega) - \mu)^2 \right] \right].$$

And therefore, since the agents execute an identical algorithm, the expectation $\mathbb{E}_\zeta \left[(y_i(t, \omega) - \mu)^2 \right]$ is the same for all $i \in \mathcal{N}(t)$ and the expected mean squared error can be written in a simple form,

$$\mathbb{E}[\text{MSE}(t, \omega)] = \mathbb{E}_\epsilon \left[\mathbb{E}_\zeta \left[(y_i(t, \omega) - \mu)^2 \right] \right] = \mathbb{E} \left[(y_i(t, \omega) - \mu)^2 \right].$$

This latter definition is used for the development of the performance limitations in Chapter 2.

To alleviate the notations, the dependency of the estimates, the set of agents and the performance on the realization ω is often omitted. Implicitly, we have for example that:

$$y_i(t, \omega) = y_i(t), \quad \text{MSE}(t, \omega) = \text{MSE}(t), \quad \text{Bias}(t, \omega) = \text{Bias}(t), \quad \text{Var}(t, \omega) = \text{Var}(t), \quad \mathcal{N}(t, \omega) = \mathcal{N}(t).$$

Sometimes, the time dependence is also omitted for these notations, but this is in general specified.

We also often suppose, without loss of generality, that the distribution mean is zero: $\mu = 0$. It implies, omitting time-dependence notation, that

$$\text{When } \mu = 0, \quad \text{Bias}^2 = \bar{y}^2 \quad \text{and} \quad \text{MSE} = \bar{y}^2 = \frac{1}{n} \sum_{i \in \mathcal{N}} y_i^2. \quad (1.3)$$

1.1.4 System events: the communications and the replacements

Since the system is open and has a constant size n , it is subject to replacements of agents. Therefore, two different types of events can occur in the system:

Communication: two agents $i, j \in \mathcal{N}(t)$ are uniformly randomly and independently selected² to exchange some information. The interactions are supposed to be symmetric, i.e. both agents share and receive information from each other. The communication scheme is asynchronous, which means that each communication is independent of the others and therefore that the agents do not have to perform exactly the same number of interactions. In brief, communications are asynchronous pairwise symmetric and random. This event may only occur if $n > 0$.

Replacement: one uniformly randomly selected agent $i \in \mathcal{N}(t)$ leaves the system and one *new* agent $j \notin \mathcal{N}(s)$ for all $s \leq t$ joins the system. It results that $\mathcal{N}(t^+) = (\mathcal{N}(t) \setminus \{i\}) \cup \{j\}$ and the size n of the system remains well constant. This event may only occur if $n > 0$.

²This includes the possibility of selecting twice the same agent.

The frequency and the distribution of these two types of events (communication and replacement) determine the behavior of the system. We suppose that the evolution of the number of communications and the number of replacements that have occurred in the system are both characterized by a different Poisson process with respective positive global communications and replacements rates λ_c^G and λ_r^G which depend on the size of the system and on individual corresponding rates. It means that the time between two communications (respectively replacements) are exponential random variables with mean $\frac{1}{\lambda_c^G}$ (respectively $\frac{1}{\lambda_r^G}$). In other words, there are in average, λ_c^G communications (respectively λ_r^G replacements) per unit of time.

The two independent Poisson processes can be merged into one Poisson process of rate $\lambda_c^G + \lambda_r^G$ that count all the events. The probability of an event to be a replacement is then $p_r = \frac{\lambda_r^G}{\lambda_r^G + \lambda_c^G}$. Conversely, a Poisson process can also be split into multiple independent Poisson processes: the replacement of each agent can be counted individually and independently with a Poisson process of identical and individual replacements rate λ_r ; and the communications of each pair of agents can be counted individually and independently with a Poisson process of identical pairwise communications rate λ_c^p . It then holds that the global replacement rate grows linearly with the size of the system n (for a fixed individual replacement rate λ_r): $\lambda_r^G = n\lambda_r$, while the global communication rate grows quadratically with n (for a fixed pairwise communications rate λ_c^p): $\lambda_c^G = \frac{n^2}{2}\lambda_c^p$. The Poisson processes that count the communications individually (and not pairwise) are not independent, but we still introduce a kind of individual communication rate λ_c (identical for all agents) that is such that the global rate grows linearly with n : $\lambda_c^G = n\lambda_c$ in order to have rates that are comparable.

For the following analyses, two size-independent measures of the behavior of the system are used:

$$\begin{aligned} \text{The probability of an event to be a replacement} \quad p_r &= \frac{\lambda_r^G}{\lambda_r^G + \lambda_c^G} = \frac{\lambda_r}{\lambda_r + \lambda_c}, \\ \text{The rate ratio} \quad \rho &= \frac{\lambda_c}{\lambda_r}. \end{aligned}$$

These two measures are linked by the following relation

$$p_r = \frac{1}{1 + \rho}.$$

The individual rate ratio ρ can be interpreted as the expected total number of communications that occur in the system between two replacements. Indeed, $\lambda_c^G = n\lambda_c$ corresponds to the expected number of communications per unit of time and $\frac{1}{\lambda_r^G} = \frac{1}{n\lambda_r}$ corresponds to the expected time between two replacements (recall that inter-occurrence time of a Poisson process follows an exponential distribution of equivalent rate). Hence, the expected number of communications between two replacements (denoted g) is given by:

$$\frac{\lambda_c^G}{\lambda_r^G} = \frac{n\lambda_c}{n\lambda_r} = \rho.$$

The individual rate ratio ρ can also be linked to the expected number of communications that a specific agent i performs before being replaced which is denoted by g_i . Indeed, the number of pairs of agents $i, j \in \mathcal{N}(t)$ that involves a same agent i is n (it does not exclude that an agent form a pair with itself). Therefore, $n\lambda_c^p$ describes the rate of communications that involve agents i and g_i can be computed as follows:

$$g_i = \frac{n\lambda_c^p}{\lambda_r} = n \frac{2\lambda_c}{n} \frac{1}{\lambda_r} = 2\rho \quad \text{for all } i.$$

The second equality uses the link that exists, by definition, between λ_c^p and λ_c : $\frac{n^2}{2}\lambda_c^p = n\lambda_c = \lambda_c^G$. In theory, g can be lower than one if the replacements are more frequent than the communications ($\lambda_r > \lambda_c$). And similarly, g_i can also be lower than one if replacements are at least twice as frequent as communications ($\lambda_r > 2\lambda_c$). However in practice, having a system with as many replacements and as few communications is not very interesting for multi-agent system applications in which, by definition, agents communicate to collaborate. For the analyses to follow, it will therefore sometimes be assumed that g is greater than one and hence $\rho > 1$ or equivalently $p_r < 1/2$. We also mainly use the individual communication and replacements rates and not so much the other related rates (global or pairwise). Therefore, they will often simply be referred to as the communication and replacement rates.

1.1.5 Problem statement limitations

As far as I know, this external averaging problem has never been addressed in the literature. Therefore, I focus on one of the simplest version of the problem, as presented in previous section. Consequently, the model presents different limitations that are briefly listed below.

- The topology of the communication network is supposed to be complete. In the problem definition, communication can occur between any pair of agents $i, j \in \mathcal{N}(t)$ at any time t in both directions. In reality, all pairs of agents in the system may not have the opportunity to communicate due to distance separation, lack of bandwidth or non-existent connection. This communication constraints could also evolve (deterministically or stochastically) with time. Asymmetric interactions or interactions involving more than two agents can also be considered. Many works in multi-agent systems study the impact of the topology of the communication network on a closed system and conditions to guarantee consensus [TJ08] [HM05]. For open multi-agent system, the convergence to a specific state will never happens (unless the arrivals and departures stops) and the system may rather approach some form of steady state behavior. [Oli18] starts to investigate the behavior of the classical averaging gossip algorithm in open multi-agent systems for different communication network topologies and provide rule between the structure of a system and the variance of the values held by the agents, i.e. a measure of the disagreement in the system.
- The replacement assumption is not valid in many applications where the size of the system may vary. This replacement assumption still makes sense when the arrival and the departure rates are similar. Many ideas and analysis developed in this master thesis under this assumption can be adapted and re-used to consider separated arrival and departures. This limitation therefore seems rather technical.
- The individual communication and replacement rates are supposed to be constant and identical for all the agents. Some applications can have time-varying communication or replacement rates where the system knows different regimes (as fluid or busy road for a car-oriented applications) or the agents can also have different communications and replacements rates from each other.
- The value of interest μ is constant. A time-varying mean $\mu(t)$ can be useful to estimate (in a decentralized manner) time-varying external value as temperature, humidity,... In that situation, the external average of the system can no longer be used to estimate a time-varying value $\mu(t)$ and the solutions presented in this document seem no longer valid, unless adapted to make agents aggregating information of a limited number of agents (the newest ones) that should depend on the variation speed of μ .

1.2 The gossip averaging algorithm: a baseline algorithm

The gossip averaging algorithm is a simple averaging consensus algorithm. It has already been briefly presented in the state-of-the-art section and its analysis in open multi-agent system is presented in Chapter 3. We mention it here since the algorithm and its analysis in open multi-agent system conducted in [HM16] and [HM17] are mainly used, as a tool, in the remaining of this master thesis, including for the validation of the simulation tool presented in the next section. The algorithm can be summarized as follows:

Gossip averaging algorithm:

Initialization for all $i \in \mathcal{N}(t)$:	$x_i(0) = z_i,$
Interaction between a pair of agents $i, j \in \mathcal{N}(t)$:	$x_i^+ = x_j^+ = \frac{x_i + x_j}{2},$
Average estimate of agent $i \in \mathcal{N}(t)$:	$y_i(t) = x_i(t).$

For this algorithm the estimates of the agents, denoted y_i , and the variables x_i coincide and can be used indifferently. In general, only x_i are used to denote both the variables of the agents and their estimates of μ (omitting the time dependence notation). In closed system, the mains result to remember is that the sum of x_i values are preserved and hence the agents are converging to a consensus which corresponds

to the average of their initial values.

The analysis of the evolution of an *open* multi-agent system subject to the gossip averaging algorithm (see Chapter 3) allows deducing the performance of the algorithm for the external averaging estimation problem. The analysis relies on the description of the system through size-independent and self-contained descriptors (omitting the time dependence notation)

$$\mathbb{E}X = \left[\mathbb{E}[\bar{x}^2], \mathbb{E}[\bar{x}] \right]^T$$

where $\bar{x} = \frac{1}{n} \sum_{i \in \mathcal{N}} x_i$ and $\bar{x}^2 = \frac{1}{n} \sum_{i \in \mathcal{N}} x_i^2$.

The expected performance (the mean squared error, the squared bias and the variance defined in Section (1.1)) that the algorithm achieves for the external averaging problem can be deduced from these descriptors. For example, when the distribution mean is zero $\mu = 0$, we have (from equation (1.3)) that

$$\mathbb{E}X = \left[\mathbb{E}[\bar{x}^2], \mathbb{E}[\bar{x}] \right]^T \underset{(\text{for } \mu = 0)}{=} [\text{Bias}^2, \text{MSE}]^T.$$

The evolution of these descriptors with an event k in the system are deduced in [HM17] and explained in Chapter 3. It is given by the following linear recurrence system:

$$\mathbb{E}X_k = A\mathbb{E}X_{k-1} + b \tag{1.4}$$

$$\text{With } A = \begin{bmatrix} 1 - \frac{2p_r}{n} & \frac{p_r}{n^2} \\ \frac{1-p_r}{n} & 1 - \frac{1}{n} \end{bmatrix}, \quad \text{and} \quad b = \sigma^2 \begin{bmatrix} \frac{p_r}{n^2} \\ \frac{p_r}{n} \end{bmatrix}.$$

This system of equations allows tracking the expected performance evolution of the averaging algorithm and deducing its steady state expected performance. These theoretical guarantees about the performance of the gossip averaging algorithm led us to use the algorithm as a baseline that new algorithms should have to beat.

1.3 Simulation tool

This section briefly presents the simulation tool created to simulate an open multi-agent system that respects the assumptions set in Section 1.1. Its validation is also presented, before reviewing the different choices about the parameters scaling that have been made for all the simulations presented in this document and that may potentially have an impact on the interpretation of the simulation results.

Utility of the simulation tool

This tool has been used to assist the design of new algorithms. Indeed, it makes it possible to observe what happens in many situations and get better intuition about the problem or the algorithms used. Since the only algorithm with theoretical guarantees for now is the gossip averaging algorithm, this simulation tool is also used to compare the algorithms with each other in different scenarios and for different system configurations.

Conception and main functionalities

Two types of simulations have been designed. Both of them consider a realization of the measurements of the agents and of the event sequence (communications and replacements) determined by a *random seed* argument which allow to reproduce the exact same realization for different simulation (e.g. to compare different algorithms). For a given algorithm and system configuration (λ_c , λ_r and n), the two types of simulations can be described as follows:

Event-based simulation loops on all events and return (among other things) the performance (for a specific realization) of the system after each event. The result depends only on the probability of an event to be a replacement $p_r = \frac{\lambda_r}{\lambda_c + \lambda_r}$ but not on each specific rate separately.

Time-based simulation loops on a discrete-time to return (among other things) the performance of the system at each sample time. The performance depends also on the probability of a event to be a replacement p_r but also on how many events occur at each time step ($n(\lambda_c + \lambda_r)$).

For both types of simulation, it is needed to average the performance over multiple realizations to obtain the expected performance of an algorithm. However, to give a visual representation of what happens in a specific realization of the system, it is possible to launch an animation of the simulation. The event-based simulation gives the evolution of the performance with the number of events that does not necessarily happen at the same moment in different realizations. Therefore, the average over multiple realizations in the event-based simulation is not really well time representative. The time-based simulation gives the evolution of the performance over time. In that case, the simulation of all the realization of sequences of events gives the performance of the system for the same discretization of the time and hence the averaging gives well the expected performance over time. Depending on the context and the purpose of the simulation, either event-based or time-based simulation is used. For the comparison of the expected performance of the different algorithms, time-based simulation is mainly used. For details about this simulations tool, its animation and other related plot scripts, you can consult the dedicated github repository: <https://github.com/sebcolla/Simulation-Tool-TFE-OMAS.git>.

Validation

To validate the sound operation of the simulation tool, the simulated expected evolution of a specific system is compared with the expected evolution predicted using the descriptor evolution system briefly presented in Section 1.2 but explained in detail in Chapter 3. As a reminder, the descriptor vector is defined as:

$$\mathbb{E}X = \left[\mathbb{E}[\bar{x}^2], \mathbb{E}[\bar{x}^2] \right]^T \quad (\text{for } \mu = 0) \quad [\text{Bias}^2, \text{MSE}]^T.$$

For the validation, it is supposed without loss of generality that $\mu = 0$ to have equivalence between the descriptors and the expected performance (squared bias and mean squared error). For computing the theoretical evolution of the system descriptors, the initial value of the descriptor vector is set to its initial expectation which is

$$\mathbb{E}X_0 = \left[\mathbb{E}[\bar{x}^2(0)], \mathbb{E}[\bar{x}^2(0)] \right]^T = \left[\frac{\sigma^2}{n}, \sigma^2 \right]. \quad (1.5)$$

Indeed, the initialization of the x values of any agent initially in the system $i \in \mathcal{N}(0)$ is given by $x_i(0) = z_i$. Remembering that the measurements z_i are independent and identically distributed (with mean $\mu = 0$ and variance σ^2) allows deducing equation (1.5).

To obtain empirical expected evolution of the performance of the system, it requires averaging the results of multiples realizations simulations. As shown in Figure 1.1, the expected performance computed theo-

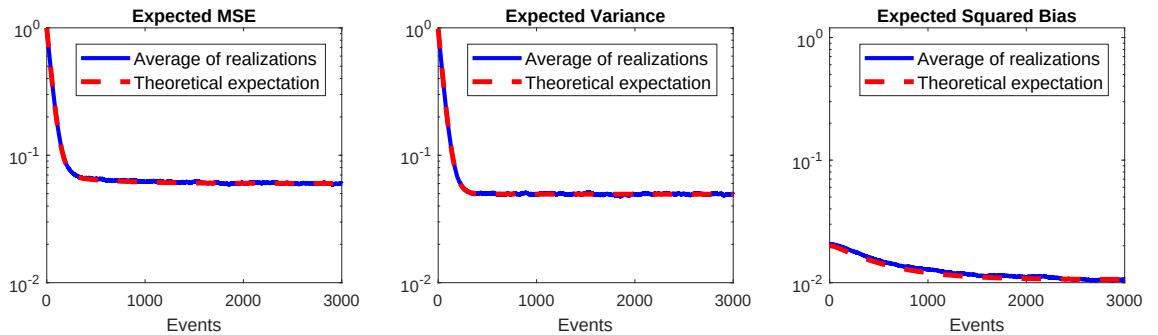


Figure 1.1: Comparison of the expected performance (MSE, squared bias and variance) for the gossip averaging algorithm computed theoretically, using equation (1.4) (and (5.8) for initialization) with the expected performance computed empirically by averaging 1000 realizations simulation, for a system of size $n = 50$ and replacement probability $p_r = 0.05$ (with constant $n\lambda_c = 10$).

retically using recurrence relation (1.4) (in dashed red) fits very well the empirical expected performance computed through simulations (in blue). When averaging 1000 realizations, for the system used on the figure ($n = 50$ and $p_r = 0.05$), the relative difference between the curves (averaged over the 3000 events) is less than 1% for the expected variance and the mean squared error and about 3% for the expected mean squared error (for which the empirical expectation requires more realizations to be a bit smoother).

This numerical experiment strengthens the validity of both the simulation tool and the theoretical evolution of the descriptors mentioned in Section 1.2, explained in Chapter 3 but initially developed in [HM17].

Parameter scaling

In order to correctly compare the results of the simulations even when the system configurations are different, we need to carefully scale the parameters of the system to avoid comparing results for which too many parameters are changing. Different choices are possible and are presented here. I have chosen for each situation a way to scale the parameters that I will use for the whole document.

First of all, it will always be assumed for the simulation that the distribution of the measurements is a normal with zero mean $\mu = 0$ and unit variance $\sigma^2 = 1$. It can be done without loss of generality since the value of the mean μ does not impact the performance of the system and the value of the variance corresponds simply to a scaling factor for these performances. The choice of the normal distribution is an arbitrary choice, motivated by its many applications. It should not change the conclusion of the results.

To evaluate the global performance of an algorithm, we often want to compute its expected performance for many different system configurations. The system configuration denotes the three main parameters of the system that define the behavior of the system over time: the system size n , the individual communication rate λ_c and the individual replacement rate λ_r .

The impact of the variation of the two following parameters on the performance of a given algorithm is often used to analyze its performance.

- The variation on the size of the system n .
- The variation of frequency of the replacements, in comparison with the frequency of the communication. It corresponds in other words to the variation of the total number of interactions that occur in the system between two replacements. This measure is interesting because it evaluates how much the system can react to a replacement before the next one takes place. As detailed in Section 1.1.4, it corresponds to the ratio between the communication and the replacement rate (called the rate ratio).

$$\rho = \frac{\lambda_c}{\lambda_r}.$$

When we vary the system size n , for a fixed rate ratio ρ , we have two different options for the parameters scaling:

1. Keep individual rates λ_c and λ_r constant. It allows having constant rate ratio ρ but it implies that the global communication and replacement rates vary linearly with n :

$$\lambda_c^G = n\lambda_c \quad \text{and} \quad \lambda_r^G = n\lambda_r. \quad (1.6)$$

In that situation, it occurs (in average) more communications and replacements (in total) in a given amount of time in a larger system.

2. Keep global rates λ_c^G and λ_r^G constant. It allows having constant rate ratio ρ but it implies that the individual communication and replacement rates vary linearly with $1/n$ (from equation (1.6)). In that situation, for a given amount of time, it occurs (in average) the same total number of communications and replacements for any system size. Therefore, when n grows, the contribution of each agent to the global rates diminished, meaning that its individual communications and replacement are less frequent.

I have chosen the second case to scale the rates parameters when system size n is varying. It corresponds to the scaling that is the toughest with large system, so if we find an algorithm that performs well in large system using this scaling, it has many chances to do so with the other parameter scaling (the first one).

When we vary the rate ratio ρ , for a fixed system size n , we also have different options to vary $\rho = \frac{\lambda_c}{\lambda_r}$:

1. Keep the replacement rate λ_r constant and vary the communication rate λ_c . Things probably always get better in a system with more communication.
2. Keep the communication rate λ_c constant and vary the replacement rate λ_r . It seems more suited to study the impact of the replacements on a system.
3. Keep the sum of the rates $\lambda_c + \lambda_r$ constant and only vary their ratio ρ . It allows always having (in average) the same number of events in a given amount of time but a different repartition between the number of communications and replacements. It is a bit arbitrary to impose that one replacement count for one communication. More generally, we can keep $\alpha_c \lambda_c + \alpha_r \lambda_r$ constant, for any $\alpha_c, \alpha_r \in \mathbb{R}$. When $\alpha_c = 0$ and $\alpha_r = 1$, we return to the first case, and similarly, we return to the second one when $\alpha_c = 1$ and $\alpha_r = 0$. Any choice of coefficients α_c and α_r is possible in practice but it does not always make much sense.

I have chosen the second option to scale rates parameters when their ratio is varying. It corresponds probably to the best choice to analyze the impact of the replacements on a system and the performance of a given algorithm.

These two choices of parameters scaling correspond to always impose a constant global communication rate $\lambda_c^G = n\lambda_c$ for any variation of n or ρ . I have arbitrarily chosen to set this constant to 10 ($n\lambda_c = 10$). These choices are used throughout this master thesis when the results of the simulations are shown and analyzed.

Chapter 2

Performance Limitations

This chapter aims to find the best expected performance that an algorithm can reach for the external averaging problem stated in Section 1.1.1. Empirical and proved lower bounds on the expected mean squared error are deduced, discussed and compared. Having a lower bound on the expected performance is very useful when designing new algorithms in order to be able to understand what improvements can still be done and in which situation. The next chapter (Chapter 3) will also show how the gossip averaging expected performance is situated in comparison with the empirical lower bound in order to illustrate the progression margin that can be achieved with new algorithms (see Section 3.3).

To find the best possible performance that an algorithm can reach, the idea is to consider an ideal algorithm and to evaluate (or bound) the performance of this ideal algorithm. This algorithm is called ideal since agents can use resources that are not allowed in the classical designed algorithm. In our situation, the algorithm is supposed to have access to an infinite memory and to the agent identifiers.

2.1 Ideal algorithm

In order to *best* estimate the distribution mean μ , agents should average as many measures z_k of μ as possible. The *best* thing an agent can do at time t is therefore to compute the external average of the system at time t that, as a reminder, is the average of the initial measurements z_k of all the agents k that have ever been in the system up to time t , even past one that are left from the system. Unfortunately, it is not guaranteed (due to replacements) that an agent has been in contact with all these agents (directly or indirectly, with multiple middle-agents) and hence has access to their measurements he needs to compute the exact external average of the system. The *best* thing an agent can do is therefore to average the initial measurements z_k of all agents k it knows. The knowledge set of an agent formalizes the acquaintance to which it has (potentially) access. This concept is introduced and used in a similar purpose in [MH20] and are studied more generally for communication graphs in [Hen14]. We defined it as follows:

Definition 2.1 (Knowledge sets). Let $i \in \mathcal{N}(t)$ be the index of an agent in the system at time t . The knowledge set of agent i at time t denoted $\mathcal{K}_i(t)$ is the set of indices of all the agents with which agent i has been in contact (directly or indirectly) since its arrival into the system at time $t_{i,0}$. The knowledge set of agent i is initialized with only its own index: $\mathcal{K}_i(0) = \{i\}$. At each communication, agents involved in the communication (i and j) make the union of their knowledge sets:

$$\mathcal{K}_i^+ = \mathcal{K}_j^+ = \mathcal{K}_i \cup \mathcal{K}_j.$$

At any time t , the knowledge set of agent i is included into the set of indices of all the agents that have ever been in the system until time t : $\mathcal{K}_i \subset \cup_{s \leq t} \mathcal{N}(s)$.

The knowledge set of an agent i allows describing the set of all agents for which agent i can have access to individual information such as their measurements z or other things (if they are memorized and shared at each communication in the same way that agents indices in the knowledge sets). Therefore, the *best* estimate of μ that each agent can have is defined as follows:

Definition 2.2 (Best ideal algorithm). Let $\mathcal{N}(t)$ be the time-varying set of agent indices of an open multi-agent system. The best ideal algorithm (for the external averaging problem stated in Section 1.1.1) is the one in which each agent uses equally all possible knowledge it gathers. In that situation, the estimate of the value of interest μ of each agent $i \in \mathcal{N}(t)$ is defined by

$$y_i^{\text{best}}(t) = \frac{1}{|\mathcal{K}_i(t)|} \sum_{k \in \mathcal{K}_i(t)} z_k \quad \text{for all } i \in \mathcal{N}(t), \quad (2.1)$$

where $\mathcal{K}_i(t)$ is the knowledge set of agent $i \in \mathcal{N}(t)$ (see Definition 2.1).

The resulting expected mean squared error of the best algorithm can be written as

$$\mathbb{E}[\text{MSE}_{\text{best}}(t)] = \mathbb{E} \left[\frac{1}{n} \sum_{i \in \mathcal{N}(t)} (y_i^{\text{best}}(t) - \mu)^2 \right] = \mathbb{E} \left[(y_i^{\text{best}}(t) - \mu)^2 \right],$$

where the last equality holds because agents are supposed to be identical and interchangeable (in expectation) (see Section 1.1.3 for details).

The sense of the word *best* we used until here should still be clarified. This is the purpose of Proposition 2.1. Since the distribution of the measurements is unknown to the agents, this estimate seems intuitively to be the *best* thing an agent can do (in terms of expected MSE). But in theory, it is not so easy and maybe it is not totally true when we consider all possible other estimates. As it will be stated and proved in Proposition 2.1, a such estimate (from equation (2.1)) corresponds to the best linear unbiased estimate that agent $i \in \mathcal{N}(t)$ can do at time t .

To execute a such algorithm, each agent i of the system ($i \in \mathcal{N}(t)$) should memorize and share, at each communication, all the measurements it has access to (i.e. z_k for all $k \in \mathcal{K}_i(t)$), and their corresponding agent indices k in order to keep their knowledge set up-to-date. Therefore, the memory required for each agent grows with time since an agent needs to store the measurement of all agents with which it has been in contact, even indirectly and even if they are gone long time ago from the system. As it will be demonstrated further, the expected size of the knowledge set of an agent (and hence the memory it needs) is upper bounded but can still be huge. That is why this algorithm cannot be used in practice. Indeed, agents are supposed to have constant bounded memory and furthermore, it is in general desirable for this memory limitation to be low for economic or practical reasons. Moreover, for this algorithm, the agents should probably make use of identifiers¹ (i.e. the indices) of the agents in order to maintain their knowledge set and its corresponding measurements, which was also forbidden in our problem statement. Chapter 7 presents an algorithm that makes use of the knowledge sets, without sharing and storing the measurements. This can make sense when the data of each agent are heavy.

This ideal algorithm is nevertheless useful since its expected mean squared error $\mathbb{E}[\text{MSE}_{\text{best}}(t)]$ provides a lower bound on the expected mean squared error of other algorithms:

Proposition 2.1 (Lower bound on the MSE of any linear unbiased algorithm). *For any algorithm that aims to solve the external averaging problem stated in Section 1.1.1 by generating estimates that corresponds to unbiased linear combinations of the measurements (z_i):*

$$y(t) = \sum_i c_i(t) z_i, \quad \text{with } \sum_i c_i(t) = 1, \quad (2.2)$$

The expected mean squared error $\mathbb{E}[\text{MSE}(t)]$ of the algorithm is lower bounded by the one of the best ideal algorithm (see Definition 2.2):

$$\mathbb{E}[\text{MSE}(t)] \geq \mathbb{E}[\text{MSE}_{\text{best}}(t)] \quad \text{for all time } t.$$

Proof. The estimate $y_i^{\text{best}}(t)$ of μ of agent i is defined by equation (2.1). It is the average of the measurements of μ of all the agents that are in its knowledge set $\mathcal{K}_i(t)$.

¹One can claim that for most of the continuous distributions, the measurement z of an agent is unique (with high probability) and can be used as identifier.

Given a sample of i.i.d. measurements from a given distribution (here the measurement of the agents in the knowledge set), the sample average is the best linear unbiased estimator (BLUE) (in terms of the MSE with respect to the given distribution) of the true mean μ of the distribution (see [Sie] for proof). Therefore, for any linear unbiased estimate y (that satisfies equation (2.2)) and for any agent $i \in \mathcal{N}(t)$, we have that

$$\mathbb{E}_\zeta[(y - \mu)^2 \mid \mathcal{K}_i(t)] \geq \mathbb{E}_\zeta[(y_i^{\text{best}}(t) - \mu)^2 \mid \mathcal{K}_i(t)] \quad \text{for any knowledge set } \mathcal{K}_i(t),$$

where $\mathbb{E}_\zeta$ denotes the expectation over all the realizations of the measurements z of the agents (which are i.i.d.).

Since this inequality is valid for any agent of the system, taking the expectation over all the possible knowledge sets, or equivalently over all the possible sequences of events ϵ , gives, for any linear unbiased estimate y (satisfying equation (2.2)):

$$\begin{aligned} \mathbb{E}_\epsilon[\mathbb{E}_\zeta[(y - \mu)^2 \mid \mathcal{K}_i(t)]] &\geq \mathbb{E}_\epsilon[\mathbb{E}_\zeta[(y_i^{\text{best}}(t) - \mu)^2 \mid \mathcal{K}_i(t)]] \\ \mathbb{E}[(y - \mu)^2] &\geq \mathbb{E}[(y_i^{\text{best}}(t) - \mu)^2]. \end{aligned} \quad (2.3)$$

Equation (2.3) means that the best estimate (in terms of the expected MSE) at time t of any agent of the system $i \in \mathcal{N}(t)$, among all the linear unbiased estimates (equation (2.2)) is given by $y_i^{\text{best}}(t)$. Therefore, the expected mean squared error of the entire system (using this estimate) is smaller than the one of any other algorithm that uses linear and unbiased estimators:

$$\mathbb{E}[\text{MSE}(t)] = \mathbb{E}[(y_i(t) - \mu)^2] \geq \mathbb{E}[(y_i^{\text{best}}(t) - \mu)^2] = \mathbb{E}[\text{MSE}_{\text{best}}(t)] \quad \text{for all time } t.$$

□

Having agents that compute an unbiased estimator of μ (denoted y_i) means that $\mathbb{E}_\zeta[y_i] - \mu = 0$ for all agents $i \in \mathcal{N}(t)$. It does not necessarily lead to a zero expected squared bias of the system as it has been defined in Section 1.1.3:

$$\mathbb{E}[(\bar{y} - \mu)^2] = \mathbb{E}_\epsilon \mathbb{E}_\zeta \left[\left(\frac{1}{n} \sum_{i \in \mathcal{N}(t)} y_i(t) - \mu \right)^2 \right].$$

It indeed computes the squared bias with respect to the estimates into the system, by taking their averaging and not with respect to the distribution of the measurements.

One can also prove that the said best ideal estimates (given by Definition 2.2) are also the best (in terms of expected MSE) among all possible estimates (even unbiased or nonlinear), if the distribution is Gaussian. This means that to be better than the said best ideal algorithm, an algorithm should exploit both the nonlinear or biased estimates and the non-Gaussianity of the distribution. For our upcoming algorithmic design, any new algorithm uses linear and unbiased estimates, so this lower bound holds for sure.

The following of this section aims to express the expected performance of the best ideal algorithm in order to develop bounds on the performance of any (linear unbiased) algorithm we could imagine.

Proposition 2.2 (Expected MSE of the best ideal algorithm). *The best ideal algorithm from Definition 2.2 achieves the following expected mean squared error.*

$$\mathbb{E}[\text{MSE}_{\text{best}}(t)] = \mathbb{E}_\epsilon \left[\frac{\sigma^2}{|\mathcal{K}_i(t)|} \right] \quad \text{for all time } t$$

where $\mathbb{E}_\epsilon$ denotes the expectation over sequences of events that occur in the system, σ^2 is the variance of the agent initial measurements distribution and $\mathcal{K}_i(t)$ the knowledge set of an agent i (from Definition 2.1).

Proof. The expected mean squared error of the algorithm is

$$\begin{aligned}\mathbb{E}[\text{MSE}_{\text{best}}(t)] &= \mathbb{E}\left[(y_i^{\text{best}}(t) - \mu)^2\right] \\ &= \mathbb{E}\left[(y_i^{\text{best}}(t))^2\right] + \mu^2 - 2\mu\mathbb{E}[y_i^{\text{best}}(t)],\end{aligned}\quad (2.4)$$

The expectations are decomposed into expectation over sequences of events ($\mathbb{E}_\epsilon$) and over the initial measurements of the agents ($\mathbb{E}_\zeta$). Therefore, using this decomposition and the definition of the estimates $y_i^{\text{best}}(t)$ of agent i used in the best ideal algorithm (see Definition 2.2), $\mathbb{E}[y_i^{\text{best}}(t)]$ can be computed as:

$$\mathbb{E}[y_i^{\text{best}}(t)] = \mathbb{E}_\epsilon \left[\frac{1}{|\mathcal{K}_i(t)|} \sum_{k \in \mathcal{K}_i(t)} \mathbb{E}_\zeta[z_k] \right] = \mu,$$

since $\mathbb{E}_\zeta[z_k] = \mu$ for any k (using that z_k are i.i.d with mean μ). Similarly, $\mathbb{E}[y_i^{\text{best}}(t)^2]$ can be computed using the fact that:

$$(y_i^{\text{best}}(t))^2 = \left(\frac{1}{|\mathcal{K}_i(t)|} \sum_{k \in \mathcal{K}_i(t)} z_k \right)^2 = \frac{1}{|\mathcal{K}_i(t)|^2} \left(\sum_{k \in \mathcal{K}_i(t)} z_k^2 + \sum_{l \in \mathcal{K}_i(t)} \sum_{\substack{m \in \mathcal{K}_i(t) \\ m \neq l}} z_l z_m \right), \quad \text{for } i \in \mathcal{N}(t)$$

We have that the initial measurements z_k are i.i.d random variables with the following first and second moments: $\mathbb{E}_\zeta[z_k] = \mu$, $\mathbb{E}_\zeta[z_k^2] = \sigma^2 + \mu^2$ (for any k) and that $\mathbb{E}_\zeta[z_i z_j] = \mu^2$ when $i \neq j$ (since the z_k are independent). Therefore, it results that

$$\mathbb{E}\left[(y_i^{\text{best}}(t))^2\right] = \mathbb{E}_\epsilon \left[\mathbb{E}_\zeta \left[(y_i^{\text{best}}(t))^2 \right] \right] = \mathbb{E}_\epsilon \left[\frac{\sigma^2}{|\mathcal{K}_i(t)|} \right] + \mu^2.$$

Substituting these expressions of $\mathbb{E}[y_i^{\text{best}}(t)]$ and $\mathbb{E}[y_i^{\text{best}}(t)^2]$ into equation (2.4) gives

$$\mathbb{E}[\text{MSE}_{\text{best}}(t)] = \mathbb{E}_\epsilon \left[\frac{\sigma^2}{|\mathcal{K}_i(t)|} \right] \quad \text{for all time } t$$

which concludes the proof. $\square$

Proposition 2.2 provides a general expression of the expected mean squared error of our ideal algorithm $\mathbb{E}[\text{MSE}_{\text{best}}(t)]$ and hence a lower bound on the mean squared error of any realistic algorithm. By realistic algorithm, we mean an algorithm that respects all the assumptions of the problem statement (concerning the use of memory and identifiers) and that uses linear and unbiased estimates.

It is however complicated to deduce an exact theoretical expression of $\mathbb{E}[\text{MSE}_{\text{best}}(t)]$ because it requires computing the expectation of a ratio $\mathbb{E}_\epsilon \left[\frac{1}{|\mathcal{K}_i(t)|} \right]$ which is in general not equal to $\frac{1}{\mathbb{E}_\epsilon[|\mathcal{K}_i(t)|]}$. However, it can be lower-bounded using Lemma 2.1 that applies the Jensen's inequality to probability.

Lemma 2.1. *Given a (strictly) positive random variable X , it holds that*

$$\mathbb{E} \left[\frac{1}{X} \right] \geq \frac{1}{\mathbb{E}[X]}.$$

Proof. We prove this lemma for the sake of completeness. We will only prove the case where X is a discrete random variable. Extension can be done to the continuous case. The Jensen's inequality (in its discrete form) [Der05] state that for a real convex function f , a set of real values x_i ($i = 1 \dots n$) in the domain of f and positive weights a_i ($i = 1 \dots n$) such that $\sum_{i=1}^n a_i = 1$, we have

$$\sum_{i=1}^n a_i f(x_i) \geq f \left(\sum_{i=1}^n a_i x_i \right).$$

It can be applied to the definition of the expectation in probability, (for a discrete random variable X)

$$\mathbb{E}[f(X)] = \sum_k f(k)P[X = k] \quad \text{where the probability density function } P[X = k] \text{ sums to 1 over } k.$$

Hence, the Jensen's inequality applied to probability [Rao11] can be stated as:

$$\mathbb{E}[f(X)] \geq f(\mathbb{E}[X]).$$

In particular, it can be applied to function $f(x) = \frac{1}{x}$ which is convex only for positive values of x . $\square$

From this lemma, we immediately deduce an upper bound on $\mathbb{E}_\epsilon \left[\frac{1}{|\mathcal{K}_i(t)|} \right]$ and then Proposition 2.2 can be extended to the following proposition.

Proposition 2.3 (Lower bound on the MSE of the best ideal algorithm). *The best ideal algorithm from Definition 2.2 has an expected mean squared error that is lower bounded by*

$$\mathbb{E}[\text{MSE}_{\text{best}}(t)] = \mathbb{E}_\epsilon \left[\frac{\sigma^2}{|\mathcal{K}_i(t)|} \right] \geq \frac{\sigma^2}{\mathbb{E}_\epsilon[|\mathcal{K}_i(t)|]} \quad \text{for any } i \in \mathcal{N}(t), \quad (2.5)$$

where $\mathbb{E}_\epsilon$ denotes the expectation over sequences of events that happens in the system, σ^2 is the variance of the agent initial measurements distribution and $\mathcal{K}_i(t)$ the knowledge set of an agent i (from Definition 2.1).

Proof. The size of the knowledge set of an agent $|\mathcal{K}_i(t)|$ is a positive discrete random variable and then, we can apply Lemma 2.1 on $\mathbb{E}_\epsilon \left[\frac{\sigma^2}{|\mathcal{K}_i(t)|} \right]$ to obtain inequality (2.5) which provides a lower bound on the expected mean squared error of the best ideal algorithm at time t (from Proposition 2.2). $\square$

To get a theoretical expression of a lower bound on performance of any *realistic* algorithm, it remains to compute (or upper bound), the expected size of the knowledge set of an agent at time t . The next section presents two ways of obtaining an upper bound on $\mathbb{E}_\epsilon[|\mathcal{K}_i(t)|]$, both based on the fact that there is an expected maximum number of agents that are connected to each other in the union graph of all communications that have taken place until time t .

Before presenting these two ways of obtaining a proved theoretical lower bound on performance of any algorithm, let's quickly analyze the behavior of the performance of this best ideal algorithm. Indeed, as it will be seen next, the proved bounds are conservative and for our algorithmic design purpose, the expected performance of an algorithm are rather compared to an *empirical lower bound* that corresponds to the expected performance of the best ideal algorithm from Definition 2.2 computed empirically (over multiple realizations). Figure 2.1 shows the time evolution of the best performance computed empirically for a specific system ($n = 50$ and $\rho = \frac{\lambda_c}{\lambda_r} = 10$). Here are some remarks illustrated by these graphs but that can be generalized to every open system configuration.

- First, the expected performance is asymptotically constant and hence, similarly than for the gossip algorithm, the expected performance of this best ideal algorithm reaches a steady state. In the situation shown in Figure 2.1, the squared bias reaches more slowly this steady state which is not achieved yet after 500 seconds.
- Thereafter, in Figure 2.1b, we remark that the expected variance of this best ideal algorithm is nonzero. This is due to the fact that the best estimate may be different for every agent since it depends on their own knowledge set. Continuous replacements ensure that the knowledge set of each agent in the system cannot be always identical. When the replacements are less frequent, the expected variance would be lower.
- Similarly, the expected squared bias converges also to a nonzero value. It means that, in general, agents cannot be aware of the initial measurement of all the agents that have been in the system since the very beginning. There is always a nonzero probability (even very small) that two sets of agents are disconnected from each other, which means that they do not exchange any information. This fact is used to develop theoretical bounds in the following section.

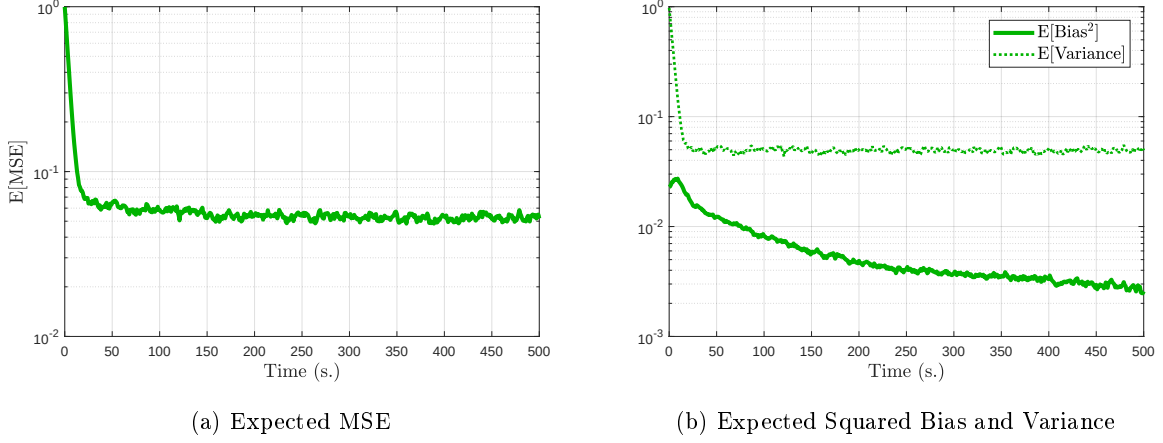


Figure 2.1: Expected performance evolution with time of the best ideal algorithm (from Definition 2.2) for a system with $n = 50$ and $\rho = \frac{\lambda_c}{\lambda_r} = 10$ (with $n\lambda_c = 10$). Expectations have been computed empirically with 500 realization simulations.

- The best ideal algorithm does not necessarily provide a lower bound on the expected variance or the expected squared bias but only on their sum which corresponds to the expected mean squared error. Indeed, we can imagine a stupid algorithm for which all agents choose immediately a same fixed value (say for example 42). This algorithm achieves a very bad expected squared bias but its expected variance goes directly to 0, which is below the expected variance of this best ideal algorithm. Concerning the expected squared bias, we observe here that it increases at the very beginning (before any replacement have occurred) before starting to decrease. This is due to the fact that the sum of the agents estimates is not necessarily preserved during communications and therefore, at some moments, some agents (or rather some agents' measurements z_i) may have more importance into the system than others, leading to a worst expected squared bias. Any algorithm that preserves the sum of the estimates of the agents during the communications (like the gossip averaging algorithm) will not increase the expected squared bias of the system during communications and it will beat the best ideal algorithm on this point, at the very beginning of the process, before the first replacement takes place. But, at any time t , the best ideal algorithm provides the best trade-off between the expected variance and the expected squared bias in the system.

2.2 Upper bound on $\mathbb{E}_\epsilon[|\mathcal{K}_i(t)|]$

In order to upper bound $\mathbb{E}_\epsilon[|\mathcal{K}_i(t)|]$, our approach finds the maximum expected size of the knowledge set $|\mathcal{K}_i(t)|$ for any agent i . Hence, it neglects the variance of the best ideal algorithm (see Figure 2.1) that is due to the fact that agents can have different knowledge set sizes and may not be of the maximum size. The idea is that the number of agents with which an agent i have been in contact directly or not until time t ($|\mathcal{K}_i(t)|$) can be upper bounded by the expected number of agents that have been in the system since the last *restart* and until time t .

Definition 2.3 (Restart). A restart event occurs at time t_{new} if a set of agents $\mathcal{N}(t_{\text{old}})$, that constituted the entire system at time t_{old} ($t_{\text{old}} < t_{\text{new}}$), has been fully replaced at time t_{new} without having had the opportunity to transmit any information to new agents. It holds that $\mathcal{N}(t_{\text{old}}) \cap \mathcal{N}(t_{\text{new}}) = \emptyset$ and that no communication has never taken place between old agents from the set $\mathcal{N}(t_{\text{old}})$ and new agents from the set $\mathcal{N}(t_{\text{new}})$ during all time t (between t_{old} and t_{new}) where the set of agents in the system $\mathcal{N}(t)$ may contained agents from both sets. In simple words, a restart thus happens whenever the entire information that was in the system before it has totally disappeared.

The expected number of agents that have been in the system since the last *restart* and until current time t can be counted as

$$n + \mathbb{E}_\epsilon[N_{\text{rpl}}(t)],$$

where n stands for the number of agents that are in the system and $\mathbb{E}_\epsilon[N_{\text{rpl}}(t)]$ for the expected number of replacements that have occurred since the last restart and until current time t . Remember from Section 1.1 that the size of the system $n = |\mathcal{N}(t)|$ is constant thanks to the replacement assumption. Therefore, an upper bound on $\mathbb{E}_\epsilon[|\mathcal{K}_i(t)|]$ can be deduced from an upper bound on $\mathbb{E}_\epsilon[N_{\text{rpl}}(t)]$:

$$\mathbb{E}_\epsilon[|\mathcal{K}_i(t)|] \leq n + \mathbb{E}_\epsilon[N_{\text{rpl}}(t)]. \quad (2.6)$$

Indeed, the size of the knowledge set of any agent $i \in \mathcal{N}(t)$ is lower than the maximum number of agents that have been potentially connected (even with multiple intermediary agents) to each other in the system at time t , which is $n + N_{\text{rpl}}(t)$.

For the following developments, that aim to find an upper bound on $\mathbb{E}_\epsilon[N_{\text{rpl}}(t)]$ via two different approaches, all expectations are supposed to be taken over the realizations of the sequence of events and hence the ϵ -index in the notation of the expectation is omitted: $\mathbb{E} = \mathbb{E}_\epsilon$. The probability of an event to be a replacement p_r is supposed to be nonzero since the system is open. If this is not the case (i.e. if $p_r = 0$), the system is close and the size of the knowledge set of any agent can simply be bounded with the size n of the system.

2.2.1 Time-dependent approach using a simplified restart definition

This first approach finds an upper bound on $\mathbb{E}[N_{\text{rpl}}(t)]$ using a simplified restart definition (see Definition 2.4). It also uses some basic probability properties that are recall here:

Proposition 2.4. *Let X and Y be two random variables, the expectation of X can be computed as*

$$\mathbb{E}[X] = \sum_y P[Y = y] \times \mathbb{E}[X \mid Y = y]$$

Proposition 2.5. *Let A and B be two probability events, we have that*

$$P[A \cap B] = P[A \mid B] \times P[B]$$

Definition 2.4 (Simplified restart). A *simplified* restart occurs when there are n different successive replacements without having any communication event. It means that in n system events, all agents that were in the system before these events have been replaced and the new agents should start from scratch, without having access to any past information. For simplification, overlapping restarts are not considered. It means that a replacement cannot count for different restarts.

A simplified restart that occurs after the i^{th} event² in the system is denoted R_i . In other words, R_i corresponds to have n successive different replacement events between events $i - n + 1$ and i . R_0 denotes the situation of the system just before starting (i.e., it is not really a restart but only a start).

This simplified definition misses a lot of real restart cases (from Definition 2.3) since it does not consider that communications can take place inside the set of old or new agents and imposes successive replacements. It also only considers succession of n different replacements (and no more than n like it could be the case if there are some repetitions before that the totality of the old agents have been replaced). But it is still a valid way to find a consistent upper bound on the expected number of replacements that have occurred since the last general restart and until current time t , denoted $\mathbb{E}[N_{\text{rpl}}(t)]$. Indeed, missing some restarts can only increase this expected number of replacements. We have that the expected number of replacements since the last general restart (from Definition 2.3) and until time t is lower than the expected number of replacements since the last simplified restart (from Definition 2.4) and until time t denoted $\mathbb{E}[N_{\text{rpl}}^s(t)]$:

$$\mathbb{E}[N_{\text{rpl}}(t)] \leq \mathbb{E}[N_{\text{rpl}}^s(t)]. \quad (2.7)$$

It is explained by the fact that the general restarts are more frequent than the simplified ones. Therefore, this approach bounds $\mathbb{E}[N_{\text{rpl}}^s(t)]$ instead of $\mathbb{E}[N_{\text{rpl}}(t)]$. It allows using the simplified restart definition (see Definition 2.4) for the theoretical developments.

²the events are counted from the beginning of the process.

Here is described the reasoning that leads to the first theoretical bound on $\mathbb{E}[N_{\text{rpl}}(t)]$ (or rather on $\mathbb{E}[N_{\text{rpl}}^s(t)]$) and hence on $\mathbb{E}[|\mathcal{K}_i(t)|]$ and $\mathbb{E}[\text{MSE}_{\text{best}}(t)]$. The resulting bound is thereafter summarized in Proposition 2.6.

Since we want to find a time-dependent bound, it is needed to convert the problem from an event-dependent one to a time-dependent one. Let $N_e(t)$ be a random variable that counts the total number of events that have occurred in the system since the very beginning and until time t . To avoid confusion with probabilistic events, the events that occur into the system (communication and replacements) are called system events in this development.

From Proposition 2.4 applied to random variables $N_{\text{rpl}}^s(t)$ and $N_e(t)$, we have that

$$\mathbb{E}[N_{\text{rpl}}^s(t)] = \sum_{n_e=1}^{\infty} P[N_e(t) = n_e] \times \mathbb{E}[N_{\text{rpl}}^s(t) \mid N_e(t) = n_e] \quad (2.8)$$

Where $P[N_e(t) = n_e] = \frac{(n(\lambda_c + \lambda_r)t)^{n_e} e^{-n(\lambda_c + \lambda_r)t}}{n_e!}$ since the system events are described with a Poisson process of rate $n(\lambda_c + \lambda_r)$.

Using again Proposition 2.4, the conditional probability of equation (2.8) can be expanded in function of the event index k at which the last restart has occurred.

$$\begin{aligned} \mathbb{E}[N_{\text{rpl}}^s(t) \mid N_e(t) = n_e] &= \sum_{k=0}^{n_e} P[\text{last restart at event } k \mid N_e(t) = n_e] \times \mathbb{E}\left[N_{\text{rpl}}^s(t) \mid \begin{array}{c} \text{last restart at event } k \\ N_e(t) = n_e \end{array}\right] \\ &= \sum_{k=0}^{n_e} P\left[(R_k) \cap (\cap_{i=k+1}^{n_e} \overline{R_i})\right] \times \mathbb{E}\left[N_{\text{rpl}}^s(t) \mid (R_k) \cap (\cap_{i=k+1}^{n_e} \overline{R_i})\right] \end{aligned} \quad (2.9)$$

Equation (2.9) uses the notation R_i for a restart event that occurs at i , as introduced in Definition 2.4 and it also ignores the assumption $N_e(t) = n_e$ because it is implicitly contained into the intersection up to n_e . Each factor of equation (2.9) can be upper-bounded independently.

Bound on the expectation in (2.9)

$$\mathbb{E}\left[N_{\text{rpl}}^s(t) \mid (R_k) \cap (\cap_{i=k+1}^{n_e} \overline{R_i})\right] \leq (n_e - k). \quad (2.10)$$

Indeed, there are $n_e - k$ system events since the last restart at event k and until event n_e . The number of replacement events since the last restart can obviously not be higher than the total number of system events since the last restart. Then the upper bound holds. It is a poor bound, especially when the general probability of a system event to be a replacement p_r is small. Actually, we strongly suspect that the upper bound is $p_r(n_e - k)$. Indeed, without other knowledge, the expected number of replacements that take place among $(n_e - k)$ system events is well $p_r(n_e - k)$. But, we want the expected number of replacements among $(n_e - k)$ system events, given that there is no restart that occurs during those events. This has an impact on the expectation that is hard to quantify (and that can be counter-intuitive) and therefore we have not been able to prove this suspected bound rigorously.

Bound on the probability in (2.9)

$$P\left[(R_k) \cap (\cap_{i=k+1}^{n_e} \overline{R_i})\right] = P[\cap_{i=k+1}^{n_e} \overline{R_i} \mid R_k] \times P[R_k] \quad \text{using Proposition 2.5}$$

$$\text{Where } P[R_k] = \begin{cases} 1 & \text{for } k = 0, \\ 0 & \text{for } 1 \leq k \leq n-1, \\ n! \left(\frac{p_r}{n}\right)^n & \text{for } k \geq n. \end{cases} \quad (2.11)$$

Indeed, R_0 is the initial start of the system and always occurs. R_k never occurs for $1 \leq k \leq n-1$ since a restart requires at least n system events (which are the successive different replacements) to be observed.

For $k \geq n$, $P[R_k]$ corresponds to the probability of having n successive different replacement.

We also have that $P[\cap_{i=k+1}^{n_e} \overline{R_i} \mid R_k] = P[\cap_{i=k+n}^{n_e} \overline{R_i}]$ since no overlapping restart are considered and that at least n system events are needed to observe a restart. In order to upper bound this probability of the intersection of no restart events $P[\cap_{i=k+n}^{n_e} \overline{R_i}]$, we restrict it to the probability of the intersection of a subset of no restart events. Instead of computing the probability of having no restart at any of the system events since the last restart (from $k+n$ to n_e), we compute the probability of having no restart only at all the system events that are multiple of n , which is a greater probability. It allows all the events of the intersection to be independent and the following property can be used:

$$\text{For independent random events } A_i, \quad P[\cap_i A_i] = \prod_i P[A_i].$$

In equation, this reasoning gives:

$$P[(\cap_{i=k+n}^{n_e} \overline{R_i})] \leq P\left[\bigcap_{l=\lfloor \frac{k+n}{n} \rfloor}^{\lfloor \frac{n_e}{n} \rfloor} \overline{R_{ln}}\right] = \prod_{l=\lfloor \frac{k+n}{n} \rfloor}^{\lfloor \frac{n_e}{n} \rfloor} (1 - P[R_{ln}]) = \left(1 - n! \left(\frac{p_r}{n}\right)^n\right)^{\lfloor \frac{n_e}{n} \rfloor - \lfloor \frac{k+n}{n} \rfloor + 1},$$

where the last equality uses the fact that $P[R_k] = n! \left(\frac{p_r}{n}\right)^n$ for all $k \geq n$.

To deal with the floor rounding in the power, we can use the fact that for any real x , we have $\lfloor x \rfloor = x - \Delta x$ with $0 \leq \Delta x < 1$.

Hence, $\lfloor x \rfloor - \lfloor y \rfloor = x - y + \Delta y - \Delta x \geq x - y - 1$. Then, we have $\lfloor \frac{n_e}{n} \rfloor - \lfloor \frac{k+n}{n} \rfloor + 1 \geq \frac{n_e - k - n}{n}$. Since the basis of the exponent in the equation is lower than one, the probability becomes:

$$P[(\cap_{i=k+n}^{n_e} \overline{R_i})] \leq \left(1 - n! \left(\frac{p_r}{n}\right)^n\right)^{\frac{n_e - k - n}{n}}. \quad (2.12)$$

Substituting upper bounds ((2.10), (2.11) and (2.12)) into equation (2.9) gives

$$\mathbb{E}[N_{\text{rpl}}^s(t) \mid N_e(t) = n_e] \leq n_e \left(1 - n! \left(\frac{p_r}{n}\right)^n\right)^{\frac{n_e - n}{n}} + \sum_{k=n}^{n_e} (n_e - k) \left(1 - n! \left(\frac{p_r}{n}\right)^n\right)^{\frac{n_e - k - n}{n}}.$$

When this equation is inserted into the first one of this development (equation (2.8)), the expected number of replacements that have occurred since the last simplified restart and until time t ($\mathbb{E}[N_{\text{rpl}}^s(t)]$) can be upper bounded as written in (2.13) and hence a fundamental bound on performance of any realistic algorithm can derived:

Proposition 2.6 (Theoretical time-dependent bound on performance). *In an open multi-agent system of size n with individual communication rate λ_c , individual replacement rate λ_r and replacement probability $p_r = \frac{\lambda_r}{\lambda_c + \lambda_r}$, the expected number of replacements that have occurred since the last simplified restart (see Definition 2.4) and until time t , denoted $\mathbb{E}[N_{\text{rpl}}^s(t)]$, can be upper bounded as follows:*

$$\mathbb{E}[N_{\text{rpl}}^s(t)] \leq \sum_{n_e=1}^{\infty} \frac{(n(\lambda_c + \lambda_r)t)^{n_e} e^{-n(\lambda_c + \lambda_r)t}}{n_e!} \left(n_e \left(1 - n! \left(\frac{p_r}{n}\right)^n\right)^{\frac{n_e - n}{n}} + \sum_{k=n}^{n_e} (n_e - k) \left(1 - n! \left(\frac{p_r}{n}\right)^n\right)^{\frac{n_e - k - n}{n}} \right). \quad (2.13)$$

The expression of this infinite sum has been computed using Mathematica and is provided in Appendix 2.A of this chapter. Asymptotic value of this bound, for time tending to infinity, provides a global bound (valid for any time t) on $\mathbb{E}[N_{\text{rpl}}^s(t)]$ (computed using Mathematica):

$$\mathbb{E}[N_{\text{rpl}}^s(t)] \leq \lim_{t \rightarrow \infty} \mathbb{E}[N_{\text{rpl}}^s(t)] \leq \frac{n! \left(\frac{p_r}{n}\right)^n \left(1 - n! \left(\frac{p_r}{n}\right)^n\right)^{\frac{1}{n} - 1}}{\left(\left(1 - n! \left(\frac{p_r}{n}\right)^n\right)^{1/n} - 1\right)^2}, \quad \text{for } 0 < p_r < 1 \text{ and for any time } t \quad (2.14)$$

This upper bound on $\mathbb{E}[N_{\text{rpl}}^s(t)]$, and hence (from equations (2.6) and (2.7)) on $\mathbb{E}[\mathcal{K}_i(t)] \leq n + \mathbb{E}[N_{\text{rpl}}^s(t)]$ allows (from Proposition 2.3) having a lower bound on the expected mean square error of the best ideal

algorithm (see Definition 2.2) and therefore on the performance ($\mathbb{E}[\text{MSE}(t)]$) of any realistic algorithm (see Proposition 2.1):

$$\mathbb{E}[\text{MSE}(t)] \geq \mathbb{E}[\text{MSE}_{\text{best}}(t)] \geq \frac{\sigma^2}{\mathbb{E}[|\mathcal{K}_i(t)|]} \geq \frac{\sigma^2}{n + \mathbb{E}[N_{\text{rpl}}^s(t)]} \quad \text{for any } i \in \mathcal{N}(t).$$

where upper bound on $\mathbb{E}[N_{\text{rpl}}^s(t)]$ is given by (2.13). Corresponding global lower bound on $\mathbb{E}[\text{MSE}(t)]$, valid for any time t , can be obtained by using global upper bound on $\mathbb{E}[N_{\text{rpl}}^s(t)]$ given at equation (2.14).

As suggests in Figure 2.2, this proved lower bound (in red) is obviously not tight to the true expected performance of the best ideal algorithm that can be computed empirically (in green). It can be explained by different reasons. The two mains reasons depend on the approach used:

- The Jensen's inequality used in Lemma 2.1 is not always tight. As suggested in Figure 2.2 when comparing green and black curves, we have

$$\mathbb{E}[\text{MSE}_{\text{best}}(t)] = \mathbb{E}\left[\frac{\sigma^2}{|\mathcal{K}_i(t)|}\right] > \frac{\sigma^2}{\mathbb{E}[|\mathcal{K}_i(t)|]},$$

This phenomenon is accentuated when the replacements are quite frequent since it implies more difference between the knowledge sets of the agents and thus a larger difference between $\mathbb{E}\left[\frac{\sigma^2}{|\mathcal{K}_i(t)|}\right]$ and $\frac{\sigma^2}{\mathbb{E}[|\mathcal{K}_i(t)|]}$.

- As mentioned, this approach, based on the restart idea, aims to find the maximum size of the knowledge set $|\mathcal{K}_i(t)|$ of any agent i and hence it neglects the variance of the best performance in part due to the fact that agents can have different knowledge set sizes, especially when the replacements are quite frequent. This variance also explains why the empirical expected performance of the best ideal algorithm has slight oscillations since it may vary abruptly with time and from one realization to another (a totally smooth curve should require much more than 500 realizations to compute the expectation).

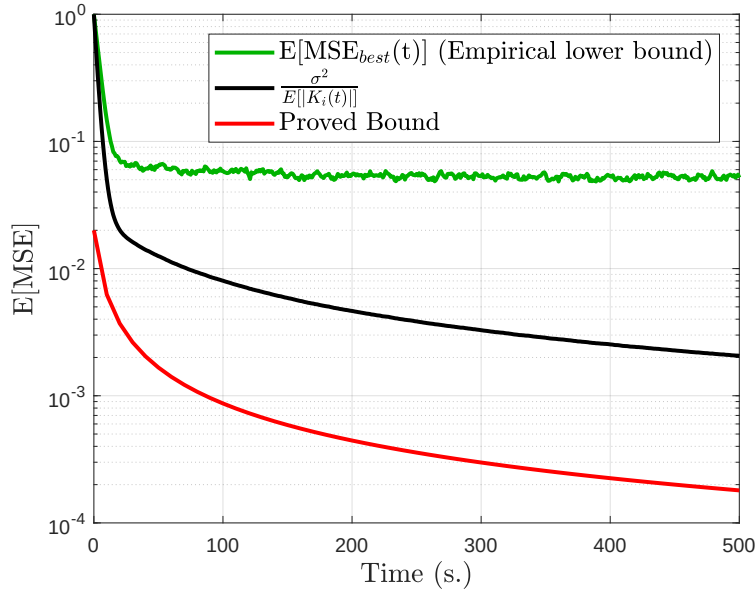


Figure 2.2: Evolution with time of the empirical (from Proposition 2.1) and theoretical (from Proposition 2.6) lower bounds on the performance (MSE) of a system with $n = 50$ and $\frac{\lambda_c}{\lambda_r} = 10$ (with $n\lambda_c = 10$). $\mathbb{E}[\text{MSE}_{\text{best}}(t)]$ have been computed empirically with 500 realization simulations.

Other reasons related to simplifications into computation, explained during the development of the bound on $\mathbb{E}[N_{\text{rpl}}(t)]$, also lead to a conservatism of the proved bound:

- The simplified restart definition which miss a lot of true restarts.
- The bound given at (2.10) on the expected number of replacements that have occur in the system until time t and since the last restart, knowing the moment of the last restart, is obviously not tight.
- Also, the bound on the probability that no restart has occurred after event k (given by (2.12)), besides using the simplified definition of a restart, discards many events to only have independent ones which allows bounding more easily the probability of the intersection. It obviously over-estimate the probability.

Figure 2.2 also shows that while empirical lower bound reaches quickly a steady state performance, the value of the theoretical bound decreases slowly. However, a global bound, valid for any time t , on $\mathbb{E}[N_{\text{rpl}}^s(t)]$ (and hence on $\mathbb{E}[\text{MSE}_{\text{best}}(t)]$) can be obtained when time goes to infinity. It corresponds to a bound on the expected number of replacements that occur between two restarts in general. This limit has been computed using Mathematica and is given by equation (2.14) in Proposition 2.6. This global version of the bound will be useful to compare this first theoretical bound with the second one presented below that aims to compute only a global bound that holds for any time t . This latter bound also aims to solve computational conservatism issues from this first theoretical bound but not the approach-based conservatism issues since the same *restart* idea is preserved.

2.2.2 Global approach using the general restart definition

This approach aims to compute a global bound on $\mathbb{E}[N_{\text{rpl}}(t)]$, the expected number of replacements that have occurred into the system since the last restart. This approach aims to use the general restart definition (Definition 2.3). The bound is called global because it is independent of the time and hence valid for any time. It actually corresponds to compute a bound on

$$\mathbb{E}[N_{\text{inter}}] = \lim_{t \rightarrow \infty} \mathbb{E}[N_{\text{rpl}}(t)],$$

where $\mathbb{E}[N_{\text{inter}}]$ is the expected number of replacements between two restarts in the system.

The resulting bound is summarized further in Proposition 2.8. The ideas and reasoning that have led to this bound are first described. The idea is to consider at any time t , a partition of the agents in the system $\mathcal{N}(t)$ into two sets of agents: the *old* ones $\mathcal{A}_{\text{old}}(t)$ that are supposed to have communicated with each other and having each (potentially) access to information of other *old* agents (and maybe *new* agents too) and a set of *new* ones $\mathcal{A}_{\text{new}}(t)$ that have not communicated yet with old agents and may generate a restart later. The partition of course respects: $\mathcal{N}(t) = \mathcal{A}_{\text{old}}(t) \cup \mathcal{A}_{\text{new}}(t)$ and $\mathcal{A}_{\text{old}}(t) \cap \mathcal{A}_{\text{new}}(t) = \emptyset$. When an agent comes into the system, it belongs to the set of new agents $\mathcal{A}_{\text{new}}$. And when a *new* agent $i \in \mathcal{A}_{\text{new}}$ communicates with any *old* agent $j \in \mathcal{A}_{\text{old}}(t)$, i passes from the set $\mathcal{A}_{\text{new}}$ to $\mathcal{A}_{\text{old}}$ since it get in contact with a *old* agent and have now potentially also access to information about all other *old* agents. A restart occurs when all the agents in the systems are *new* ones: $\mathcal{N}(t) = \mathcal{A}_{\text{new}}(t)$. Since the system size is constant $n = |\mathcal{N}(t)|$, it is sufficient to analyze the size of each set of agents.

Markov chain We define an absorbing Markov chain with state $k = |\mathcal{A}_{\text{new}}(t)|$ which is the number of *new* agents that are in the system. If $k = n$, it means that all the agents in the system are *new* and have not exchanged any information with past agents of the system. This state corresponds thus to observing a restart. This is the absorbing state of our chain. There is a total of $n + 1$ different states: $k = 0, 1, \dots, n$. Using absorbing Markov chain theory, it is possible to compute the expected number steps (i.e. events) needed to reach the absorbing state (i.e. a restart).

Let us first formalize the Markov Chain and its transition probability matrix. Let p_r denote the probability for an event to be a replacement (and hence probability $1 - p_r$ to be a communication) and let c_k denote the probability for a communication to be a communication between the two sets of agents ($\mathcal{A}_{\text{old}}(t)$ and $\mathcal{A}_{\text{new}}(t)$) at state k . Since there is $k = |\mathcal{A}_{\text{new}}(t)|$ *new* agents and $n - k = |\mathcal{A}_{\text{old}}(t)|$ *old* ones, the probability, at state k , that a communication involves a *new* agent and an *old* is

$$c_k = \frac{2k(n - k)}{n(n - 1)}.$$

Here are listed the transition probabilities of the Markov chain which is represented in Figure 2.3.

- When the system reaches the absorbing state ($k = n$), it remains there with probability 1.
- The system passes from any state $k > 0$ to state $k - 1$ with probability $(1 - p_r)c_k$ which is the probability of an event to be a communication between $i \in \mathcal{A}_{old}(t)$ and $j \in \mathcal{A}_{new}(t)$.
- The system passes from any state $k < n$ to $k + 1$ with probability $\frac{n-k}{n}p$ which is the probability of an event to be a replacement of an *old* agent $i \in \mathcal{A}_{old}(t)$
- The system remains at the same state k with probability $\frac{k}{n}p_r + (1 - p_r)(1 - c_k)$ which is the probability of an event to be a replacement of a *new* agent $i \in \mathcal{A}_{new}(t)$ or an intern communication between two *new* or two *old* agents).

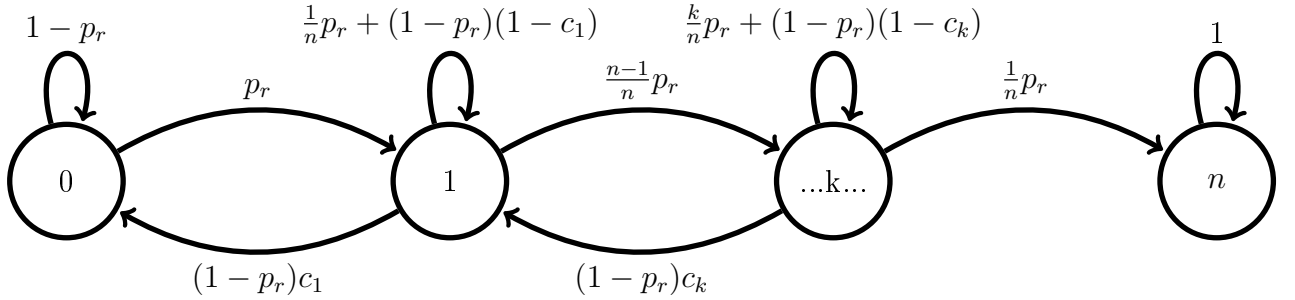


Figure 2.3: Representation of the absorbing Markov chain.

Results on Markov chain [GS06] allows to compute the expected number of visits to each transient state before reaching the final absorbing state ($k = n$) for a given starting state. For our problem, the starting state is always $k = 0$, meaning that all the agents in the system at the beginning are considered to (potentially) know each other and to be *old* agents. The beginning can correspond to the system after any of its starts or restarts. At each transient state, the system evolves to another state through a replacement with probability p_r and through a communication with probability $1 - p_r$. It is therefore possible to count the expected number of replacements that occur in the system before reaching the absorbing state, i.e. a restart.

Let P be the transition matrix of the Markov chain in a canonical form, since there is only one absorbing state, P can be expressed as:

$$P = \begin{pmatrix} Q^{n \times n} & R^{n \times 1} \\ 0^{1 \times n} & 1 \end{pmatrix}.$$

Theorem 2.2 (The fundamental matrix in absorbing Markov Chain [GS06]). *For an absorbing Markov chain with a canonical transition matrix P , the matrix $I - Q$ has an inverse N that is such that*

$$N = I + Q + Q^2 + \dots = \sum_{i=0}^{\infty} Q^i = (I - Q)^{-1}.$$

I is the identity matrix. N is called the fundamental matrix of the Markov Chain. The ij -entry n_{ij} of the matrix N is the expected number of visits to transient state j (until being absorbed), given that it starts in transient state i . The initial state is counted if $i = j$.

Here, we are only interested in the first line n_0^T of the fundamental matrix N which corresponds to starting from state $k = 0$. The system to solve to find n_0 is

$$N(I - Q) = I \Leftrightarrow (I - Q)^T n_0 = e_0, \quad e_0 \text{ is the vector } (1, 0, \dots, 0)^T. \quad (2.15)$$

where n_0 is a column vector corresponding to the first line of the fundamental matrix N . It corresponds to the expected number of visits to each transient state ($k = 0, \dots, n - 1$) before reaching the final absorbing state ($k = n$). The system to solve is tridiagonal and even if its size n is unspecified, it can be written as a second order recurrence equation, which allows to deduce a general expression for n_0 . From

the expression of n_0 is possible to compute the expected number of events and replacements that occur in the system between two restarts as stated in Proposition 2.7. *Between two restarts* means actually since the beginning (state $k = 0$) and until the first restart (absorbing state $k = n$). The starting state is considered to be always $k = 0$ since when a restart occurs, all the new agents that fully constitute the system ($\mathcal{N}(t) = \mathcal{A}_{new}(t)$) can potentially already know each other³.

Proposition 2.7. *The expected number of events $\mathbb{E}[M]$ that occurs in a system of size n between two restarts (see Definition 2.3), can be computed as*

$$\mathbb{E}[M] = \sum_{i=0}^{n-1} n_{0i} \quad (2.16)$$

where n_{0i} are the solution of the system (2.15).

The corresponding expected number of replacements that occurs in the system between two restarts (denoted $\mathbb{E}[N_{inter}]$) can be computed as

$$\mathbb{E}[N_{inter}] = p_r \mathbb{E}[M], \quad (2.17)$$

where p_r is the probability of an event to be a replacement.

In order to prove that proposition, we recall the Wald's Identity (from renewal processes theory)

Theorem 2.3 (Wald's Identity from [Hei]). *Let $X_1, X_2, \dots$ be i.i.d random variables with common finite mean, and M be a stopping time with $\mathbb{E}[M] < \infty$, then $\mathbb{E}[X_1 + \dots + X_M] = \mathbb{E}[M]\mathbb{E}[X_1]$.*

The random variable M is called a stopping time for the stochastic process $\{X_k, k \geq 1\}$ if M takes values in the positive integers set and for each positive integer m , the event $\{M \leq m\}$ is independent of $X_{m+1}, X_{m+2}, \dots$

Proof of Proposition 2.7. The first equation (2.16) of the proposition holds by definition of $\mathbb{E}[M]$ and of the fundamental matrix N from Theorem 2.2.

The second equation (2.17) is an application of the Wald's identity. In our situation, we observe system events until a restart occurs. The i.i.d random variables X_k are the number of replacements that occur in the system at event k . It can only be 0 or 1. We have $\mathbb{E}[X_k] = p_r$ for any event k . The number of replacements in the system from the beginning until a restart occurs can be computed as:

$$N_{inter} = \sum_{k=1}^M X_k.$$

where M corresponds to the index of the first system event where a restart occurs. The value of M only depends on what happens in events 1 to M (and not after). It is therefore a stopping time and the Wald's identity can be used to conclude the proof. $\mathbb{E}[N_{inter}] = \mathbb{E}[M]\mathbb{E}[X_1] = \mathbb{E}[M]p_r$. $\square$

From system to recurrence: We now explain how to solve the system of unspecified size from (2.15). The idea is to convert it into a recurrence equation. From the above description of the transition probabilities, the matrix Q can be summarized as follows:

$$Q_{ij} = \begin{cases} (1 - p_r)c_i & \text{(lower diagonal) if } j = i - 1 \text{ (and } i \neq 0), \\ \frac{ip_r}{n} + (1 - p_r)(1 - c_i) & \text{(main diagonal) if } j = i, \\ \frac{n-i}{n}p_r & \text{(upper diagonal) if } j = i + 1 \text{ and } i \neq n - 1, \\ 0 & \text{otherwise.} \end{cases} \quad \text{for } \begin{matrix} i = 0, \dots, n - 1 \\ j = 0, \dots, n - 1 \end{matrix}$$

Each row of Q sums up to 1, except the last one (row $n - 1$) which corresponds to the only state that has a nonzero probability to jump (directly) toward the absorbing state ($k = n$). The matrix $(I - Q)^T$ that corresponds to the matrix of the system to solve (2.15) is

$$(I - Q)_{ij}^T = \begin{cases} -\frac{n-j}{n}p_r & \text{(lower diagonal) if } j = i - 1 \text{ (and } i \neq 0), \\ 1 - \frac{jp_r}{n} - (1 - p_r)(1 - c_j) & \text{(main diagonal) if } j = i, \\ -(1 - p_r)c_j & \text{(upper diagonal) if } j = i + 1 \text{ and } i \neq n - 1, \\ 0 & \text{otherwise.} \end{cases} \quad \text{for } \begin{matrix} i = 0, \dots, n - 1 \\ j = 0, \dots, n - 1 \end{matrix}$$

³Except maybe the last agent that arrives into the system and leads to the restart. A slightly more precise bound can maybe be derived by choosing initial state as $k = 1$

i stands for the equation indices and j for the unknown indices. Since the matrix is tridiagonal, each equation $i = 1, \dots, n-2$ involves 3 unknowns n_{0i-1} , n_{0i} and n_{0i+1} . Therefore, these equations can be written as a second order recurrence equation:

$$-\left(\frac{n-i+1}{n}\right)p_r n_{0i-1} + \left(1 - \frac{ip_r}{n} - (1-p_r)(1-c_i)\right)n_{0i} - (1-p_r)c_{i+1}n_{0i+1} = 0 \quad \text{for } i = 1, \dots, n-2. \quad (2.18)$$

It requires two initial conditions to be solved. Actually, the two last unknowns n_{0n-1} and n_{0n-2} can be computed easily. Note that for matrix $(I-Q)^T$, each column sums to 0, except the last one (column $n-1$). Indeed, $(I-Q)^T = I-Q^T$ and each column (except column $n-1$) of both matrices I and Q^T sums to 1. Summing all equations together allows therefore to eliminate all unknown except the last one n_{0n-1} it results the following equation :

$$\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (I-Q)_{ij}^T n_{0j} = \sum_{i=0}^{n-1} e_{1,i} \quad \Leftrightarrow \quad \left(- (1-p_r)(c_{n-1}) + 1 - \frac{n-1}{n}p_r - (1-p_r)(1-c_{n-1})\right)n_{0n-1} = 1$$

$$\Leftrightarrow \quad n_{0n-1} = \frac{n}{p_r} \quad (2.19)$$

The relation $n_{0n-1} = \frac{n}{p_r}$ have been verified using Matlab for different systems configurations. Finally, last equation (equation $n-1$) of the system (2.15) involves only two unknowns : n_{0n-1} and n_{0n-2} . Solving it allows to deduce an expression for n_{0n-2} (since equation (2.19) already gives an expression for n_{0n-1}) that can be used as second initial condition to solve recurrence (2.18) and get an expression of any n_{0i} for all i . Equation $n-1$ from (2.15) is:

$$-\frac{2}{n}p_r n_{0n-2} + \left(1 - \frac{n-1}{n}p_r - (1-p_r)(1-c_{n-1})\right)n_{0n-1} = 0 \quad \Leftrightarrow \quad n_{0n-2} = \frac{n(2-p_r)}{2p_r^2}. \quad (2.20)$$

The recurrence equation (2.18) with initial conditions (2.19) and (2.20) has been solved using Mathematica. The solution n_{0i} (for $i = 0, \dots, n-1$) is not an easy expression and hence is not shown here but is available in Appendix 2.A of this chapter. Having an expression for each n_{0i} (for $i = 0, \dots, n-1$) allows computing the expected number of replacements that occur between two restarts thanks to Proposition 2.7 that states that $\mathbb{E}[N_{\text{inter}}] = p_r \sum_{i=0}^{n-1} n_{0i}$, and hence deriving the following fundamental lower bound on performance.

Proposition 2.8 (Theoretical global bound on performance). *In an open multi-agent system of size n , the expected number of replacements that have occurred in the system since the last restart and until time t ($\mathbb{E}[N_{\text{rpl}}(t)]$) is bounded by the expected number of replacements that occur in the system between two restarts ($\mathbb{E}[N_{\text{inter}}]$). See Definition 2.3 for the definition of a restart. From Proposition 2.7, we have:*

$$\mathbb{E}[N_{\text{rpl}}(t)] \leq \mathbb{E}[N_{\text{inter}}] = p_r \sum_{i=0}^{n-1} n_{0i}, \quad \text{for any time } t. \quad (2.21)$$

where expression of n_{0i} can be computed using the recurrence equation (2.18) with initial conditions (2.19) and (2.20). The expression of the solution of this recurrence is available in Appendix 2.A of this chapter. This global upper bound on $\mathbb{E}[N_{\text{rpl}}(t)]$, and hence (from equation (2.6)) on $\mathbb{E}[|\mathcal{K}_i(t)|] \leq n + \mathbb{E}[N_{\text{rpl}}(t)]$ allows (from Proposition 2.3) having a global lower bound on the expected mean squared error of the best ideal algorithm (see Definition 2.2) and therefore on the performance ($\mathbb{E}[\text{MSE}(t)]$) of any realistic algorithm (see Proposition 2.1).

$$\mathbb{E}[\text{MSE}(t)] \geq \mathbb{E}[\text{MSE}_{\text{best}}(t)] \geq \frac{\sigma^2}{\mathbb{E}[|\mathcal{K}_i(t)|]} \geq \frac{\sigma^2}{n + \mathbb{E}[N_{\text{rpl}}(t)]} \quad \text{for any } i \in \mathcal{N}(t),$$

where a global upper bound on $\mathbb{E}[N_{\text{rpl}}(t)]$ is given by (2.21).

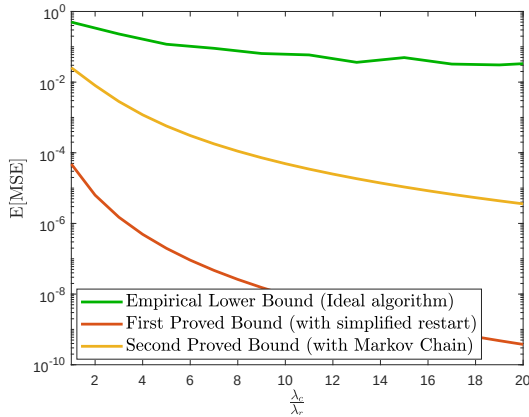
2.3 Comparison of the different lower bounds on performance

The three different lower bounds on the performance (the *true* one from Proposition 2.1 for which no exact theoretical expression is available, the time-dependent one from Proposition 2.6 and the global

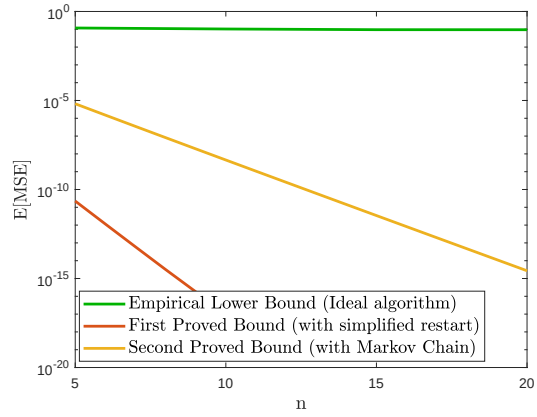
one from Proposition 2.8) are compared in Figure 2.4. To allow a fair comparison, they are compared on their global aspect which means that we compare them at the time t for which they have the lowest value, i.e. for large time t . In concrete terms, Figure 2.4 shows in green, the value of the true bound from Proposition 2.1 after 500 seconds⁴ computed empirically through numerical simulations, in red, the value of the first time-dependent theoretical bound from Proposition 2.6 when time goes to infinity (i.e. in its global form) and in yellow the value of second theoretical bound which is valid for any time t .

As expected, the two theoretical bounds (in yellow and red) are not tight at all to the true bound (in green) derived from the best ideal algorithm (see Definition 2.2) for which the expected performance is computed empirically (using different numerical simulations). The second one (Proposition 2.8) is still better than the first one (Proposition 2.6) since it computes the exact expected number of replacements that occurs between two restarts $\mathbb{E}[N_{\text{inter}}]$ while the first approach can only upper bound it. The worst bound (from Proposition 2.6) is however the one with the simplest theoretical expression (see equation (2.14) in Proposition 2.6).

The bounds evolve logically with the systems parameters: both theoretical proved lower bounds decrease exponentially (note that the y-scale is logarithmic) with the size n of the system: the larger the system, the lower the chance of observing a restart. Similarly, they also decrease with the rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$ that corresponds to the expected number of communications that occurs between two replacements. Intuitively, if there are more communications, the replacement probability is lower and so is the chance of observing a restart into the system. The performance of the best ideal algorithm, for its part, also decreases (much less) when the replacement become less frequent (i.e. the rate ratio ρ increases) and it is almost insensitive to the variation of the system size n . Indeed, even its expected squared bias is in general lower for larger system, it is counterbalanced by the increase of the expected variance in the system (since it takes longer for the information to spread throughout the entire system) and the total expected mean squared error is almost constant with n .



(a) Evolution with $\rho = \frac{\lambda_c}{\lambda_r}$ for fixed size $n = 5$



(b) Evolution with n for fixed rate ratio $\rho = \frac{\lambda_c}{\lambda_r} = 5$

Figure 2.4: Comparison for different global empirical and proved lower bounds on the expected mean squared error for different system configurations (with constant $n\lambda_c = 10$). In green, the empirical lower bound from Proposition 2.1 after 500s computed empirically with 500 realization simulations. In red, the first proved bound when $t \rightarrow \infty$ from Proposition 2.6. And in yellow, the second proved global bound from Proposition 2.8.

⁴As suggested in Figure 2.1, the expected mean squared error of the best ideal algorithm has already reached a steady state after 500 seconds. This may be not totally the case for some systems that implies very long transient state (e.g. very large system with small pr).

Conclusion about the performance limitations

In conclusion, this chapter proves (via two different ways) the existence of a lower bound on the expected mean square error of any algorithm that aims to solve the external averaging problem by generating linear unbiased estimates. The proofs rely on the lower bounding of the expected mean squared error of the best ideal algorithm that solves the problem. These theoretical bound are useless in practice since they are not tight enough. However, the performance of the best ideal algorithm, that provides a very good lower bound on performance of any realistic algorithm, can be computed empirically using the simulation tool I developed. This lower bound is called the *Empirical Lower Bound* in the next chapters, even if it still relies on theoretical aspects. This *Empirical Lower Bound* will be abundantly used for the design of new algorithms, presented in the next chapters, since it allows to analyze the progression margin that is still achievable. The following chapter shows, among other things, that the gossip averaging algorithm is quite far from the lower bound in many system configurations and that there is room for new algorithms that achieve this progression margin.

2.A Appendix for theoretical expressions in the performance lower bounds

First proved bound from Proposition 2.6

The first time-dependent proved bound on the expected mean squared error (from Proposition 2.6) depends on the upper bound on $\mathbb{E}[N_{\text{rpl}}(t)]$ (see equation (2.13)) for which the expression is an infinite sum. This sum can be computed using Mathematica and it gives the following expression for the upper bound on $\mathbb{E}[N_{\text{rpl}}(t)]$:

$$\mathbb{E}[N_{\text{rpl}}(t)] \leq \frac{(1-f)^{\frac{1}{n}-2} e^{-\lambda^G t}}{\left((1-f)^{1/n} - 1\right)^2} \left[f e^{\lambda^G t (1-f)^{1/n}} \left((\lambda^G t - n) \left((1-f)^{1/n} - 1 \right) - 1 \right) + f n \left((1-f)^{1/n} - 1 \right) + f e^{\lambda^G t} \right. \\ \left. - f^2 \left(e^{\lambda^G t} - 1 \right) + \lambda^G t \left((1-f)^{1/n} - 1 \right)^2 e^{\lambda^G t (1-f)^{1/n}} \right]$$

where $f = n! \left(\frac{p_r}{n} \right)^n$, and $\lambda^G = n(\lambda_c + \lambda_r)$.

Second proved bound from Proposition 2.8

The second lower bound on the performance from Proposition 2.8 involves the solution n_{0i} of a recurrence relation (see equation (2.18)) that has been computed using Mathematica:

$$n_{0i} = \frac{n e^{-q} q^i}{2 p_r (p_r - 1) (i - n) \Gamma[i + 2]} \left[(-1)^{i+1} (n - 1) p_r \Gamma[i + 2] \Gamma[-i, -q] \right. \\ \left. + (i + 1) (-1)^n \Gamma[n] \left((n - 1) (p_r - 2) \Gamma[1 - n, -q] + 2(p_r - 1) \Gamma[2 - n, -q] \right) \right]$$

where $q = \frac{(n-1)p_r}{2(1-p_r)}$ and where the Gamma function $\Gamma[z]$ is defined as:

$$\Gamma[z] = \int_0^\infty t^{z-1} e^{-t} dt \quad \text{for } z \in \mathbb{C} \text{ and } \mathcal{R}(z) > 0.$$

It is a prolongation of the factorial function to any complex number with positive real part. For any positive integer input n , we have $\Gamma[n] = (n - 1)!$

The (upper) incomplete Gamma function $\Gamma[z, x]$ is defined as:

$$\Gamma[z, x] = \int_x^\infty t^{z-1} e^{-t} dt \quad \text{for } z \in \mathbb{C} \text{ and } \mathcal{R}(z) > 0.$$

We have that $\Gamma[z, 0] = \Gamma[z]$.

Chapter 3

Gossip averaging algorithm

The gossip averaging algorithm is a simple averaging consensus algorithm. It has already been briefly presented in the state-of-the-art section and in Section 1.2 from Chapter 1. The algorithm can be summarized as follows:

Gossip averaging algorithm:

$$\text{Initialization for all } i \in \mathcal{N}(t): \quad x_i(0) = z_i, \quad (3.1)$$

$$\text{Interaction between a pair of agents } i, j \in \mathcal{N}(t): \quad x_i^+ = x_j^+ = \frac{x_i + x_j}{2}, \quad (3.2)$$

$$\text{Estimate of agent } i \in \mathcal{N}(t): \quad y_i(t) = x_i(t).$$

For this algorithm the estimates of the agents, denoted y_i , and the variables x_i coincide and can be used indifferently. In general, only x_i are used to denote both the variables of the agents and their estimates of μ (omitting the time dependence notation). In *closed* system, the main result to remember is that the sum of x_i values are preserved and hence the agents are converging to a consensus which corresponds to the average of their initial values.

This chapter first presents the analysis of an *open* multi-agent system subject to the gossip averaging algorithm that are detailed in [HM16] and [HM17] and deduces the performance of the algorithm for the external averaging estimation problem (see the problem statement in Section 1.1.1). It also presents the main issues that the algorithm encounters to solve this problem. These results are largely used in the following of this master thesis: first to use the gossip averaging algorithm as a baseline algorithm for which theoretical guarantees are available and second to exploit the principles of its analysis in open systems and its main issues to develop new efficient algorithms. That is why this entire chapter is dedicated to this algorithm. The last section of the chapter compares the performance of the averaging algorithm with the empirical lower bound developed in Chapter 2 in order to show the possible progression margin that can be achieved by new algorithms.

3.1 Analysis of the gossip averaging algorithm

This section presents the results concerning the evolution of an *open* multi-agent system subject to the gossip averaging algorithm that are detailed in [HM16] and [HM17]. This allows deducing the performance of the algorithm for the external averaging estimation problem. The analysis pursued to obtain these results is also presented in order to be able to re-use similar reasonings for the design of new algorithms.

The analysis relies on the description of the system through size-independent and self-contained descriptors:

$$\mathbb{E}[\bar{x}^2] \quad \text{and} \quad \mathbb{E}[\bar{x}^2] \quad \text{and possibly also} \quad \mathbb{E}[\bar{x}]$$

$$\text{where} \quad \bar{x} = \frac{1}{n} \sum_{i \in \mathcal{N}} x_i \quad \text{and} \quad \bar{x}^2 = \frac{1}{n} \sum_{i \in \mathcal{N}} x_i^2.$$

The expected performance (the mean squared error, the squared bias and the variance defined in Section 1.1.3) that the algorithm achieves for the external averaging problem can be deduced from these descriptors:

$$\begin{aligned}\mathbb{E}[\text{Bias}^2] &= \mathbb{E}[(\bar{x} - \mu)^2] = \mathbb{E}[\bar{x}^2] + \mu^2 - 2\mu\mathbb{E}[\bar{x}], \\ \mathbb{E}[\text{Var}] &= \mathbb{E}\left[\frac{1}{n} \sum_{i \in \mathcal{N}} (x_i - \bar{x})^2\right] = \mathbb{E}[\bar{x}^2] - \mathbb{E}[\bar{x}]^2, \\ \mathbb{E}[\text{MSE}] &= \mathbb{E}[\text{Bias}^2] + \mathbb{E}[\text{Var}] \quad \text{from equation (1.2)}\end{aligned}$$

Note that the time and realization dependence of the different variables is omitted here and in the following development.

3.1.1 Descriptor evolution system

For the following, the distribution mean μ is supposed to be zero, without loss of generality. It is therefore sufficient to track the evolution of the expected vector $\mathbb{E}X = [\mathbb{E}[\bar{x}^2], \mathbb{E}[\bar{x}]]^T$ to describe the performance of the algorithm: $\mathbb{E}[\text{Bias}^2] = [1, 0]\mathbb{E}X$, $\mathbb{E}[\text{Var}] = [-1, 1]\mathbb{E}X$ and $\mathbb{E}[\text{MSE}] = [0, 1]\mathbb{E}X$. The idea developed in [HM17] is to consider a discrete time evolution so that the time samples correspond to the instants where an event occurs. $\mathbb{E}X_k$ denotes the value of the descriptor vector after event k . The evolution of vector $\mathbb{E}X_k$ can be written as a linear recurrence system:

$$\mathbb{E}X_k = A\mathbb{E}X_{k-1} + b.$$

The initial vector is $\mathbb{E}X_0 = \left[\frac{\sigma^2}{n}, \sigma^2\right]^T$ since the initial value of agent each agent i is set to its own measurement z_i ($x_i(0) = z_i$) that is independently drawn from an identical distribution with a zero mean $\mu = 0$ and a variance σ^2 .

The event k can be either a replacement with probability p_r or a communication with probability $1 - p_r$. The evolution of $\mathbb{E}X$ will be different in both cases. Only the mean ideas of the reasoning are explained. For complete proofs of the results, see [HM17].

A communication in gossip algorithm preserves the sum of x_i and therefore preserves $\bar{x}$. The bias is thus unchanged. However, each gossip communication reduces a bit the variance ($\mathbb{E}[\text{Var}]' = (1 - \frac{1}{n})\mathbb{E}[\text{Var}]$) and hence also a bit the global MSE. The linear update of $\mathbb{E}X$ vector for a gossip communication is described in the following lemma.

Lemma 3.1 (Update of the descriptor vector $\mathbb{E}X$ with a communication (for the Gossip algorithm) [HM17]). *Let k be a communication event in an open multi-agent system of size n and with $\mu = 0$ subject to the gossip averaging algorithm (see the problem statement in Section 1.1.1 for a general definition of a communication event or an agent). For a communication in the gossip algorithm, the update is defined by equation (3.2) and the value of the descriptor vector $\mathbb{E}X = [\mathbb{E}[\bar{x}^2], \mathbb{E}[\bar{x}]]^T$ is updated according to*

$$\mathbb{E}X_k = A_c\mathbb{E}X_{k-1}, \quad \text{with} \quad A_c = \begin{bmatrix} 1 & 0 \\ \frac{1}{n} & 1 - \frac{1}{n} \end{bmatrix}.$$

A replacement impacts both components of the $\mathbb{E}X$ since it implies the departure of an agent and the arrival of a new one. The evolution of the descriptor vector $\mathbb{E}X$ is described in the following lemma.

Lemma 3.2 (Update of the descriptor vector $\mathbb{E}X$ with a replacement (for the Gossip algorithm) [HM17]). *Let k be a replacement event in an open multi-agent system of size n and with $\mu = 0$ subject to the gossip averaging algorithm (see the problem statement in Section 1.1.1 for a general definition of a replacement event or an agent). For a replacement, the gossip algorithm initializes the new agent i according to equation (3.1) and the value of the descriptor vector $\mathbb{E}X = [\mathbb{E}[\bar{x}^2], \mathbb{E}[\bar{x}]]^T$ is updated according to*

$$\mathbb{E}X_k = A_r\mathbb{E}X_{k-1} + b_r, \quad \text{with} \quad A_r = \begin{bmatrix} 1 - \frac{2}{n} & \frac{1}{n^2} \\ 0 & 1 - \frac{1}{n} \end{bmatrix}, \quad b_r = \sigma^2 \begin{bmatrix} \frac{1}{n^2} \\ \frac{1}{n} \end{bmatrix}. \quad (3.3)$$

The independent term b_r appears due to arrival of a new agent i with its own measurement z_j of variance σ^2 .

The resulting linear recurrence system that fully describes the evolution of $\mathbb{E}X$ is therefore described in the following theorem.

Theorem 3.3 (Update of the descriptor vector $\mathbb{E}X$ with an event (for the Gossip algorithm) [HM17]). *Let an open multi-agent system satisfying the assumptions of the problem statement (see Section 1.1.1), subject to the gossip algorithm and with $\mu = 0$. The k^{th} event that occurs in the system can be either a communication (with probability $1 - p_r$) or a replacement (with probability p_r) (see the problem statement for the general definitions of the events). The value of the descriptor vector $\mathbb{E}X = [\mathbb{E}[\bar{x}^2], \mathbb{E}[\bar{x}^2]]^T$ is updated according to the following recurrence relations for the k^{th} event:*

$$\mathbb{E}X_k = A\mathbb{E}X_{k-1} + b \quad (3.4)$$

$$\text{With} \quad A = (1 - p_r)A_c + p_rA_r = \begin{bmatrix} 1 - \frac{2p_r}{n} & \frac{p_r}{n^2} \\ \frac{1-p_r}{n} & 1 - \frac{1}{n} \end{bmatrix}, \quad \text{and} \quad b = p_rb_r = \sigma^2 \begin{bmatrix} \frac{p_r}{n^2} \\ \frac{p_r}{n} \end{bmatrix}.$$

Sketch of Proof. The general evolution of $\mathbb{E}X$ with an event k can be decomposed as

$$\mathbb{E}X_k = (1 - p_r)\mathbb{E}[X_{k-1} \mid k \text{ is a gossip}] + p_r\mathbb{E}[X_{k-1} \mid k \text{ is a repl.}] = (1 - p_r)A_c\mathbb{E}X_k + p_r(A_r\mathbb{E}X_k + b_r).$$

Where the last equality uses both Lemmas 3.1 and 3.2. Therefore, we have that $A = (1 - p_r)A_c + p_rA_r$ and $b = p_rb_r$, where A_c , A_r and b_r are defined in Lemmas 3.1 and 3.2. $\square$

3.1.2 Steady state expected performance and convergence rate

The steady state expected performance of the gossip algorithm (for $\mu = 0$) corresponds to the fixed point $\mathbb{E}X_{eq}$ of the linear update system of the descriptors (see equation (3.4)), i.e the solution of equation $\mathbb{E}X_{eq} = A\mathbb{E}X_{eq} + b$, which is:

$$\mathbb{E}[\text{Bias}^2]_{eq} = \mathbb{E}[\bar{x}^2]_{eq} = \frac{p_r + 1}{p_r + 2n - 1}\sigma^2 \quad \text{and} \quad \mathbb{E}[\text{MSE}]_{eq} = \mathbb{E}[\bar{x}^2]_{eq} = \frac{1 + p_r(2n - 1)}{p_r + 2n - 1}\sigma^2. \quad (3.5)$$

It results from this solution that the steady state expected variance is given by

$$\mathbb{E}[\text{Var}]_{eq} = \mathbb{E}[\bar{x}^2]_{eq} - \mathbb{E}[\bar{x}]_{eq}^2 = \frac{2p_r(n - 1)}{p_r + 2n - 1}\sigma^2.$$

Interpretation of these expressions can be made easily in some limit cases. First when $p_r = 1$, meaning that all events are replacements and that no communication ever takes places, the expected squared bias is $\mathbb{E}[\text{Bias}^2]_{eq} = \frac{\sigma^2}{n}$ and the expected variance $\mathbb{E}[\text{Var}]_{eq} = (1 - \frac{1}{n})\sigma^2$. This is also a fixed point of the replacement update (3.3) (from Lemma 3.2). This corresponds simply to the performance of a closed system with n agents with i.i.d z values (that correspond to their initial measurements) with mean 0 and variance σ^2 .

When $p_r \rightarrow 0$, the number of gossip communications between two replacements tends to infinity and the agents converge to the exact average of the system before any replacement. In that case, it results that $\mathbb{E}[\text{Bias}^2]_{eq} \rightarrow \frac{\sigma^2}{2n-1}$ and $\mathbb{E}[\text{Var}]_{eq} \rightarrow 0$. The expected variance tends toward zero, which means that agents eventually reach a consensus on their x values (that estimate μ). The final expected squared bias is lower than what would be obtained by averaging n i.i.d values. Indeed, the average of the values of the system is actually a weighted average of the values of all the agents that have ever been part of the system (at any past or present time). This statement (which is true for any value of p_r) and its implications are detailed in the next subsection.

When the size system is large ($n \rightarrow \infty$) with constant p_r , the opposite situation happens: the expected squared bias $\mathbb{E}[\text{Bias}^2]_{eq}$ goes to zero but the expected variance does not: $\mathbb{E}[\text{Var}]_{eq} \rightarrow p_r\sigma^2$.

The rate of convergence of the descriptors (and hence of the expected performance) toward this steady state behavior can be analyzed using the eigenvalues of the update matrix A . Let us introduce Δ_k which

corresponds to the difference of the vector $\mathbb{E}X_k$ with the fixed point $\mathbb{E}X_{eq}$, such that any descriptor vector $\mathbb{E}X_k$ can be expressed as $\mathbb{E}X_k = \mathbb{E}X_{eq} + \Delta_k$. The recurrence system from equation (3.4) becomes:

$$\mathbb{E}X_{eq} + \Delta_k = A(\mathbb{E}X_{eq} + \Delta_{k-1}) + b \quad \Leftrightarrow \quad \Delta_k = A\Delta_{k-1} = A^k\Delta_0.$$

The rate of convergence of Δ_k to 0 corresponds to the rate of convergence of $\mathbb{E}X_k$ to $\mathbb{E}X_{eq}$. Let λ_1, λ_2 be the two eigenvalues of matrix A and v_1, v_2 the corresponding eigenvectors (that are supposed to be independent). Any initial difference vector Δ_0 can be expressed in the eigenvector basis as $\Delta_0 = \alpha_1 v_1 + \alpha_2 v_2$ with $\alpha_1, \alpha_2 \in \mathbb{R}$. Therefore,

$$\Delta_k = A^k \Delta_0 = A^k(\alpha_1 v_1 + \alpha_2 v_2) = \alpha_1 \lambda_1^k v_1 + \alpha_2 \lambda_2^k v_2.$$

The difference vector Δ_k goes to 0 if and only if both eigenvalues of A are (strictly) lower than one (in absolute value). Moreover, the spectral radius of A , denoted $\rho(A)$, which is the greatest eigenvalue of A (in absolute value), gives the corresponding decreasing rate. If a component of the corresponding eigenvector is near to zero, it implies that this component converges more quickly than the others. According to [HM17]¹, the eigenvalues of matrix A are

$$\lambda_{1,2} = \frac{2n - 2p_r - 1 \pm \sqrt{\Delta}}{2n}, \quad \text{with} \quad \Delta = (1 - 2p_r)^2 + \frac{4p_r(1 - p_r)}{n}.$$

Note that the discriminant Δ has no link with the difference vector Δ_k introduced above. For large n , we have that $\Delta \approx |1 - 2p_r|$, and thus $\lambda_1 \approx 1 - \frac{2p_r}{n}$ and $\lambda_2 \approx 1 - \frac{1}{n}$, which corresponds to the diagonal elements of A . It makes sense because A is almost triangular for large n (since $\frac{p_r}{n^2} \approx 0$). For $p_r \in [0, 1]$, we have $\Delta \in [0, 1]$. Hence, for $n \geq 2$, it holds that $0 \leq \lambda_2 < \lambda_1 \leq 1$. Therefore, the spectral radius of matrix A is $\rho(A) = \lambda_1$ and it corresponds to the convergence rate of the expected performance toward the steady state expected performance. This convergence rate strongly depends on the replacement probability p_r . The convergence rate increases when p_r decreases (meaning that the convergence is slower) and even goes to 1 when p_r goes to 0 (meaning that the convergence never happens). It means that when the replacements are less frequent, the expected performance converges slower.

3.2 Main issues of the gossip algorithm for the external averaging problem

New agents have too much weight in the system

In the gossip averaging algorithm, the x_i value of each agent i at time t can be seen as a weighted average of all the initial values $x_k(0) = z_k$ of all the agents that have been present in the system at time t or before:

$$x_i(t) = \sum_{k \in \cup_{s \leq t} \mathcal{N}(s)} \phi_{i,k}(t) z_k \quad \text{with} \quad \sum_{k \in \cup_{s \leq t} \mathcal{N}(s)} \phi_{i,k}(t) = 1,$$

where $\phi_{i,k}(t)$ denotes the weight that agent $i \in \mathcal{N}(t)$ gives to initial value of agent $k \in \cup_{s \leq t} \mathcal{N}(s)$ at time t . These weights evolve with the algorithm as follows:

Initialization: $\phi_{i,k} = \begin{cases} 1 & \text{if } i = k, \\ 0 & \text{otherwise.} \end{cases} \quad \text{for all } i, k \in \mathcal{N}(0)$

Gossip communication between $i \in \mathcal{N}(t)$ and $j \in \mathcal{N}(t)$: $\phi_{i,k}^+ = \phi_{j,k}^+ = \frac{\phi_{i,k} + \phi_{j,k}}{2} \quad \text{for all } k \in \cup_{s \leq t} \mathcal{N}(s).$

Replacement of agent l by agent m :

All remaining agents are setting a weight of zero for the new agent m : $\phi_{i,m} = 0$ for all $i \neq m$.

The new agent m has only its own weight as nonzero weight: $\phi_{m,k} = \begin{cases} 1 & \text{if } m = k, \\ 0 & \text{otherwise.} \end{cases}$

¹And confirmed by Matlab and Mathematica

To compute the exact external average of the system, each agent $i \in \mathcal{N}(t)$ should give an equal weight to the initial value of all the present and previous agents $j \in \cup_{s \leq t} \mathcal{N}(s)$: $\phi_{i,j}(t) = \frac{1}{|\cup_{s \leq t} \mathcal{N}(s)|}$. In practice, this is impossible to achieve at any time since the agents cannot agree on a common value due to perpetual replacements that disturb the system. The best possible performance is discussed in Chapter 2. However, if the replacements stop at time t_s , each agent in the system will converge to the system average:

$$\bar{x}(t_s) = \frac{1}{n} \sum_{i \in \mathcal{N}(t_s)} x_i(t_s) = \frac{1}{n} \sum_{i \in \mathcal{N}(t_s)} \sum_{k \in \cup_{s \leq t_s} \mathcal{N}(s)} \phi_{i,k}(t_s) z_k.$$

This system average can also be written as a weighted average of the initial values of all agents:

$$\bar{x}(t_s) = \frac{1}{n} \sum_{k \in \cup_{s \leq t_s} \mathcal{N}(s)} \bar{\phi}_k(t_s) z_k, \quad \text{with} \quad \bar{\phi}_k(t_s) = \frac{1}{n} \sum_{i \in \mathcal{N}(t_s)} \phi_{i,k}.$$

It is therefore interesting to analyze the weights of each initial measurement z_k into the system average $\bar{x}(t)$ for any time t , in other words to analyze $\bar{\phi}_k(t) = \frac{1}{n} \sum_{i \in \mathcal{N}(t)} \phi_{i,k}$ for a fixed k . It will namely provide some interpretation and information about the bias of the system.

After the initialization of the system, for each agent $k \in \mathcal{N}(0)$, it holds that $\bar{\phi}_k(0) = \frac{1}{n}$. Each agent has an equal weight into the system average. The gossip communications preserve $\bar{\phi}_k$ for all k . When a replacement takes place at time t_r , the incoming agent m (that sets its own weight to 1) has an average weight of $\bar{\phi}_m(t_r^+) = \frac{1}{n}$ into the system average. The leaving agent l takes away its own weights $\phi_{l,k}(t_r^-)$ (for all k) that are not all zero since they sum up to 1. Consequently, $\bar{\phi}_k$ decreases for at least one $k \in \cup_{s < t_r} \mathcal{N}(s)$. More precisely, we have that, for a given leaving agent l ,

$$\bar{\phi}_k(t_r^+) = \bar{\phi}_k(t_r^-) - \frac{\phi_{l,k}(t_r^-)}{n} \quad \text{for all } k \in \cup_{s < t_r} \mathcal{N}(s). \quad (3.6)$$

The average weight of an agent k into the system average is maximum ($\frac{1}{n}$) when the agent k enters the system and may decrease at each new replacement. All agents cannot maintain a maximal weight of $\frac{1}{n}$ since at least a part of the old agents have decreasing weight at each replacement. Therefore, with gossip algorithm, the agents do not compute, even in average, the exact external average of the system that is (as a reminder) the average of the initial values of all agents that have ever been in the system until current time. Indeed, the algorithm gives more weights to the new agents and progressively forgets the oldest ones.

If we suppose that the replacements are sufficiently infrequent (p_r is closed to zero) so that a convergence is achieved between each replacement, it is possible to specify the decrease of the average weights of the agents with a replacement. In that situation, between each replacement, each x_i converges toward $\bar{x}$ and hence each $\phi_{i,k}$ converges toward $\bar{\phi}_k$ (for all k). The evolution of the weight of each agent in the system average $\bar{x}$ can be characterized more precisely. Initially, it still holds that $\bar{\phi}_k(t_{k,0}) = \frac{1}{n}$ where $t_{k,0}$ is the time at which agent k enters into the system. When a replacement occurs at time t_r the update of $\bar{\phi}_k$ from equation (3.6) becomes:

$$\bar{\phi}_k(t_r^+) = \left(\frac{n-1}{n} \right) \bar{\phi}_k(t_r^-) \quad \text{for all } k \in \cup_{s < t_r} \mathcal{N}(s).$$

This defines a recurrence relation with an initial condition for which the solution gives an expression for $\bar{\phi}_k(t_{k,r})$: the average weight of agent k in the system after r new replacements since its arrival at time $t_{k,0}$:

$$\bar{\phi}_k(t_{k,r}) = \left(\frac{n-1}{n} \right)^r \bar{\phi}_k(t_{k,0}) = \left(\frac{n-1}{n} \right)^r \frac{1}{n}. \quad (3.7)$$

It means that the weight of an agent into the system average decreases with new replacements at a rate of $(1 - \frac{1}{n})$. [HM16] shows that it is thereafter possible to re-deduce the theoretical expression of the steady state expected square bias $\mathbb{E}[\bar{x}^2]$ from equation (3.5) (in this limit case where $p_r \rightarrow 0$).

This reasoning shows that the gossip algorithm does not compute what is desired (the external average of the system) since it does not give an equal average weight to all the agents that have ever been in the system. Actually, it does not compute the intrinsic average of the system either (that only cares about the agent currently in the system) since it does not forget the previous agents directly. For the external averaging problem, the algorithm therefore has an expected squared bias that is quite large and seems improvable. This statement is confirmed by Figures 3.1b, 3.2a and 3.2b that each compares, for a different specific system, the gossip expected squared bias (the solid black line) with the one of the best ideal algorithm (the solid green line) that provides a lower bound on the expected mean squared error of any algorithm (but not necessarily on the expected squared bias or the expected variance that sum to the expected MSE, see Chapter 2). We observe that the expected squared bias of the gossip algorithm converges toward a steady state value much larger than the expected squared bias of the best ideal algorithm, especially when the system is small or has a small rate ratio ρ .

Replacements disturb the system

In the gossip averaging algorithm, a new agent is among the ones that have the more weight into the system average. But, its estimate (that corresponds only to its own measurements) is probably not very good. Therefore, each replacement disturbs largely the system and implies a steady state expected variance that seems also improvable as illustrated in Figure 3.1b. Indeed, the steady state variance of the gossip algorithm (the dotted black line) is quite larger than the variance of the best ideal algorithm (the dotted green line).

In the situation of Figure 3.1, the second issue of the gossip algorithm, related to the expected variance,

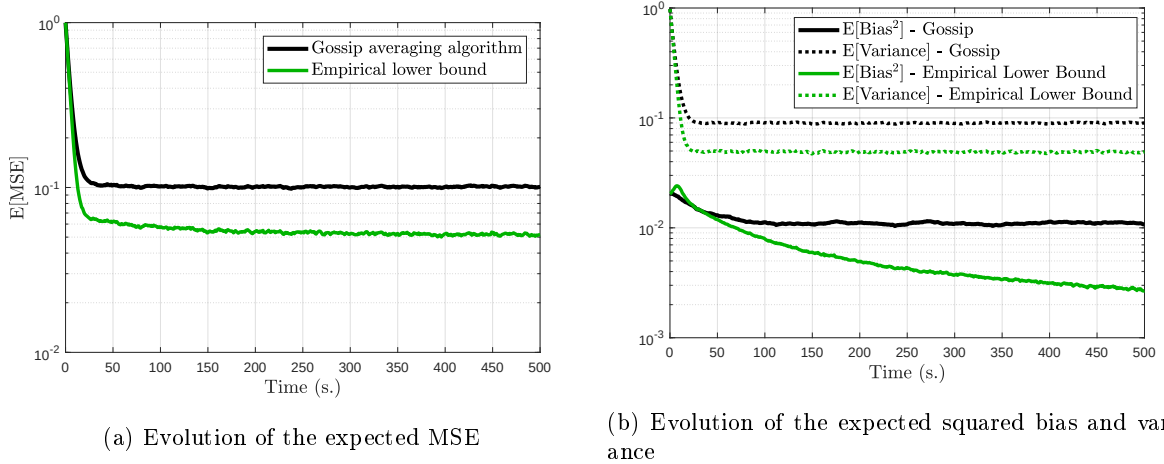


Figure 3.1: Comparison between the expected performance evolution with time of the gossip averaging algorithm and the best ideal algorithm for a system with $n = 50$ and $\rho = \frac{\lambda_c}{\lambda_r} = 10$ ($n\lambda_c = 10$). Expectations have been computed empirically with 5000 realization simulations.

seems the most important to solve, since the expected variance largely dominates the expected squared bias. But, Figure 3.2 shows that it is not always the case. Depending on the system configuration, the expected variance of the gossip averaging algorithm can be of the same order of magnitude than the expected squared bias (as in Figure 3.2a) or largely below it (as in Figure 3.2b). With these three figures, we also observe that the (multiplicative) difference between the expected variance of the gossip averaging algorithm and the one of the best ideal algorithm seems not varying much from one system to another (on the contrary of the expected squared bias).

It seems therefore important to solve both issues of the gossip averaging algorithm in order to obtain a robustly efficient algorithm. Having different algorithms that solve both issues separately is already a good way of solving the many situations where one of the two issues is dominating the other.

Let us make general and non-rigorous comments about the importance of the expected variance and the expected squared bias into the expected mean squared error for any algorithm.

- For small rate ratios ρ (as in Figure 3.1b), there are only a few communications between each replacement and the expected variance in general dominates the expected squared bias, especially if the system is large, as explained at the second point.
- Larger system has in general a lower expected squared bias since the system contains more agents and hence more information about the value of interest μ . In a such system, the expected squared bias is in general less important. The comparison of Figure 3.2a and Figure 3.1b where the size of the system passes from $n = 10$ to $n = 50$ illustrates this.
- On the contrary, when the system is small and its rate ratio is large, the large number of communications in which each agent is involved between each replacement allows the expected variance to be relatively low, and in general lower than the expected squared bias, as illustrated in Figure 3.2b.

These general comments are used to better understanding the *final* expected mean squared comparison presented in the next section.

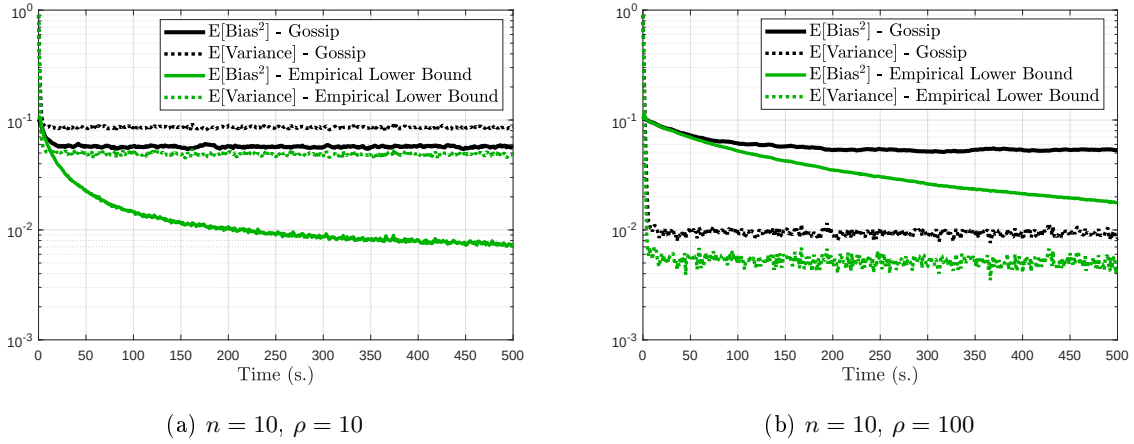


Figure 3.2: Comparison between the expected variance and the expected squared bias of the gossip averaging algorithm and the best ideal algorithm for two different systems with a fixed size $n = 10$ and two different rate ratios ($\rho = 10$ versus $\rho = 100$) (with constant $n\lambda_c = 10$). Expectations have been computed empirically with 5000 realization simulations.

3.3 Comparison between the gossip averaging algorithm and the best ideal algorithm

Figure 3.3 compares the expected mean squared error (MSE) after 500 seconds of the gossip averaging algorithm with the one of the best ideal algorithm (see Definition 2.2) for a variety of different system configurations. Concretely, Figures 3.3a and 3.3b show the evolution of the performance with the rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$ respectively for a small and a large system ($n = 15$ and $n = 75$) and Figures 3.3c and 3.3d show the evolution of the performance with the system size n respectively for a small and a large rate ratio ($\rho = 4$ and $\rho = 50$). The parameter scaling is done with constant $n\lambda_c = 10$, as explained in Section 1.3. Moreover, the theoretical expression of the steady state expected mean squared error of the gossip averaging algorithm from equation (3.5) is also plotted (dotted black line) in all these situations and fits well the points of the gossip expected MSE after 500 seconds. This means that the steady state of the algorithm is reached after 500 seconds, for almost all these system configurations.

These figures show that the gossip averaging algorithm does not perform too badly but that a significant progression margin is possible in most of the cases.

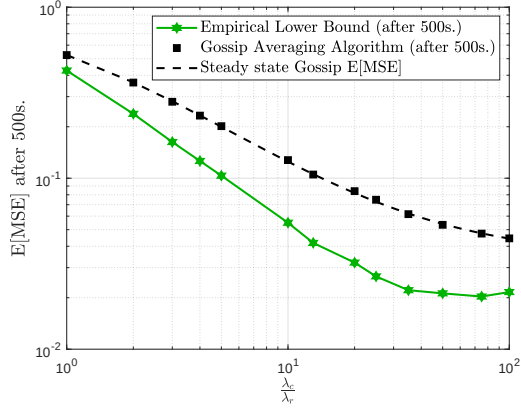
- Figure 3.3c (with a constant small ρ) corresponds to a situation where the shown expected mean squared error is mainly influenced by the expected variance of the system. Figure shows that the

expected variance of the system with both algorithms does not seem to be much impacted by the size of the system n . In that case, the difference between the expected performance of the two algorithms is constant.

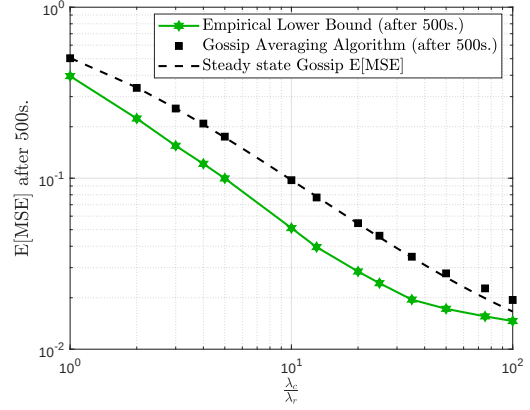
- Larger system has in general a lower expected squared bias. Figure 3.3b (with constant large n) corresponds therefore also to a situation where the expected mean squared error is mainly influenced by the expected variance of the system. The space for improvement seems nearly identical for any rate ratio ρ , in a logarithmic scale, meaning that the *multiplicative* difference between the performance of both algorithms is nearly constant for all rate ratio ρ .
- In Figure 3.3d, the constant rate ratio is large ($\rho = 50$) and hence the impact of the expected squared bias on the expected mean squared error is more significant. We observe that the expected MSE of the gossip algorithm moves away from the empirical lower bound when n becomes small. This observation matches with the intuition about the issue of the gossip algorithm relative to the expected squared bias developed in Section 3.2. Indeed, the speed at which the gossip forgets progressively the old agents (see equations (3.6) and (3.7)) decrease when the size of the system n increases and hence the expected squared bias achieved by the algorithm get closer to the expected squared bias of the empirical lower bound when n increases.
- In Figure 3.3a, we can observe that the *multiplicative* difference between the expected performance of the gossip algorithm and the empirical lower bound increases with the rate ratio ρ . This can be explained by the fact that the expected squared bias (for which the gossip algorithm is far from the optimal one since n is small), have more importance into the expected MSE when the rate ratio ρ is large (which implies generally a low expected variance). In Figure 3.3a, we can also observe that the performance of the best ideal algorithm after 500 seconds does not always decrease when the rate ratio ρ increases. Having fewer replacements in the system means less disturbance in the system which explain why the performance get generally better when ρ increase. But having fewer replacements also means having less information (about the interest value μ) coming in the system (in a given amount of time) and hence implies a worst expected squared bias (after a given amount of time). For too large rate ratio ρ , the gain on the variance is not sufficient to counterbalance the loss of information and it leads to a larger MSE after a given amount of time (here 500 seconds). Therefore, in these situations (when ρ is very large), the expected performance of the best ideal algorithm has not totally reached its steady state.

Conclusion about the gossip averaging algorithm

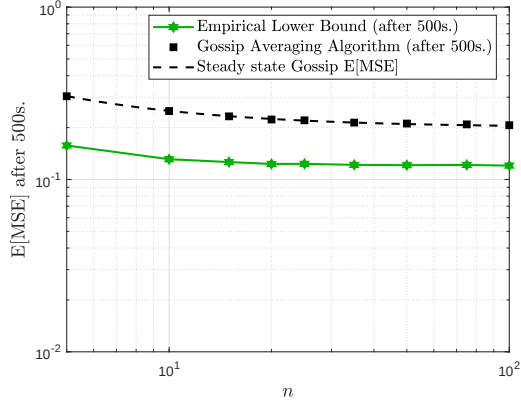
In a nutshell, the gossip averaging algorithm for which the evolution in open multi-agent system is well known, presents two main issues for solving the external averaging problem stated in Section 1.1.1. The first issue is related to the expected squared bias and is due to the importance given to new agents. This problem gets worse for small systems. The second issue is related to the variance of the system and its importance seems constant for any system configuration. These two problems, as illustrated in Figure 3.3, leaves room to new algorithms for improving the achieved performance and getting closer to the empirical lower bound. Chapter 4 presents the design of a new algorithm that aims to solve the expected squared bias issue of the gossip averaging algorithm. Chapter 5 concerns the design of a new algorithm which is intended to solve the expected variance issue of the gossip. Thereafter, Chapter 6 presents the combination of both algorithms in one in order to solve all the issues encountered in the gossip averaging algorithm in once. The gossip algorithm can still be used as a baseline with theoretical guarantees that new algorithms should have to beat. The design of new algorithms has also made use of the thoughts on its issues in order to solve efficiently the external averaging problem in open multi-agent systems.



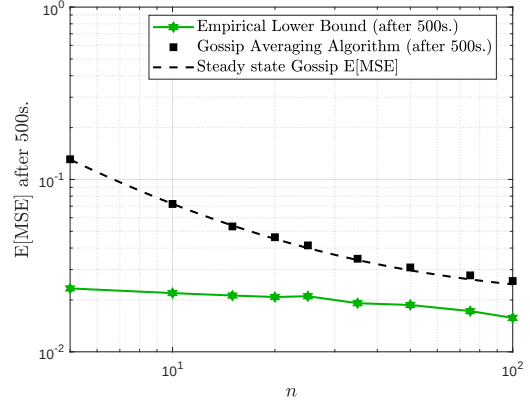
(a) Evolution with rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$, for $n = 15$.



(b) Evolution with rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$, for $n = 75$.



(c) Evolution with system size n , for $\rho = \frac{\lambda_c}{\lambda_r} = 4$.



(d) Evolution with system size n , for $\rho = \frac{\lambda_c}{\lambda_r} = 50$.

Figure 3.3: Comparison between the expected mean squared error after 500 seconds of the gossip averaging algorithm and the one of the best ideal algorithm (that provides an empirical lower bound) for a variety of different systems (with constant $n\lambda_c = 10$). Expectations have been computed empirically with 1000 realization simulations. The theoretical expression of the steady state mean squared error of the gossip algorithm (see equation (3.5)) is also plotted (dashed black line).

Chapter 4

Symmetric Push-Sum algorithm

This chapter presents an algorithm that aims to solve the first issue of the gossip averaging algorithm for the external averaging problem (stated in Section 1.1.1), that has been presented in Section 3.2 from Chapter 3. In short, this issue states that the gossip averaging algorithm cannot compute the external average of the system since it gives too much importance to the new agents that come into the system and therefore forgets progressively the old agents. The idea is therefore to design an algorithm capable of assigning a weight to each agent in order to be able to choose wisely the weights of the incoming agents. The push-sum algorithm, presented in the state-of-the-art, introduces a weight to each agent. Our new algorithm will therefore be push-sum based. This chapter presents the design of this new algorithm and evaluates its performance using simulations results. It also analyses (partially) the theoretical evolution of an open multi-agent system subject to this push-sum based algorithm, without succeeding in having guarantees on its performance. Finally, an implementable version of the algorithm is discussed.

4.1 Algorithm description and framework

The push-sum algorithm was initially designed to handle asymmetric communications in the averaging problem in closed multi-agent system which the averaging gossip does not enable. The algorithm has been studied for example in [KDG03] or [GH19]. See equation (2) for a description of this algorithm.

For our problem, we suppose that the communications are symmetric (see the problem statement in Section 1.1.1) and therefore, the push-sum algorithm can be written in a non-directed way (see below). Recall that for our problem, each agent i in the system $i \in \mathcal{N}(t)$ has an initial measurement z_i independently drawn from an identical distribution of mean μ and variance σ^2 . For the algorithm, each agent $i \in \mathcal{N}(t)$ also holds a time-varying value $x_i(t)$ that represents the aggregation of information held by i and a time-varying weight $w_i(t)$ that represents the volume of information that i holds within its variable x_i . At any time, the estimate of μ of each agent is given by $y_i(t) = \frac{x_i(t)}{w_i(t)}$. The general framework of the symmetric push-sum algorithm is the following.

General symmetric push-sum algorithm:

Initialization for all $i \in \mathcal{N}(t)$: $x_i(0) = w_i(0)z_i$, and choose $w_i(0) \in \mathbb{R}^+$. (4.1)

Interaction between agents $i \in \mathcal{N}(t)$ and $j \in \mathcal{N}(t)$:
$$x_i^+ = x_j^+ = \frac{x_i + x_j}{2}, \quad \text{and} \quad w_i^+ = w_j^+ = \frac{w_i + w_j}{2}. \quad (4.2)$$

Estimate of agent $i \in \mathcal{N}(t)$:
$$y_i(t) = \frac{x_i(t)}{w_i(t)}. \quad (4.3)$$

This algorithm can be interpreted in terms of the estimates of the agents $y_i(t)$. Using the definition of the estimates (equation (4.3)), a communication update can be understood as a weighted sum of both estimates of communicating agents ($i, j \in \mathcal{N}(t)$):

$$y_i^+ = y_j^+ = \frac{x_i + x_j}{w_i + w_j} = \frac{w_i y_i + w_j y_j}{w_i + w_j}. \quad (4.4)$$

But the weight of each agent is also updated during each communication in order to keep up-to-date the importance of its estimate into the system.

On the point of view of the x values and the weights w , this algorithm simply amounts to perform two distinct gossip averaging algorithms: one with the x values and one with the w values. It allows the algorithm to inherit properties of the gossip averaging algorithm, such as the preservation of the sum of x values and the sum of weights w during communications. In the push-sum algorithm presented in the state-of-the-art section, each agent begins with the same initial weight ($w_i(0) = 1$). Here we let the possibility to choose a different initial weight for each agent in order to prepare the adaptation to the *open* system case. The initial x value of each agent i is then fixed to $x_i(0) = w_i(0)z_i$ to ensure that its initial estimate of the solution corresponds to its own measurement of μ : $y_i(0) = \frac{x_i(0)}{w_i(0)} = z_i$. When the system is *closed*, meaning that the set of agents $\mathcal{N}(t)$ that constitutes the system remains constant throughout the entire process, the following proposition can be deduced.

Proposition 4.1 (The symmetric push-sum algorithm in closed system). *In a closed multi-agent system (with fixed set of agents $\mathcal{N}$) subject to the symmetric push-sum with symmetric, pairwise, random communications (as stated in Section 1.1.1), the estimates $y_i(t)$ of μ of each agent i converges to a weighted average of the initial estimate $y_j(0)$ of all agent $j \in \mathcal{N}$:*

$$\lim_{t \rightarrow \infty} y_i(t) = \frac{\sum_{j \in \mathcal{N}} w_j(0) y_j(0)}{\sum_{j \in \mathcal{N}} w_j(0)} \quad \text{for all } i \in \mathcal{N}$$

In general, all agents have been initialized such that their initial estimate $y_i(0)$ corresponds to their own measurement of μ , as in equation (4.1), $y_j(0) = z_j$.

The starting time of the system is considered to be 0 here, but it can actually be any (finite) time.

Proof. Using convergence results of the averaging gossip algorithm ([Boy+06] or [Blo+05]), we have that the x value of each agent, for which the algorithm performs a gossip average, converges toward the average of the initial x values:

$$\lim_{t \rightarrow \infty} x_i(t) = \frac{1}{n} \sum_{j \in \mathcal{N}} x_j(0) \quad \text{for all } i \in \mathcal{N}.$$

Similarly, for the weights we have that:

$$\lim_{t \rightarrow \infty} w_i(t) = \frac{1}{n} \sum_{j \in \mathcal{N}} w_j(0) \quad \text{for all } i \in \mathcal{N}.$$

Therefore, the estimate of each agent $i \in \mathcal{N}$ becomes (using definition of $y_i(t)$ given by equation (4.3)):

$$\lim_{t \rightarrow \infty} y_i(t) = \lim_{t \rightarrow \infty} \frac{x_i(t)}{w_i(t)} = \frac{\sum_{j \in \mathcal{N}} x_j(0)}{\sum_{j \in \mathcal{N}} w_j(0)} = \frac{\sum_{j \in \mathcal{N}} w_j(0) y_j(0)}{\sum_{j \in \mathcal{N}} w_j(0)} \quad \text{for all } i \in \mathcal{N}$$

□

If all initial weights are identical (for example 1), the estimates of the agents converge toward the exact average of initial measurements of all agents in the system: $\frac{1}{n} \sum_{j \in \mathcal{N}} z_j$. In that case, we remark that the algorithm is strictly equivalent to the gossip averaging algorithm since the weight of the agents remains unchanged during the communications (see equation (4.2)) and becomes useless.

The individual weights of the agents become useful when the multi-agent system is *open*. Indeed, since we can set the initial weight of any agent, the idea is to wisely choose the initial weight of the new agents that arrive into the system during its operation. These initial weights should be set in a such way that the agents tend to evaluate the external average of the system and then have a good estimate of μ at any time t . As a reminder, the external average of the system at time t is defined as

$$\bar{z}_{\text{ext}}(t) = \frac{1}{|\cup_{s \leq t} \mathcal{N}(s)|} \sum_{j \in \cup_{s \leq t} \mathcal{N}(s)} z_j.$$

And the intrinsic average at time t is defined as $\bar{z}(t) = \frac{1}{n} \sum_{j \in \mathcal{N}(t)}$.

For notation convenience, let us suppose that the n agents in the system at the beginning are denoted with indices from 1 to n : $\mathcal{N}(0) = \{1, \dots, n\}$. Each new agent that comes in the system during its operation is supposed to have an index $n + r$ where r is its order of arrival (e.g. 1 for the first agent to arrive during the first replacement). It allows writing the external average as:

$$\bar{z}_{\text{ext}}(t) = \frac{1}{N_{\text{tot}}(t)} \sum_{j=1}^{N_{\text{tot}}(t)} z_j,$$

where $N_{\text{tot}}(t)$ is the total number of agents that have ever been in the system until time t .

For the symmetric push-sum algorithm in *open* multi-agent system, we suppose that the initial weight of the agents initially present in the system is 1 and the next ones need to be wisely set:

$$w_i(0) = 1 \quad \text{for all } i \in 1, \dots, n \quad \text{and} \quad \text{choose } w_{n+r}(0) \quad \text{for } r = 1, 2, \dots$$

It ensures that the algorithm acts as the gossip averaging algorithm, until the first replacement, and hence that the estimates of the agents start tending correctly (at least until the first replacement) to the average of their initial measurements $\bar{z}$, which corresponds to the external average $\bar{z}_{\text{ext}}$ since no replacement has taken place yet.

Setting correctly the initial weight $w_{n+k}(0)$ of all the new agents $n + k$ that arrive into the system is therefore a key to solve the expected squared bias issue of the gossip averaging algorithm. The next section presents different ways of choosing these initial weights.

4.2 Setting the initial weight of the incoming agents

To set correctly the initial weights of the incoming agents, this section presents a first reasoning, assuming that the convergence occurs in the system between each replacement, which corresponds to an ideal case. Secondly, the problem is written as an optimization problem to reveal if other better weights can be found.

4.2.1 Convergence assumption

Assumption 4.1. We assume that the convergence occurs between two replacements in the system. It means that the replacement probability of an event tends to 0 ($p_r \rightarrow 0$), or in other words, that the expected number of interactions between two replacements (given by the rate ratio ρ) tends to infinity ($\rho \rightarrow \infty$).

Proposition 4.2 (Initial weights under Assumption 4.1). *In an open multi-agent system subject to the symmetric push-sum algorithm and under Assumption 4.1, if the initial weight of the agents are set according to:*

$$w_i(0) = 1 \quad \text{for all } i \in \{1, \dots, n\} \quad \text{and} \quad w_{n+r}(0) = \left(\frac{n-1}{n}\right)^r \quad \text{for } r = 1, 2, \dots, \quad (4.5)$$

where $\{1, \dots, n\}$ corresponds to the set of agents initially present in the system ($\mathcal{N}(0)$) and $n + r$ (for $r = 1, 2, \dots$) corresponds to indices of all the new agents that enter the system, in their index order; Then, the estimates of the agents in the system $y_i(t)$ for $i \in \mathcal{N}(t)$ converge to the external average of the system $\bar{z}_{\text{ext}}(t)$, between each replacement.

Proof. Let us make a recursive reasoning that uses the conclusion of the proposition as recursion assumption. Imposing that this assumption remains valid before each replacement allows re-deducing the expression of the initial weights given by (4.5).

Let t_r be the moment of the r^{th} replacement in the system and suppose that the estimate of each agent has converged to the external average, just before the replacement occurs (at time t_r^-). Using notation

$N_{\text{tot}} = n + r - 1$ (the total number of agent in the system at t_r^- , just before the r^{th} replacement), we have for all $i \in \mathcal{N}(t_r^-)$:

$$\begin{aligned} x_i(t_r^-) &= \bar{x}(t_r^-) = \frac{1}{n} \sum_{j \in \mathcal{N}(t_r^-)} x_j(t_r^-) && \text{using Assumption 4.1,} \\ w_i(t_r^-) &= \bar{w}(t_r^-) = \frac{1}{n} \sum_{j \in \mathcal{N}(t_r^-)} w_j(t_r^-) && \text{using Assumption 4.1,} \end{aligned} \quad (4.6)$$

$$y_i(t_r^-) = \bar{z}_{\text{ext}}(t_r^-) = \frac{1}{N_{\text{tot}}} \sum_{j=1}^{N_{\text{tot}}} z_j, \quad \text{new recursion assumption.} \quad (4.7)$$

Remark that the only thing that changes in the system between time t_r^- and t_r is the replacement of one randomly selected agent of the system by a new one. Therefore, we have for all $i \in \mathcal{N}(t_r^-)$,

$$x_i(t_r^-) = x_i(t_r), \quad w_i(t_r^-) = w_i(t_r), \quad y_i(t_r^-) = y_i(t_r) \quad \text{for all } i \in \mathcal{N}(t_r^-). \quad (4.8)$$

But the total number of agents that have ever been in the system changes and increases of one unit:

$$N_{\text{tot}}(t_r) = N_{\text{tot}}(t_r^-) + 1 = n + r.$$

Choosing unit initial weight for the set of agents initially present in the system: $w_i(0) = 1$ for all $i \in \{1, \dots, n\}$, allows the recursion assumption (4.7) to be satisfied before the first replacement in the system.

Therefore, the assumption is verified for the initial case where $N_{\text{tot}} = n$. Let us choose the weight $w_{N_{\text{tot}}+1}$ of the new agent $N_{\text{tot}} + 1$ to ensure that the system after the replacement re-converges toward the new external average $\bar{z}_{\text{ext}}(t_r)$ and therefore that the assumption (4.7) remains satisfied before the $r+1^{\text{th}}$ replacement occurs.

Thanks to Assumption 4.1, the system can be considered as *closed* between the replacement at time t_r and the next replacement. Therefore, the convergence result of Proposition 4.1 can be used to determine the convergence value of estimates of the agents, using t_r as starting time. The initial weight of the new agent $N_{\text{tot}} + 1$ should be such that the recursion assumption (4.7) remains satisfied for the next replacement:

$$\frac{\sum_{j \in \mathcal{N}(t_r)} w_j(t_r) y_j(t_r)}{\sum_{j \in \mathcal{N}(t_r)} w_j(t_r)} = \frac{1}{N_{\text{tot}} + 1} \sum_{j=1}^{N_{\text{tot}}+1} z_j = \bar{z}_{\text{ext}}(t_r)$$

Let l be the index of the leaving agent, and let us suppose that all variables for which no time is specified is evaluated at time t_r (except for N_{tot} that is evaluated at t_r^-). The left-hand side equation can be expanded to isolate the terms related to the new agent $N_{\text{tot}} + 1$. The right-hand side can be expanded to make appear the external average of the system before the replacement occurs (i.e. at time t_r^-).

$$\frac{\sum_{j \in \mathcal{N}(t_r)} w_j y_j - w_l y_l + w_{N_{\text{tot}}+1} y_{N_{\text{tot}}+1}}{\sum_{j \in \mathcal{N}(t_r)} w_j - w_l + w_{N_{\text{tot}}+1}} = \frac{N_{\text{tot}}}{N_{\text{tot}} + 1} \left(\frac{1}{N_{\text{tot}}} \sum_{j=1}^{N_{\text{tot}}} z_j \right) + \frac{1}{N_{\text{tot}} + 1} z_{N_{\text{tot}}+1}$$

Using convergence assumptions (equations (4.6) and (4.7)) and equation (4.8) (for the left-hand side), it becomes

$$\frac{(n-1)\bar{w}(t_r^-)\bar{z}_{\text{ext}}(t_r^-) + w_{N_{\text{tot}}+1}z_{N_{\text{tot}}+1}}{(n-1)\bar{w}(t_r^-) + w_{N_{\text{tot}}+1}} = \frac{N_{\text{tot}}}{N_{\text{tot}} + 1} (\bar{z}_{\text{ext}}(t_r^-)) + \frac{1}{N_{\text{tot}} + 1} z_{N_{\text{tot}}+1}$$

Both sides of this last equation corresponds to a weighted average of $\bar{z}_{\text{ext}}(t_r^-)$ and $z_{N_{\text{tot}}+1}$. The equality holds for any value of $\bar{z}_{\text{ext}}(t_r^-)$ and $z_{N_{\text{tot}}+1}$ if and only if their weight ratio is the same:

$$\frac{w_{N_{\text{tot}}+1}}{(n-1)\bar{w}(t_r^-)} = \frac{1}{N_{\text{tot}}} \Leftrightarrow w_{N_{\text{tot}}+1} = \frac{(n-1)\bar{w}(t_r^-)}{N_{\text{tot}}}. \quad (4.9)$$

This leads to an expression of the initial weight of the new agent $w_{N_{\text{tot}}+1}$.

To obtain the expression (4.5) of the proposition, it simply remains to find an expression of $\bar{w}(t_r^-)$. The

average of the weights in the system only depends on the number of replacements that already have occurred in the system since communications preserve the sum of weights (and then their average). We denote $\bar{w}_r$ the average of the weights in the system after r replacements. Thus, we have $\bar{w}(t_r^-) = \bar{w}_{r-1}$ since t_r^- is the time just before the r^{th} replacement. Equation (4.9) can be written in function of the rank r of the considered replacement (remembering that $N_{\text{tot}} = n + r - 1$):

$$w_{n+r}(0) = \frac{(n-1)\bar{w}_{r-1}}{n+r-1}. \quad (4.10)$$

Remark that we note $w_{n+r}(0)$ at time zero since it corresponds to the *initial* weight of the r^{th} new agent but actually, it could have been any time until its arrival into the system at time t_r . The expression of the average weight $\bar{w}_{r-1}$ can be computed as a recurrence equation:

$$\begin{aligned} \text{Initially:} \quad & \bar{w}_0 = 1 \\ \text{after the } r^{th} \text{ replacement:} \quad & \bar{w}_r = \frac{1}{n} \left(\sum_{j \in \mathcal{N}(t_r^-)} w_j(t_r^-) - w_l(t_r^-) + w_{n+r}(t_r) \right) \\ & = \frac{1}{n} \left((n-1)\bar{w}_{r-1} + \frac{(n-1)\bar{w}_{r-1}}{n+r-1} \right) \\ & = \frac{n-1}{n} \left(1 + \frac{1}{n+r-1} \right) \bar{w}_{r-1} \end{aligned}$$

To write this recurrence, we use the expression of the initial weight of new agent (4.10) and the convergence assumption (4.6): each weight are equal to their average before any replacement. The solution of this recurrence equation is given by

$$\bar{w}_r = \frac{n+r}{n} \left(\frac{n-1}{n} \right)^r.$$

It can be used to replace $\bar{w}_{r-1}$ into equation (4.10) to give the initial weights stated in the proposition:

$$w_{n+r}(0) = \frac{(n-1) \frac{n+r-1}{n} \left(\frac{n-1}{n} \right)^{r-1}}{n+r-1} = \left(\frac{n-1}{n} \right)^r.$$

□

This results about the initial weights of the incoming agents matches perfectly with the results obtained in Section 3.2 about the gossip averaging algorithm issues. Indeed, when the gossip averaging algorithm is used, the weight of each agent in the average of the system decreases with a factor $\frac{n-1}{n}$ at each replacement (see equation (3.7)). Decreasing the weights of the incoming agents in a similar way seems therefore natural. It represents the fact that at each replacement, only $n-1$ (over n) old agents remains in the system. From another point of view, it seems logical that we attribute less and less weights to new agents for which the initial estimate always contains the same amount of information, coming from their initial measurement of μ , while the estimates of agents in the system gather progressively more and more information about μ .

The convergence assumption (see Assumption 4.1) is however ideal and it is not really achievable by a true system. It can still be *nearly* achieved when the rate ratio ρ is large. Figure 4.1 shows the performance of this symmetric push-sum algorithm (using initial weights from Proposition 4.2) when $\rho = 100$, in comparison with the gossip averaging algorithm and the empirical lower bound (on the expected MSE) given by the best ideal algorithm. When $\rho = 100$ there is in average 100 communications between two replacements in the system. Therefore, using results about the gossip algorithm from Section 3.1.1, we can state that the expected variance of x and w decreases in average approximately of a factor $\left(\frac{n-1}{n} \right)^\rho \approx 3 \times 10^{-5}$ (for $n = 10$) between two replacements, meaning that the values of the agents are almost all the same, and then that the convergence is almost reached.

Figure 4.1 first confirms that, even when the replacements are infrequent, the gossip averaging algorithm (in black) does not perform so well since it gives too much importance to new agents and then

forgets progressively the old ones, as explained in Section 3.2. The symmetric push-sum algorithm (in red) that aims to correct this issue by setting decreasing weights to new agents fills its mission perfectly: its expected squared bias fits very well the one of the best ideal algorithm (in green). It confirms therefore the correctness of Proposition 4.2.

To measure how much the curves match together, we introduce a measure of relative difference between two curves. This has no real theoretical value, but still allows to quantify how well the symmetric push-sum algorithm performs, in comparison with the best ideal algorithm. The relative difference can be computed at a specific time t or averaged over each discrete time of the simulation (that uses a time-step of one second $t_s = 1$).

$$\begin{aligned} \mathbb{E}[\text{Bias}^2] \text{ relative difference at time } t & \quad \frac{\mathbb{E}[\text{Bias}_{\text{sps}}^2(t)] - \mathbb{E}[\text{Bias}_{\text{best}}^2(t)]}{\mathbb{E}[\text{Bias}_{\text{best}}^2(t)]}, \\ \mathbb{E}[\text{Bias}^2] \text{ relative difference (averaged over time)} & \quad \frac{1}{J_f} \sum_{j=1}^{J_f} \frac{\mathbb{E}[\text{Bias}_{\text{sps}}^2(jt_s)] - \mathbb{E}[\text{Bias}_{\text{best}}^2(jt_s)]}{\mathbb{E}[\text{Bias}_{\text{best}}^2(jt_s)]} \end{aligned}$$

Similar relative differences measures can be computed for the expected variance and the expected mean squared error. In Figure 4.1b, the relative difference (averaged over time) between the expected squared bias of the symmetric push-sum algorithm and the one of the best ideal algorithm is about 0.007 (or 0.7%), while similar relative difference (averaged over time) for the gossip averaging algorithm is 0.84 (or 84%).

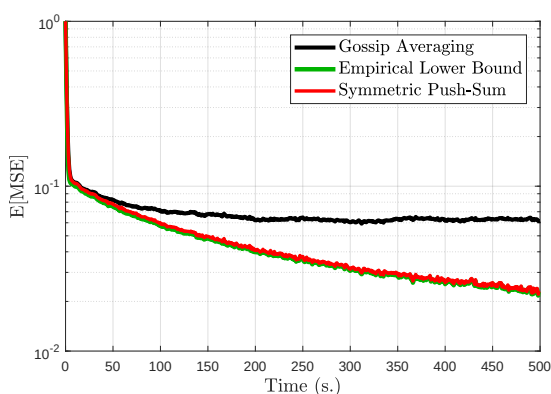
Secondly, the symmetric push-sum algorithm presents an expected variance that is close to the one of the gossip algorithm at the beginning but that decreases with time and seems to tend to the one of the empirical lower bound. A decreasing expected variance is also understandable with the weights of Proposition 4.2 since each new agent has a bit less importance than the previous one, it perturbs each time a bit less the system.

As shown in Figure 4.1a, the resulting expected mean squared error of the algorithm (dominated by its squared bias component) is very close to the one of the best algorithm (with an average relative difference of 0.03 only (or 3%)). In that situation where the Assumption 4.1 is almost satisfied, the symmetric push-sum algorithm with initial weights from Proposition 4.2 seems therefore almost optimal since its expected performance (MSE, squared bias and variance) is very close (or tends to be very close for the variance) to the one of the best ideal algorithm.

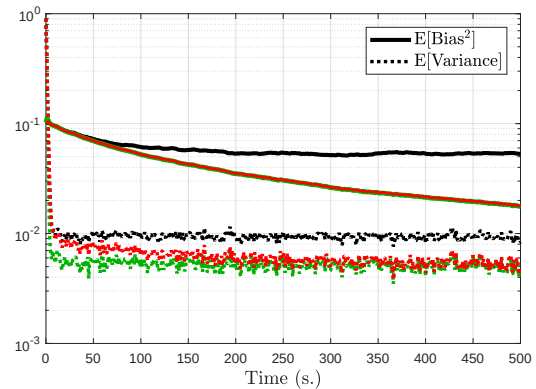
Implementing this symmetric push-sum algorithm with the weights given in Proposition 4.2 requires new agents to know (or estimate) the size of the system n and their arrival rank (r) in the system (or equivalently the number of agents that have ever been in the system before its arrival). These details are discussed in Section 4.5. For other sections (including this one), we focus on the *theoretical* version of the algorithm, where the new agents are supposed to know perfectly the system size n and their arrival rank r .

A natural question arises about the relaxation of Assumption 4.1. What happens when the assumption is totally relaxed and that the replacements in the systems are frequent (in comparison with the communications)? Figure 4.2 is similar to the previous one (Figure 4.1) and presents results for a system of identical size ($n = 10$) but subject to more frequent replacements ($\rho = 5$). It figures out that the symmetric push-sum (using initial weights of Proposition 4.2) still performs very well.

The same conclusions as before can be drawn for the expected bias that is close to the one of the best ideal algorithm (in green) and for the expected variance that tends to the one of the best ideal algorithm (in dotted green). The average relative difference for the expected squared bias is about 0.08 (or 8%), which is a bit more than in the previous situation (from Figure 4.1b) but still very good. The expected variance tends much more quickly to one the best ideal algorithm since a higher replacement frequency implies that the weights of the incoming agents decrease faster (with time). What is different in this scenario of frequent replacements, is that the expected mean squared error is dominated by its expected variance term. This explains why the expected mean squared error of the algorithm (from Figure 4.2a) does not totally fit the one of the best ideal algorithm at the beginning (but it is still very close).

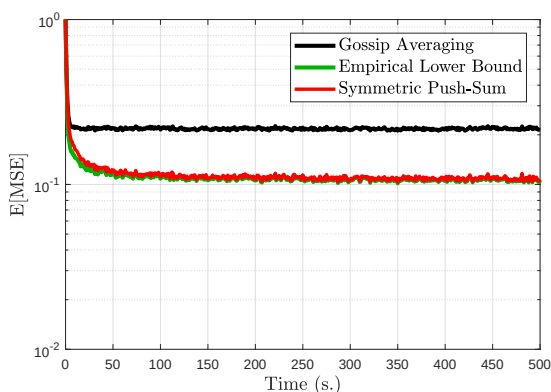


(a) Evolution of the expected MSE

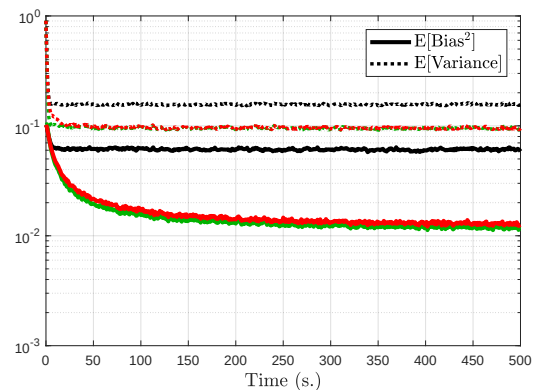


(b) Evolution of the expected squared bias and variance

Figure 4.1: Comparison between the expected performance time evolution of the symmetric push-sum algorithm (using initial weights given in Proposition 4.2), the gossip averaging algorithm and the best ideal algorithm (that provides the empirical lower bound on $\mathbb{E}[\text{MSE}(t)]$) for a system with $n = 10$ and $\rho = \frac{\lambda_c}{\lambda_r} = 100$ (with $n\lambda_c = 10$). Expectations have been computed empirically with 5000 realization simulations.



(a) Evolution of the expected MSE



(b) Evolution of the expected squared bias and variance

Figure 4.2: Comparison between the expected performance time evolution of the symmetric push-sum algorithm (using initial weights given in Proposition 4.2), the gossip averaging algorithm and the best ideal algorithm (that provides the empirical lower bound on $\mathbb{E}[\text{MSE}(t)]$) for a system with $n = 10$ and $\rho = \frac{\lambda_c}{\lambda_r} = 5$ (with $n\lambda_c = 10$). Expectations have been computed empirically with 5000 realization simulations.

As further illustrated in Section 4.3, the performance of the symmetric push-sum algorithm are very good with the decreasing initial weights given in Proposition 4.2. Are these initial weights really optimal in any situation? Is it possible to find better ones? That is what we aim to find out in the next subsection.

4.2.2 Optimal initial weights

This subsection aims to model the problem of setting the initial weights of incoming agents as an optimization problem. Since an optimization problem can only manage a fixed number of variables, we fix in advanced the number of replacements r that are considered.

The optimization problem chooses the initial weight $w_i(0)$ of each new agent $i = n + 1 \dots, n + r$ that

enters the system. All the problem variables are stacked (in order) in a vector $\mathbf{w}_0$. These variables are optimized such that the resulting *expected* mean squared error is minimum after all the replacements have taken place. The *expectation* is computed as the average over a fixed number n_{real} of realization ω_k . The notation t_{r_k} denotes the time after r replacements for a given realization ω_k . The problem can be written as follows:

$$\underset{\mathbf{w}_0}{\text{minimize}} \quad \frac{1}{n_{\text{real}}} \sum_{k=1}^{n_{\text{real}}} \left(\frac{1}{n} \sum_{i \in \mathcal{N}(t_{r_k})} (\mu - y_i(t_{r_k}, \omega_k))^2 \right)$$

where $y_i(t_{r_k}, \omega_k)$ denotes the estimate of agent $i \in \mathcal{N}(t_{r_k})$ in realization ω_k at the time t_{r_k} and is defined (for the symmetric push-sum algorithm) as: $y_i(t_{r_k}, \omega_k) = \frac{x_i(t_{r_k}, \omega_k)}{w_i(t_{r_k}, \omega_k)}$. The n initial agents of the systems have weights initialized to 1 which are not decision variables. To be able to solve this problem, the estimates of the agents in a given realization ω_k need to be expressed in terms of the variable $\mathbf{w}_0$ of the problem.

For a given realization ω_k , a simulation of a system (in a given configuration n, λ_c, λ_r) subject to the symmetric push-sum algorithm is launched with unit initial weight for all the agents initially in the system at the beginning, but without fixing the initial weight of any new incoming agent. The simulation rather keeps track of all the coefficients of these initial weights that allows computing x_i and w_i (for all $i \in \mathcal{N}(t)$) as a linear combination of $\mathbf{w}_0$, with an independent term that represents the influence of the initial agents (for which the initial weight is set to 1):

$$x_i(t, \omega_k) = \mathbf{c}_i^T(t, \omega_k) \mathbf{w}_0 + \tilde{c}_i(t, \omega_k) \quad \text{and} \quad w_i(t, \omega_k) = \mathbf{d}_i^T(t, \omega_k) \mathbf{w}_0 + \tilde{d}_i(t, \omega_k).$$

The coefficient can be computed for any discrete time t of the simulation (note that it could better have been realized using the event-based simulation). The components of $\mathbf{c}_i(t, \omega_k)$ and $\mathbf{d}_i(t, \omega_k)$ give the coefficients of the corresponding components of $\mathbf{w}_0$, respectively for $x_i(t, \omega_k)$ and $w_i(t, \omega_k)$. Similarly, the values of $\tilde{c}_i(t, \omega_k)$ and $\tilde{d}_i(t, \omega_k)$ give their corresponding independent term. The coefficients of interest for our optimization problem are the one at the final time of the simulation, after r replacement (at time t_{r_k}). Therefore, we use the following simplified notation for these final coefficients

$$\mathbf{c}_{i,k} = \mathbf{c}_i(t_{r_k}, \omega_k), \quad \tilde{c}_{i,k} = \tilde{c}_i(t_{r_k}, \omega_k) \quad \mathbf{d}_{i,k} = \mathbf{d}_i(t_{r_k}, \omega_k) \quad \text{and} \quad \tilde{d}_{i,k} = \tilde{d}_i(t_{r_k}, \omega_k).$$

Multiple realization ω_k are considered to cancel the random effects and find values of $\mathbf{w}_0$ adapted to a general case. This problem is still dependent on the fixed configuration of the system (system size n , communication and replacement rates λ_c, λ_r).

The resulting optimization problem is written in the following equation

$$\underset{\mathbf{w}_0}{\text{minimize}} \quad \frac{1}{n_{\text{real}}} \sum_{k=1}^{n_{\text{real}}} \left(\frac{1}{n} \sum_{i \in \mathcal{N}(t_{r_k})} \left(\mu - \frac{\mathbf{c}_{i,k}^T \mathbf{w}_0 + \tilde{c}_{i,k}}{\mathbf{d}_{i,k}^T \mathbf{w}_0 + \tilde{d}_{i,k}} \right)^2 \right) \quad (4.11)$$

where the coefficients of the input weights for a given realization $\mathbf{c}_{i,k}, \tilde{c}_{i,k}, \mathbf{d}_{i,k}$ and $\tilde{d}_{i,k}$ have been computed with a simulation of a given multi-agent system (with fixed n, λ_c, λ_r). The problem minimizes the expected mean squared error of the estimates of the agents right after a fixed number of replacements r , computed over n_{real} iterations.

A variant of this optimization problem considers also a fixed number r of replacements in the system but minimizes the expected mean square error of the estimates a long time after that the last replacement have occurred. In other words, after the estimates of the agents have converged toward a common value, after the r^{th} replacement in the system (these estimates do not have any variance anymore). This convergence value can be computed using Proposition 4.1 with t_{r_k} as starting time:

$$\lim_{t \rightarrow \infty} y_i(t) = \frac{\sum_{j \in \mathcal{N}(t_{r_k})} x_j(t_{r_k})}{\sum_{j \in \mathcal{N}(t_{r_k})} w_j(t_{r_k})} \quad \text{for all } i \in \mathcal{N}(t_{r_k}).$$

Using that results, it is not needed to simulate the evolution of the system beyond the time t_{r_k} of the r^{th} replacement and the second variant optimization problem can be written as in equation (4.12).

$$\underset{\mathbf{w}_0}{\text{minimize}} \quad \frac{1}{n_{\text{real}}} \sum_{k=1}^{n_{\text{real}}} \left(\mu - \frac{\sum_{i \in \mathcal{N}(t_{r_k})} \mathbf{c}_{i,k}^T \mathbf{w}_0 + \tilde{c}_{i,k}}{\sum_{i \in \mathcal{N}(t_{r_k})} \mathbf{d}_{i,k}^T \mathbf{w}_0 + \tilde{d}_{i,k}} \right)^2 \quad (4.12)$$

where the coefficients of the input weights for a given realization $\mathbf{c}_{i,k}, \tilde{c}_{i,k}, \mathbf{d}_{i,k}$ and $\tilde{d}_{i,k}$ have been computed with a simulation of a given multi-agent system (with fixed n, λ_c, λ_r).

The solutions of these optimization problems have been computed using the `fminunc` function of Matlab, that uses the quasi-newton method. The starting point is randomly selected (between 0 and 1). These latter comments are valid for all the upcoming optimization problems (i.e. with regularization and parametrization).

Figure 4.3 shows the solutions of both optimization problems for 50 replacements averaged over $n_{\text{real}} = 200$ realizations for two different systems of given size $n = 10$ and with two different rate ratios ($\rho = 100$ and $\rho = 5$). We observe a similar trend that with our theoretical initial weights from Proposition 4.2 but with random spikes probably due to over-optimization (or overfitting) of some realizations. To solve this problem, there are different potential solutions we can test:

- Consider more realizations to reduce the effects of randomness and the risk of overfitting of some realizations. Some tests have been performed with more realizations (up to 1000) but it does not improve the look of the solution much. To obtain smooth solutions, it would probably require too many realizations to be viable. Therefore, this option has been abandoned.
- Add a regularization term in the objective function (4.11) and (4.12) to try obtaining a smooth evolution of the initial weights of the incoming agents. Regularization penalizes solution with abrupt variations of successive $\mathbf{w}_{0,i}$. Different regularizations have been tested and are presented below.
- Parametrize the problem in order to express the problem variable $\mathbf{w}_0$ as a parametrized function: $\mathbf{w}_0 = f(\alpha, \beta, \dots)$, for a chosen function $f: \mathbb{R}^s \rightarrow \mathbb{R}^r$ (for s parameters and r replacements). The optimization problems (4.11) and (4.12) can be written only with the s parameters as decision variable. This solution has been tested and is presented below.
- Try solving the problem using a stochastic gradient descent that loops multiple times over the different realizations separately (randomly ordered) in order to update the current iterate, only based on one realization. It may result in a better efficiency in the resolution and may also reduce overfitting. This potential solution has not been tested.

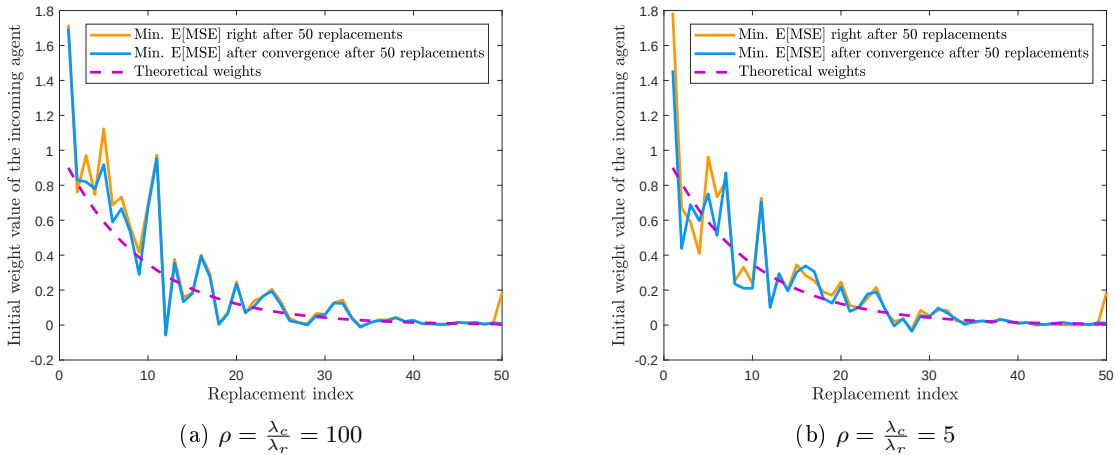


Figure 4.3: Solution of both optimization problems (4.11) and (4.12) for $r = 50$ replacements and $n_{\text{real}} = 200$ realizations, for a given system size $n = 10$ and two different rate ratios $\rho = \frac{\lambda_c}{\lambda_r}$.

Regularization Different regularizations can be considered. Let $\lambda > 0$ be the importance of the regularization term and let $\mathbf{w}_{0,0}$ denote the initial weights of all agents initially present in the system at the beginning which is not a decision variable since it is fixed to $\mathbf{w}_{0,0} = 1$. For $i = 1, \dots, r$, $\mathbf{w}_{0,i}$ denotes the i^{th} component of the variable vector $\mathbf{w}_0$, and hence the initial weight of the i^{th} agent to enter the system. Different regularization terms can be added to the initial objective value (from equation (4.11) or (4.12)):

$$\text{Absolute: } \lambda \sum_{i=1}^r |\mathbf{w}_{0,i-1} - \mathbf{w}_{0,i}|, \quad \text{Squared: } \lambda \sum_{i=1}^r (\mathbf{w}_{0,i-1} - \mathbf{w}_{0,i})^2.$$

Since the initial weights of the incoming agents should be decreasing with the rank i of their arrival (according to our initial design idea), it is useful to consider logarithmic regularizations that penalize less the variation when the weights are large (in order to allow them to decrease potentially exponentially).

$$\text{Absolute Log: } \lambda \sum_{i=1}^r |\log(\mathbf{w}_{0,i-1}) - \log(\mathbf{w}_{0,i})|, \quad \text{Squared Log: } \lambda \sum_{i=1}^r (\log(\mathbf{w}_{0,i-1}) - \log(\mathbf{w}_{0,i}))^2 \quad (4.13)$$

In most of the situations, the squared logarithmic regularization solves well the problem by smoothing the successive initial weights.

Figure 4.4 shows the solution of both optimization problems ((4.11) and (4.12)) with a squared logarithmic regularization term with coefficient $\lambda = 0.01$ in two different situations: for a system with infrequent replacements ($n = 10$ and $\rho = 100$) and a system with frequent replacements ($n = 10$ and $\rho = 5$). The ordinate scale is logarithmic to allow a better comparison of the curves.

In both situations, the optimal solutions are quite similar and looks like to the theoretical weights (from Proposition 4.2) but with a decrease that is a bit slower. It is probably due to the regularization term since it does not perform better on real simulations (on the contrary, it is a bit worst in expectation). We still note that for frequent replacements (Figure 4.4b on the right), to optimize the expected MSE right after the 50 replacements, the solution starts to decrease faster the initial weights a bit before the 50th replacement. This allows to reduce faster the expected variance that each new agent brings into the system but should impact the expected bias at long term (after the 50th replacement). It is understandable that the optimal solution tries to reduce the expected variance faster since it is the dominant term of the expected mean squared error in this frequent replacements situation (as shown in Figure 4.2).

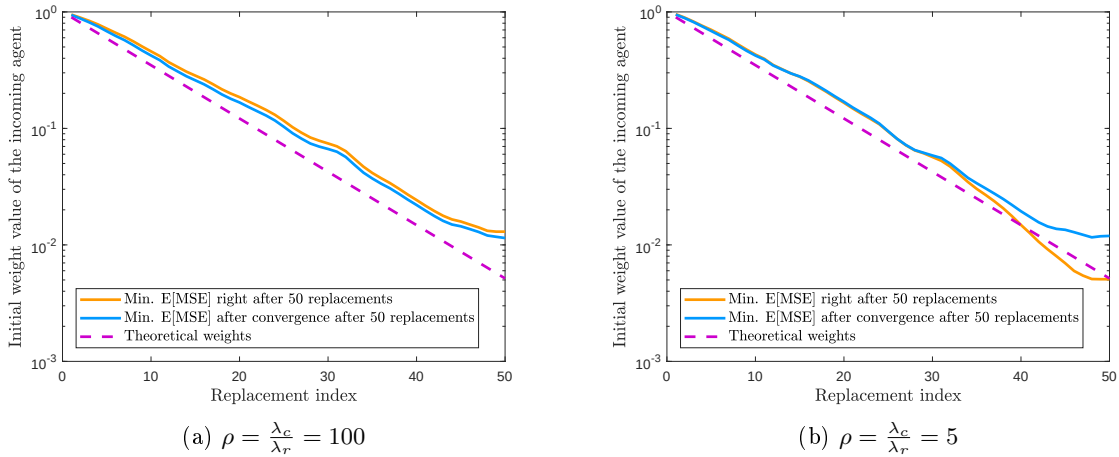


Figure 4.4: Solution of both regularized optimization problems (4.11) and (4.12) with squared logarithmic regularization term (with $\lambda = 0.01$) (from equation (4.13)) for $r = 50$ replacements and $n_{\text{real}} = 200$ realizations, for a given system size $n = 10$ and two different rate ratios $\rho = \frac{\lambda_c}{\lambda_r}$. The ordinate scale is logarithmic to allow a better comparison of the curves.

Parametrization This paragraph presents a parametrization of the problem in order to eliminate the issues related to overfitting or to regularization. It obviously reduces the possibilities of the solution weights, but it can still give a good idea of optimal solution behavior. Parametrization also allows to extend the results behind the 50th replacements on which we optimize, to observe if it generalizes well to any number of replacements.

Using the parametrized initial weights, $\mathbf{w}_0 = f(\alpha, \beta, \dots)$, the optimization problem (4.11) becomes:

$$\underset{\alpha, \beta, \dots}{\text{minimize}} \quad \frac{1}{n_{\text{real}}} \sum_{k=1}^{n_{\text{real}}} \left(\frac{1}{n} \sum_{i=1}^n \left(\mu - \frac{\mathbf{c}_{i,k}^T f(\alpha, \beta, \dots) + \tilde{c}_{i,k}}{\mathbf{d}_{i,k}^T f(\alpha, \beta, \dots) + \tilde{d}_{i,k}} \right)^2 \right) \quad (4.14)$$

And the second optimization problem (4.12) presents similar changes:

$$\underset{\alpha, \beta, \dots}{\text{minimize}} \quad \frac{1}{n_{\text{real}}} \sum_{k=1}^{n_{\text{real}}} \left(\mu - \frac{\sum_{i=1}^n \mathbf{c}_{i,k}^T f(\alpha, \beta, \dots) + \tilde{c}_{i,k}}{\sum_{i=1}^n \mathbf{d}_{i,k}^T f(\alpha, \beta, \dots) + \tilde{d}_{i,k}} \right)^2 \quad (4.15)$$

We need to choose the parametrized function f to use. For s parameters and r replacements, it needs to be a real function $f: \mathbb{R}^s \rightarrow \mathbb{R}^r$. Intuitively, we think about a function that would have similar shape as the theoretical initial weights obtained in Proposition 4.2: $w_{n+r}(0) = \left(\frac{n-1}{n}\right)^r$. We can for example choose:

$$\mathbf{w}_{0,i} = f_i(\alpha, \beta, \gamma) = (\alpha + \beta i)(\gamma)^i + \theta. \quad (4.16)$$

For the theoretical weights, it corresponds to set $\alpha = 1$, $\beta = 0$, $\gamma = \frac{n-1}{n}$ and $\theta = 0$.

Figure 4.5 presents the solutions of these parametrized problems ((4.14) and (4.15)) for two different systems (the same as previously): one with infrequent replacements ($n = 10$ and $\rho = 100$) and a one with frequent replacements ($n = 10$ and $\rho = 5$).

For infrequent replacements (Figure 4.5a), the optimal solutions correspond (almost perfectly) to the theoretical initial weights. Indeed, from a random starting point, the optimization method has converged to a similar solution for both problems (rounded to 2 decimal places): $\alpha = 0.97$, $\beta = 0$, $\gamma = 0.90$ and $\theta = 0$. The most impacting parameter is the decreasing rate γ and it is very close to the theoretical one ($\gamma = \frac{n-1}{n} = 0.9$). Moreover, the optimal weights improve the objective value of the theoretical weights of only 0.2% for optimization problem (4.14) and 0.02% for (4.15).

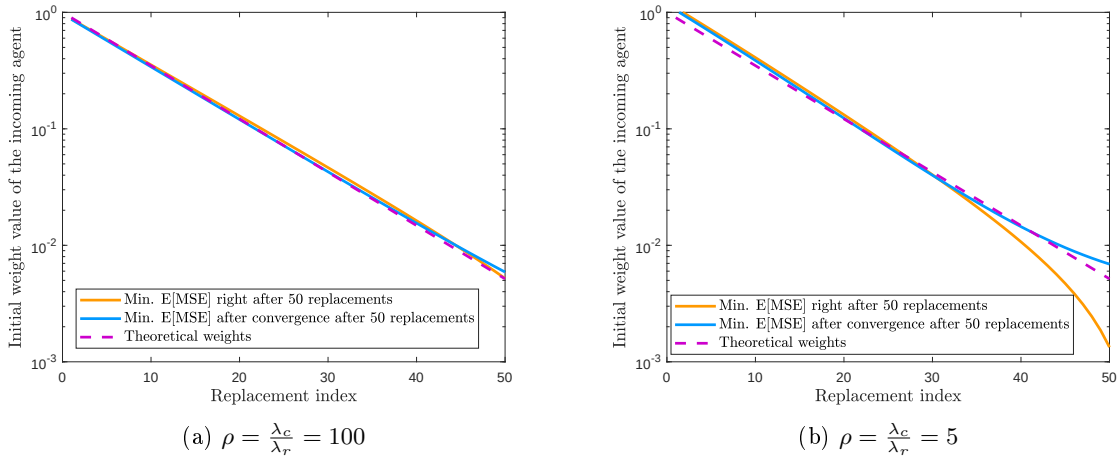


Figure 4.5: Solution of both parametrized optimization problems (4.14) and (4.15) (with parametrized function from equation (4.16)) for $r = 50$ replacements and $n_{\text{real}} = 200$ realizations, for a given system size $n = 10$ and two different rate ratios $\rho = \frac{\lambda_c}{\lambda_r}$. The ordinate scale is logarithmic to allow a better comparison of the curves.

For frequent replacements (Figure 4.5b), the improvement of the objective value (in comparison with theoretical weights) is slightly more significant (about 1%) for both problems. The optimal weights still

fit pretty well the theoretical ones. The optimal decreasing rate γ (for both problems) is about 0.88 (a bit lower than the theoretical 0.9). It leads to a slightly faster decrease (in comparison with the theoretical weights), compensated in a first time by the factor α that is a bit larger than 1 (respectively 1.15 and 1.22). Their independent term (θ) is close to zero but still impact the curve when its values becomes small, i.e. after about 30 replacements. Here, as for the regularized problems, the optimal weights of the first problem decrease heavily before the 50th replacement in order to ensure a lower expected variance. This is not the case for the second optimization problem that optimize the expected MSE after convergence (after 50 replacements) and for which the expected variance part is always zero. These independent non-zero terms θ imply a bad generalization after 50 replacements. Even without these independent terms, these optimal solutions do not generalize well after 50 replacements, even if some minor improvements are indeed present before the 50th replacement.

The theoretical initial weights from Proposition 4.2 seems to be nearly optimal in different situations, with frequent and infrequent replacements. Figure 4.6 shows the evolution with the system's rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$ of the relative difference between the optimal objective value and the objective value provided by the theoretical weights (for both optimization problems).

Figure 4.6a (on the left) is applied to the regularized problems (with the same squared logarithmic regularization $\lambda = 0.01$). For both optimization problems (4.11) and (4.12) (with regularization), the relative difference remains almost constant with the rate ratio ρ , meaning that the theoretical weights are as close to the true optimum for every rate ratio. When minimizing the expected MSE right after 50 replacements with a squared logarithmic regularization term, the relative difference is about 0.01, meaning that the optimal solution improves the objective value of the theoretical weight only of 1%. For the variant problem, it is between 5 and 10%. These improvements may be related to the expected MSE term or the regularization term of the objective function. This seems to indicate that our theoretical weights are closer to optimizing the expected value of the MSE right after a fixed number of replacements (problem (4.11)) than after the convergence (problem (4.12)).

Figure 4.6b (on the right) is applied to the parametrized problems (with parametrization from (4.16)). The relative difference between the objectives is not constant with the rate ratio of the system: larger improvements are possible for small rate ratio ρ . They remain very small (less than 1% for problem (4.11) and than 5% for problem (4.12)). The separation between the two variants of the optimization problem is not as obvious as in Figure 4.6a but it still seems that the relative difference of objectives for the first problem (4.11) is globally below the relative difference of objectives for the variant problem (4.12).

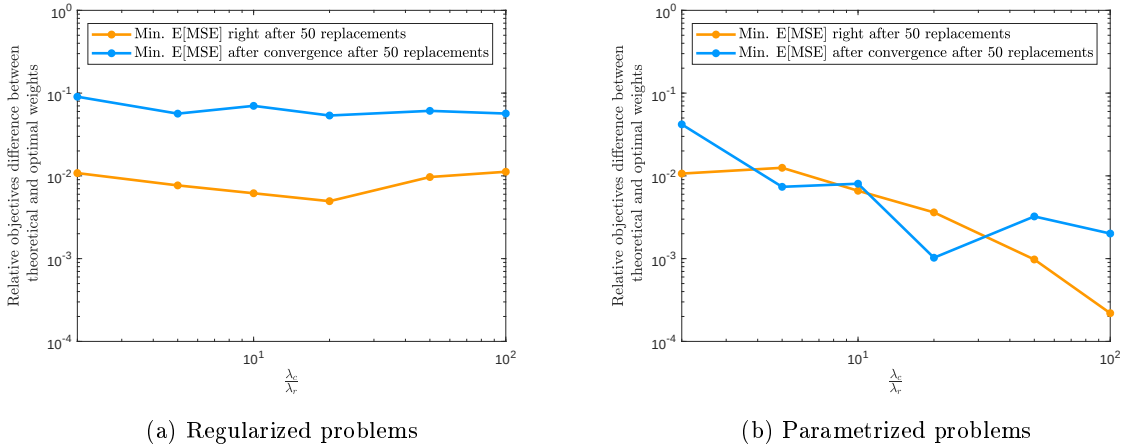


Figure 4.6: Relative difference between the objective value given by the theoretical weights (from Proposition 4.2) and the optimal objective value for both regularized optimization problems ((4.11) and (4.12) with squared logarithmic regularization from equation (4.13) with $\lambda = 0.01$) (plotted on the right) and both parametrized optimization problems ((4.14) and (4.15) with parametrized function from equation (4.16)) (on the left), for $r = 50$ replacements and $n_{\text{real}} = 200$ realizations, for a given system size $n = 10$ and for different rate ratios $\rho = \frac{\lambda_c}{\lambda_r}$. The scale is logarithmic to allow a better comparison of the curves.

Using a mix of simulations and optimization, this subsection has illustrated the fact that the theoretical weights of incoming agents from Proposition 4.2 are close to optimize the expected value of the mean squared error in the system in all system configurations (frequent or infrequent replacements) and I do not find any other initial weights that performs significantly better for any number of replacements.

4.3 Performance analysis

Here is a summary of the symmetric push-sum algorithm that we have designed and for which the performance is analyzed in this section. It incorporates the initial weights given in Proposition 4.2 into the general framework of the algorithm described by equations (4.1), (4.2) and (4.3). Let us remember that we suppose indexing of agents such that the initial set of agents in the system (at time $t = 0$) is $\mathcal{N}(0) = \{1, \dots, n\}$ and the r^{th} agent to arrive into the system during its operation have index $n + r$.

Designed symmetric push-sum algorithm (SPS):

Initialization of agent i before it enters the system

$$x_i(0) = w_i(0)z_i, \quad \text{for all } i \quad \text{and} \quad w_i(0) = \begin{cases} 1 & \text{for } i = 1, \dots, n \\ \left(\frac{n-1}{n}\right)^{n-i} & \text{for } i = n+1, n+2, \dots \end{cases} \quad (4.17)$$

Interaction between agents $i \in \mathcal{N}(t)$ and $j \in \mathcal{N}(t)$:

$$x_i^+ = x_j^+ = \frac{x_i + x_j}{2}, \quad \text{and} \quad w_i^+ = w_j^+ = \frac{w_i + w_j}{2}. \quad (4.18)$$

Estimate of agent $i \in \mathcal{N}(t)$:

$$y_i(t) = \frac{x_i(t)}{w_i(t)}. \quad (4.19)$$

This section first analyzes the time evolution of the performance of the symmetric push-sum algorithm for four different representative system configurations and in comparison to the performance of other algorithms. In a second time, the performance of the symmetric push-sum algorithm after 500 seconds is analyzed in a larger variety of system configurations to show how its (nearly) steady state performance is situated in comparison with other algorithms (the gossip averaging algorithm and the best ideal algorithm).

4.3.1 Time evolution of the expected performance

Figure 4.7 shows the time evolution of the expected mean squared error ($\mathbb{E}[\text{MSE}(t)]$) of the symmetric push-sum algorithm, in comparison with the one of the gossip averaging algorithm (from Chapter 3) and with the empirical lower bound on the mean squared error (from Chapter 2), in four different scenarios corresponding to four different system configurations ($n = 10$ or $n = 50$ and $\rho = 5$ or $\rho = 100$). In each scenario, the expected MSE of the symmetric push-sum (in red) is quite better than the one of the gossip averaging algorithm (in black) and seems tending to the empirical lower bound (in green) and having a similar steady state value which is very good news. But the time needed for the expected mean squared error of the symmetric push-sum to stick definitively the empirical lower bound is different from one scenario to another. This can be explained by observing the decomposition of the expected mean squared error into expected squared bias and expected variance in each of these scenarios, as shown in Figure 4.8. We can observe that

- The expected squared bias of the symmetric push-sum algorithm sticks very well to the one of the best ideal algorithm (that provides an empirical lower bound on the MSE) since the beginning time and for all the different system configurations considered in Figure 4.8. This is due to the setting of the initial weights of the incoming agents that has been designed in Section 4.2 with the ideal assumption that convergence occurs between each replacement (i.e. replacement are extremely infrequent) and that works surprisingly properly when this assumption is relaxed.
- The expected variance of the symmetric push-sum algorithm does not stick directly to the one of the best ideal algorithm but decreases and tends to it more or less quickly depending on the system

configuration. This variance decrease depends mainly on the decrease of the initial weights of new agents (in comparison with the average weight of the agents in the system). Indeed, the lower the weight of a new agent is (in comparison with others), the less influence it has on the estimates of other agents (and on itself), since a symmetric push-sum communication corresponds to a weighted average of the estimates (see equation (4.4)). The speed of decrease of the initial weights of new agents with time depends on the decreasing rate at each replacement, given by $\frac{n-1}{n}$ but also on the frequency of the replacements. For large n , the initial weights decrease slower at each replacement and for large ρ there are less replacements (in average) per unit of time. Indeed, when we use our scaling of the parameters that keeps $n\lambda_c = 10$ (see Section 1.3 for discussion), a larger ρ implies a lower λ_r . Therefore, the decrease of the initial weights, and hence of the expected variance, is slower for system with large size n and/or large rate ratios ρ . That is well what we observe in Figure 4.8.

- For any algorithm, the expected mean squared bias and the expected variance have more or less importance into the expected mean squared error depending on the system configuration. The expected variance is in general smaller for systems with larger rate ratio (which is equivalent to less frequent replacements with our scaling $n\lambda_c = 10$). For example, one can compare expected variance in Figures 4.8a and 4.8c where ρ passes from 5 to 100. The expected squared bias is in general smaller for larger systems. For example, one can compare Figures 4.8a and 4.8b where the system size passes from $n = 10$ to $n = 50$.

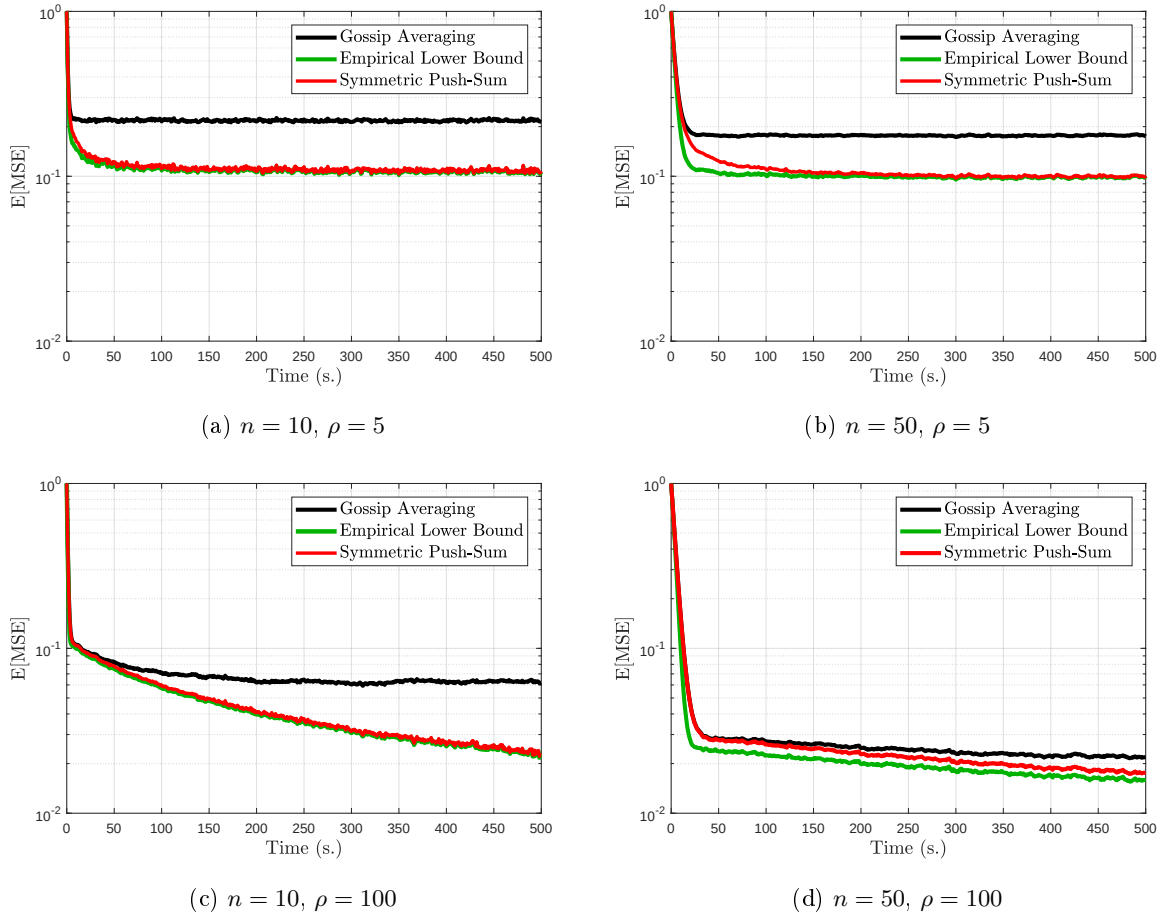
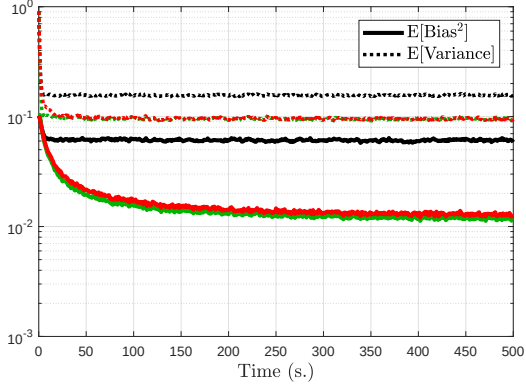
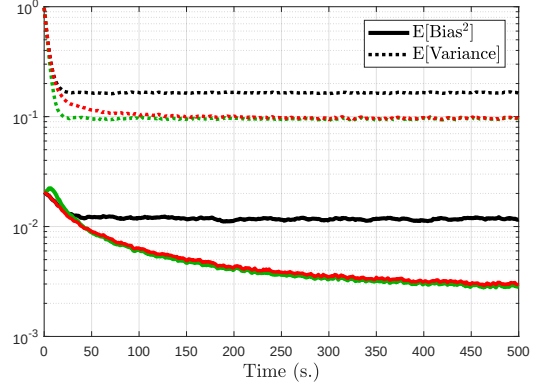


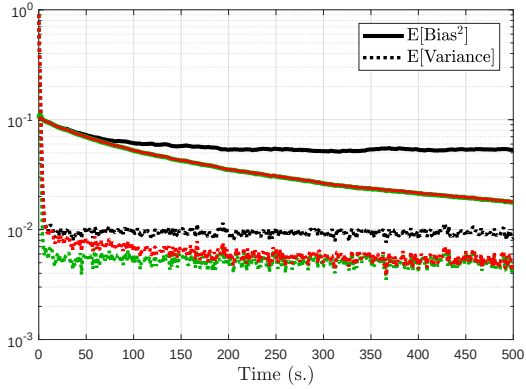
Figure 4.7: Comparison between the expected mean squared error time evolution ($\mathbb{E}[\text{MSE}(t)]$) of the symmetric push-sum algorithm, the gossip averaging algorithm and the best ideal algorithm (that provides the empirical lower bound on $\mathbb{E}[\text{MSE}(t)]$) for four different system configurations (n and $\rho = \frac{\lambda_c}{\lambda_r}$) with scaling $n\lambda_c = 10$. Expectations have been computed empirically with 5000 realization simulations.



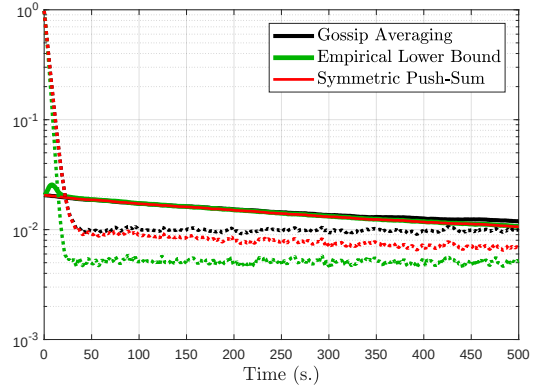
(a) $n = 10, \rho = 5$



(b) $n = 50, \rho = 5$



(c) $n = 10, \rho = 100$



(d) $n = 50, \rho = 100$

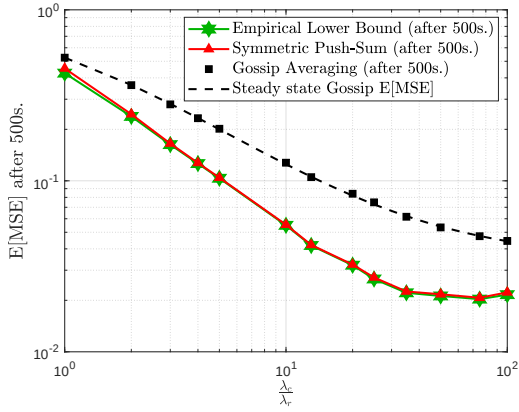
Figure 4.8: Comparison between the expected squared bias (solid line) and the expected variance (dotted line) time evolution of the symmetric push-sum algorithm (in red), the gossip averaging algorithm (in black) and the best ideal algorithm (in green) (that provides the empirical lower bound on $\mathbb{E}[\text{MSE}(t)]$) for four different system configurations (n and $\rho = \frac{\lambda_c}{\lambda_r}$) with scaling $n\lambda_c = 10$. Expectations have been computed empirically with 5000 realization simulations.

Knowing all this helps to better understand why the symmetric push-sum mean squared error does not fit directly the empirical lower bound when the system size n is large (as in Figure 4.7b) and why it is worst when in addition, the rate ratio ρ is large (as in Figure 4.7d). The next subsection analyzes the performance of the algorithms after 500 seconds in a larger variety of system configurations to show if the performance of symmetric push-sum algorithm tends to the empirical lower bound in all situations, as it has been observed until here.

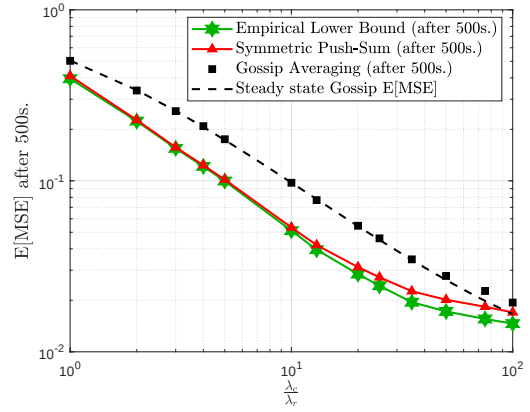
4.3.2 Expected performance after 500 seconds

Figure 4.9 shows the performance (expected mean squared error) achieved by the symmetric push-sum algorithm after 500 seconds for different system configurations. This performance is compared to the one of the gossip averaging algorithm (from Chapter 3) and the best ideal algorithm (from Chapter 2). Actually it is the same figure than Figure 3.3 with an additional algorithm: the symmetric push-sum that we have just described. As a reminder, these figures show the impact of the system size n and the rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$ on the expected mean squared error of the algorithms after 500 seconds. The parameter scaling is done with constant $n\lambda_c = 10$, as explained in Section 1.3. Concretely, Figures 4.9a and 4.9b show the evolution with the rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$ respectively for a small and a large system ($n = 15$ and $n = 75$) and Figures 4.9c and 4.9d show the evolution with the system size n respectively for a small and a large rate ratio ($\rho = 4$ and $\rho = 50$).

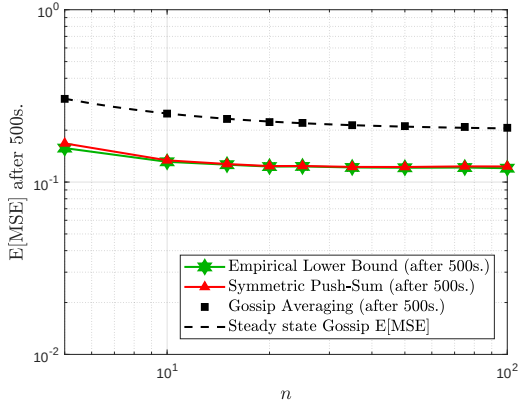
We observe that the value of the expected mean squared error of a system subject to this symmetric push-sum algorithm (in red) after 500 seconds is always better than the gossip averaging algorithm (in black) and almost always reaches the empirical lower bound (in green). After 500 second, the performance of each algorithm has, in many situations, almost reached their steady state, as suggested by previous figures (see Figure 4.7). But there are some situations where the convergence to the steady state is slower and this steady state is not reached in 500 seconds (see Figure 4.7d). It is the case for systems with too large rate ratios ρ , implying a too small replacement rate (using our scaling $n\lambda_c = 10$). Therefore, for large rate ratios, the curves correspond to performance of the algorithms before the steady state is reached (except maybe for the gossip algorithm that converges faster). Hence, the gap between the expected mean squared error of the symmetric push-sum algorithm and the empirical lower bound that appears for large n and large ρ illustrates, using another perspective, the main fact illustrated and explained in the previous sub-section that analyzes time evolution of the performance: the symmetric push-sum mean squared error takes more time to fit the empirical lower bound when the system size n is large (as in Figure 4.7b) and when in addition, the rate ratio ρ is large, due to the slow decrease of its expected variance in these cases.



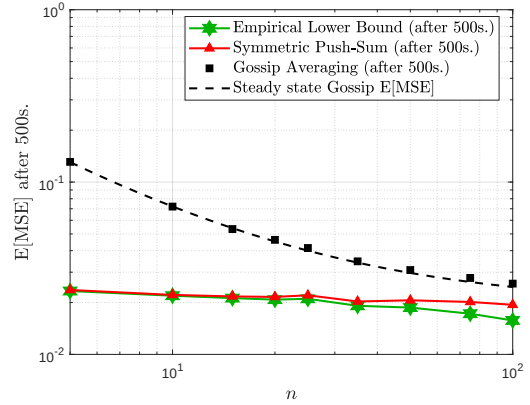
(a) Evolution with rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$, for $n = 15$.



(b) Evolution with rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$, for $n = 75$.



(c) Evolution with system size n , for $\rho = \frac{\lambda_c}{\lambda_r} = 4$.



(d) Evolution with system size n , for $\rho = \frac{\lambda_c}{\lambda_r} = 50$.

Figure 4.9: Comparison between the expected mean squared error after 500 seconds of the symmetric push-sum algorithm, the averaging gossip algorithm and the one of the best ideal algorithm (that provides an empirical lower bound) for a variety of different systems (with constant $n\lambda_c = 10$). Expectations have been computed empirically with 1000 realization simulations. The theoretical expression of the steady state mean squared error of the gossip algorithm (see equation (3.5)) is also plotted (dashed black line).

In conclusion of this section, we have seen that the symmetric push-sum has much better performance than the gossip averaging algorithm for solving the external averaging problem stated in Section 1.1.1.

The expected mean squared error of the algorithm seems to reach closely the empirical lower bound (from Chapter 2) in steady state for any system configuration. Improvement is still possible concerning the speed at which the expected mean squared error of the symmetric push-sum algorithm reaches the empirical lower bound, mainly due to its expected variance part. Chapter 5 presents another algorithm that reduces very quickly the expected variance and Chapter 6 combines the idea of this new algorithm with the one of the symmetric push-sum in order to obtain all their benefits.

Before exploring all of this, let us first try to analyze theoretically the evolution of a system subject to the symmetric push-sum algorithm. We will also discuss the practical implementation of the algorithm.

4.4 Theoretical evolution of the system

In this section, we start to describe the theoretical evolution of an open multi-agent system subject to the symmetric push-sum algorithm. The obtained description is partial (only about the x and w values and not the estimates) and does not lead to performance guarantees.

The performance achieved by an algorithm can be expressed using some descriptors of the distribution of the estimates of the agents in the system. As a reminder, omitting time and realization dependence, we have

$$\begin{aligned}\mathbb{E}[\text{Bias}^2] &= \mathbb{E}[(\bar{y} - \mu)^2] = \mathbb{E}[\bar{y}^2] + \mu^2 - 2\mu\mathbb{E}[\bar{y}] \quad (\text{for } \mu = 0) \quad \mathbb{E}[\bar{y}^2], \\ \mathbb{E}[\text{Var}] &= \mathbb{E}\left[\frac{1}{n} \sum_{i \in \mathcal{N}} (y_i - \bar{x})^2\right] = \mathbb{E}[\bar{y}^2] - \mathbb{E}[\bar{y}]^2, \\ \mathbb{E}[\text{MSE}] &= \mathbb{E}[\text{Bias}^2] + \mathbb{E}[\text{Var}] \quad (\text{for } \mu = 0) \quad \mathbb{E}[\bar{y}^2],\end{aligned}$$

$$\text{where} \quad \bar{y} = \frac{1}{n} \sum_{i \in \mathcal{N}} y_i \quad \text{and} \quad \bar{y}^2 = \frac{1}{n} \sum_{i \in \mathcal{N}} y_i^2.$$

Without loss of generality, we suppose that $\mu = 0$ and hence it is sufficient to describe the evolution of $\mathbb{E}[\bar{y}^2]$ and $\mathbb{E}[\bar{y}]$. Unfortunately describing evolution of these descriptors for the symmetric push-sum algorithm is not as easy as for the gossip averaging algorithm (see Section 3.1.1) since the estimates are ratios of two variables: $y_i = \frac{x_i}{w_i}$. The first subsection describes the difficulties when applying a similar reasoning to the one we used for the gossip averaging algorithm (see Section 3.1.1). Thereafter, the system evolution is still analyzed through the separate analysis of the x values and the weights of the agents.

Difficulties to describe evolution of $\mathbb{E}[\bar{y}^2]$ and $\mathbb{E}[\bar{y}]$

Let us denote the descriptors into one vector: $\mathbb{E}Y = [\mathbb{E}[\bar{y}^2], \mathbb{E}[\bar{y}]]$. As for the gossip, we can consider the evolution of this vector with the events that occur in the system (communications and replacements). $\mathbb{E}Y_k$ denotes the vector after event k . The purpose is to write the evolution of this vector as a system of recurrence relations (i.e. $\mathbb{E}Y_k$ in function of $\mathbb{E}Y_{k-1}$ (and possibly other function of k)). Let us analyze what happens for each kind of event separately.

A replacement implies the arrival of one new agent i . The algorithm is such that its initial estimate $y_i(0)$ is equal to its own measurement of μ : z_i (see equation (4.17)). There is also an agent l that leaves the system with all its personal information (y_l, x_l, w_l). Therefore, the situation is similar to the one with the gossip averaging algorithm and the Lemma 3.2 presented in Section 3.1.1 can be reused. Hence, equation (4.20) can be written for the update of descriptor vector $\mathbb{E}Y_k$.

$$\mathbb{E}Y_k = A_r \mathbb{E}Y_{k-1} + b_r, \quad \text{with} \quad A_r = \begin{bmatrix} 1 - \frac{2}{n} & \frac{1}{n^2} \\ 0 & 1 - \frac{1}{n} \end{bmatrix}, \quad b_r = \sigma^2 \begin{bmatrix} \frac{1}{n^2} \\ \frac{1}{n} \end{bmatrix}. \quad (4.20)$$

The independent term b_r appears due to arrival of a new agent i with its own measurement z_i of variance σ^2 and mean $\mu = 0$.

A communication between agent i and j implies a different evolution of the descriptor vector (than for the gossip algorithm). For the following, the $+$ in the exponent indicates the values after the communication (and just before it without the $+$ exponent). We can compute $\mathbb{E}[(\bar{y}^2)^+]$ and $\mathbb{E}[\bar{y}^{2+}]$ by decomposing the expectation in two: the expectation over all possible sequences of events ϵ until this communication that allows knowing the y , x and w values of all agents just before the communication occurs, and the expectation over the two agents i, j selected for the communication. We have that $\mathbb{E}[(\bar{y}^2)^+] = \mathbb{E}_\epsilon[\mathbb{E}_{i,j}[(\bar{y}^2)^+ | y, x, w]]$ and $\mathbb{E}[\bar{y}^{2+}] = \mathbb{E}_\epsilon[\mathbb{E}_{i,j}[\bar{y}^{2+} | y, x, w]]$. After a communication between agents i and j with the symmetric push-sum, $(\bar{y}^2)^+$ and $\bar{y}^{2+}$ can be expressed using previous values of y and w using the weighted average vision of the symmetric push-sum update for a communication (see equation (4.4)) that combines the communication update from equation (4.18) and the definition of the estimates from equations (4.19).

$$(\bar{y}^2)^+ = \left(\bar{y} + \frac{1}{n} \left(-y_i - y_j + 2 \frac{w_i y_i + w_j y_j}{w_i + w_j} \right) \right)^2 \quad \text{and} \quad \bar{y}^{2+} = \bar{y}^2 + \frac{1}{n} \left(-y_i^2 - y_j^2 + 2 \left(\frac{w_i y_i + w_j y_j}{w_i + w_j} \right)^2 \right)$$

Since both communicating agents are selected uniformly randomly and independently among the agents of the system, we have that $\mathbb{E}_{i,j}[y_i^2 | y] = \bar{y}^2$ and that $\mathbb{E}_{i,j}[y_i y_j | y] = \bar{y}^2$. Therefore, we have

$$\begin{aligned} \mathbb{E}_{i,j}[(\bar{y}^2)^+ | y, x, w] &= \left(1 + \frac{2}{n^2} - \frac{4}{n} \right) \bar{y}^2 + \frac{2}{n^2} \bar{y}^2 + \frac{4}{n^2} \mathbb{E}_{i,j} \left[\left(\frac{w_i y_i + w_j y_j}{w_i + w_j} \right)^2 \middle| y, x, w \right] \\ &\quad + \frac{4}{n} \bar{y} \mathbb{E}_{i,j} \left[\frac{w_i y_i + w_j y_j}{w_i + w_j} \middle| y, x, w \right] - \frac{8}{n^2} \mathbb{E}_{i,j} \left[\left(\frac{w_i y_i + w_j y_j}{w_i + w_j} \right) y_i \middle| y, x, w \right], \\ \mathbb{E}_{i,j}[\bar{y}^{2+} | y, x, w] &= \left(1 - \frac{2}{n} \right) \bar{y}^2 + \frac{2}{n} \mathbb{E}_{i,j} \left[\left(\frac{w_i y_i + w_j y_j}{w_i + w_j} \right)^2 \middle| y, x, w \right] \end{aligned}$$

Unfortunately, to follow the evolution of the two descriptors $\mathbb{E}[\bar{y}^2]$ and $\mathbb{E}[\bar{y}^{2+}]$, for a communication, it is required to follow the evolution of many others descriptors such as $\mathbb{E}[\bar{y}]$ or other more intricate ones involving $\frac{w_i y_i + w_j y_j}{w_i + w_j}$:

$$\mathbb{E}_\epsilon \mathbb{E}_{i,j} \left[\frac{w_i y_i + w_j y_j}{w_i + w_j} \middle| y, x, w \right], \quad \mathbb{E}_\epsilon \mathbb{E}_{i,j} \left[\left(\frac{w_i y_i + w_j y_j}{w_i + w_j} \right)^2 \middle| y, x, w \right] \quad \text{and} \quad \mathbb{E}_\epsilon \mathbb{E}_{i,j} \left[\left(\frac{w_i y_i + w_j y_j}{w_i + w_j} \right) y_i \middle| y, x, w \right].$$

These descriptors are much more complicated to analyze and to compute since they involve the expectation of a ratio that imply both the weights and the estimates of the agents. The reasoning used for analyzing the gossip algorithm leads here to a complex situation that looks like a stalemate. Hence, we cannot deduce the theoretical performance of the symmetric push-sum algorithm in that way. Other simplified situations should be considered to hope to unlock the situation, like having cyclic events: exactly ρ communication events (assuming that ρ is an integer) between two replacement events. This has not been explored. However, analyzing the evolution of both agent-dependent variables x and w independently can allow to describe the main tendency of the system evolution, even if it cannot really be extended to characterization of the performance.

Evolution of $\mathbb{E}[\bar{x}^2]$, $\mathbb{E}[\bar{x}^2]$ and $\mathbb{E}[\bar{w}]$

The motivation to study the evolution of the descriptor vector $\mathbb{E}X = [\mathbb{E}[\bar{x}^2], \mathbb{E}[\bar{x}^2]]$ is first to try to describe the evolution of the system in a simpler way. As for the gossip, we can consider the evolution of this vector with events that occur in the system (communications and replacements). $\mathbb{E}X_k$ denotes the vector after event k . The purpose is to write the evolution of this vector as a system of recurrence relations (i.e. $\mathbb{E}X_k$ in function of $\mathbb{E}X_{k-1}$ (and possibly other function of k)). Let us analyze what happens for each kind of event separately. We again suppose, without loss of generality that $\mu = 0$.

A communication modifies the x values exactly in the same way as the gossip algorithm (see equation (4.18)). The evolution of the descriptor vector is therefore the same, and the Lemma 3.1 can be re-written to describe the linear update of $\mathbb{E}X$ for a communication with the symmetric push-sum algorithm:

Lemma 4.1 (Update of the descriptor vector $\mathbb{E}X$ with a communication (for the SPS algorithm)). *Let k be a communication event in an open multi-agent system of size n and with $\mu = 0$ subject to the symmetric push-sum algorithm (see the problem statement in Section 1.1.1 for a general definition of a communication event or an agent). For a communication in this algorithm, the update is defined by equation (4.18) and the value of the descriptor vector $\mathbb{E}X = [\mathbb{E}[\bar{x}^2], \mathbb{E}[\bar{x}^2]]^T$ is updated according to*

$$\mathbb{E}X_k = A_c \mathbb{E}X_{k-1}, \quad \text{with} \quad A_c = \begin{bmatrix} 1 & 0 \\ \frac{1}{n} & 1 - \frac{1}{n} \end{bmatrix}.$$

Proof. In the symmetric push-sum algorithm, the x values are evolving according to the scheme of the gossip averaging algorithm during a communication event (see equation (4.18)) and therefore Lemma 3.1 can be applied to the symmetric push-sum algorithm to describe the evolution of the descriptor vector $\mathbb{E}X$, that only involves the x values (except for the initialization but it does not impact the result), during a communication event. $\square$

A replacement updates the descriptors differently from the gossip case (in Lemma 3.2), since the initial $x_i(0)$ of the new incoming agent i is not equal to z_i (as for the gossip) but to $w_i(0)z_i$. The update of the descriptor vector $\mathbb{E}X$ for a replacement can be summarized by the following lemma.

Lemma 4.2 (Update of the descriptor vector $\mathbb{E}X$ with a replacement (for the SPS algorithm)). *Let k be a replacement event in an open multi-agent system of size n and with $\mu = 0$ subject to the symmetric push-sum algorithm (see the problem statement in Section 1.1.1 for a general definition of a replacement event or an agent). For a replacement, the algorithm initializes the new agent i according to equation (4.17) and the value of the descriptor vector $\mathbb{E}X = [\mathbb{E}[\bar{x}^2], \mathbb{E}[\bar{x}^2]]^T$ is updated according to*

$$\mathbb{E}X_k = A_r \mathbb{E}X_{k-1} + b_r \left(1 - \frac{1}{n}\right)^2 (c_1^{k-1} + c_2^{k-1})$$

with $A_r = \begin{bmatrix} 1 - \frac{2}{n} & \frac{1}{n^2} \\ 0 & 1 - \frac{1}{n} \end{bmatrix}$, $b_r = \sigma^2 \begin{bmatrix} \frac{1}{n^2} \\ \frac{1}{n} \end{bmatrix}$, $c_1 = 1 - \frac{2n-1}{n^2} p_r$, and $c_2 = -(1 - p_r)$.

The matrix A_r and the vector b_r are the same as for the gossip averaging algorithm (see Lemma 3.2). The update differs only with the decreasing factor that multiplies the vector b_r .

Proof. For the following developments, the $+$ in the exponent indicates the values after the replacement (and just before it without the $+$ exponent). We can compute $\mathbb{E}[(\bar{x}^2)^+]$ and $\mathbb{E}[\bar{x}^{2+}]$ by decomposing the expectation in two: the expectation over all possible sequences of events ϵ until this replacement that allows knowing the number of replacements $r - 1$ that have already occurred in the system and the x values of all agents just before the replacement occurs; and the expectation over the information related to this replacement (denoted $\mathbb{E}_{\text{repl}}$): the index of the agents l selected to leave the system and the measurement z_i of the incoming agent. We have that $\mathbb{E}[(\bar{x}^2)^+] = \mathbb{E}_\epsilon[\mathbb{E}_{\text{repl}}[(\bar{x}^2)^+ | x, r]]$ and $\mathbb{E}[\bar{x}^{2+}] = \mathbb{E}_\epsilon[\mathbb{E}_{\text{repl}}[\bar{x}^{2+} | x, r]]$. After a replacement of the agent l by the new agent i in the symmetric push-sum, $(\bar{x}^2)^+$ and $\bar{x}^{2+}$ can be expressed using previous values of x and the one of the new agent $x_i = w_i(0)z_i$ (see initialization equation (4.17)).

$$(\bar{x}^2)^+ = \left(\bar{x} + \frac{1}{n}(w_i(0)z_i - x_l)\right)^2 \quad \text{and} \quad \bar{x}^{2+} = \bar{x}^2 + \frac{1}{n}((w_i(0)z_i)^2 - x_l^2)$$

Since the leaving agent is selected independently and uniformly randomly in the system for the replacement, we have that $\mathbb{E}_{\text{repl}}[x_l^2 | x, r] = \bar{x}^2$ and that $\mathbb{E}_{\text{repl}}[\bar{x}x_l | x, r] = \bar{x}^2$. Using the definition of the initial weight of the new incoming agent i (see equation (4.17)) and of its initial measurement

z_i (independently drawn from a distribution of variance σ^2 and of mean $\mu = 0$), we also have that $\mathbb{E}_{\text{repl}}[(w_i(0)z_i)^2 \mid x, r] = \sigma^2 \left(\frac{n-1}{n}\right)^{2r}$ and that $\mathbb{E}_{\text{repl}}[x_l w_i(0)z_i \mid x, r] = 0$ (since $\mathbb{E}[z_i] = \mu = 0$). Using these results, we have that

$$\begin{aligned}\mathbb{E}_{\text{repl}}[(\bar{x}^2)^+ \mid x, r] &= \left(1 - \frac{2}{n}\right)\bar{x}^2 + \frac{1}{n^2}\bar{x}^2 + \frac{\sigma^2}{n^2}\left(\frac{n-1}{n}\right)^{2r}, \\ \mathbb{E}_{\text{repl}}[\bar{x}^{2+} \mid x, r] &= \left(1 - \frac{1}{n}\right)\bar{x}^2 + \frac{\sigma^2}{n}\left(\frac{n-1}{n}\right)^{2r}\end{aligned}$$

It remains only to take the expectation $\mathbb{E}_\epsilon$ on both expressions to deduce a recurrence system that describes the update of the descriptor after a communication.

$$\mathbb{E}X_k = A_r \mathbb{E}X_{k-1} + b_r \mathbb{E}_\epsilon \left[\left(\frac{n-1}{n} \right)^{2r} \right] \quad \text{with} \quad A_r = \begin{bmatrix} 1 - \frac{2}{n} & \frac{1}{n^2} \\ 0 & 1 - \frac{1}{n} \end{bmatrix}, \quad b_r = \sigma^2 \begin{bmatrix} \frac{1}{n^2} \\ \frac{1}{n} \end{bmatrix}. \quad (4.21)$$

The term $\mathbb{E}_\epsilon \left[\left(\frac{n-1}{n} \right)^{2r} \right]$ corresponds to the expectation of the squared initial weight of the new agent i that enters the system during this replacement ($\mathbb{E}_\epsilon[(w_i(0))^2]$). The expectation is taken over the sequence of events that have occurred before this replacement that is the k^{th} event of the system, i.e. over the sequence of events from the beginning until event $k-1$ (denoted ϵ). In fact, the only thing that depends on the realization of this sequence of the events is the order of arrival r of the new agent. Counting the number of replacements that have taken place between the first event and until event $k-1$ and add 1 allows obtaining the value of r . Recalling that each event has a probability p_r to be a replacement, it can be computed as:

$$\mathbb{E}_\epsilon[(w_i(0))^2] = \sum_{l=1}^{k-1} \frac{(k-1)!}{l!(k-1-l)!} p_r^l (1-p_r)^{k-1-l} \left(\frac{n-1}{n} \right)^{2(l+1)} = \left(1 - \frac{1}{n}\right)^2 \left(\left(1 - \frac{2n-1}{n^2} p_r\right)^{k-1} - (1-p_r)^{k-1} \right).$$

Substituting this last result in the update of equation (4.21) gives the update described in the lemma. $\square$

Using these two lemmas, the resulting recurrence system that fully describes the evolution of $\mathbb{E}X$ with an event can be described easily.

Theorem 4.3 (Update of the descriptor vector $\mathbb{E}X$ with an event (for the SPS algorithm)). *Let an open multi-agent system satisfying the assumptions of the problem statement (see Section 1.1.1), subject to the symmetric push-sum algorithm and with $\mu = 0$. The k^{th} event that occurs in the system can be either a communication (with probability $1-p_r$) or a replacement (with probability p_r) (see the problem statement for the general definitions of the events). The value of the descriptor vector $\mathbb{E}X = [\mathbb{E}[\bar{x}^2], \mathbb{E}[\bar{x}^2]]^T$ is updated according to the following recurrence relations for the k^{th} event:*

$$\mathbb{E}X_k = A \mathbb{E}X_{k-1} + b \left(1 - \frac{1}{n}\right)^2 (c_1^{k-1} + c_2^{k-1})$$

$$\text{With} \quad A = \begin{bmatrix} 1 - \frac{2p_r}{n} & \frac{p_r}{n^2} \\ \frac{1-p_r}{n} & 1 - \frac{1}{n} \end{bmatrix}, \quad b = \sigma^2 \begin{bmatrix} \frac{p_r}{n^2} \\ \frac{p_r}{n} \end{bmatrix}, \quad c_1 = 1 - \frac{2n-1}{n^2} p_r, \quad \text{and} \quad c_2 = -(1-p_r).$$

Here also, the matrix A and the vector b are the same as for the gossip averaging algorithm (see Theorem 3.3). The update differs only with the decreasing factor that multiply the vector b .

Proof. The general evolution of $\mathbb{E}X$ with the k^{th} event (that has a probability p_r to be a replacement) can be decomposed as

$$\mathbb{E}X_k = (1-p_r)\mathbb{E}[X_k \mid k \text{ is a communication}] + p_r\mathbb{E}[X_k \mid k \text{ is a repl.}] \quad (4.22)$$

The updates to obtain $\mathbb{E}[X_k \mid k \text{ is a communication}]$ and $\mathbb{E}[X_k \mid k \text{ is a repl.}]$ are respectively described in Lemmas 4.1 and 4.2. Therefore, combining these updates with equation (4.22) gives the update described in the theorem with $A = (1-p_r)A_c + p_r A_r$ and $b = p_r b_r$. $\square$

The update of the descriptor vector $\mathbb{E}X$ given in Theorem 4.3 allows showing that both descriptors tend to 0 when the number of events k goes to infinity. Indeed, the value of the descriptor vector after any event k can be described from the initial one:

$$\mathbb{E}[X_k] = A^k \mathbb{E}[X_0] + \sum_{i=0}^{k-1} A^i b \left(1 - \frac{1}{n}\right)^2 (c_1^{k-i-1} + c_2^{k-i-1}).$$

If the initial vector $\mathbb{E}[X_0]$ is finite, the vector $\mathbb{E}[X_k]$ tends to 0 (when k tends to ∞) if the spectral radius of A (denoted $\rho(A)$), $|c_1|$ and $|c_2|$ are all (strictly) lower than 1. The spectral radius of matrix A is similar than for the gossip algorithm (since the matrix is the same), and hence from Section 3.1.2, we have that $\rho(A) < 1$ (for $p_r > 0$).

In the update of Theorem 4.3, contrarily to the one of the gossip (Theorem 3.3), the second term, independent of any components of the descriptor vector $\mathbb{E}X$, is also decreasing with the events k . We have that $|c_1| < 1$ and $|c_2| < 1$ (for $p_r > 0$) and therefore both descriptors tend to 0 when k tends to infinity:

$$\lim_{k \rightarrow \infty} \mathbb{E}[X_k] = [0, 0]^T \quad \text{for } p_r > 0.$$

The rate of convergence of $\mathbb{E}[X_k]$ to 0 is limited by the larger value between $|c_1|$, $|c_2|$ and $\rho(A)$. As a reminder from Section 3.1.2, the spectral radius of matrix A is given by

$$\rho(A) = \lambda_1 = \frac{2n - 2p_r - 1 + \sqrt{\Delta}}{2n}, \quad \text{with } \Delta = (1 - 2p_r)^2 + \frac{4p_r(1 - p_r)}{n}. \quad (4.23)$$

Since $p_r \in (0, 1]^1$, and if $n \geq 2$, it is larger than $|c_2| = (1 - p_r)$ (obviously) and larger than $|c_1| = 1 - \frac{2n-1}{n^2} p_r$ (not obviously but verified numerically).

Therefore, in an open multi-agent system (with $p_r > 0$) subject to the symmetric push-sum algorithm, the descriptors of the x values ($\mathbb{E}[\bar{x}^2]$ and $\mathbb{E}[\bar{x}^2]$) tend to zero as fast as they converge to their steady state value (in general different from zero) with the gossip averaging algorithm. But the performance of the symmetric push-sum algorithm, that involves the estimate $y_i = \frac{x_i}{w_i}$ of each agent i in the system, does not tend to 0, as suggested on the simulations results (see for example figures of Section 4.3). That can be explained by the fact that the weights of the agents also tend to 0 since the initial weight of an incoming agent decreases at each replacement and tends to zero (see equation (4.17)). Analyzing how the weights of the agents decrease (in expectation) can help to give an idea of the general evolution of the system. We can for example compute the expected average of the weight of the agents in the system denoted $\mathbb{E}[\bar{w}]$ (where $\bar{w}(t) = \frac{1}{n} \sum_{i \in \mathcal{N}(t)} w_i(t)$, but the time dependence is again omitted).

Evolution of $\mathbb{E}[\bar{w}]$ can be computed easily using the same reasoning as for other descriptor evolutions. Let $\mathbb{E}[\bar{w}_k]$ be the value of $\mathbb{E}[\bar{w}]$ after the k^{th} event that occurs in the system. The update of $\mathbb{E}[\bar{w}]$ for an event k that occurs in the system is given in the following theorem.

Theorem 4.4 (Update of $\mathbb{E}[\bar{w}]$ with an event (for the SPS algorithm)). *Let an open multi-agent system satisfying the assumptions of the problem statement (see Section 1.1.1), subject to the symmetric push-sum algorithm. The k^{th} event that occurs in the system can be either a communication (with probability $1 - p_r$) or a replacement (with probability p_r) (see the problem statement for the general definitions of the events). The value of the expected average weight of the agents in the system $\mathbb{E}[\bar{w}]$ is updated according to the following recurrence equation for the k^{th} event*

$$\mathbb{E}[\bar{w}_k] = (a_1 + a_2) \mathbb{E}[\bar{w}_{k-1}] + \frac{p_r}{n} \left(1 - \frac{1}{n}\right) (c_3^{k-1} - c_4^{k-1})$$

$$\text{With } a_1 = p_r \left(1 - \frac{1}{n}\right), \quad a_2 = (1 - p_r), \quad c_3 = 1 - \frac{p_r}{n} \quad \text{and} \quad c_4 = -(1 - p_r).$$

¹We have excluded the case $p_r = 0$ corresponding to a closed system and where the descriptor vector $\mathbb{E}[X_k]$ does not tend to 0.

Proof. The general evolution of $\mathbb{E}[\bar{w}]$ with the k^{th} event (that has a probability p_r to be a replacement) can also be decomposed as

$$\mathbb{E}[\bar{w}_k] = (1 - p_r)\mathbb{E}[\bar{w}_k \mid k \text{ is a communication}] + p_r\mathbb{E}[\bar{w}_k \mid k \text{ is a repl.}] \quad (4.24)$$

Therefore, the update of $\mathbb{E}[\bar{w}]$ when the k^{th} event is a communication or a replacement is analyzed independently.

A communication of the symmetric push-sum algorithm preserves the sum of weights in the system (see equation (4.18)). Therefore, if the k^{th} event is an communication, we have that

$$\mathbb{E}[\bar{w}_k \mid k \text{ is a communication}] = \mathbb{E}[\bar{w}_{k-1}]. \quad (4.25)$$

A replacement implies modifications due to the departure of an agent l with its own weight w_l and the arrival of a new agent i into the system with its initial weight $w_i(0) = \left(\frac{n-1}{n}\right)^r$ (r denotes the order of arrival of i). For the following developments, the $+$ in the exponent indicates the values after the replacement (and just before it without the $+$ exponent).

The expectation is decomposed in two: the expectation over all possible sequences of events ϵ until this replacement that allows knowing the number of replacement $r - 1$ that have already occurred in the system and the w values of all agents just before the replacement occurs; and the expectation over the information related to this replacement (denoted $\mathbb{E}_{\text{repl}}$): the index of the agents l selected to leave the system and the measurement z_i of the incoming agent. We have that $\mathbb{E}[\bar{w}^+] = \mathbb{E}_{\epsilon}[\mathbb{E}_{\text{repl}}[\bar{w}^+ \mid w, r]]$. A replacement of agent l by a new agent i impacts $\bar{w}$ as follows:

$$\bar{w} = \bar{w} + \frac{1}{n}(-w_l + w_i(0)) \quad \text{and} \quad \mathbb{E}_{\text{repl}}[\bar{w}^+ \mid w, r] = \left(1 - \frac{1}{n}\right)\bar{w} + \frac{1}{n}\left(\frac{n-1}{n}\right)^r, \quad (4.26)$$

Using that $\mathbb{E}_{\text{repl}}[\bar{w}_l \mid w, r] = \bar{w}$ and that $\mathbb{E}_{\text{repl}}[w_i(0) \mid w, r] = \left(\frac{n-1}{n}\right)^r$ (using definition of $w_i(0)$). It remains to compute the expectation over the sequence of events that occur before this replacement ϵ to obtain a recurrence equation that defines $\mathbb{E}[\bar{w}_k]$. It nevertheless requires to compute $\mathbb{E}_{\epsilon}\left[\left(\frac{n-1}{n}\right)^r\right]$ which corresponds to the expectation of the initial weight of the new agent i that enters the system during this replacement ($\mathbb{E}_{\epsilon}[w_i(0)]$) which can be done in a similar manner than in the proof of Lemma 4.2 that evaluate the expectation of the squared initial weight of a new agent i ($\mathbb{E}_{\epsilon}[(w_i(0))^2]$). The expectation is taken over the sequence of events that have occurred before this replacement that is the k^{th} event of the system, i.e. over the sequence of events from to beginning until event $k - 1$ (denoted ϵ). In fact, the only thing that depends on the realization of this sequence of the events is the order of arrival r of the new agent. Counting the number of replacements that have taken place between the first event and until event $k - 1$ and add 1 allows obtaining the value of r . Recalling that each event has a probability p_r to be a replacement, it can be computed as:

$$\mathbb{E}_{\epsilon}[w_i(0)] = \sum_{l=1}^{k-1} \frac{(k-1)!}{l!(k-1-l)!} p_r^l (1-p_r)^{k-1-l} \left(\frac{n-1}{n}\right)^{l+1} = \left(1 - \frac{1}{n}\right) \left(\left(1 - \frac{p_r}{n}\right)^{k-1} - (1-p_r)^{k-1} \right).$$

Using this result, the expectation $\mathbb{E}_{\epsilon}$ can be computed on the equation (4.26) and the update of $\mathbb{E}[\bar{w}]$ with a replacement can be described as follows:

$$\mathbb{E}[\bar{w}_k \mid k \text{ is a replacement}] = \left(1 - \frac{1}{n}\right)\mathbb{E}[\bar{w}_{k-1}] + \frac{1}{n}\left(1 - \frac{1}{n}\right)(c_3^{k-1} + c_4^{k-1}), \quad (4.27)$$

with $c_3 = 1 - \frac{p_r}{n}$ and $c_4 = -(1 - p_r)$.

Substituting equations (4.25) and (4.27) into the decomposition provided in equation (4.24) gives the result stated in the theorem. $\square$

This theorem allows to show that the expected average weight of the agents $\mathbb{E}[\bar{w}]$ tends to zero when the number of events k tends to infinity.

Indeed, for $p_r > 0$ and $n \geq 2$, we have that $|a_1| < |a_2| = |c_4| < |c_3| < 1$ and therefore we have that $\mathbb{E}[\bar{w}]$ tend well to 0 when k tends to ∞ . The decreasing rate of $\mathbb{E}[\bar{w}]$ is given by $|c_3| = 1 - \frac{p_r}{n}$ (for $p_r > 0$ and

$n \geq 2$).

One should notice that the decreasing rate of $(\mathbb{E}[\bar{w}])^2$, which is $(1 - \frac{p_r}{n})^2$, is very close to the decreasing rate of $\mathbb{E}[\bar{x}^2]$ or $\mathbb{E}[\bar{x}^2]$, especially for large n .

For large n , we have (from the definitions of $\rho(A)$ and Δ recalled in equation (4.23)) that

$$\sqrt{\Delta} \approx |1 - 2p_r| \quad \text{and therefore} \quad \rho(A) = \lambda_1 \approx 1 - \frac{2p_r}{n},$$

which have already been noticed in Chapter 3 concerning the analysis of the gossip averaging algorithm. Concerning the decreasing rate of $(\mathbb{E}[\bar{w}])^2$, we have that

$$c_3^2 = \left(1 - \frac{p_r}{n}\right)^2 = 1 - \frac{2p_r}{n} + \frac{p_r^2}{n^2} \quad (\text{for large } n) \approx 1 - \frac{2p_r}{n}.$$

For large system size n , the decreasing rates of $(\mathbb{E}[\bar{w}])^2$, $\mathbb{E}[\bar{x}^2]$ and $\mathbb{E}[\bar{x}^2]$, which are $|c_3|$ and $\rho(A)$, are all very close to $1 - \frac{2p_r}{n}$.

Hence, the ratio $\frac{\mathbb{E}[\bar{x}^2]}{(\mathbb{E}[\bar{w}])^2}$ may give something that approaches roughly the expected squared bias of the symmetric push-sum algorithm (for $\mu = 0$) which is $\mathbb{E}[\bar{y}^2]$. This is not really a valid approximation for different reasons:

- The approximation $\mathbb{E}[\bar{y}^2] \approx \frac{\mathbb{E}[\bar{x}^2]}{(\mathbb{E}[\bar{w}])^2}$ could only be valid when all agents have a weight close to the weight average $\bar{w}$, which is needed to have that

$$\mathbb{E}[\bar{y}^2] = \mathbb{E}\left[\left(\frac{1}{n} \sum_{i \in \mathcal{N}} \frac{x_i}{w_i}\right)^2\right] \approx \mathbb{E}\left[\frac{\bar{x}^2}{\bar{w}^2}\right].$$

But the weight average $\bar{w}$ should also be similar from one realization to another in order to ensure that $\mathbb{E}\left[\frac{\bar{x}^2}{\bar{w}^2}\right] \approx \frac{\mathbb{E}[\bar{x}^2]}{(\mathbb{E}[\bar{w}])^2}$. This approximation is therefore rarely valid.

- Moreover, the ratio $\mathbb{E}[\bar{x}^2]$ and $(\mathbb{E}[\bar{w}])^2$ have close decreasing rates (especially for large n). But they are never exactly equal and therefore asymptotically, when the number of events k goes to infinity, the ratio $\frac{\mathbb{E}[\bar{x}^2]}{(\mathbb{E}[\bar{w}])^2}$ goes either to zero (if $\mathbb{E}[\bar{x}^2]$ decreases slightly faster) or to infinity (if $(\mathbb{E}[\bar{w}])^2$ decreases slightly faster, which seems to be the case according to some numerical experiences). None of these cases are observed in reality for $\mathbb{E}[\bar{y}^2]$.

It would be interesting to analyze the variance of the weights of the agents in the system (and maybe other moments) which is needed to approximate the expectations where the weights are present in the denominator, such as for the estimates of the agents. Indeed, having no variance between the weights means that they are all the same and that the symmetric push-sum becomes equivalent to the gossip algorithm but when weights are different, the symmetric push-sum updates performs weighted average, as explained in the beginning of the chapter (see equation (4.4)).

In conclusion, this section has studied the evolution of a multi-agent system subject to the symmetric push-sum algorithm. The reasoning that uses classical descriptors evolutions, as for the gossip algorithm (see Chapter 3 or [HM17]), cannot be easily used to analyze the evolution of the performance of the algorithm in the system since it involves expectations of ratios (due to definition of the estimates of the agents $y_i = \frac{x_i}{w_i}$). Nevertheless, the agent dependent variables (x and w) can be analyzed separately. It results that expected average of the weights go to zero since the initial weights of new agents are and the x descriptors $\mathbb{E}[\bar{x}^2]$ and $\mathbb{E}[\bar{x}^2]$ also tends to zero. Future works and analyses are required in order to deduce theoretical description of the performance of the algorithm.

4.5 Implementable version of the algorithm

In order to have an implementable version of the algorithm, each incoming agent $n + r$ should know the size of the system n and its own order of arrival r to allow itself to set its initial weight according to equation (4.17). Equivalently, it is sufficient for a new agent to know the size of the system n and the total number of agents that have ever been in the system since the beginning before its arrival, denoted $N_{\text{tot}} = n + r - 1$. A solution is to let each agent in the system store an approximation of these values (n and N_{tot}). The incoming agent will then receive the approximation of a peer during its first communication. The new agents do not need to set their initial weights before their first communication. In fact, a new agent cannot *really* be considered part of the system until it has communicated with another agent in the system.

How do agents approximate n ?

Each agent initially in the system ($i \in \mathcal{N}(0)$) is supposed to know the size of the system n . It can be achieved beforehand through different processes based on a probabilistic counting [Luc+15] [Sha+12] [GY13] [KKL15]. Since we consider only replacements (see the problem statement in Section 1.1.1), the size of the system never changes and it can simply be transmitted to new agents that enter the system during its operation (when $t > 0$).

How do agents approximate N_{tot} ?

From the last paragraph, each agent initially in the system $i \in \mathcal{N}(0)$ is supposed to know the size of the system n and can therefore starts with a correct initial estimate of N_{tot} , denoted $N_{\text{tot},i}$: $N_{\text{tot},i} = n$ for all $i \in \mathcal{N}(0)$. The estimate $N_{\text{tot},i}$ of each agent i , as well as the true value of N_{tot} evolve with the system events and hence with time but this time dependence notation is omitted for simplicity. During their communications, the agents perform a maximum consensus on their $N_{\text{tot},i}$ estimate: they share their estimates and keep the highest one (as described in the state-of-the-art). When a new agent join the system, during its first communication with an agent $i \in \mathcal{N}(t)$, it receives its estimate $N_{\text{tot},i}$ and both will memorize $N_{\text{tot},i} + 1$ as new estimate of N_{tot} . This new value will be propagated to other agents thanks to the maximum consensus. It can be summarized as follows:

Max-consensus algorithm to estimate N_{tot} :

Initialization for all initially in the system $i \in \mathcal{N}(0)$:	$N_{\text{tot},i}(0) = n,$
During the first communication of a new agent j with $i \in \mathcal{N}(t)$	$N_{\text{tot},j}(0) = N_{\text{tot},i}^+ = N_{\text{tot},i} + 1,$
Classic interaction between a pair of agents $i, j \in \mathcal{N}(t)$:	$N_{\text{tot},i}^+ = N_{\text{tot},j}^+ = \max(N_{\text{tot},i}, N_{\text{tot},j}).$

This method can miss some new agents into the estimates of N_{tot} , especially if multiple new agents arrive in a short interval of time. Let us see a simple example. In a system where all the agents have an identical estimate of N_{tot} , let us consider a new agent j that first communicates with $i \in \mathcal{N}(t)$. The value of $N_{\text{tot},i} + 1$ will gradually spread to other agents. If another new agent m arrives and first communicates with $k \in \mathcal{N}(t)$ that has not yet converged to $N_{\text{tot},i} + 1$, it will add 1 to its estimate of N_{tot} for the new agent m but not for j . The maximum consensus is not able to detect this error that will be propagated into the system. In this simple example, two agents have arrived in the system but only one of them has been added to the different estimates of N_{tot} .

Figure 4.10 illustrates this problem by comparing the expected true value of N_{tot} with the expected average estimate of the agents $\bar{N}_{\text{tot},i} = \frac{1}{n} \sum_{i \in \mathcal{N}(t)} N_{\text{tot},i}(t)$. When the rate ratio is large (as in Figure 4.10a with $\rho = 100$ and $n = 20$), the expected average estimate of N_{tot} stays relatively close to the expected true value of N_{tot} . But when the rate ratio is smaller (as in Figure 4.10b with $\rho = 10$ and $n = 20$), the expected average estimate of N_{tot} is much lower than the expected true value of N_{tot} (more than two times lower). Figure 4.11 shows, in both situations, the impact of the estimation error on N_{tot} on the expected mean squared error achieved by the implementable version of the symmetric push-sum algorithm. For $\rho = 100$ (Figure 4.11a), the impact is very small and the implementable version of the

symmetric push-sum (in orange) performs almost as well as the theoretical version of the algorithm (with the true value of N_{tot}). For $\rho = 10$ (Figure 4.11b), the impact is more important and the implementable version of the symmetric push-sum (in orange) performs not as well as the theoretical version of the algorithm and is closer (in terms of performance) to the gossip averaging algorithm (in black). As it will be explained below, this problem about the estimation of N_{tot} can be corrected using bias correction mechanism, but another problem which cannot be corrected exists. Hopefully, this is quite an infrequent problem.

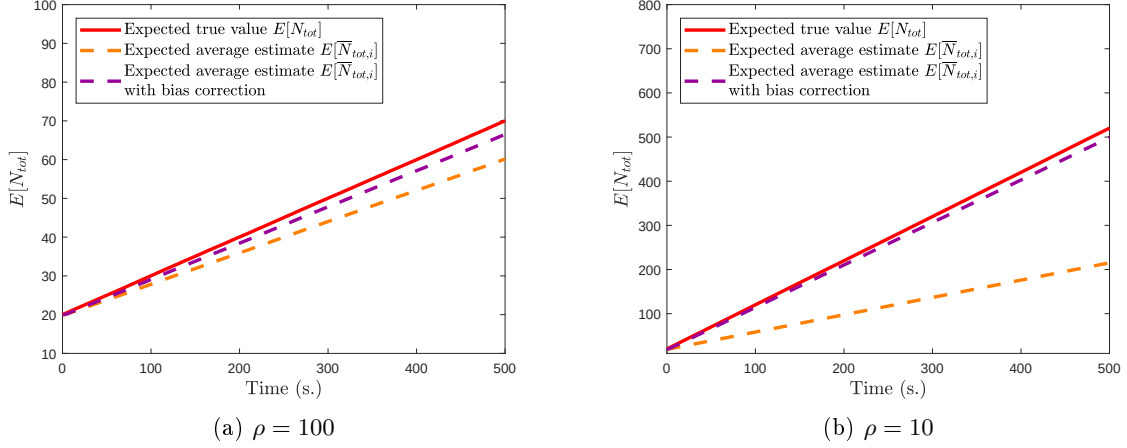


Figure 4.10: Time evolution of the expected total number of agents that have ever been in the system (N_{tot}), in comparison with the expected average estimate of N_{tot} with and without bias correction (using correction factor from (4.28)). The system size is fixed to $n = 20$. The plot is shown for two different rate ratios ($\rho = \frac{\lambda_c}{\lambda_r}$): $\rho = 100$ and $\rho = 10$, with constant $n\lambda_c = 10$. Expectations have been computed empirically with 5000 realization simulations.

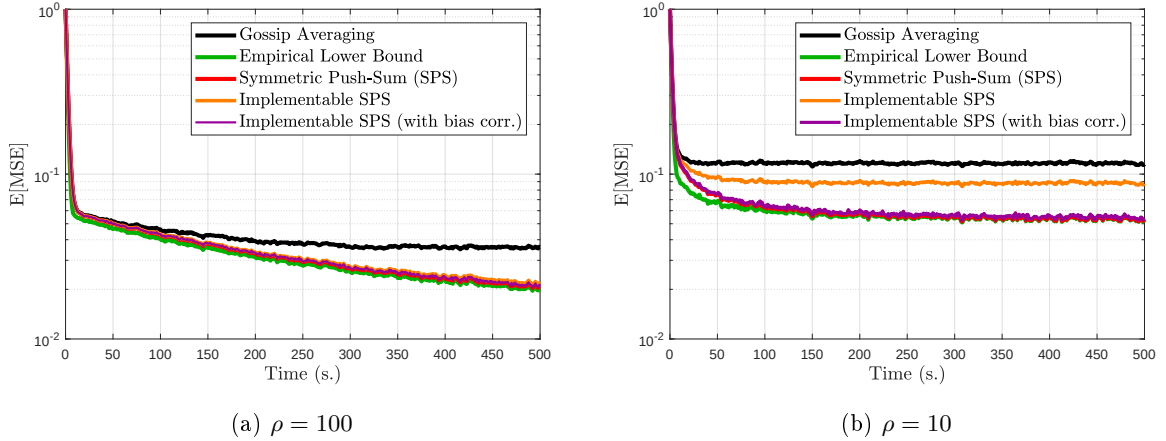


Figure 4.11: Time evolution of the expected mean squared error for the symmetric push-sum algorithm in its theoretical version and implementable version (with and without bias correction (using correction factor from (4.28))), in comparison with the gossip averaging algorithm and the empirical lower bound. The system size is fixed to $n = 20$. The plot is shown for two different rate ratios ($\rho = \frac{\lambda_c}{\lambda_r}$): $\rho = 100$ and $\rho = 10$, with constant $n\lambda_c = 10$. Expectations have been computed empirically with 5000 realization simulations.

This other problem corresponds to the situation where the first communication of an agent is with another new agent. None of them have any estimation of the system parameters and the only thing

they can do is a classical gossip interaction, waiting for a communication with someone that has information about the system. This situation rarely happens, except when the rate ratio ρ is (very) small (i.e. the replacements are frequent) and/or when the system is (very) small. For example, when $n = 20$ and $\rho = 100$, only (about) 0.03% of the communications occur between two new agents that have no information about the system. When ρ passes from 100 to 10, the percentage of these communications becomes about 0.54% and it becomes again a bit more (about 0.88%) when, in addition, the size of the system n passes from 20 to 10. These situations are therefore in general negligible.

However, as explained in Chapter 2, there is always a chance (even small) that all agent are replaced successively without having opportunity to transmit any information to new agents. It is called a *restart* (see Definition 2.3 for details). Usually a new agent waits for communicating with another *old* agent that has information about the system (meanwhile, it only performs gossip averaging interactions with other new agents it encounters). After a restart, none of the agents have any information and if they do not detect the restart the agents could perform gossip averaging interactions between them for eternity. Detecting the restart (with some probability) allows re-initializing the system as if it is the first start. Supposing an agent detects a restart and has access to the value of n^2 , it can initialize its own weight to one and its estimate of N_{tot} to n in order to start counting correctly the new agents that come in the system. The probability of having a restart in a system, studied in Chapter 2, increases with the probability of an event to be a replacement ($p_r = \frac{1}{1+\rho}$), with time and when the system size n is decreased. Indeed, the smaller the system is, the fewer replacements are needed to make a restart, and, the more replacements it occurs, the more chance there is to have a sequence of events that generate a restart.

A possible way for an agent (that know the system size n) to detect a restart would be to count the number of *unsuccessful* communications it has performed and to compare it with the system size n . An *unsuccessful* communication refers to a communication with another new agent that does not have any information about N_{tot} , which is needed to initialize their weight according to the symmetric push-sum algorithm. This has not been explored further and the following simply ignores the possibility of a restart which is rather unlikely in many systems.

Bias correction can be used to correct the estimates of N_{tot} of the agents and having a efficient implementable version of the symmetric push-sum algorithm. The estimates of N_{tot} of the agents miss some new agents and then are biased below the true value, as explained before and as shown on Figure 4.10. The sign of this bias³ remains identical (since estimates are all below the truth) and therefore a bias correction can be considered. It has been performed empirically by finding a factor q such that:

$$\mathbb{E}[N_{\text{tot}}(t)] - n \approx q \left(\mathbb{E} \left[\frac{1}{n} \sum_{i \in \mathcal{N}(t)} N_{\text{tot},i}(t) \right] - n \right),$$

where n has been subtracted from the expectation of N_{tot} (and its expected average estimate) because n is supposed to be perfectly known by each agent. It amounts to compare the expected number of replacements that have occurred in the system with the expected number of replacements that the agents have perceived. An identical q can probably not be found for any time t . Large time are considered to reach the steady state situation of the system, where the multiplicative gap between the true and the estimate number of replacements that have occurred becomes constant (in expectation). The expectations are taken over all the possible sequences of events and have been approximated using simulations of many different realizations. Once a such value of q have been found (in advance), it can be used by the new agents to correct the estimate of N_{tot} they receive when they arrive in the system and correctly set their initial weight. Concretely, when agent j enters the system and first communicates with i , instead of using the estimate $N_{\text{tot},i}$ as estimate of N_{tot} to initialize its own weight, it will use the following correction

$$q(N_{\text{tot},i} - n) + n.$$

²Other than by communicating with a former agent, which is impossible since a restart have occurred.

³As a reminder, the bias is the difference between the expected estimate and the true value. Here it is therefore defined as $\mathbb{E} \left[\frac{1}{n} \sum_{i \in \mathcal{N}(t)} N_{\text{tot},i}(t) \right] - \mathbb{E}[N_{\text{tot}}(t)]$.

Note that the max-consensus to estimate N_{tot} is not modified. The correction only takes place when the new agents use the value of N_{tot} in order to compute their order of arrival and initialize their weight.

Interpolating the values of q found for different system configurations (ρ and n), allows deriving an expression for q that seems to be working well:

$$q = 0.8p_r n + 1, \quad (4.28)$$

where p_r is the probability of an event to be a replacement $p_r = \frac{1}{1+\rho}$. The value of p_r , which is supposed to be fixed, is supposed to be known by the agents in the system. It could maybe also be observed by the agents in the system using another decentralized protocol. In-depth analysis of this max-consensus scheme in open multi-agent system could allow obtaining an exact expression of the expected bias on the estimates of N_{tot} and therefore on the correction factor q , but this empirical expression (4.28) is already satisfying.

Figure 4.10 shows the time evolution of $\mathbb{E}[N_{\text{tot}}(t)]$ and $\mathbb{E}[\bar{N}_{\text{tot},i}] = \mathbb{E}\left[\frac{1}{n} \sum_{i \in \mathcal{N}(t)} N_{\text{tot},i}(t)\right]$ with and without bias correction for two different rate ratios ($\rho = 100$ and $\rho = 10$). This figure first confirms that the factor q is constant for large time since both $\mathbb{E}[N_{\text{tot}}(t)]$ (in red) and $\bar{N}_{\text{tot},i}$ (in orange) increase linearly (starting at the same value n). Secondly, it shows that the correction factor proposed in equation (4.28) performs well since the expected corrected values of $\mathbb{E}[\bar{N}_{\text{tot},i}]$ almost fit the true values of $\mathbb{E}[N_{\text{tot}}(t)]$, even when the gap without correction is larger (as in Figure 4.10b).

The resulting mean squared error of this implementable version of the symmetric push-sum algorithm with bias correction is shown in purple in Figure 4.11 in both cases $\rho = 100$ and $\rho = 10$ (with fixed system size $n = 20$). As expected, when the estimates of N_{tot} are corrected (using correction factor from equation (4.28)), the implementable version of the push-sum algorithm perform as well as the theoretical version that has been developed in this chapter. When the rate ratio is large (such as in Figure 4.11a), meaning that the replacement probability p_r is small, the correction factor q is close to one and the bias correction makes almost no difference.

In this section, we have seen that it is possible to implement the symmetric push-sum algorithm by preserving its good performance, even if the agents do not know their rank of arrival in the system (which is required to initialize their weight). It requires the agents to know the exact size of the system n (at least the initial agents) and to maintain estimates of the total number of agents that have ever been in the system N_{tot} . To have an implementable version of the algorithm that is as efficient as the theoretical one in any situation, the agents (at least the initial ones) should also know the replacement probability p_r (or equivalently the rate ratio ρ) in order to correct the bias of the N_{tot} estimate when it is needed. This may still fail if a *restart* occurs in the system, which is less frequent, except in extreme system configurations ((very) small system with (very) frequent replacements). *Restart* detection should be studied in order to allow the system and the agents to restart correctly their estimates of N_{tot} when a restart occurs.

Conclusion about the symmetric push-sum algorithm

In this chapter, we have designed an algorithm to solve the external averaging problem stated in Section 1.1.1 with the idea of resolving the expected squared bias issue of the gossip averaging algorithm. As explained in Chapter 3, the gossip algorithm gives too much importance to the information brought by the new agents that enter the system and therefore it progressively forgets the information of the old agents, which is not desirable for computing the external average of the system. The symmetric push-sum algorithm is an algorithm that allows giving a weight to each agent. The initial weight of an agent represents the importance of its own measurement in the system. During the algorithm, the weight of each agent is updated to indicate the amount of information it carries. Initial weight of the agents (mainly the ones that arrive into the system during its operation) has been designed under the ideal assumption that convergence occurs in the system between two replacements, meaning that the replacements are extremely infrequent (see Assumption 4.1). As expected, we have obtained initial weight that decreases for each new agent. It decreases of $\frac{n-1}{n}$ from the previous arrival: $w_{n+r}(0) = (\frac{n-1}{n})^r$. Under the design convergence assumption, the algorithm is guaranteed to converge toward the external

average of the system between each replacement (see Proposition 4.2). These initial weights allow the symmetric push-sum algorithm to perform very well, even when the design assumption does not hold at all (i.e. when the replacements are frequent). To explain this, the problem of finding initial weights of arrival agents that minimize the expected mean squared error has been studied and it seems that the theoretical designed weights are nearly optimal in many different situations.

In terms of performance, based on simulations results, we have seen that the symmetric push-sum algorithm totally solves the expected squared bias issue of the gossip averaging algorithm since its expected squared bias always fits the one of the best ideal algorithm (see Definition 2.2 in Chapter 2) that provides a lower bound on the expected mean squared error. The expected variance of the algorithm tends to the one of the best ideal algorithm more or less quickly depending on the system configuration (it is slower when the replacements are less frequent or when the system is larger). The resulting expected mean squared error achieved by the algorithm always tends to the empirical lower bound provided by the best ideal algorithm. The only remaining improvement concerns the speed at which the expected variance (and then the expected mean squared error) tends toward the one of the best ideal algorithm. The characterization of the expected performance is unfortunately difficult to analyze theoretically since agents compute their estimate with a ratio of two variables $y_i = \frac{x_i}{w_i}$. Nevertheless, the expected evolution of both variables (x and w) has been derived (namely the evolution of descriptors $\mathbb{E}[\bar{x}^2]$, $\mathbb{E}[x^2]$ and $\mathbb{E}[\bar{w}]$). Further analysis (and/or simplification of the problem) are needed to be able to derive consistent bounds on the expected performance. Finally, this chapter has presented a way for the agents to estimate the total number of agents N_{tot} that have ever been in the system in order to allow new agents to know their rank of arrival and to initialize their weight correctly. It is based on a maximum consensus protocol. It makes the algorithm implementable, provided that the agents (at least the initial ones) know (or estimate) the size n of the system and its rate ratio ρ (or equivalently the probability of replacement p_r).

The next chapter presents a new idea, applied to the gossip averaging algorithm, in order to reduce rapidly the expected variance of the system. This idea is thereafter applied to this symmetric push-sum algorithm in Chapter 6.

Chapter 5

Gossip averaging with difference tracking

This chapter presents an algorithm that has been designed to reduce the expected variance due to the arrival of new agents in open multi-systems. As observed in Section 3.2, the gossip averaging algorithm does not perform very well for solving the external averaging problem (stated in Section 1.1.1) in part because of the expected variance it achieves, that represents the degree of disagreement between the agents and the impact of replacements on the system. Having a good expected variance from the beginning is also the main possible improvement for the symmetric push-sum algorithm presented in Chapter 4. This new algorithm relies on the idea of difference tracking applied the gossip averaging algorithm. The next chapter (Chapter 6) applies it to the symmetric push-sum algorithm. The idea of the difference tracking is that the new agents, in order not to disturb the system abruptly, pretend to have the value of an agent already in the system and retain the difference with their true value to add it progressively into the system and spread out the arrival of new information.

5.1 Algorithm description

The main design intention of this algorithm is to adapt the gossip averaging algorithm detailed in Chapter 3 in order to reduce the expected variance it achieves.

As a reminder, for the external averaging problem stated in Section 1.1.1, each agent i in the system $i \in \mathcal{N}(t)$ has an initial measurement z_i independently drawn from an identical distribution of mean μ and variance σ^2 and aims to estimate μ as well as it can. In the gossip algorithm, each agent i also holds a time varying value $x_i(t)$ to be exchanged with the others and that corresponds to its estimate $y_i(t)$ of μ . That is why for gossip-based algorithm (on the contrary of push-sum based algorithm), the x notation is used to denote the estimates of the agent.

The idea of the algorithm for reducing the disturbance brought by new agents when they enter the system is that a new agent i should pretend to have the same estimate as another agent j in the system, which is probably better than its initial estimate (that only contains its own measurement z_i).

$$x_i(0) = x_j.$$

In order not to lose its own information, but to spread out and add it progressively into the system, the new agent i also retains an additional variable, denoted δ_i , which correspond to the difference between its initial measurement z_i and its initial estimate that comes from another agent.

$$\delta_i(0) = z_i - x_i(0) = z_i - x_j.$$

This difference should be spread out and added progressively into the system in order to incorporate the new measurement of μ into the estimates of the agents. A similar additional variable has already been used in the literature under the name of *surplus* or *companion* variable. It has been used to develop, as

an alternative to the push-sum algorithm, other averaging consensus algorithms that works for a general digraph communication network [CI12] [CI14] [SR12] (recall that the gossip averaging algorithm only works for symmetric interactions).

As explained, the algorithm, for which an explicit definition is provided below, relies on the fact that a new agent has access to the estimate of another agent in the system to initialize its own variables (x and δ). That is why the following assumption is considered for algorithmic design and analysis.

Assumption 5.1 (Arrival with communication). When a new agent i enters into an open multi-agent system with set of agents $\mathcal{N}(t)$, it directly communicates with an agent $j \in \mathcal{N}(t)$ that was in the system before the arrival of i .

Under this assumption, the definition of replacement from the problem statement in Section 1.1.1 needs to be *modified*.

Definition 5.1 (Replacement with communication). Under Assumption 5.1, a replacement at time t denotes the succession of the departure of a randomly uniformly selected agent $l \in \mathcal{N}(t^-)$, the arrival of a new agent $i \notin \mathcal{N}(t^-)$, and a communication between the new agent i and another randomly uniformly selected agent $j \in \mathcal{N}(t^-) \setminus \{l\}$ that is still in the system.

All notations related to a replacement are modified with a tilde in order to show that it refers to this modified replacement definition. For example, the individual replacement rate, with this new definition of a replacement is denoted $\tilde{\lambda}_r$, the rate ratio (with this new replacement rate) is denoted $\tilde{\rho}$ and the probability of an event to be a (modified) replacement is denoted $\tilde{p}_r$.

This assumption is set up principally for designing and analysis purpose. Section 5.2 presents the theoretical evolution of the system, including the evolution of the expected performance achieved by the algorithm, under this assumption. The relaxation of this assumption is studied in Section 5.4 and the performance of the algorithm is analyzed through simulation results without this assumption in Section 5.5.

As we can imagine, the performance of any algorithm under Assumption 5.1 is improved. Indeed, imposing a new agent to communicate directly when it enters the system increase the total number of communications and avoid all the problems related with the *restarts* of the system (i.e. when all the agents are replaced without communicating their information to the new ones, see Definition 2.3) that are impossible under this assumption. It is a quite natural assumption that new agents communicate when they enter the system, since it is actually the reason of their entry: communicating with other agents in order to compute a better approximation of μ . Similarly, some may also think that an agent is not *really* part of the system before its first communication.

Under this assumption, the proved lower bounds (based on the nonzero restart probability) do not hold anymore. However, the best ideal algorithm (from Definition 2.2) remains valid under this assumption and still provides a lower bound on the expected mean squared error of any realistic algorithm under this assumption (see Proposition 2.1). The performance of this algorithm can be computed empirically and therefore it is still called the *empirical lower bound*.

Under Assumption 5.1, the gossip averaging with difference tracking can be described as follows.

Gossip averaging with difference tracking (under Assumption 5.1)

Initialization for any agent $i \in \mathcal{N}(0)$: $x_i(0) = z_i$ and $\delta_i(0) = 0$, setting parameter $\beta \in [0, 1]$.

Replacement (at time t) of agent $l \in \mathcal{N}(t^-)$ by agent i that first communicates with agent $j \in \mathcal{N}(t)$:

$$\begin{aligned} \mathcal{N}(t) &= \mathcal{N}(t^-) \setminus \{l\} && \text{Departure of agent } l, \\ x_i(0) &= x_j(t) \quad \text{for } j \in \mathcal{N}(t) && \text{Initial pretending } x \text{ value of agent } i, \end{aligned} \quad (5.1)$$

$$\delta_i(0) = z_i - x_i(0) \quad \text{Difference with its true measurement,} \quad (5.2)$$

$$\mathcal{N}(t^+) = \mathcal{N}(t) \cup \{i\} \quad \text{Arrival of agent } i,$$

The new agent i and the other (old) agent j perform a communication together (see below).

Communication between agents $i \in \mathcal{N}(t)$ and $j \in \mathcal{N}(t)$:

$$x_i^+ = x_j^+ = \frac{x_i + x_j}{2} + \beta \left(\frac{\delta_i + \delta_j}{2} \right) \quad (5.3)$$

$$\delta_i^+ = \delta_j^+ = \frac{\delta_i + \delta_j}{2} (1 - \beta) \quad (5.4)$$

Equation (5.3) averages the x values of both communicating agents and adds a part (β) of their respective difference variable δ . Equation (5.4) averages the remaining differences of both communicating agents in order that the difference variable δ of each agent spreads through the system during future interactions.

The parameter $\beta \in [0, 1]$ denotes the part of the difference variable δ_i , each agent i adds to its estimate x_i during each of its communications. It is called the *incorporation rate*. This parameter is supposed to be the same for each agent and to be constant with time. When $\beta = 1$, the entire difference of a new agent is directly added to its estimate during its first communication and the algorithm amounts to the classical gossip averaging algorithm (under the arrival with communication assumption (Assumption 5.1)).

By definition of the communication update (equations (5.3) and (5.4)), the sum of all x values with all δ values is preserved into the system. If $+$ exponent denotes the value after a communication, we have that

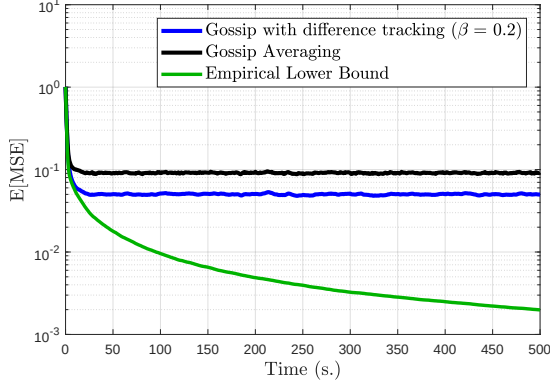
$$\sum_{i \in \mathcal{N}(t)} x_i^+ + \delta_i^+ = \sum_{i \in \mathcal{N}(t)} x_i + \delta_i.$$

Supposing that β is not too small (relatively to the system size and the rate ratio), the value δ_i of each agent i in the system will decrease to zero before the next replacement, meaning that all the information has been transmitted into the estimates x of the agents. In that situation, the δ values in the system only contain information from the last arriving agent that is spread among the agents and progressively incorporated into their estimates before the next replacement. This should allow the system to spread out the impact of an arrival of a new agent without modifying the amount of information it brings to the system. The expected variance should therefore be improved, without modifying the expected squared bias. Figure 5.1b shows that it is well the case. The system size of the simulation was $n = 10$ and the rate ratio $\tilde{\rho} = \frac{\lambda_c}{\lambda_r} = 10$ (using the *modified* definition of a replacement (see Definition 5.1) as expressed with the tilde notation). The value of the incorporation rate used is $\beta = 0.2$.

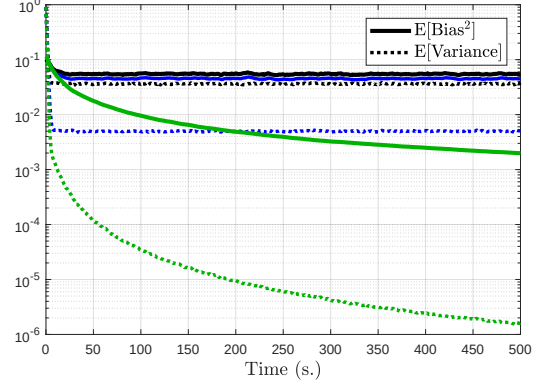
Comparing the evolution of the best ideal algorithm (in green) and the gossip averaging algorithm (in black) in Figure 5.1b with their evolution in Figure 3.2a from Chapter 3 allows observing that the assumption of arrival with communication reduces mainly the expected variance of the algorithms. Indeed, this assumption adds a communication for each replacement, and moreover, this communication involves the agent that was probably the less in agreement with the others since it has just arrived. The improvement is more important for the best ideal algorithm.

The difference tracking on the gossip averaging (with $\beta = 0.2$) allows reducing the expected variance almost by a factor 10 in this case, but it remains quite far from the expected variance of the empirical lower bound. It does not modify (or a very little) the expected squared bias. Hence, in that situation, the algorithm still presents the same issue as the gossip averaging algorithm concerning its expected squared bias (see Section 3.2 for details about this issue). Indeed, the agents that arrive into the system still have too much weight after their information has been completely incorporated into the system, in comparison with the older ones. The resulting expected mean squared error of the gossip averaging with difference tracking shown in Figure 5.1a is not improved that much in comparison to the basic gossip (it decreases by a factor 2). In that system configuration, the expected mean squared error is largely influenced by the value of the expected squared bias, which is not improved by the difference tracking with relatively large incorporation rate β (here $\beta = 0.2$). Improving the expected variance is good but it is not enough for this system configuration.

If β is smaller (relatively to the system size and the rate ratio), the δ values are incorporated slower into the estimates and they may be nonzero before a replacement. Therefore, the δ values may contain



(a) Expected mean squared error

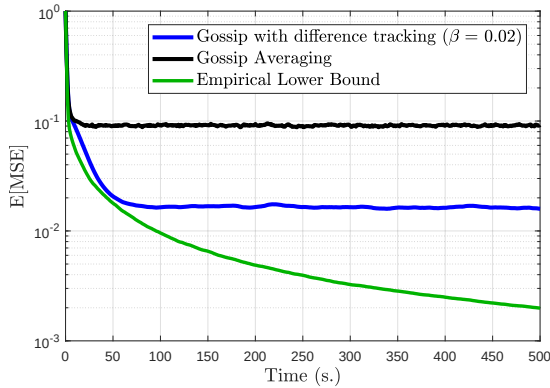


(b) Expected variance and squared bias

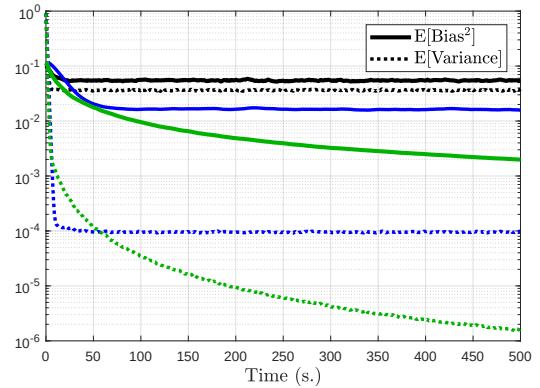
Figure 5.1: Time evolution of the expected performance of the gossip averaging with difference tracking (with $\beta = 0.2$) for an open multi-agent system with $n = 10$, $\bar{\rho} = \frac{\lambda_c}{\lambda_r} = 10$ (with $n\lambda_c = 10$). Performance of the classical gossip averaging algorithm and the best ideal algorithm (that provides a lower bound on the mean squared error) are also shown in comparison (also under Assumption 5.1). Expectations have been computed empirically with 5000 realization simulations.

information coming from different previous arrivals. If a leaving agent l has a nonzero δ_l value, it means that some information, concerning some previous arrivals will never be incorporated into the agents estimates. The information contained in the δ concerns the most recently arrived agents, the ones that had too much weights when all their information was incorporated. The loss of some of their information may therefore be beneficial for the expected squared bias. In the meantime, the initialization of a new agent in the system requires the new agent to pretend to have the estimate of another agent, which gives it temporarily more importance in the system and may also impact the expected square bias, especially if an important part of the difference δ is lost during a departure of an agent (which may occur for very small β).

Figure 5.2 is similar to Figure 5.1 but with a smaller incorporation rate $\beta = 0.02$. The resulting expected variance decreases drastically (even below the one of the best ideal algorithm at the beginning). The combined effects of the algorithm on the expected squared bias due to the departure of agents with nonzero δ values makes it difficult to interpret.



(a) Expected mean squared error



(b) Expected variance and squared bias

Figure 5.2: Time evolution of the expected performance of the gossip averaging with difference tracking (with $\beta = 0.02$) for an open multi-agent system with $n = 10$, $\bar{\rho} = \frac{\lambda_c}{\lambda_r} = 10$ (with $n\lambda_c = 10$). Performance of the classical gossip averaging algorithm and the best ideal algorithm (that provides a lower bound on the mean squared error) are also shown in comparison (also under Assumption 5.1). Expectations have been computed empirically with 5000 realization simulations.

For the gossip with difference tracking (in blue), we can observe that the expected squared bias first increases a bit (probably due to the fact that the estimates of some agents are duplicated by new agents to not disturb the system) before stabilizing below the one of the classical gossip averaging algorithm. It results in a significant improvement of the classical gossip averaging algorithm performance (almost by a factor 10).

Just for giving an idea of what happens when the incorporation rate β is very small, let us analyze Figure 5.3 which represents the same system as previous figures but for $\beta = 0.002$. The information of the new agents is incorporated extremely slowly into the system and a major part is probably lost due to replacement. It results in a drastic decrease of the expected variance, largely below the one of the best ideal algorithm for a while. The expected squared bias also decreases, further than for larger values of β but much more slowly. It also begins by increasing largely above the one of the gossip averaging algorithm, before it starts to decrease. It takes more than 500 seconds to reach a steady state. As it will be explained in the next section, this is due to the low importance attributed to the measurements brought by new agents.

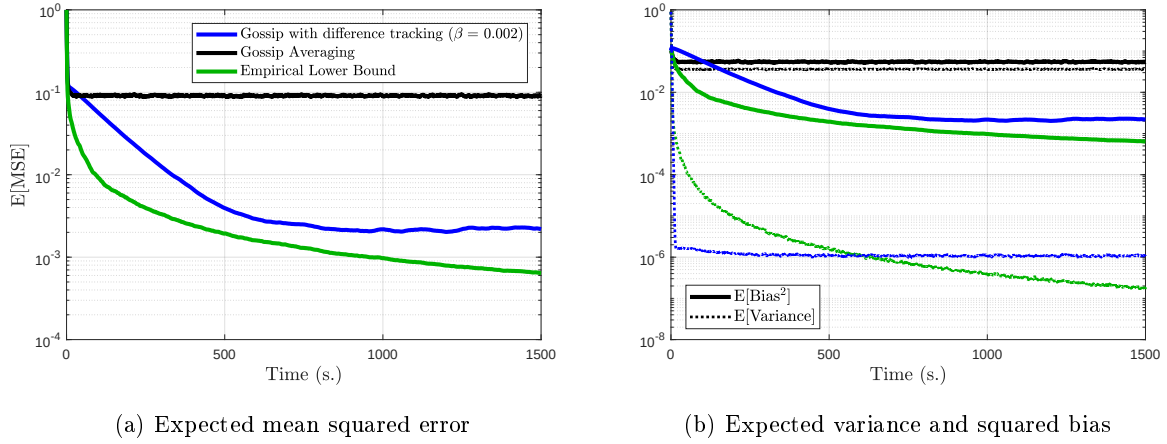


Figure 5.3: Time evolution of the expected performance of the gossip averaging with difference tracking (with $\beta = 0.002$) for an open multi-agent system with $n = 10$, $\tilde{\rho} = \frac{\lambda_c}{\lambda_r} = 10$ (with $n\lambda_c = 10$). Performance of the classical gossip averaging algorithm and the best ideal algorithm (that provides a lower bound on the mean squared error) are also shown in comparison (also under Assumption 5.1). Expectations have been computed empirically with 5000 realization simulations.

These three examples show that the value for the incorporation rate β has a significant impact on the performance of the algorithm. The tuning of this β parameter is therefore important. Depending on the objective we aim to achieve with this algorithm based on difference tracking idea (e.g. decreasing only the expected variance or achieving the best possible mean squared error after a given time), different strategies of β tuning should be used. In any cases, tuning the incorporation rate β in a reliable manner is easier with a theoretical description of an open multi-agent system subject to this gossip averaging with difference tracking. Next section aims to develop a such theoretical description based on the same reasoning as the theoretical description of the classical gossip averaging algorithm, presented in Section 3.1. It also allows giving guarantees on expected performance achieved by the algorithm (under Assumption 5.1)

5.2 Theoretical evolution of the system

Under Assumption 5.1, the performance achieved by the gossip averaging with difference tracking can be expressed using some descriptors of the distribution of the estimates x of the agents in the system. As a reminder, omitting time and realization dependence, we have

$$\begin{aligned}\mathbb{E}[\text{Bias}^2] &= \mathbb{E}[(\bar{x} - \mu)^2] = \mathbb{E}[\bar{x}^2] + \mu^2 - 2\mu\mathbb{E}[\bar{x}] \quad (\text{for } \mu = 0) \quad \mathbb{E}[\bar{x}^2], \\ \mathbb{E}[\text{Var}] &= \mathbb{E}\left[\frac{1}{n} \sum_{i \in \mathcal{N}} (x_i - \bar{x})^2\right] = \mathbb{E}[\bar{x}^2] - \mathbb{E}[\bar{x}]^2, \\ \mathbb{E}[\text{MSE}] &= \mathbb{E}[\text{Bias}^2] + \mathbb{E}[\text{Var}] \quad (\text{for } \mu = 0) \quad \mathbb{E}[\bar{x}^2],\end{aligned}$$

$$\text{where} \quad \bar{x} = \frac{1}{n} \sum_{i \in \mathcal{N}} x_i \quad \text{and} \quad \overline{x^2} = \frac{1}{n} \sum_{i \in \mathcal{N}} x_i^2.$$

Without loss of generality, we suppose that $\mu = 0$ and hence it is sufficient to describe the evolution of $\mathbb{E}[\bar{x}^2]$ and $\mathbb{E}[\bar{x}]$ to obtain a description of the expected mean squared error and its decomposition into expected variance and squared bias. Expressing the evolution of both descriptors $\mathbb{E}[\bar{x}^2]$ and $\mathbb{E}[\bar{x}]$ with an event of the system requires tracking the evolution of 6 different descriptors, involving both variables x_i and δ_i of each agent:

$$\mathbb{E}[X] = \left[\mathbb{E}[\bar{x}^2], \quad \mathbb{E}[\bar{x}], \quad \mathbb{E}[\bar{\delta}^2], \quad \mathbb{E}[\bar{\delta}], \quad \mathbb{E}[\bar{\delta}\bar{x}], \quad \mathbb{E}[\bar{\delta}x] \right]^T \quad (5.5)$$

$$\text{where} \quad \bar{\delta} = \frac{1}{n} \sum_{i \in \mathcal{N}} \delta_i, \quad \bar{\delta}^2 = \frac{1}{n} \sum_{i \in \mathcal{N}} \delta_i^2 \quad \text{and} \quad \bar{\delta x} = \frac{1}{n} \sum_{i \in \mathcal{N}} \delta_i x_i.$$

5.2.1 Descriptor evolution system

The evolution of the system subject to the gossip averaging with difference tracking can be described using a linear system of recurrence equations of the form

$$\mathbb{E}X_k = A\mathbb{E}X_{k-1} + b.$$

It describes the effect of one event k in the system on the descriptor vector $\mathbb{E}X_{k-1}$, which denotes the value of the descriptors after event $k-1$ in the system. Once again, to deduce an expression of matrix A and vector b , we consider the two possible types of event in the system separately: a communication (with probability $1 - \tilde{p}_r$) or a replacement (with probability $\tilde{p}_r$ under Assumption 5.1).

When the event is a communication, the following lemma holds.

Lemma 5.1 (Update of the descriptor vector $\mathbb{E}X$ with a communication (for the gossip algorithm with difference tracking)). *Let k be a communication event in an open multi-agent system of size n and with $\mu = 0$ subject to the gossip averaging algorithm with difference tracking (see the problem statement in Section 1.1.1 for a general definition of a communication event). For a communication in this algorithm, the update is defined by equations (5.3) and (5.4) and the value of the descriptor vector $\mathbb{E}X$ (defined by equation (5.5)) is updated according to*

$$\mathbb{E}X_k = A_c \mathbb{E}X_{k-1}, \quad \text{with} \quad A_c = \begin{bmatrix} 1 & 0 & \frac{2\beta^2}{n^2} & \frac{2\beta^2}{n^2} & \frac{4\beta}{n} & 0 \\ \frac{1}{n} & 1 - \frac{1}{n} & \frac{\beta^2}{n^2} & \frac{\beta^2}{n^2} & \frac{2\beta}{n} & \frac{2\beta}{n} \\ 0 & 0 & 1 + \frac{2\beta^2}{n^2} - \frac{4\beta}{n} & \frac{2\beta^2}{n^2} & 0 & 0 \\ 0 & 0 & \frac{(1-\beta)^2}{n} & 1 - \left(\frac{2-(1-\beta)^2}{n} \right) & 0 & 0 \\ 0 & 0 & \frac{2\beta}{n} - \frac{\beta^2}{n^2} & -\frac{2\beta^2}{n^2} & 1 - \frac{2\beta}{n} & 0 \\ 0 & 0 & \frac{(1-\beta)\beta}{n} & \frac{(1-\beta)\beta}{n} & \frac{1-\beta}{n} & 1 - \left(\frac{1+\beta}{n} \right) \end{bmatrix},$$

where β is the incorporation rate parameter of the gossip averaging with difference tracking.

Proof. The proof of this lemma is based on reasonings similar to that of the analysis of the classical averaging algorithm from [HM17] and is given in appendix of this chapter (see Appendix 5.A). $\square$

This results shows that, on the contrary of the classical gossip algorithm, a communication does not preserve the expected squared bias ($\mathbb{E}[\bar{x}^2]$ for $\mu = 0$). It is due to the addition of a part β of the difference variables δ of the communicating agents during each communication. The four first descriptors are always positive. This is not the case of the two last ones. Depending on the sign (and the amplitude) of $\mathbb{E}[\delta\bar{x}]$, the expected squared bias can increase or decrease with a communication event.

When the event is a replacement, the following lemma holds.

Lemma 5.2 (Update of the descriptor vector $\mathbb{E}X$ with a replacement (for the gossip algorithm with difference tracking)). *Let k be a modified replacement event (from Definition 5.1) in an open multi-agent system of size n and with $\mu = 0$ subject to the gossip averaging algorithm with difference tracking. The effect of a replacement can be decomposed into the effect of departure of one random agent from the system that changes its size from n to $n-1$ momentarily and the effect of arrival of a new agent into the system (initialized according to (5.1) and (5.2)) that directly communicates with one of the $n-1$ remaining agents and restores the system size to n . Therefore, when event k is a replacement, the descriptor vector $\mathbb{E}X$, defined by equation (5.5), is updated according to*

$$\mathbb{E}X_k = A_a A_d \mathbb{E}X_{k-1} + b_a,$$

$$\text{with } A_d = \begin{bmatrix} \frac{n^2-2n}{(n-1)^2} & \frac{1}{(n-1)^2} & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{n^2-2n}{(n-1)^2} & \frac{1}{(n-1)^2} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{n^2-2n}{(n-1)^2} & \frac{1}{(n-1)^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_a = \begin{bmatrix} \frac{(n-1)(-2\beta+n+1)}{n^2} & \frac{(1-\beta)^2}{n^2} & 0 & \frac{\beta^2}{n^2} & \frac{2\beta(n-1)}{n^2} & \frac{2(1-\beta)\beta}{n^2} \\ 0 & \frac{\frac{\beta^2}{2}-2\beta+n}{n} & 0 & \frac{\beta^2}{2n} & 0 & \frac{(2-\beta)\beta}{n} \\ 0 & \frac{(1-\beta)^2}{n^2} & \frac{(n-1)(n-2\beta-1)}{n^2} & \frac{\beta^2}{2n} & -\frac{2(1-\beta)(n-1)}{n^2} & \frac{2(1-\beta)\beta}{n^2} \\ 0 & \frac{(1-\beta)^2}{2n} & 0 & \frac{\frac{1}{2}(1-\beta)^2+n-2}{n} & 0 & -\frac{(1-\beta)^2}{n} \\ -\frac{(1-\beta)(n-1)}{n^2} & -\frac{(1-\beta)^2}{2n} & \frac{\beta(n-1)}{n^2} & -\frac{\beta^2}{n^2} & \frac{(n-1)(n-2\beta)}{n^2} & -\frac{2(1-\beta)\beta}{n^2} \\ 0 & -\frac{(1-\frac{\beta}{2})(1-\beta)}{n} & 0 & \frac{(1-\beta)\beta}{2n} & 0 & \frac{n+(1-\beta)^2-2}{n} \end{bmatrix}$$

$$b_a = \sigma^2 \begin{bmatrix} \frac{\beta^2}{n^2} & \frac{\beta^2}{2n} & \frac{(1-\beta)^2}{n^2} & \frac{(1-\beta)^2}{2n} & \frac{(1-\beta)\beta}{n^2} & \frac{(1-\beta)\beta}{2n} \end{bmatrix}^T,$$

where β is the incorporation rate parameter of the gossip averaging with difference tracking.

Proof. The proof of this lemma is based on reasonings similar to that of the analysis of the classical averaging algorithm from [HM17] and is given in appendix of this chapter (see Appendix 5.B). $\square$

These two lemmas are combined into the following theorem that describes the evolution of system descriptors with an event, for the gossip averaging algorithm with difference tracking.

Theorem 5.3 (Update of the descriptor vector $\mathbb{E}X$ with an event (for the gossip algorithm with difference tracking)). *Let an open multi-agent system satisfying the assumptions of the problem statement (see Section 1.1.1) and Assumption 5.1, subject to the gossip averaging algorithm with difference tracking and with $\mu = 0$. The k^{th} event that occurs in the system can be either a communication (with probability $1 - \tilde{p}_r$) or a modified replacement, including a communication due to Assumption 5.1 (with probability $\tilde{p}_r$) (see the problem statement in Section 1.1.1 for the general definitions of the events and Definition 5.1 for the modified replacement definition). The value of the descriptor vector $\mathbb{E}X$ (defined by equation (5.5)) is updated according to the following recurrence relations for the k^{th} event:*

$$\mathbb{E}X_k = A \mathbb{E}X_{k-1} + b, \quad \text{where} \quad A = (1 - \tilde{p}_r)A_c + \tilde{p}_r A_a A_d \quad \text{and} \quad b = \tilde{p}_r b_a, \quad (5.6)$$

with A_c from Lemma 5.1 and A_a , A_d and b_a from Lemma 5.2.

Proof. The general evolution of $\mathbb{E}X$ with the k^{th} event (that has a probability $\tilde{p}_r$ to be a replacement) can be decomposed as

$$\mathbb{E}X_k = (1 - \tilde{p}_r)\mathbb{E}[X_k \mid k \text{ is a communication}] + \tilde{p}_r\mathbb{E}[X_k \mid k \text{ is a repl.}] \quad (5.7)$$

The updates to obtain $\mathbb{E}[X_k \mid k \text{ is a communication}]$ and $\mathbb{E}[X_k \mid k \text{ is a repl.}]$ are respectively described in Lemmas 5.1 and 5.2. Therefore, combining these updates with equation (5.7) gives the update described in the theorem with $A = (1 - \tilde{p}_r)A_c + \tilde{p}_rA_aA_d$ and $b = \tilde{p}_rb_a$. $\square$

5.2.2 Validation

In order to validate the theoretical evolution of the system from Theorem 5.3, we compare it to the simulation results for some specific systems. For computing the theoretical evolution of the descriptors, the initial value of the descriptor vector is set to its initial expectation value which is

$$\mathbb{E}X_0 = \left[\mathbb{E}[\bar{x}^2(0)], \mathbb{E}[\bar{x}(0)], \mathbb{E}[\bar{\delta}^2(0)], \mathbb{E}[\bar{\delta}(0)], \mathbb{E}[\bar{\delta}(0)\bar{x}(0)], \mathbb{E}[\bar{\delta}\bar{x}(0)] \right]^T = \left[\frac{\sigma^2}{n}, \sigma^2, 0, 0, 0, 0 \right]^T. \quad (5.8)$$

Indeed, the initialization of the x and δ values of any agent initially in the system $i \in \mathcal{N}(0)$ is given by $x_i(0) = z_i$ and $\delta_i(0) = 0$. Remembering that the measurements z_i are independent and identically distributed (with mean $\mu = 0$ and variance σ^2) allows deducing equation (5.8).

As shown in Figure 5.4, the expected performance computed theoretically using Theorem 5.3 (in dashed red) fits very well the empirical expected performance computed through simulations (in blue). When averaging 1000 realizations, for the system used on the figure ($n = 10$ and $\tilde{\rho} = 100$), the relative difference between the curves is about 1% for the expected squared bias and the mean squared error and 5% for the expected variance (for which the empirical expectation requires more realizations to be smoother). Simulations validate the theoretical results from Theorem 5.3 which should allow tuning the incorporation rate parameter β of the algorithm more easily, as presented in Section 5.3.

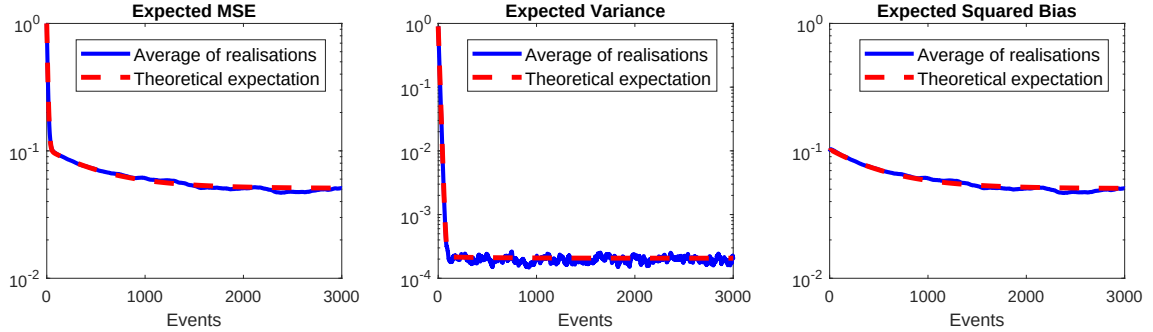


Figure 5.4: Comparison of the expected performance (MSE, squared bias and variance) for the gossip averaging algorithm with difference tracking computed theoretically, using Theorem 5.3 (and (5.8) for initialization) with the expected performance computed empirically by averaging 1000 realizations simulation, for a system of size $n = 10$ and rate ratio $\tilde{\rho} = 100$ (with constant $n\lambda_c = 10$).

5.2.3 Steady state expected performance and convergence rate

Theorem 5.3 allows computing the steady state expected performance achieved by the gossip averaging with difference tracking. Similarly than for the classical gossip averaging, the steady state expected performance can be obtained as the fixed point of the system of recurrence relations that describes the evolution of the descriptor vector with an event, which is given by equation (5.6) for the gossip algorithm with difference tracking. Concretely, we have that the fixed point of equation (5.6) can be computed as follows:

$$\mathbb{E}X_{eq} = A\mathbb{E}X_{eq} + b.$$

And from that steady state descriptor vector, steady state expected performance are simply deduced:

$$\mathbb{E}[\text{Bias}^2]_{eq} = \mathbb{E}[\bar{x}^2]_{eq}, \quad \mathbb{E}[\text{MSE}]_{eq} = \mathbb{E}[\bar{x}^2]_{eq} \quad \text{and} \quad \mathbb{E}[\text{Var}]_{eq} = \mathbb{E}[\bar{x}^2]_{eq} - \mathbb{E}[\bar{x}]_{eq}^2.$$

Unfortunately, the analytical expression of $\mathbb{E}X_{eq}$ is complicated. Each of its components is a quotient of multivariate polynomials (in β , n and $\tilde{p}_r$). Their values can however be computed easily for a given system. Some of them are shown and commented below, with their corresponding convergence rate (see Figures 5.5b and 5.6b).

As for the analysis of the classical gossip averaging algorithm in Section 3.1.2, the spectral radius of the matrix A gives the convergence rate of the descriptor vector to its fixed point and therefore the convergence rate of the expected performance to their steady state.

Let us introduce Δ_k which corresponds to the difference of vector $\mathbb{E}X_k$ to the fixed point $\mathbb{E}X_{eq}$, such that any descriptor vector $\mathbb{E}X_k$ can be expressed as $\mathbb{E}X_k = \mathbb{E}X_{eq} + \Delta_k$. The recurrence system (5.6) becomes:

$$\mathbb{E}X_{eq} + \Delta_k = A(\mathbb{E}X_{eq} + \Delta_{k-1}) + b \quad \Leftrightarrow \quad \Delta_k = A\Delta_{k-1} = A^k\Delta_0.$$

The rate of convergence of Δ_k to 0 corresponds to the rate of convergence of $\mathbb{E}X_k$ to $\mathbb{E}X_{eq}$ and therefore strongly depends on the spectral properties of the matrix A . Let λ_j (for $j = 1 \dots, 6$) be the six eigenvalues of the matrix A (indexed in absolute descending order) and v_j their corresponding eigenvectors (that are supposed to be independent). Any initial difference vector Δ_0 can be expressed in the eigenvector basis: $\Delta_0 = \sum_{j=1}^6 \alpha_j v_j$ with $\alpha_j \in \mathbb{R}$ for $j = 1, \dots, 6$. Therefore, the difference vector after k events Δ_k can be expressed in terms of the eigenvalues and eigenvectors of the matrix A .

$$\Delta_k = A^k\Delta_0 = A^k \left(\sum_{j=1}^6 \alpha_j v_j \right) = \sum_{j=1}^6 \alpha_j \lambda_j^k v_j. \quad (5.9)$$

It tends toward 0 only if all the eigenvalues λ_j ($j = 1 \dots, 6$) of the matrix A are strictly lower than one (in absolute value). Each term of the sum in (5.9) converges to zero with a rate λ_j . The largest eigenvalue (in absolute value) λ_1 , which is the spectral radius of the matrix, gives therefore the limiting convergence rate of Δ_k . Analyzing components of the dominant eigenvector v_1 , especially finding the one that are close to zero, allows determining the descriptors that converge more quickly to their steady state than others.

The analytical expression of the eigenvalues of matrix A are not explicitly given, since they correspond to the roots of a 6th degree polynomial function. Their corresponding eigenvectors have therefore no explicit expression either. Their values can however be computed easily for a given system.

Figures 5.5 and 5.6 show the evolution of the steady state expected performance of the algorithm and its corresponding convergence rate, given by the spectral radius of the matrix A , with the incorporation rate β , for different system sizes and rate ratios. As shown in Figures 5.5a and 5.6a, the spectral radius of the matrix A is lower than 1 (for $\beta > 0$), meaning that the convergence of the descriptors to their steady state occurs, but it starts increasing to one for small values of β , which implies a slower convergence. For $\beta = 0$, the spectral radius of the matrix A is always equal to 1 and the convergence to a steady state never happens. This is shown in Figures 5.5a and 5.6a for different specific systems but it has been validated by Matlab for a general value of n and $\tilde{p}_r$.

For its part, the steady state expected mean squared error decreases when β decreases (see Figures 5.5b and 5.6b), mainly due to the decrease of the steady state expected variance which is almost linear for not-to-small β values (see Figures 5.5c and 5.6c). The steady state expected variance naturally tends to zero when β goes to zero. Indeed, having $\beta = 0$ means that all new agents ignore completely their own value and will never incorporate it into the system. The expected squared bias is almost constant for large β values and starts decreasing for small β values. Of course, this steady state is only reached after a long time since the convergence rate gets closer to 1 for smaller value of β . Surprisingly, the expected squared bias tends to zero too when β goes to zero. Of course, this is not achievable in practice since having $\beta = 0$ implies an unit spectral radius, meaning that the convergence never occurs.

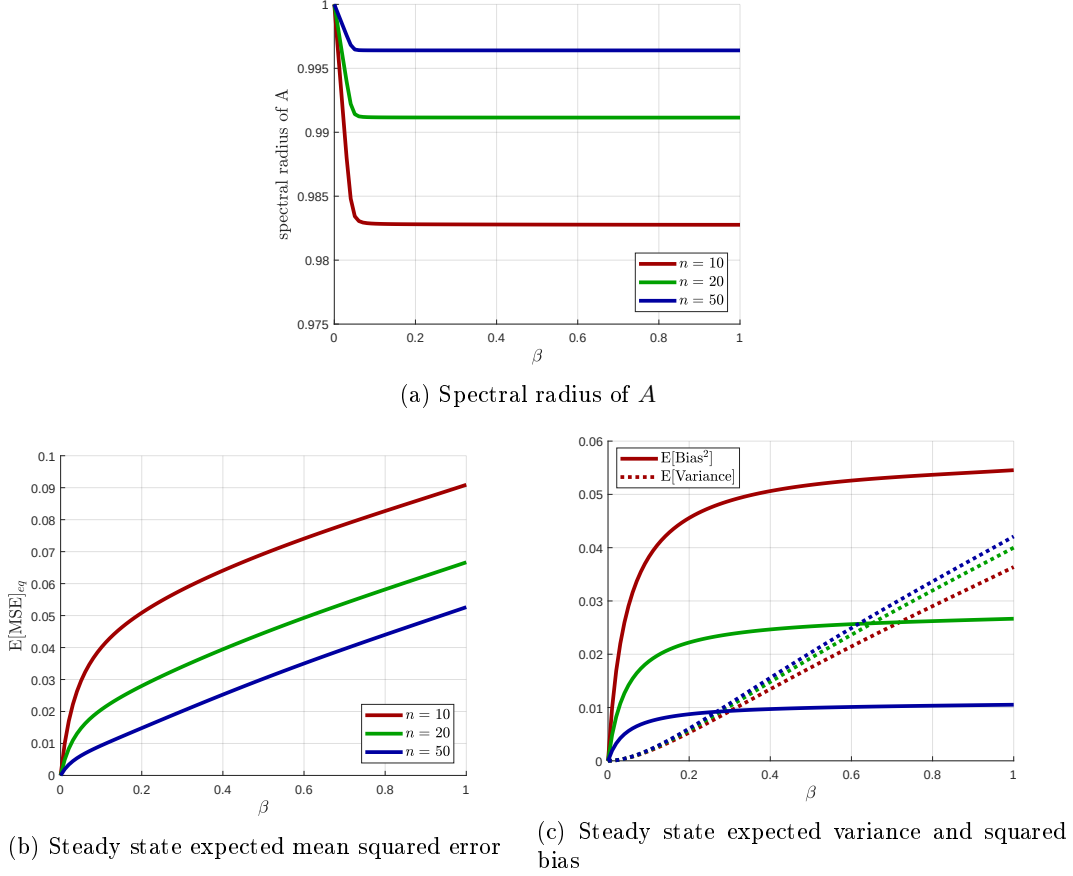


Figure 5.5: Evolution of the steady state expected mean squared error and its corresponding convergence rate (given by the spectral radius of A) with the incorporation rate β for different system sizes n and a fixed rate ratio $\tilde{\rho} = 10$.

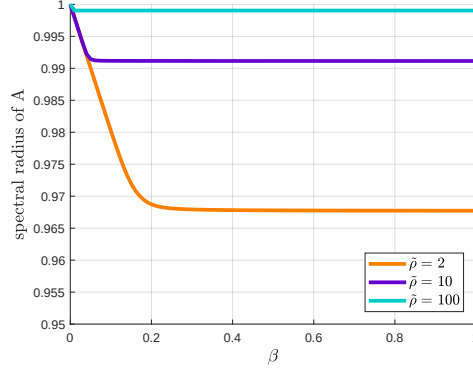
Analyzing the huge expression of the steady state expected squared bias, variance and mean squared error allows to see that they effectively tend to 0 when β goes to 0 since their form are the following:

$$\mathbb{E}[\text{MSE}]_{eq} = \sigma^2 \beta \left(\frac{q_1(\beta, n, \tilde{p}_r)}{q_d(\beta, n, \tilde{p}_r)} \right), \quad \mathbb{E}[\text{Bias}^2]_{eq} = \sigma^2 \beta \left(\frac{q_2(\beta, n, \tilde{p}_r)}{q_d(\beta, n, \tilde{p}_r)} \right) \quad \text{and} \quad \mathbb{E}[\text{Var}]_{eq} = \sigma^2 \beta \left(\frac{q_3(\beta, n, \tilde{p}_r)}{q_d(\beta, n, \tilde{p}_r)} \right).$$

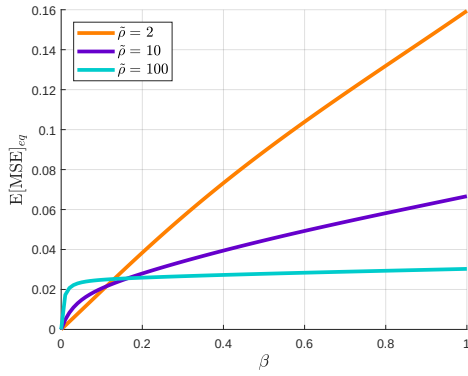
where q_1 , q_2 , q_3 and q_d are multivariate polynomial (depending on β , n and $\tilde{p}_r$) such that $q_1 = q_2 + q_3$ (since $\text{MSE} = \text{Bias}^2 + \text{Var}$), and q_d is different from 0 for $\beta = 0$. These expressions show well that having β tending to zero allows the steady state expected performance tending to zero too.

One could select a positive β sufficiently small such that the algorithm eventually achieves an expected mean squared error below $\epsilon > 0$ (for any positive value of ϵ) after a time that grows exponentially with $\frac{1}{\epsilon}$. This is possible only thanks to Assumption 5.1 under which no *restart* occurs. The algorithm can potentially perform poorly in a first time like this is the case in Figure 5.3b.

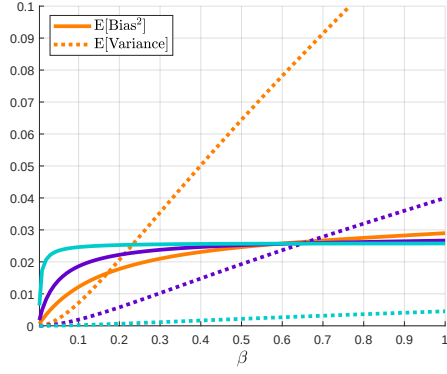
An intuitive way of explaining this is to consider the value of β as a kind of weight that is given to the measurements of μ of the new agents in the system. This weight is effective only after a delay that corresponds to the time needed for all difference variables that contain a part of the measurement either to be incorporated, either to leave the system. It can continue to increase after this delay due to the pretending values of the future new agents. For large β , the measurements are totally incorporated into the estimates of the agents in the system, which is equivalent of having a weight of $\frac{1}{n}$ in the average of the estimates. For low β , depending on the rate ratio $\tilde{\rho}$ of the system, only a part of the measurements of the new agents is incorporated into the estimates of the agents in the system, meaning that they have a weight smaller than $\frac{1}{n}$ (in the average of the estimates).



(a) Spectral radius of A



(b) Steady state expected mean squared error



(c) Steady state expected variance and squared bias

Figure 5.6: Evolution of the steady state expected mean squared error and its corresponding convergence rate (given by the spectral radius of A) with the incorporation rate β for different rate ratios $\hat{\rho}$ and a fixed system size $n = 20$.

For a given rate ratio, we have that the weight of a new agent is about $\frac{1}{n}$ for the not-too-small values of β , meaning that most of the information of the new agents is transmitted to the estimates of the agents. This explains why the convergence rate is nearly constant for not-too-small values of β (see Figure 5.5a and 5.6a).

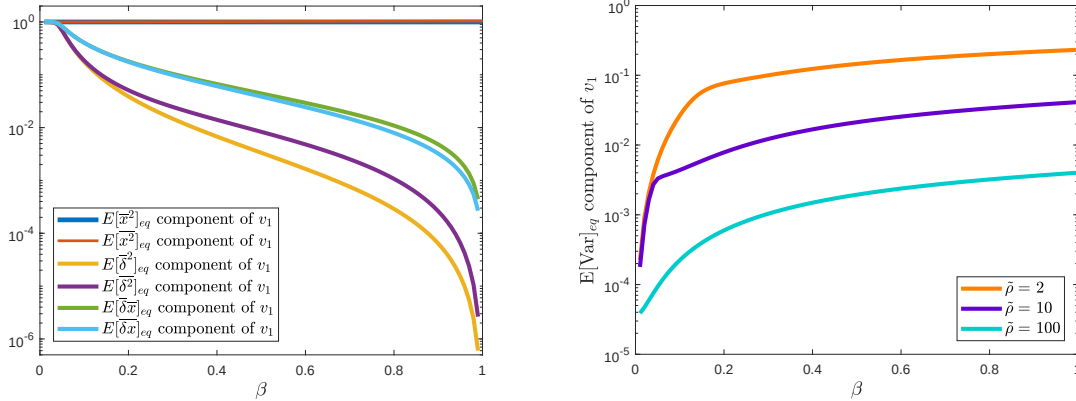
When β becomes small, the weight of the new agents decreases, and reaches zero when $\beta = 0$. Since we aim to compute the external average of the system (see the problem statement in Section 1.1.1), a fixed β , leading in steady state to give an identical weight to each incoming agents, is only suited for a given total number of agents that have ever been in the system (N_{tot}). The smaller β , the smaller the weight of new agents and the larger the suited N_{tot} . The expected squared bias decreases hence slowly until the correct value of N_{tot} is reached (i.e. when the correct number of replacements have occurred). When β (and hence the weight of a new agent) tends to zero, the suited N_{tot} tends to infinity. This suited N_{tot} leads to the averaging of an infinity of measures of μ , which leads well to an expected squared bias of zero. However, this suited N_{tot} require an infinite time to be reached.

This explanation misses some details about the delay and the evolution of the weight of an agent into the system but still allows giving a good intuitive explanation of the evolution of the expected squared bias with the value of the incorporation rate β .

In practice, it is therefore required to find a value of β that is a trade-off between the steady state performance it allows and the convergence rate to this steady state performance. Or even better, the value of the incorporation rate β could be continually adapted to the growing value of N_{tot} in the system. This problem is addressed in the next section that aims to tune the incorporation rate β for this algorithm.

In situations where the spectral radius of the matrix A is close to one (for β close to zero), the descriptors generally tends slowly to their steady state, but we observe that the expected variance always converges quite rapidly (as suggested in Figures 5.2b and 5.3b). This is due to the dominant eigenvector v_1 of the matrix A . Figure 5.7a shows the evolution of the different components of this dominant eigenvector¹ v_1 with the incorporation rate β for a given system ($n = 10$ and $\tilde{\rho} = 10$). Here are some comments about the values of this dominant eigenvector based on observations made on this figure.

- The four last components of v_1 are quite small and approach zero for large beta (close to one), which makes these components of the descriptor vector converging more rapidly than others to their steady state. Indeed, it cancels the contribution of the dominant eigenvalue for these components (see equation (5.9)). This is consistent with the fact that for $\beta = 1$, the algorithm directly incorporates the entire difference δ into the system and therefore these components (related to the δ values) directly reach a steady state of 0.
- The two first components of v_1 are very closed to each other, which means that their difference is close to zero. The two first components are related to $\mathbb{E}[\bar{x}^2]_{eq}$ and $\mathbb{E}[x^2]_{eq}$ and their difference corresponds to the expected variance: $\mathbb{E}[\text{Var}]_{eq} = \mathbb{E}[\bar{x}^2]_{eq} - \mathbb{E}[x^2]_{eq}$. The relative difference between these two components is shown in Figure 5.7b for different rate ratios. It shows that it is close to zero, especially for small values of β and it therefore means the expected variance is almost not impacted by the dominant eigenvalue and converges more rapidly to its steady state (in comparison to other components, such as the expected squared bias), especially for small β . This is consistent with observations of Figures 5.2b and 5.2b.



(a) Evolution of the v_1 components for a specific system: $n = 10$ and $\tilde{\rho} = 10$.

(b) Evolution of the variance component of v_1 (corresponding to the difference of the $\mathbb{E}[\bar{x}^2]_{eq}$ and $\mathbb{E}[x^2]_{eq}$ components) for a system of fixed size $n = 10$ and different rate ratios $\tilde{\rho}$.

Figure 5.7: Evolution of the different components of the dominant eigenvector v_1 with the incorporation rate β . The vector is normalized according to its first component (relative to $\mathbb{E}[\bar{x}^2]_{eq}$).

¹The evolution of the dominant eigenvector can have different regimes corresponding to different dominant eigenvalues. The transition between these regimes can be more or less smooth.

5.3 Tuning of the incorporation rate β

In light of the theoretical results presented in the previous section, it seems that tuning the β parameter requires to find a trade-off between the convergence speed (worsening with small values of β) and the quality of the final performance (improving with small values of β).

The first subsection aims to find a fixed value of β that correspond to a consistent trade-off. The second subsection analyzes the possibility of having a value of β decreasing with time. Indeed, this can benefit from the advantage of having a large β in a first time, with rapid convergence and hence a rapid decrease of the expected mean squared error; and a smaller β thereafter in order to continue decreasing, less rapidly, the expected mean squared error. Both tuning lead to improving the performance of the classical gossip averaging algorithm but as it will be explained, the decreasing β strategy allows solving the issue related to the expected squared bias, which cannot be done robustly with a fixed value of β .

5.3.1 Fixed incorporation rate β

This subsection presents a consistent trade-off for the value of the incorporation rate β . The idea is to achieve the best steady state performance, without compromising the performance during the transient state. In other words, it should avoid having a first period of time during which the algorithm performs poorly, like it is the case in Figure 5.3. In that figure the period of time during which the poor performance occurs is quite short but it can become much longer when the replacements are less frequent (i.e. when the rate ratio $\tilde{\rho}$ is larger) implying (for any algorithms) a slower decrease of the expected mean squared error.

A good way of avoiding poor performance at the beginning is to ensure a convergence rate of the expected performance toward its steady state that is close to the one of the gossip averaging algorithm, that corresponds to the case of $\beta = 1$ (in which all δ values are always equal to zero) and that never increases the expected squared bias. Figures 5.5a and 5.6a show that the spectral radius of matrix A (that gives the convergence rate) is nearly constant for all value of β greater than some threshold value that depends on the system configuration: $\beta \geq \beta_{thresh}(n, \tilde{\rho})$. The convergence rate starts increasing rapidly to 1 for β values below this threshold. Like suggested in Figure 5.5a, this threshold value of β seems independent of the system size n .

Choosing the value of β as equal to this threshold value allows decreasing the most possible the expected mean squared error (principally by decreasing the expected variance), while keeping a convergence rate very close to the best one (for $\beta = 1$), which corresponds to the one of the gossip averaging algorithm. For $\beta \geq \beta_{thresh}$, it seems that most of the information of new agents is transmitted to the estimates of new agents and therefore the expected squared bias is almost not modified.

Interpolating the β_{thresh} value for different rate ratios $\tilde{\rho}$ allows obtaining an approximate expression for β_{thresh} :

$$\beta_{thresh} \approx \frac{1}{\tilde{\rho} + 1} = \tilde{p}_r.$$

This expression can be validated with Figure 5.6a. It is consistent with the domain of definition of the incorporation rate $\beta \in [0, 1]$ since the replacement probability is also between 0 and 1. It is also consistent with our intuition: if $\tilde{\rho}$ is large, there are many communications between each replacement in the system and even with a small incorporation rate β , the entire δ values are in general transmitted into the estimates of the agents. On the contrary, when there are only a few communications between each replacement (i.e. when $\tilde{\rho}$ is small), the incorporation rate β should be larger in order not to lose too much of the δ values when an agent leaves the system. Using $\beta_{thresh} = \tilde{p}_r$ to set the value of β of course requires the agents (at least the initial ones) to know (or estimate) the value of $\tilde{p}_r$ (or equivalently $\tilde{\rho}$).

Figure 5.8 shows the time evolution of the simulated expected performance of the gossip averaging algorithm with difference tracking, with $\beta = \beta_{thresh} = \tilde{p}_r$, for the same system ($n = 10$ and $\tilde{\rho} = 10$) as in Figures 5.1, 5.2 and 5.3. As expected, it converges as quickly as the classical gossip algorithm and it mainly decreases its expected variance. Therefore, as desired, the expected squared bias (and then the

expected mean squared error) does not increase at the beginning (or a very little but it is not visible) but it does not improve much the expected squared bias of the gossip averaging.

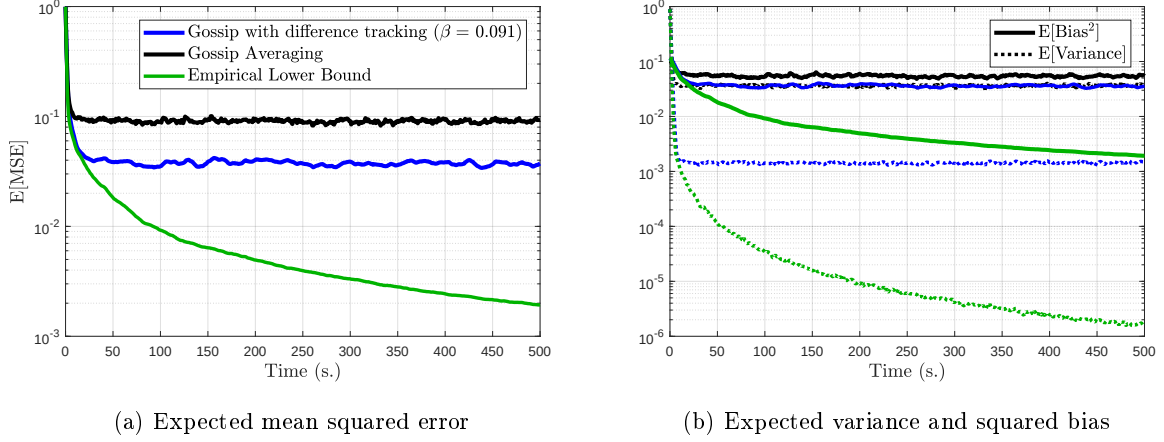


Figure 5.8: Time evolution of the expected performance of the gossip averaging with difference tracking (with $\beta = \beta_{\text{thresh}} = \frac{1}{\tilde{\rho}+1}$) for an open multi-agent system with $n = 10$, $\tilde{\rho} = \frac{\lambda_c}{\lambda_r} = 10$ (with $n\lambda_c = 10$). Performance of the classical gossip averaging algorithm and the best ideal algorithm (that provides a lower bound on the mean squared error) are also shown in comparison (also under Assumption 5.1). Expectations have been computed empirically with 1000 realization simulations.

Even if it is still far from the performance of the best ideal algorithm, this gossip algorithm with difference tracking, using $\beta = \tilde{\rho}_r$, is guaranteed (under Assumption 5.1) to beat the classical gossip averaging algorithm after a short amount of time since it has a better steady state performance and an almost identical convergence rate. A characterization of this *short amount of time* and this *almost identical* convergence rate should be studied in order to obtain rigorous theoretical guarantees.

The algorithm with a fixed β value really improves the expected variance of the classical gossip averaging algorithm but it does not allow solving the issue of the gossip algorithm related to the expected squared bias. This issue is related to the fact that the new agents entering the system have too much importance with the gossip algorithm and therefore the estimates of the agents progressively forgets the measurement of former agents, which is not desirable for computing the external average of the system. This issue is not solved with the difference tracking idea with fixed β since, in the steady state, any new agent has an identical fixed importance into the system, meaning that the same amount of their own measurement is integrated into the system. The constant importance of new agents cannot be adapted to the system for all time since the total number of agents that have ever been in the system keeps growing with time.

The idea of difference tracking with fixed incorporation rate β can still be applied to the symmetric push-sum algorithm that efficiently solves this issue related to the expected squared bias of the system. Having a tuning of β that does not impact too much the importance of new agents into the system and hence the expected squared bias (like the one presented in this subsection) is needed in order to avoid the two algorithms to collide. If both algorithms are combined efficiently, it could allow solving both expected squared bias and variance issues of the classical gossip and getting very close to the performance of the best ideal algorithm in most of the case. This idea is explored in Chapter 6.

Another way of solving this squared bias issue is to consider β decreasing with time, in order to decrease the part of the new measurements of μ that are integrated into the system. It is somewhat similar to the decreasing initial weight of the symmetric push-sum algorithm (see Chapter 4). This is discussed in the following subsection.

5.3.2 Decreasing incorporation rate β

As explained previously, a fixed incorporation rate β allows decreasing the expected variance in the system and sets a constant importance to new incoming agents which cannot be adapted to the system for any time, since the total number of agents that have ever been in the system N_{tot} keeps growing. At some time, the importance of new agents may be too low for the actual value of N_{tot} and after some time, when N_{tot} has increased, it becomes too large and it results in the progressive forgetting of the oldest agents. Having a value of β that is adapted with the actual value of N_{tot} in the system seems therefore better. Since the replacement events occurs according to a Poisson clock (see the problem statement in Section 1.1.1), the value of N_{tot} increases linearly with time (which have been confirmed in Figure 4.10) and therefore the value of β can evolve with time to be suited to the value of N_{tot} . Since N_{tot} increases with time, the importance of each new agent should decrease. Decreasing the value of β with time allows decreasing progressively the importance of new agents in the system. This is somewhat similar to the decreasing weight of new agents of the symmetric push-sum algorithm (see Chapter 4).

Having a decreasing incorporation rate β seems also a good idea given the theoretical results from Section 5.2. Indeed, the system can benefit from the advantage of having a large β in a first time, with rapid convergence and hence a rapid decrease of the expected mean squared error; and a smaller β thereafter in order to continue decreasing, less rapidly, the expected mean squared error.

Deducing a general theoretical expression of the decreasing $\beta(t)$ in order to achieve a good expected mean squared error at any time is not an easy task. Decreasing β values can be computed for a specific system using a succession of optimization problems. Let us suppose we know the value of the descriptor vector from equation (5.5) after event k . Then the value of β to use for the next event can be computed greedily by minimizing the expected mean squared error after the next event (supposing $\mu = 0$):

$$\underset{\beta}{\text{minimize}} \quad \mathbf{e}_2^T (A\mathbb{E}X_k + b)$$

with matrix A and vector b from Theorem 5.3 and where $\mathbf{e}_2$ denotes the vector with only a 1 in the second component. The second component corresponds to the component of $\mathbb{E}[\bar{x}^2]$ in the descriptor vector, which is the expected mean squared error when $\mu = 0$ (which we assume). The initial value of $\mathbb{E}X_0$ is given by equation (5.8).

This can be seen as a greedy method since after each event, the next β value is chosen in order to optimize the expected mean squared error after the next event. Alternatively, one can also minimize the expected mean squared error after a larger number of events, called the span s . This objective function can be computed using the recurrence relations from Theorem 5.3.

$$\underset{\beta}{\text{minimize}} \quad \mathbf{e}_2^T \left(A^s \mathbb{E}X_k + \sum_{j=0}^{s-1} A^j b \right) \quad (5.10)$$

Figure 5.9a shows the evolution of the β values when solving this optimization problem recursively, using equation (5.8) for the initialization, for different span values s and for the specific system $n = 10$ and $\tilde{\rho} = 10$ (which corresponds to a replacement probability of $\tilde{p}_r = \frac{1}{11}$). The resulting β values are smaller when the span s is larger but the decreasing rate is almost similar for the four considered spans. Figure 5.9b on the right shows the corresponding theoretical expected means squared error achieved by the obtained β decreasing values for each s . The figure shows that the greediest methods (with the smaller span s) does not perform the best on long term. And the solution of the problem with larger spans, such as $s = 200$, seems to achieve a better expected mean squared error on long term, but not for the first events where the expected mean squared error decreases a bit slower.

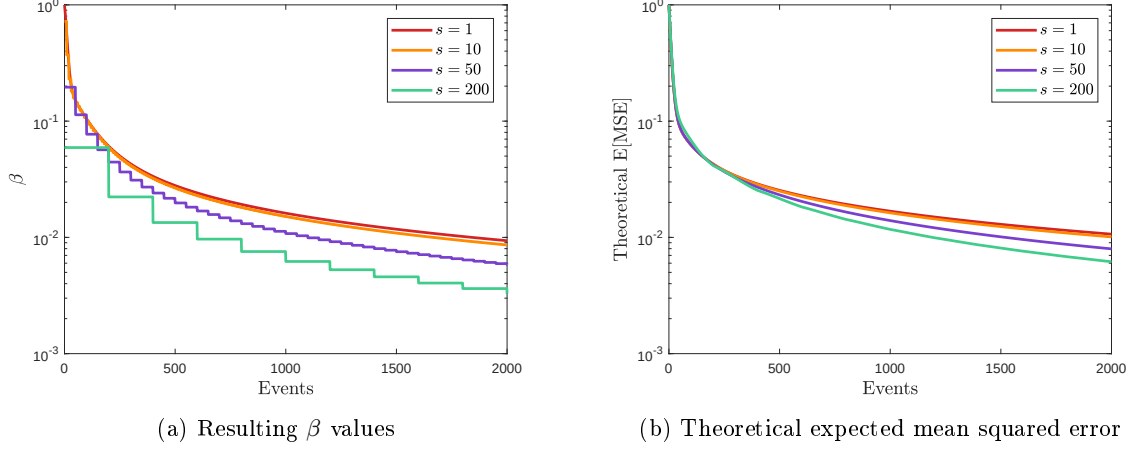


Figure 5.9: Evolution of the β value obtained recursively as solution of equation (5.10) (initialized using equation (5.8)) with different span values s for a specific system ($n = 10$, $\tilde{\rho} = 10$) (on the left) and evolution of the resulting theoretical expected mean squared error (on the right).

Implementing the result as it is, requires the agents to have access to the number of events that have already occurred in the system. But a similar decrease of β can be achieved if the agents have an intern time clock and know the global rate of the events in the system which is given by $\tilde{\lambda}^G = n(\lambda_c + \tilde{\lambda}_r)$. Indeed, it allows converting the time into an expected number of events since $\tilde{\lambda}^G$ corresponds to the expected number of events that occurs per seconds.

For the system in Figure 5.9 ($n = 10$ and $\tilde{\rho} = 10$), the β solutions for $s = 200$ have been fitted with the following function (that depends on the index of the current event k):

$$\beta(k) = \frac{0.06}{1 + \left(\frac{k}{121}\right)^{1.02}}.$$

A similar decrease of β with time can be expressed as:

$$\beta(t) = \frac{0.06}{1 + \left(\frac{t\tilde{\lambda}^G}{121}\right)^{1.02}}. \quad (5.11)$$

This decrease of β with time has been tested on simulations of the system and Figure 5.10 shows that the result are very good. The expected mean squared error fits the empirical lower bound almost all the time. It achieves a slightly different trade-off between the expected variance and squared bias since its expected variance is slightly lower than the one of the best ideal algorithm and its expected squared bias is slightly greater than the one of the best ideal algorithm (which is not really visible on the figure due to the scale). With a well-tuned decreasing β , the gossip averaging with difference tracking definitively and largely beats the classical averaging gossip algorithm. In this situation, the algorithm is as good as the symmetric push-sum algorithm (in red) presented in Chapter 4 (simulated under Assumption 5.1), in terms of the expected mean squared error. It achieves an expected variance that decreases very quickly below the one of the symmetric push-sum but its expected squared bias decreases slightly slower in the beginning and remains slightly above. This may benefit to one or the other of the algorithms depending on the system configuration that is considered.

This way of tuning the decreasing values of β requires a different tuning for each system configuration. Solving the tuning problem for many different system configurations (n and $\tilde{\rho}$) should allow interpolating the different decreases of β obtained in order to deduce an expression of the same form than (5.11) but with coefficients depending on n and $\tilde{\rho}$ (or $\tilde{\rho}_r$). One could expect a time-decrease of the β that is more rapid for smaller rate ratio $\tilde{\rho}$, since there are less communications between each replacement. This has not been done during this master thesis and may be a good idea to explore for the future. Therefore, the performance analysis of Section 5.5 only considers the algorithm with a fixed β , as tuned in the previous subsection.

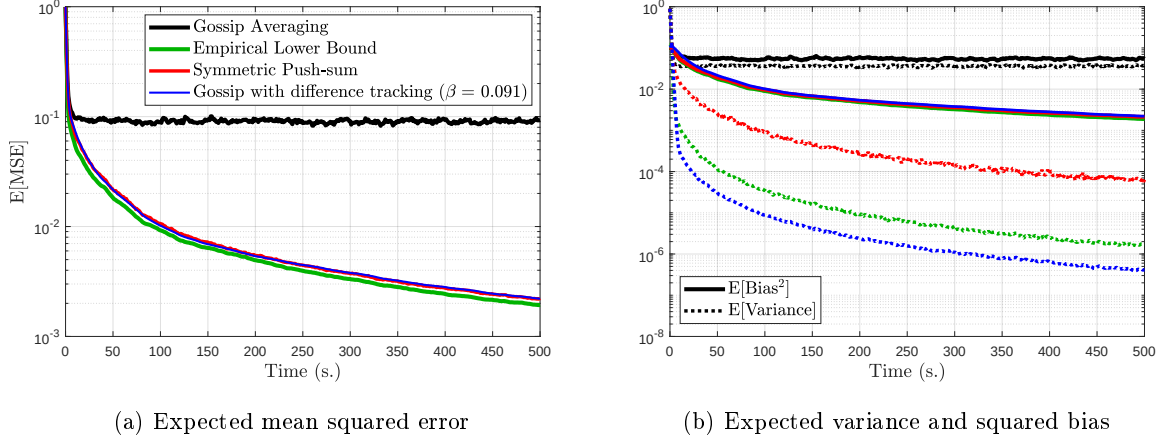


Figure 5.10: Time evolution of the expected performance of the gossip averaging with difference tracking (with decreasing β) for an open multi-agent system with $n = 10$, $\tilde{\rho} = \frac{\lambda_c}{\lambda_r} = 10$ (with $n\lambda_c = 10$). Performance of the classical gossip averaging algorithm, the symmetric push-sum algorithm and the best ideal algorithm (that provides a lower bound on the mean squared error) are also shown in comparison (also under Assumption 5.1). Expectations have been computed empirically with 1000 realization simulations.

5.4 Relaxation of Assumption 5.1 (arrival with communication)

Without Assumption 5.1, the new agents are not guaranteed to communicate with an agent in the system directly after their arrival. The initial pretending value x_i and the corresponding difference δ_i of a new agent i should therefore wait for the first communication of agent i in order to be correctly set up. The algorithm can be written as (omitting the time dependence notation for x and δ):

Gossip averaging with difference tracking (without Assumption 5.1)

Initialization for any agent $i \in \mathcal{N}(0)$: $x_i(0) = z_i$ and $\delta_i(0) = 0$, setting parameter $\beta \in [0, 1]$.

Replacement (at time t) of agent $l \in \mathcal{N}(t^-)$ by agent i

$$\begin{aligned} \mathcal{N}(t) &= \mathcal{N}(t^-) \setminus \{l\} && \text{Departure of agent } l, \\ x_i(0) &= z_i \\ \delta_i(0) &= 0 \\ \mathcal{N}(t^+) &= \mathcal{N}(t) \cup \{i\} && \text{Arrival of agent } i. \end{aligned}$$

Before the first communication of a new agent $i \in \mathcal{N}(t)$ with an (*old*) agent $j \in \mathcal{N}(t)$:

$$\begin{aligned} \delta_i &= x_i - x_j && \text{Difference with its true } x_i, \\ x_i &= x_j && \text{Initial pretending } x \text{ value of agent } i, \end{aligned}$$

The new agent i and the other agent j perform a communication together (see below).

Communication between agents $i \in \mathcal{N}(t)$ and $j \in \mathcal{N}(t)$:

$$\begin{aligned} x_i^+ &= x_j^+ = \frac{x_i + x_j}{2} + \beta \left(\frac{\delta_i + \delta_j}{2} \right) \\ \delta_i^+ &= \delta_j^+ = \frac{\delta_i + \delta_j}{2} (1 - \beta) \end{aligned}$$

It is very similar to the algorithm under Assumption 5.1 described at the beginning of the chapter, but the true initialization of the x_i and δ_i of a new agent i is not done before its first communication

with another agent of the system. It is kind of like the agent does not enter the system until its first communication.

But what should an agent do if its first communication occurs with another recent agent that has also not yet communicated? This problem is similar that the one encountered in Section 4.5, where agents maintain an estimate of the total number of agents that have ever been in the system (N_{tot}). There exists two principal ways of answering the latter question.

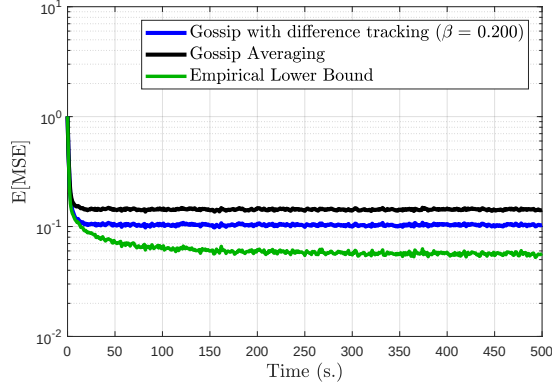
- Selecting one of the two communicating agents to pretend to have the x value of the other and continuing the algorithm normally.
- Communicating agents do not pretend anything about their x values and perform a classical communication that corresponds to average their x values (as for the gossip algorithm). These new agents wait for the first communication with an *old* agent in order to pretend to have its x value. An *old* agent is an agent that has already been fully initialized: either an agent from $\mathcal{N}(0)$, initially present in the system, or an agent that has already communicated with an *old* agent.

As explained in Section 4.5, this type of communication that occurs between two new agents is quite rare, except in small systems subject to frequent replacements. That is why the second solution is preferred in most of the cases. Indeed, the behavior of the first one is unknown and may impact badly both the variance and the squared bias of the system. But this first solution keeps working even when a *restart* occurs in the system (i.e. when all the agents are replaced, see Definition 2.3). The second solution is more suited when these communications are not too frequent (like it is mostly the case) but badly react to a *restart* of the system since the agents will simply performing gossip interaction, waiting for communication with an *old* agent, which does not exist anymore into the system. As suggested in Section 4.5, a restart detection protocol should be beneficial for this solution. For the following analyses, the second solution has been implemented (without any *restart* detection). This is the solution we refer to when we talk about the gossip averaging algorithm with difference tracking without Assumption 5.1. The issues related to the restart of the system, which is rare in most situations, is ignored for the following.

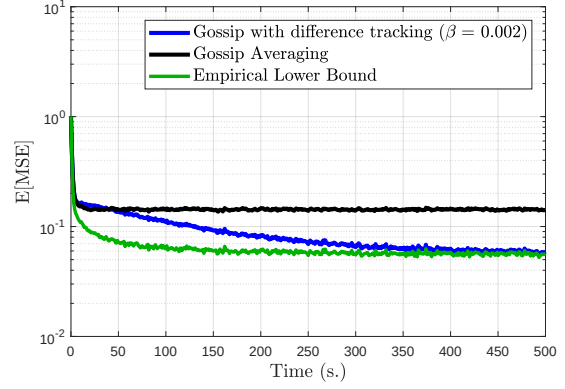
Without this assumption of arrival with communication, the impact of the incorporation rate β on the behavior of the algorithm remains quite similar. As shown in Figure 5.11a, too large value of β still implies small improvements on the expected mean squared error. On the contrary, too small value of β , like on Figure 5.11b, still implies that too little importance is attributed to new agents and consequently, the expected mean squared error decreases further but more slowly. The algorithm waits for the total number of agents that have ever been in the system (N_{tot}) to be adapted with the fixed importance the algorithm attributes to new agents. The suited value of N_{tot} (that can be (very) large for small β) may never be reached due to the chance of *restart* it exists in the system. Indeed, a restart imply recounting the total number of agents that have ever been in the system from zero. Despite this difference due to the little chance of *restart* in the system, the impact of the incorporation rate β remains quite similar without Assumption 5.1.

Therefore, the tuning of β with a fixed value (see Section 5.3) still works in a similar way. The exact value of β_{thresh} (which is the threshold above which the convergence rate is nearly constant) is probably not identical but approximating it with p_r seems still working, as suggested in Figure 5.12a. Indeed, in this figure, the gossip averaging with difference tracking (in blue) converge as quickly as the classical gossip (in black) but quite lower. Having a slightly incorrect threshold β_{thresh} is not dramatic since in reality, there is no true threshold and the transition is relatively smooth. The threshold $\beta_{\text{thresh}} = p_r$ is probably a bit too low. Since there are less communications, it requires a β slightly higher in order to incorporate all the information of the new agents in the estimates of the agents in the system. Therefore, having $\beta_{\text{thresh}} = p_r$ can therefore impact the expected squared bias and make it slightly worse at the beginning, especially when the rate ratio is low (which is not the case of the system simulated in Figure 5.12a).

Concerning the decreasing β tuning (see Section 5.3), it should also work well with the same decreasing as under Assumption 5.1, except when a *restart* occurs. It is therefore not recommended for systems where the *restarts* are relatively frequent ((very) small system with (very) low rate ratio). For the system of size $n = 10$ and rate ratio $\rho = 10$ for which the decreasing weight has been designed in Section



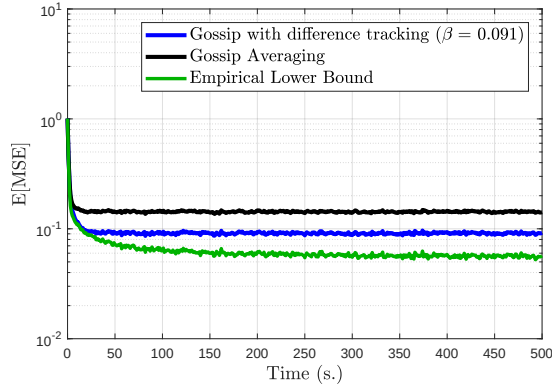
(a) $\beta = 0.2$



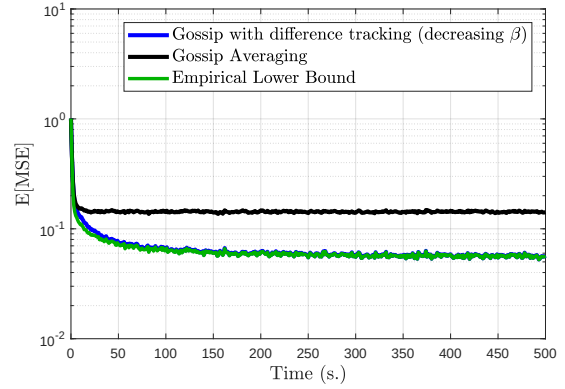
(b) $\beta = 0.002$

Figure 5.11: Impact of the incorporation rate β on the expected mean squared error of the gossip averaging with difference tracking, without Assumption 5.1, for a system of size $n = 10$ and with rate ratio $\tilde{\rho} = 10$ (with $n\lambda_c = 10$). Expectations have been computed empirically with 5000 realization simulations.

5.3 (under Assumption 5.1), Figure 5.12b shows that it performs very well and fit almost directly the empirical lower bound without the designing assumption.



(a) Fixed β value: $\beta = p_r = 0.91$



(b) Decreasing β value

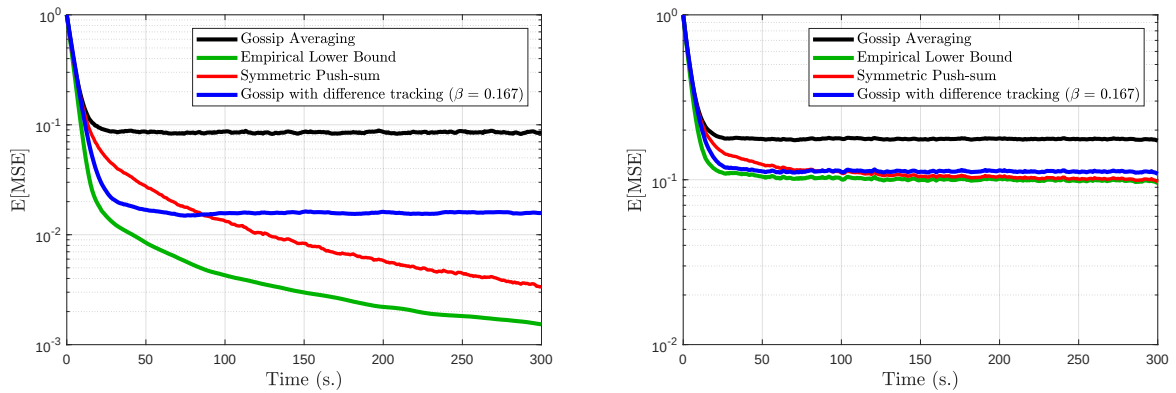
Figure 5.12: Impact of the tuning solutions of the incorporation rate β (from Section 5.3) on the expected mean squared error of the gossip averaging with difference tracking, without Assumption 5.1 for a system of size $n = 10$ and with rate ratio $\tilde{\rho} = 10$ (with $n\lambda_c = 10$). Expectations have been computed empirically with 5000 realization simulations.

5.5 Performance analysis

This section analyzes the performance of the gossip averaging algorithm with difference tracking, using a fixed incorporation rate $\beta = p_r$ (as tuned in Section 5.3). In order to have results comparable to previous chapters, most of the analysis focuses on the case where Assumption 5.1 is relaxed. Even if the theoretical description of the algorithm has been developed under this assumption, previous section has shown that the behavior of the algorithm and the impact of the incorporation rate β is quite similar in most of the cases. This section presents quickly the time evolution of the expected performance of the algorithm and shows the advantage of converging rapidly to a steady state situation (in expectation). Thereafter, the expected mean squared error of the algorithm after 500 seconds is compared to the other algorithms for a variety of different systems in order to have a good vision of the *nearly* final performance of the algorithm in comparison to others.

5.5.1 Time evolution of the expected performance

In reason of the tuning of a fixed β that has been done in Section 5.3, the expected performance of the algorithm converges almost as quickly as the gossip averaging algorithm (corresponding to $\beta = 1$) in any situation. Figure 5.13 illustrates this with and without the assumption of arrival with communication. That is advantageous for the algorithm to quickly decrease its expected mean squared error below the one of the classical gossip, even if it does not reach the empirical lower bound. The symmetric push-sum algorithm (from Chapter 4) that reaches the empirical lower bound in most cases, can sometimes converge slowly to it and can perform not directly in an optimal way. This is the case in Figure 5.13, and the gossip with difference tracking achieves a better expected mean squared error than the symmetric push-sum during the first 75 seconds. This observation is valid with or without the Assumption 5.1 (see respectively Figures 5.13a and 5.13b). This shows that both algorithms complement each other, even if there are still situations where the push-sum algorithm directly fits the empirical lower bound (see Chapter 4).



(a) Under Assumption 5.1 (arrival with communication)

(b) Without Assumption 5.1

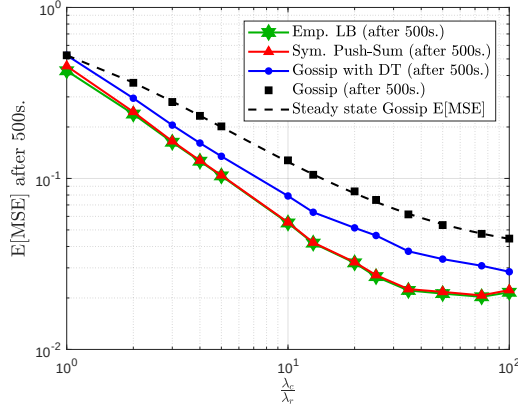
Figure 5.13: Time evolution of the expected mean squared error of the gossip averaging with difference tracking (using fixed $\beta = p_r = \tilde{p}_r$) with and without Assumption 5.1 for a system with $n = 50$ and $\rho = 5$ (or $\tilde{\rho} = 5$ for the figure on the left) with $n\lambda_c = 10$. Performance of the classical gossip averaging algorithm, the symmetric push-sum algorithm and the best ideal algorithm (that provides a lower bound on the mean squared error) are also shown in comparison. Expectations have been computed empirically with 1000 realization simulations.

5.5.2 Expected performance after 500 seconds

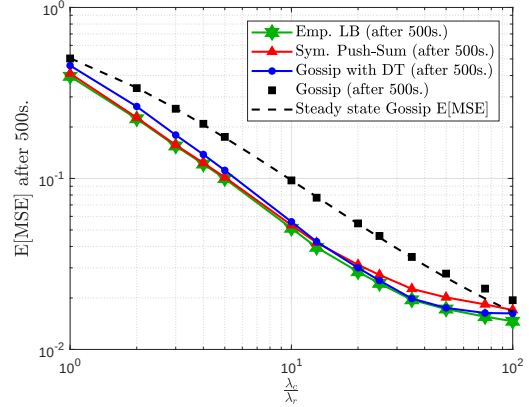
Figure 5.14 shows the performance (the expected mean squared error) achieved by the gossip averaging algorithm with difference tracking (without Assumption 5.1 and with $\beta = p_r$) after 500 seconds for different system configurations. This performance is compared to the one of the gossip averaging algorithm

(from Chapter 3), the symmetric push-sum algorithm (from Chapter 4) and the best ideal algorithm (from Chapter 2).

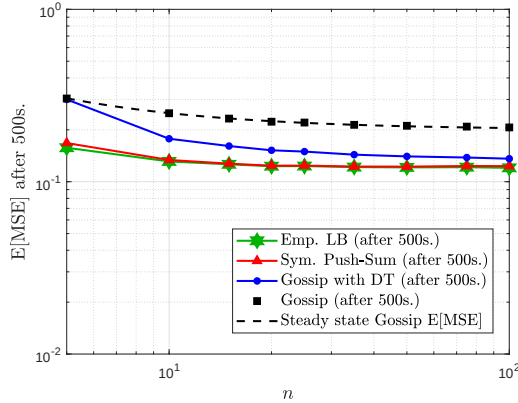
Actually it is the same figure than Figure 4.9 with an additional algorithm (in blue): the gossip algorithm with difference tracking that we have just described in Section 5.4. As a reminder, these figures show the impact of the system size n and the rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$ on the expected mean squared error of the algorithms after 500 seconds. The parameter scaling is done with constant $n\lambda_c = 10$, as explained in Section 1.3. Concretely, Figures 5.14a and 5.14b show the evolution with the rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$ respectively for a small and a large system ($n = 15$ and $n = 75$) and Figures 5.14c and 5.14d show the evolution with the system size n respectively for a small and a large rate ratio ($\rho = 4$ and $\rho = 50$).



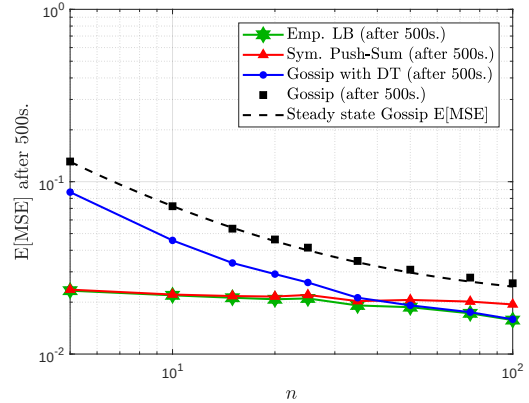
(a) Evolution with rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$, for $n = 15$.



(b) Evolution with rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$, for $n = 75$.



(c) Evolution with system size n , for $\rho = \frac{\lambda_c}{\lambda_r} = 4$.



(d) Evolution with system size n , for $\rho = \frac{\lambda_c}{\lambda_r} = 50$.

Figure 5.14: Comparison of the expected mean squared error after 500 seconds of the gossip averaging algorithm with difference tracking (denoted Gossip with DT) (using $\beta = p_r$ and without Assumption 5.1) with the one of other algorithms (best ideal algorithm, classical gossip, symmetric push-sum) for a variety of different systems (with constant $n\lambda_c = 10$). Expectations have been computed empirically with 1000 realization simulations. The theoretical expression of the steady state mean squared error of the gossip algorithm (see equation (3.5)) is also plotted (dashed black line).

We observe that the gossip algorithm with difference tracking (in blue) achieves a better expected mean squared error after 500 seconds than the classical gossip algorithm in any cases (or equal in some extreme cases). Of course, since the algorithm (with fixed $\beta = p_r$) mainly improve the expected variance, it gets closer to the empirical lower bound for system configurations where the expected variance has a great impact on the expected mean squared error of the algorithms. This is the case for large systems (i.e with large n) that have generally a small expected squared bias (such as in Figure 5.14b) or for systems with low rate ratio ρ for which the expected variance is generally large (such as in Figure 5.14c).

Moreover, the algorithm beats the symmetric push-sum algorithm (in red) in the only situations where it does not perform perfectly: when the symmetric push-sum algorithm decreases the expected variance slowly (for large rate ratio ρ) and when this expected variance impact the expected mean squared error (for large system size n) (see Chapter 4 for details). This can be observed in Figures 5.14b and 5.14d. This suggests that the combination of both algorithms (presented in Chapter 6), if it succeeds to benefit from the best of each algorithm, could perform very well and be almost as good as the empirical lower bound in any situation.

Conclusion about the gossip averaging algorithm with difference tracking

This chapter has presented a new gossip-based algorithm to solve the external averaging problem as stated in Section 1.1.1. The algorithm applies the idea of *difference tracking* to the gossip averaging algorithm in order to allow new agents to incorporate progressively their own information about μ into the estimates of the agents in the system and hence to reduce the expected variance due to repeated arrivals. The difference tracking idea, in brief, is that new agents pretend to have a x value of another agent in the system when they arrive and keep track of the difference δ with their true value that will be spread out and incorporated into the system progressively.

To study the algorithm theoretically, we have considered the assumption of arrival with direct communication. This assumption can be relaxed in practice, even if it implies troubles related to possible *restart* of the system. Theoretical results have allowed generalizing observations: the incorporation rate β impacts the steady state expected variance almost linearly (the less quickly the new information is incorporated into the system, the less variance it has (in expectation)). When the incorporation rate is too small (in comparison with the rate ratio), some agents leave the system with nonzero difference, meaning that some information is lost. For these small values of β the importance attributed to new agents is impacted and therefore the steady state expected squared bias of the algorithm is modified (in comparison with the gossip averaging algorithm). This also implies slower convergence of the algorithm toward its expected performance and a partially poor expected squared bias in a first time. Understanding the impact of the incorporation rate β have allowed to tune it consistently.

A fixed value of β implies choosing a compromise between the quality of the convergence speed and the quality of the steady state expected performance. Having a decreasing value of β with time allows not compromising either. It corresponds to giving a decreasing importance to new agents that arrives into the system (in the same idea as for the symmetric push-sum from Chapter 4) and therefore it improves significantly the expected squared bias of the gossip averaging algorithm. Further works are required in order to deduce a general decreasing expression of β . A well-tuned fixed β still allows improving the expected mean squared error of the gossip averaging algorithm (mainly by decreasing its expected variance) without compromising about the convergence rate toward the expected performance. It does not often reach the empirical lower bound for the expected mean squared error, but it does so in the only situations where the symmetric push-sum does not. In this regard, both algorithms complement each other and it seems natural to try combining them. That is the subject of the next chapter (Chapter 6).

5.A Appendix for the proof of Lemma 5.1

Proof of Lemma 5.1. For the following developments, the $+$ in the exponent indicates the values after the communication (and just before it without the $+$ exponent). We can compute $\mathbb{E}X^+$ by decomposing the expectation in two: the expectation over all possible sequences of events ϵ until this communication that allows knowing the x and δ values of all agents just before the communication have occur; and the expectation over the two agents i, j selected for the communication (denoted $\mathbb{E}_{i,j}$). We have that $\mathbb{E}X^+ = \mathbb{E}_\epsilon[\mathbb{E}_{i,j}[X^+ | x, \delta]]$. To compute this expectation for each descriptor, we use the fact that both communicating agents are selected uniformly randomly and independently among the agents of the system and then

$$\begin{aligned}\mathbb{E}_{i,j}[x_i^2 | x, \delta] &= \mathbb{E}_{i,j}[x_j^2 | x, \delta] = \overline{x^2}, & \mathbb{E}_{i,j}[x_i x_j | x, \delta] &= \mathbb{E}_{i,j}[x_i \bar{x} | x, \delta] = \mathbb{E}_{i,j}[x_j \bar{x} | x, \delta] = \overline{x^2}, \\ \mathbb{E}_{i,j}[\delta_i^2 | x, \delta] &= \mathbb{E}_{i,j}[\delta_j^2 | x, \delta] = \overline{\delta^2}, & \mathbb{E}_{i,j}[\delta_i \delta_j | x, \delta] &= \mathbb{E}_{i,j}[\delta_i \bar{\delta} | x, \delta] = \mathbb{E}_{i,j}[\delta_j \bar{\delta} | x, \delta] = \overline{\delta^2}, \\ \mathbb{E}_{i,j}[\delta_i \bar{x} | x, \delta] &= \mathbb{E}_{i,j}[\delta_j \bar{x} | x, \delta] = \overline{\delta \bar{x}}, & \mathbb{E}_{i,j}[\delta_i x_i | x, \delta] &= \mathbb{E}_{i,j}[\delta_j x_j | x, \delta] = \overline{\delta x}, \\ \mathbb{E}_{i,j}[\bar{\delta} x_i | x, \delta] &= \mathbb{E}_{i,j}[\bar{\delta} x_j | x, \delta] = \overline{\delta \bar{x}}.\end{aligned}$$

Using these relations, and the communication scheme definition for the algorithm (equations (5.3) and (5.4)), we can compute the expectation for each descriptor. For $\mathbb{E}[(\bar{x}^2)^+]$, we have

$$\begin{aligned}(\bar{x}^2)^+ &= \left(\frac{1}{n} \sum_{k \in \mathcal{N}} x_k^+ \right)^2 = \left(\frac{1}{n} \sum_{k \in \mathcal{N}} x_k - x_i - x_j + x_i^+ + x_j^+ \right)^2 = \left(\bar{x} + \frac{\beta}{n}(\delta_i + \delta_j) \right)^2 \\ &= \bar{x}^2 + \frac{\beta^2}{n^2}(\delta_i^2 + \delta_j^2 + 2\delta_i \delta_j) + 2\frac{\beta}{n}\bar{x}(\delta_i + \delta_j), \\ \mathbb{E}_{i,j}[(\bar{x}^2)^+ | x, \delta] &= \bar{x}^2 + \frac{\beta^2}{n^2}(2\overline{\delta^2} + 2\overline{\delta^2}) + 4\frac{\beta}{n}\overline{\delta \bar{x}}.\end{aligned}$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the first line of matrix A_c .

For $\mathbb{E}[\bar{x}^2]^+$, we have

$$\begin{aligned}\bar{x}^2^+ &= \frac{1}{n} \sum_{k \in \mathcal{N}} (x_k^+)^2 = \bar{x}^2 + \frac{1}{n} \left(2 \left(\frac{x_i + x_j}{2} + \beta \left(\frac{\delta_i + \delta_j}{2} \right) \right)^2 - x_i^2 - x_j^2 \right) \\ &= \bar{x}^2 + \frac{1}{n} \left(\frac{x_i^2}{2} + \frac{x_j^2}{2} + \frac{\beta^2 \delta_i^2}{2} + \frac{\beta^2 \delta_j^2}{2} + x_i x_j + \beta x_i \delta_i + \beta x_i \delta_j + \beta x_j \delta_i + \beta x_j \delta_j + \beta^2 \delta_i \delta_j - x_i^2 - x_j^2 \right),\end{aligned}$$

$$\mathbb{E}_{i,j}[\bar{x}^2^+ | x, \delta] = \frac{1}{n}\bar{x}^2 + \left(1 - \frac{1}{n}\right)\bar{x}^2 + \frac{\beta^2}{n}\overline{\delta^2} + \frac{\beta^2}{n}\overline{\delta^2} + \frac{2\beta}{n}\overline{\delta \bar{x}} + \frac{2\beta}{n}\overline{\delta \bar{x}}.$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the second line of matrix A_c .

For $\mathbb{E}[(\bar{\delta}^2)^+]$, we have

$$\begin{aligned}(\bar{\delta}^2)^+ &= \left(\frac{1}{n} \sum_{k \in \mathcal{N}} \delta_k^+ \right)^2 = \left(\bar{\delta} + \frac{(1-\beta)}{n}(\delta_i + \delta_j) - \frac{1}{n}(\delta_i + \delta_j) \right)^2 = \left(\bar{\delta} - \frac{\beta}{n}(\delta_i + \delta_j) \right)^2 \\ &= \bar{\delta}^2 + \frac{\beta^2}{n^2}(\delta_i^2 + \delta_j^2 + 2\delta_i \delta_j) - \frac{2\beta}{n}\bar{\delta}(\delta_i + \delta_j), \\ \mathbb{E}_{i,j}[(\bar{\delta}^2)^+ | x, \delta] &= \left(1 + \frac{2\beta^2}{n^2} - \frac{4\beta}{n}\right)\bar{\delta}^2 + \frac{2\beta^2}{n^2}\overline{\delta^2}.\end{aligned}$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the third line of matrix A_c .

For $\mathbb{E}[\bar{\delta}^{2+}]$, we have

$$\begin{aligned}\bar{\delta}^{2+} &= \frac{1}{n} \sum_{k \in \mathcal{N}} (\delta_k^+)^2 = \bar{\delta}^2 + \frac{1}{n} \left(2 \left((1-\beta) \left(\frac{\delta_i + \delta_j}{2} \right) \right)^2 - \delta_i^2 - \delta_j^2 \right) \\ &= \bar{\delta}^2 + \frac{1}{n} \left((1-\beta)^2 \left(\frac{\delta_i^2}{2} + \frac{\delta_j^2}{2} + \delta_i \delta_j \right) - \delta_i^2 - \delta_j^2 \right), \\ \mathbb{E}_{i,j}[\bar{x}^{2+} \mid x, \delta] &= \frac{(1-\beta)^2}{n} \bar{\delta}^2 + \left(1 + \frac{(1-\beta^2) - 2}{n} \right) \bar{\delta}^2.\end{aligned}$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the fourth line of matrix A_c .

For $\mathbb{E}[\bar{\delta} \bar{x}]$, we have

$$\begin{aligned}\bar{\delta}^+ \bar{x}^+ &= \left(\frac{1}{n} \sum_{k \in \mathcal{N}} \delta_k^+ \right) \left(\frac{1}{n} \sum_{k \in \mathcal{N}} x_k^+ \right) = \left(\bar{\delta} - \frac{\beta}{n} (\delta_i + \delta_j) \right) \left(\bar{x} + \frac{\beta}{n} (\delta_i + \delta_j) \right) \\ &= \bar{\delta} \bar{x} + \frac{\beta}{n} (\bar{\delta} \delta_i + \bar{\delta} \delta_j) - \frac{\beta}{n} (\bar{x} \delta_i + \bar{x} \delta_j) - \frac{\beta^2}{n^2} (\delta_i^2 + \delta_j^2 + \delta_i \delta_j), \\ \mathbb{E}_{i,j}[\bar{x}^{2+} \mid x, \delta] &= \left(\frac{2\beta}{n} - \frac{\beta^2}{n^2} \right) \bar{\delta}^2 - \frac{2\beta^2}{n^2} \bar{\delta}^2 + \left(1 - \frac{2\beta}{n} \right) \bar{\delta} \bar{x}.\end{aligned}$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the fifth line of matrix A_c .

For $\mathbb{E}[\bar{\delta} x]$, we have

$$\begin{aligned}\bar{\delta} x^+ &= \frac{1}{n} \sum_{k \in \mathcal{N}} \delta_k^+ x_i^+ = \bar{\delta} x + \frac{1}{n} \left(2 \left(\frac{\delta_i + \delta_j}{2} (1-\beta) \right) \left(\frac{x_i + x_j}{2} + \beta \frac{\delta_i + \delta_j}{2} \right) - \delta_i x_i - \delta_j x_j \right) \\ &= \bar{\delta} x + \frac{(1-\beta)}{2n} (\delta_i x_i + \delta_i x_j + \delta_j x_i + \delta_j x_j) + \frac{(1-\beta)\beta}{2n} (\delta_i^2 + \delta_j^2 + 2\delta_i \delta_j) - \frac{1}{n} \delta_i x_i - \frac{1}{n} \delta_j x_j, \\ \mathbb{E}_{i,j}[\bar{x}^{2+} \mid x, \delta] &= \frac{(1-\beta)\beta}{n} \bar{\delta}^2 + \frac{(1-\beta)\beta}{n} \bar{\delta}^2 + \frac{1-\beta}{n} \bar{x} \bar{\delta} + \left(1 + \frac{-\beta-1}{n} \right) \bar{\delta} x.\end{aligned}$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the sixth line of matrix A_c .

□

5.B Appendix for the proof of Lemma 5.2

Proof of Lemma 5.2. The effect of a replacement can be decomposed into the effect of departure of one random agent from the system that changes its size from n to $n-1$ momentarily and the effect of arrival of a new agent into the system (initialized according to (5.1) and (5.2)) that directly communicates with one of the $n-1$ remaining agents and restores the system size to n . Let us first analyze the evolution of the descriptors with the departure of a randomly uniformly selected agent from the system.

Departure For the following developments, the $+$ in the exponent indicates the values after the departure (and just before it without the $+$ exponent). We can compute $\mathbb{E}X^+$ by decomposing the expectation in two: the expectation over all possible sequences of events ϵ until this departure that allows knowing the x and δ values of all agents just before the departure have occurred; and the expectation over the agent i selected for leaving (denoted $\mathbb{E}_i$). We have that $\mathbb{E}X^+ = \mathbb{E}_\epsilon[\mathbb{E}_i[X^+ | x, \delta]]$. Using the fact that the leaving agent i is selected independently and randomly uniformly selected among the agents of the system, we have that

$$\begin{aligned}\mathbb{E}_i[x_i^2 | x, \delta] &= \bar{x}^2, & \mathbb{E}_i[x_i \bar{x} | x, \delta] &= \bar{x}^2, & \mathbb{E}_i[\delta_i^2 | x, \delta] &= \bar{\delta}^2, & \mathbb{E}_i[\delta_i \bar{\delta} | x, \delta] &= \bar{\delta}^2, \\ \mathbb{E}_i[\delta_i \bar{x} | x, \delta] &= \mathbb{E}_i[\bar{\delta} x_i | x, \delta] &= \bar{\delta} \bar{x}, & & \mathbb{E}_i[\delta_i x_i | x, \delta] &= \bar{\delta} \bar{x}.\end{aligned}$$

These relations are used to compute the evolution of each descriptor with a departure.

For $\mathbb{E}[(\bar{x}^2)^+]$, we have

$$\begin{aligned}(\bar{x}^2)^+ &= \left(\frac{1}{n-1} \left(\sum_{k \in \mathcal{N}} x_k - x_i \right) \right)^2 = \frac{1}{(n-1)^2} (n^2 \bar{x}^2 + x_i^2 - 2n \bar{x} x_i), \\ \mathbb{E}[(\bar{x}^2)^+ | x, \delta] &= \frac{n^2 - 2n}{(n-1)^2} \bar{x}^2 + \frac{1}{(n-1)^2} \bar{x}^2.\end{aligned}$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the first line of matrix A_d .

For $\mathbb{E}[\bar{x}^{2+}]$, we have

$$\bar{x}^{2+} = \frac{1}{n-1} \left(\sum_{k \in \mathcal{N}} x_k^2 - x_i^2 \right) = \frac{1}{(n-1)} (n \bar{x}^2 - x_i^2) \quad \text{and} \quad \mathbb{E}[\bar{x}^{2+} | x, \delta] = \bar{x}^2.$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the second line of matrix A_d .

For $\mathbb{E}[(\bar{\delta}^2)^+]$, we have

$$\begin{aligned}(\bar{\delta}^2)^+ &= \left(\frac{1}{n-1} \left(\sum_{k \in \mathcal{N}} \delta_k - \delta_i \right) \right)^2 = \frac{1}{(n-1)^2} (n^2 \bar{\delta}^2 + \delta_i^2 - 2n \bar{\delta} \delta_i), \\ \mathbb{E}[(\bar{\delta}^2)^+ | x, \delta] &= \frac{n^2 - 2n}{(n-1)^2} \bar{\delta}^2 + \frac{1}{(n-1)^2} \bar{\delta}^2.\end{aligned}$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the third line of matrix A_d .

For $\mathbb{E}[\bar{\delta}^{2+}]$, we have

$$\bar{\delta}^{2+} = \frac{1}{n-1} \left(\sum_{k \in \mathcal{N}} \delta_k^2 - \delta_i^2 \right) = \frac{1}{(n-1)} (n \bar{\delta}^2 - \delta_i^2) \quad \text{and} \quad \mathbb{E}[\bar{\delta}^{2+} | x, \delta] = \bar{\delta}^2.$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the fourth line of matrix A_d .

For $\mathbb{E}[\bar{\delta}^+ \bar{x}^+]$, we have

$$\begin{aligned}\bar{\delta}^+ \bar{x}^+ &= \left(\frac{1}{n-1} \left(\sum_{k \in \mathcal{N}} \delta_k - \delta_i \right) \right) \left(\frac{1}{n-1} \left(\sum_{k \in \mathcal{N}} x_k - x_i \right) \right) = \frac{1}{(n-1)^2} (n \bar{\delta} - \delta_i)(n \bar{x} - x_i), \\ &= \frac{1}{(n-1)^2} (n^2 \bar{\delta} \bar{x} - n(\bar{\delta} x_i + \delta_i \bar{x}) + \delta_i x_i) \\ \mathbb{E}[\bar{\delta}^+ \bar{x}^+ | x, \delta] &= \frac{n^2 - 2n}{(n-1)^2} \bar{\delta} \bar{x} + \frac{1}{(n-1)^2} \bar{\delta} \bar{x}.\end{aligned}$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the fifth line of matrix A_d .

For $\mathbb{E}[\overline{\delta x}^+]$, we have

$$\overline{\delta x}^+ = \frac{1}{n-1} \left(\sum_{k \in \mathcal{N}} \delta_k x_k - \delta_i x_i \right) = \frac{1}{(n-1)} (n \overline{\delta x} - \delta_i x_i) \quad \text{and} \quad \mathbb{E}[\overline{\delta x}^+ | x, \delta] = \overline{\delta x}.$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the sixth line of matrix A_d .

After this departure, we have that $\mathbb{E}[X^+] = A_d \mathbb{E}[X]$. For a replacement, the departure of an agent is followed by the arrival of a new agent that directly communicates.

Arrival with communication Let us analyze the evolution of the descriptors $\mathbb{E}[X]$ with the arrival of a new agent that directly communicate with a randomly uniformly selected agent from the system. A departure is supposed to precedes this arrival and therefore the size of the system just before the arrival is $n-1$. For the following developments, the $+$ in the exponent indicates the values just after the arrival with communication (and just before it without the $+$ exponent). We can compute $\mathbb{E}X^+$ (i.e. $\mathbb{E}X$ after the arrival with communication) by decomposing the expectation in two: the expectation over everything that happened before this arrival (denoted $\mathbb{E}_\epsilon$) (including the sequence of events and the departure that precedes this arrival) that allows knowing the x and δ values of all agents just before the arrival occurs; and the expectation over the information related to this arrival with communication (denoted $\mathbb{E}_{\text{arr}}$): the measurement z_i of the incoming agent i and the agent j selected to communicate with the new agent i . We have that $\mathbb{E}X^+ = \mathbb{E}_\epsilon[\mathbb{E}_{\text{arr}}[X^+ | x, \delta]]$.

Combining initialization of the new agent from equations (5.1) and (5.2) with the communications equation of the algorithm ((5.3) and (5.4)), the value of x_i and x_j after the arrival with communication are defined as:

$$x_i^+ = x_j^+ = \frac{x_i(0) + x_j}{2} + \beta \left(\frac{\delta_i(0) + \delta_j}{2} \right) = x_j + \beta \left(\frac{z_i - x_j + \delta_j}{2} \right) \quad (5.12)$$

$$\delta_i^+ = \delta_j^+ = \frac{\delta_i(0) + \delta_j}{2} (1 - \beta) = \frac{z_i - x_j + \delta_j}{2} (1 - \beta). \quad (5.13)$$

Let us analyze the evolution of each descriptor separately. This analysis, in addition of using of the definition of the new x and δ values given by these two equations ((5.12) and (5.13)), also uses the fact that the measurement z_i of the new agent i is independently drawn from a distribution of mean $\mu = 0$ and variance σ^2 and that the communicating agent j is independently and randomly uniformly selected among the agents of the system. It implies that

$$\mathbb{E}_{\text{arr}}[x_j^2 | x, \delta] = \overline{x^2}, \quad \mathbb{E}_{\text{arr}}[x_j \overline{x} | x, \delta] = \overline{x^2}, \quad \mathbb{E}_{\text{arr}}[\delta_j^2 | x, \delta] = \overline{\delta^2}, \quad \mathbb{E}_{\text{arr}}[\delta_j \overline{\delta} | x, \delta] = \overline{\delta^2},$$

$$\mathbb{E}_{\text{arr}}[\delta_j \overline{x} | x, \delta] = \mathbb{E}_{\text{arr}}[\overline{\delta} x_j | x, \delta] = \overline{\delta} \overline{x}, \quad \mathbb{E}_{\text{arr}}[\delta_j x_j | x, \delta] = \overline{\delta x}, \quad \mathbb{E}_{\text{arr}}[z_i^2 | x, \delta] = \sigma^2 \quad \mathbb{E}_{\text{arr}}[z_i | x, \delta] = 0.$$

Let us remember that the size of the system before this arrival is $n-1$. For $\mathbb{E}[(\overline{x}^2)^+]$, we have

$$\begin{aligned} (\overline{x}^2)^+ &= \left(\frac{1}{n} \left(\sum_{k \in \mathcal{N}} x_k + x_i^+ + x_j^+ - x_j \right) \right)^2 = \frac{1}{n^2} ((n-1)\overline{x} + (1-\beta)x_j + \beta\delta_j + \beta z_i)^2 \\ &= \frac{1}{n^2} \left[(n-1)^2 \overline{x}^2 + (1-\beta)^2 x_j^2 + \beta^2 \delta_j^2 + \beta^2 z_i^2 + 2(n-1)(1-\beta)\overline{x}x_j + 2(n-1)\beta\overline{x}\delta_j + \right. \\ &\quad \left. + 2(n-1)\beta\overline{x}z_i + 2\beta(1-\beta)\delta_j x_j + 2\beta(1-\beta)x_j z_i + 2\beta^2 \delta_j z_i \right], \\ \mathbb{E}[(\overline{x}^2)^+ | x, \delta] &= \frac{(n-1)^2 + 2(n-1)(1-\beta)}{n^2} \overline{x}^2 + \frac{(1-\beta)^2}{n^2} \overline{x^2} + \frac{\beta^2}{n^2} \overline{\delta^2} + \frac{2(n-1)\beta}{n^2} \overline{x\delta} + \frac{2\beta(1-\beta)}{n^2} \overline{\delta x} + \frac{\beta^2}{n^2} \sigma^2. \end{aligned}$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the first line of matrix A_a and vector b_a .

For $\mathbb{E}[\overline{x^2}^+]$, we have

$$\begin{aligned}\overline{x^2}^+ &= \frac{1}{n} \left(\sum_{k \in \mathcal{N}} x_k^2 + (x_i^+)^2 + (x_j^+)^2 - x_j^2 \right) = \frac{1}{n} \left((n-1)\overline{x^2} + 2 \left(x_j + \beta \left(\frac{z_i - x_j + \delta_j}{2} \right) \right)^2 - x_j^2 \right) \\ &= \frac{1}{n} \left((n-1)\overline{x^2} + \frac{\beta^2}{2} (z_i - x_j + \delta_j)^2 + 2\beta x_j (z_i - x_j + \delta_j) + x_j^2 \right), \\ \mathbb{E}[\overline{x^2}^+ | x, \delta] &= \frac{(n-1) + \frac{\beta^2}{2} - 2\beta + 1}{n} \overline{x^2} + \frac{\beta^2}{2n} \overline{\delta^2} + \frac{-\beta^2 + 2\beta}{n} \overline{\delta x} + \frac{\beta^2}{2n} \sigma^2\end{aligned}$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the second line of matrix A_a and vector b_a .

For $\mathbb{E}[(\overline{\delta^2})^+]$, we have

$$\begin{aligned}(\overline{\delta^2})^+ &= \left(\frac{1}{n} \left(\sum_{k \in \mathcal{N}} \delta_k + \delta_i^+ + \delta_j^+ - \delta_j \right) \right)^2 = \frac{1}{n^2} ((n-1)\overline{\delta} + (z_i - x_j)(1 - \beta) + \beta\delta_j)^2 \\ &= \frac{1}{n^2} \left[(n-1)^2 \overline{\delta^2} + (1 - \beta)^2 (z_i - x_j)^2 + \beta^2 \delta_j^2 + 2(n-1)(1 - \beta)(z_i - x_j)\overline{\delta} + \right. \\ &\quad \left. + 2(n-1)\beta\overline{\delta}\delta_j + 2(1 - \beta)\beta(z_i - x_j)\delta_j \right], \\ \mathbb{E}[(\overline{\delta^2})^+ | x, \delta] &= \frac{(1 - \beta)^2}{n^2} \overline{x^2} + \frac{(n-1)^2 - 2(n-1)\beta}{n^2} \overline{\delta^2} + \frac{\beta^2}{n^2} \overline{\delta^2} - \frac{2(n-1)(1 - \beta)}{n^2} \overline{x\delta} + \frac{2(1 - \beta)\beta}{n^2} \overline{\delta x} + \frac{(1 - \beta)^2}{n^2} \sigma^2.\end{aligned}$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the third line of matrix A_a and vector b_a .

For $\mathbb{E}[\overline{\delta^2}^+]$, we have

$$\begin{aligned}\overline{\delta^2}^+ &= \frac{1}{n} \left(\sum_{k \in \mathcal{N}} \delta_k^2 + (\delta_i^+)^2 + (\delta_j^+)^2 - \delta_j^2 \right) = \frac{1}{n} \left((n-1)\overline{\delta^2} + 2 \left(\frac{z_i - x_j + \delta_j}{2} \right)^2 - \delta_j^2 \right) \\ &= \frac{1}{n} \left((n-1)\overline{\delta^2} + \frac{(1 - \beta)^2}{2} (z_i^2 + x_j^2 + \delta_j^2 - 2z_i x_j - 2z_i \delta_j + 2\delta_j x_j) - \delta_j^2 \right), \\ \mathbb{E}[\overline{\delta^2}^+ | x, \delta] &= \frac{(1 - \beta)^2}{2n} \overline{x^2} + \frac{(n-1) + \frac{(1 - \beta)^2}{2} - 1}{n} \overline{\delta^2} + \frac{(1 - \beta)^2}{n} \overline{\delta x} + \frac{(1 - \beta)^2}{2n} \sigma^2\end{aligned}$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the fourth line of matrix A_a and vector b_a .

For $\mathbb{E}[\overline{\delta^+ x^+}]$, we have

$$\begin{aligned}\overline{\delta^+ x^+} &= \left(\frac{1}{n} \left(\sum_{k \in \mathcal{N}} x_k + x_i^+ + x_j^+ - x_j \right) \right) \left(\frac{1}{n} \left(\sum_{k \in \mathcal{N}} \delta_k + \delta_i^+ + \delta_j^+ - \delta_j \right) \right) \\ &= \frac{1}{n^2} ((n-1)\overline{x} + x_j + \beta(z_i - x_j + \delta_j))((n-1)\overline{\delta} + (z_i - x_j + \delta_j)(1 - \beta) - \delta_j) \\ &= \frac{1}{n^2} ((n-1)\overline{x} + (1 - \beta)x_j + \beta(z_i + \delta_j))((n-1)\overline{\delta} + (z_i - x_j)(1 - \beta) - \beta\delta_j) \\ &= \frac{1}{n^2} \left[(n-1)^2 \overline{x\delta} - \beta(n-1)\overline{x\delta}_j + (n-1)(1 - \beta)(z_i - x_j)\overline{x} + (n-1)(1 - \beta)\overline{\delta}x_j - (1 - \beta)\beta\delta_j x_j + \right. \\ &\quad \left. + (1 - \beta)^2 (z_i - x_j)x_j + (n-1)\beta\overline{\delta}(z_i + \delta_j) - \beta^2(z_i + \delta_j)\delta_j + (1 - \beta)\beta(z_i + \delta_j)(z_i - x_j) \right], \\ \mathbb{E}[\overline{\delta^+ x^+} | x, \delta] &= -\frac{(n-1)(1 - \beta)}{n^2} \overline{x^2} - \frac{(1 - \beta)^2}{n^2} \overline{x^2} + \frac{(n-1)\beta}{n^2} \overline{\delta^2} - \frac{\beta^2}{n^2} \overline{\delta^2} \\ &\quad + \frac{(n-1)(n-2\beta)}{n^2} \overline{x\delta} - \frac{2(1 - \beta)\beta}{n^2} \overline{\delta x} + \frac{(1 - \beta)\beta}{n^2} \sigma^2.\end{aligned}$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the fifth line of matrix A_a and vector b_a .

For $\mathbb{E}[\overline{\delta x}^+]$, we have

$$\begin{aligned}
\overline{\delta x}^+ &= \frac{1}{n} \left(\sum_{k \in \mathcal{N}} \delta_k x_k + \delta_i^+ x_i^+ + \delta_j^+ x_j^+ - \delta_j x_j \right) \\
&= \frac{1}{n} \left((n-1)\overline{\delta x} + 2 \left(x_j + \beta \left(\frac{z_i - x_j + \delta_j}{2} \right) \right) \left((1-\beta) \left(\frac{z_i - x_j + \delta_j}{2} \right) \right) - \delta_j x_j \right) \\
&= \frac{1}{n} \left((n-1)\overline{\delta x} + (1-\beta)x_j(z_i - x_j + \delta_j) + \frac{\beta(1-\beta)}{2}(z_i - x_j + \delta_j)^2 - \delta_j x_j \right) \\
\mathbb{E}[\overline{\delta x}^+ \mid x, \delta] &= \frac{-(1-\beta) + \frac{\beta(1-\beta)}{2}}{n} \overline{x^2} + \frac{\beta(1-\beta)}{2n} \overline{\delta^2} + \frac{(n-1) + (1-\beta)^2 - 1}{n} \overline{\delta x} + \frac{\beta(1-\beta)}{2n} \sigma^2
\end{aligned}$$

Taking $\mathbb{E}_\epsilon$ of the last equation gives the sixth line of matrix A_a and vector b_a .

□

Chapter 6

Symmetric Push-Sum algorithm with difference tracking

In order to contextualize this chapter, let us briefly summarize the main results of Chapters 3, 4 and 5. Chapter 3 presents the well-known gossip averaging algorithm and the two main issues it has for solving the external averaging problem (see the problem statement in Section 1.1.1). The first issue is related to the expected squared bias and is due to the fact that new agents have too much importance when they arrive in the system. Chapter 4 presents an algorithm (the symmetric push-sum) that aims to solve this issue by giving a decreasing weight to new incoming agents. It also partially solves the second issue of the gossip algorithm which is related to the expected variance. Chapter 5 presents the idea of difference tracking applied to the gossip algorithm for reducing quickly the expected variance of the system. It is possible to combine these two algorithms into one in order to solve rapidly both issues of the gossip averaging algorithm in the same time and to get an expected mean squared error very close to the empirical lower bound. This symmetric push-sum with difference tracking is presented in the first section of chapter. Thereafter, its performance is analyzed using simulation results in order to show how much the combination of both algorithms is successful.

6.1 Algorithm description

In order to apply the difference tracking idea on the symmetric push-sum algorithm (from Chapter 4), it requires each agent to have two additional difference variables: one related to the x variable (called δ , as in previous chapter) and another related to the w variable (called δ_w). When a new agent i arrives into the system and first communicates with another (old) agent j , it pretends to have both its x_j value and its weight w_j in order to obtain the same estimate $y_i = y_j = \frac{x_j}{w_j}$. Using the same initial weights as for the symmetric push-sum from Chapter 4, the algorithm can be described as follows (omitting time dependence notation for x , w , δ and δ_w):

Initialization for any agent $i \in \mathcal{N}(0)$: $x_i(0) = z_i$, $w_i(0) = 1$, $\delta_{w,i} = 0$, and $\delta_i(0) = 0$,
setting parameter $\beta \in [0, 1]$.

Replacement (at time t) of agent $l \in \mathcal{N}(t^-)$ by agent i . It is the k^{th} replacement in the system:

$\mathcal{N}(t) = \mathcal{N}(t^-) \setminus \{l\}$	$\delta_{w,i}(0) = 0$	Departure of agent l ,
$w_i(0) = \left(\frac{n-1}{n}\right)^k$,	$\delta_i(0) = 0$	Initial weight of agent i
$x_i(0) = w_i z_i$,		Initial x value of agent i ,
$\mathcal{N}(t^+) = \mathcal{N}(t) \cup \{i\}$		Arrival of agent i .

Before the first communication of a new agent $i \in \mathcal{N}(t)$ with an (*old*) agent $j \in \mathcal{N}(t)$

$$\begin{array}{ll} \delta_i = x_i - x_j & \text{Difference with its true measurement,} \\ x_i = x_j & \text{Initial pretending } x \text{ value of agent } i, \\ \delta_{w,i} = w_i - w_j & \text{Difference with its true weight,} \\ w_i = w_j & \text{Initial pretending weight of agent } i, \end{array}$$

The new agent i and the other agent j perform a communication together (see below).

Communication between agents $i \in \mathcal{N}(t)$ and $j \in \mathcal{N}(t)$:

$$x_i^+ = x_j^+ = \frac{x_i + x_j}{2} + \beta \left(\frac{\delta_i + \delta_j}{2} \right) \quad (6.1)$$

$$\delta_i^+ = \delta_j^+ = \frac{\delta_i + \delta_j}{2} (1 - \beta) \quad (6.2)$$

$$w_i^+ = w_j^+ = \frac{w_i + w_j}{2} + \beta \left(\frac{\delta_{w,i} + \delta_{w,j}}{2} \right) \quad (6.3)$$

$$\delta_{w,i}^+ = \delta_{w,j}^+ = \frac{\delta_{w,i} + \delta_{w,j}}{2} (1 - \beta) \quad (6.4)$$

Equation (6.1) averages the x values of both communicating agents and adds a part (β) of their difference variable δ . Equation (6.2) averages the remaining differences of both communicating agents in order that the difference δ of each agent spreads through the system during future interactions. The weights w and their related difference δ_w are updated similarly (see equations (6.3) and (6.4)).

This algorithm works either under the assumption of arrival with direct communication (see Assumption 5.1) or without this assumption to be more general. Without this assumption, the same remarks as in Section 5.4 can be made and the agents wait for their first communication with an *old* agent to pretend having different x and w values and truly initialize their difference variables. If two new agents are communicating with each other before meeting any *old* agent, they perform a communication without pretending anything about their x or w (i.e. $\delta = \delta_w = 0$). To be clear, let us recall from Section 5.4 that an *old* agent is an agent that has already been fully initialized: either an agent from $\mathcal{N}(0)$, initially present in the system, or an agent that has already communicated with an *old* agent. Relaxing Assumption 5.1 allows the system to have a chance (often little) to *restart* (i.e. when all the agents are replaced without communicating their information to the new ones, see Definition 2.3). When it happens, if no restart detection protocol is set up, the agents will simply perform an algorithm equivalent to the symmetric push-sum, without any difference tracking. Restart detection protocol has not been studied in this master thesis and could consist in a future work. Fortunately, it is quite unlikely to happen in most system configuration, unless waiting very long time.

The incorporation rate β corresponds to the part of the difference variables that is incorporated into the x and w variables at each communication. The value of β is identical for all agents and fixed at the initialization. The same incorporation rate is used for δ and δ_w in order to avoid having the x or the w values that is evolving more quickly than the other and to ensure keeping consistent estimates. More experiments could be conduct about this.

For this algorithm, the importance of each new agent is properly decided by its initial weight and should not be modified through the difference tracking. On the contrary of the gossip with difference tracking, it is therefore important that the information of a new agent (its x and w) is entirely (or in major part) incorporated into the system in order to only impact the expected variance of the symmetric push-sum algorithm, without modifying its expected squared bias (which seems already almost perfect). That is why, the fixed value of $\beta = p_r$ obtained in Section 5.3 based on threshold above which the convergence rate is nearly constant should be increased to ensure that the expected squared bias is (almost) not modified at all. Within this constraint, the value of β still needs to be the lowest possible

in order to decrease the expected variance the most possible. The consistent choice we have made is

$$\beta = 2p_r = \frac{2}{1 + \rho}.$$

We have well that $\beta \in [0, 1]$ for $p_r \in [0, \frac{1}{2}]$ (which we suppose, since having $p_r > \frac{1}{2}$ (or $\rho < 1$) corresponds to situations where the replacements are so frequent that it does not really make sense (see Section 1.1.4)). The factor 2 is a kind of security factor and is a bit arbitrary. More elaborated analyses could conduct to another tuning of the β . The important is that it seems working very well in many cases, as illustrated in the following section.

Implementing this algorithm requires the agents to know (or to estimate) the size n of the system and their rank of arrival in the system in order to initialize correctly their own weight, and the value of ρ (or equivalently p_r) in order to set up correctly the value of the incorporation rate β . As detailed in Section 4.5, the agents can estimate their rank of arrival efficiently by knowing the rate ratio ρ (or equivalently p_r) and the system size. This estimation protocol may still fail if a *restart* of the system occurs.

6.2 Performance analysis

This section analyzes the performance of the symmetric push-sum algorithm with difference tracking, in comparison with other algorithms described in previous chapters, including the symmetric push-sum (from Chapter 4) which already achieves very good performance in many situations (i.e. close to the empirical lower bound on the expected MSE). To be able to compare the results with those of the previous chapters, the performance results are presented without Assumption 5.1 about the arrival with communication and the problem solved by the algorithm is therefore exactly the one described in the problem statement (see Section 1.1.1).

We will see that the symmetric push-sum algorithm with difference tracking successfully and rapidly solves both issues of the gossip algorithm (related to the expected variance and the expected squared bias). Therefore, it has an expected mean squared error better (or equal) than any other algorithm we have analyzed and that gets rapidly very close to the empirical lower bound in any situation. To show this, the section first analyzes the time evolution of the expected performance of the algorithm (in comparison with the others) and thereafter, it presents its expected mean squared error after 500 seconds for a variety of different system configurations (n and ρ).

6.2.1 Time evolution of the expected performance

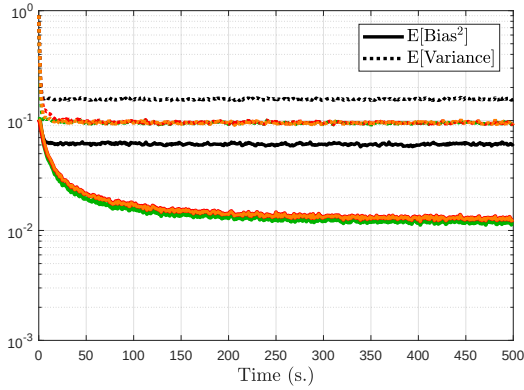
Figure 6.1 and 6.2 show the evolution of the performance of the algorithm for the same four different system configurations we used for the analysis of the symmetric push-sum algorithm in Section 4.3. Figure 6.1 shows the expected squared bias and expected variance and Figure 6.2 shows the expected mean squared error. As expected, the symmetric push-sum with difference tracking (in orange) achieves an expected squared bias which is as good as the one of the symmetric push-sum (in red) in all these situations (see Figure 6.1). It means that the difference tracking does not imply a real loss of information in expectation, but it still improves the expected variance of the symmetric push-sum algorithm. Indeed, in Figure 6.1, the expected variance of the symmetric push-sum with difference tracking decreases quickly toward the one of the best ideal algorithm, which is not always the case of the symmetric push-sum, as shown in Figures 6.1b, 6.1c and 6.1d.

It results in an improvement of the expected mean squared error of the symmetric push-sum error in these situations if it is significantly impacted by the expected variance (such as in Figures 6.2b and 6.2d). In other cases (Figures 6.2a and 6.2c), the symmetric push-sum algorithm already fits the empirical lower bound. In conclusion, the difference tracking allows the symmetric push-sum to fit the expected mean squared error lower bound more rapidly, without compromising about its steady state performance.

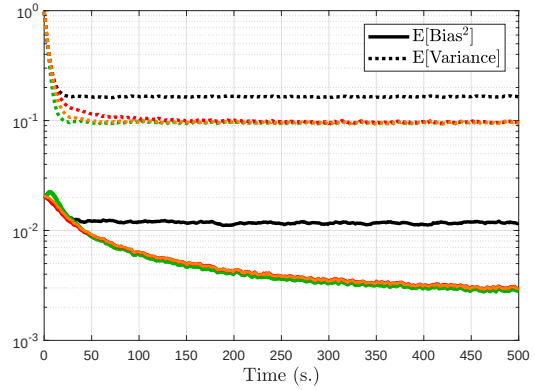
It remains a tiny improvement margin for the algorithm in the situations where the expected variance does not fit directly the one of the best ideal algorithm. This is visible when the system is large and has a great rate ratio (such as in Figures 6.2b and 6.2d or in their corresponding bias-variance decomposition in Figures 6.1b and 6.1d). This is probably due, in part, to the variance decrease that is achieved at each communication in the symmetric push-sum algorithm which is not totally optimal. The parameter scaling $n\lambda_c = 10$ that is used may thus explain why it is more visible for large system since it ensures that the global rate of communications events in the system remains identical, even if the system size increases, which implies that the expected variance in the system decreases less rapidly (in time) due to communication.

Moreover, in this algorithm, the arrival of a new agent also implies a decreasing of the expected variance: due to its pretending behavior but also to its smaller initial weights. Larger rate ratio ρ (with our parameter scaling choice) implies having a smaller replacement rate, which lead to a slower decrease of the variance due to arrival.

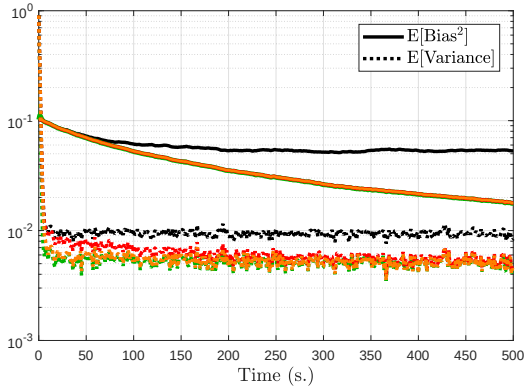
As shown in Figures 6.2b and 6.2d, the remaining possible improvement is very limited, but it exists. Next section shows that after a quite long time (500 seconds), the expected mean squared error of the algorithm fits the empirical lower bound almost perfectly.



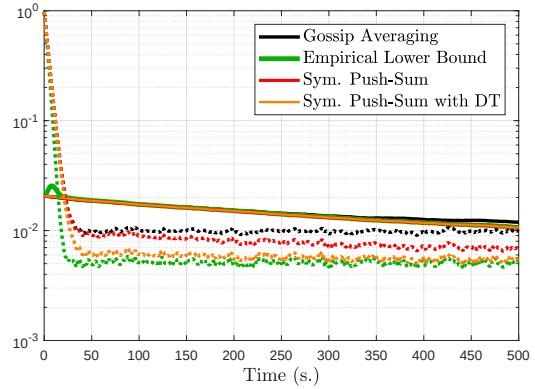
(a) $n = 10, \rho = 5$



(b) $n = 50, \rho = 5$



(c) $n = 10, \rho = 100$



(d) $n = 50, \rho = 100$

Figure 6.1: Comparison between the expected squared bias (solid line) and the expected variance (dotted line) time evolution of the symmetric push-sum algorithm with difference tracking (in orange), the symmetric push-sum algorithm (in red), the gossip averaging algorithm (in black) and the best ideal algorithm (in green) (that provides the empirical lower bound on $\mathbb{E}[\text{MSE}(t)]$) for four different system configurations (n and $\rho = \frac{\lambda_c}{\lambda_r}$) with scaling $n\lambda_c = 10$. Expectations have been computed empirically with 5000 realization simulations.

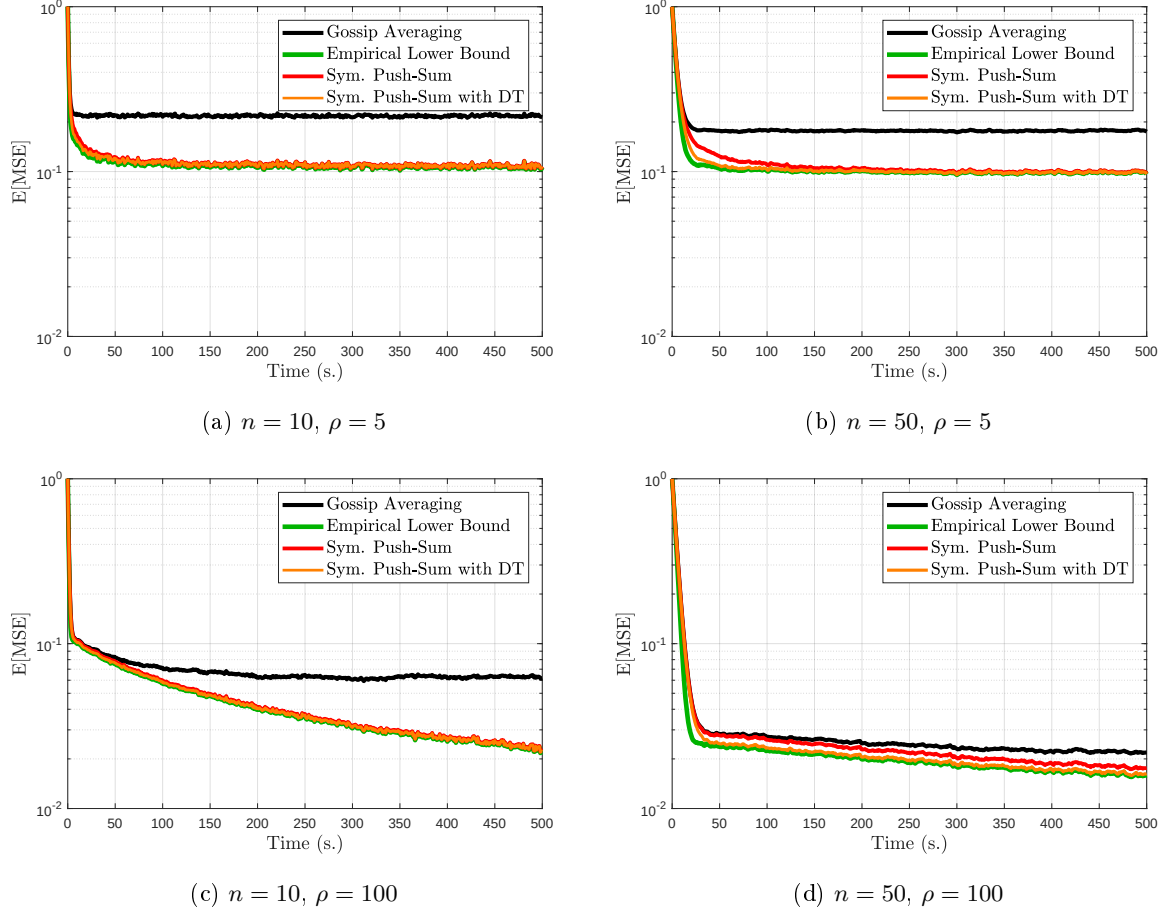


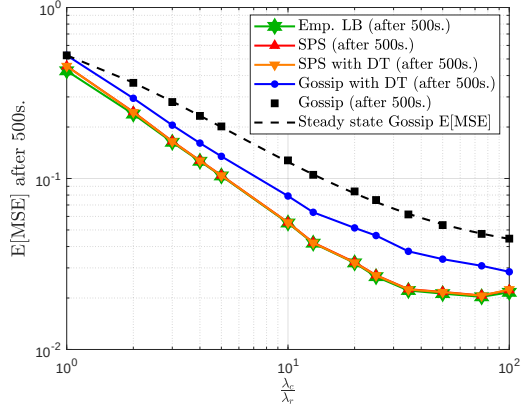
Figure 6.2: Comparison between the expected mean squared error time evolution ($\mathbb{E}[\text{MSE}(t)]$) of the symmetric push-sum algorithm with difference tracking (Sym. Push-Sum with DT), the symmetric push-sum algorithm, the gossip averaging algorithm and the best ideal algorithm (that provides the empirical lower bound on $\mathbb{E}[\text{MSE}(t)]$) for four different system configurations (n and $\rho = \frac{\lambda_c}{\lambda_r}$) with scaling $n\lambda_c = 10$. Expectations have been computed empirically with 5000 realization simulations.

6.2.2 Expected performance after 500 seconds

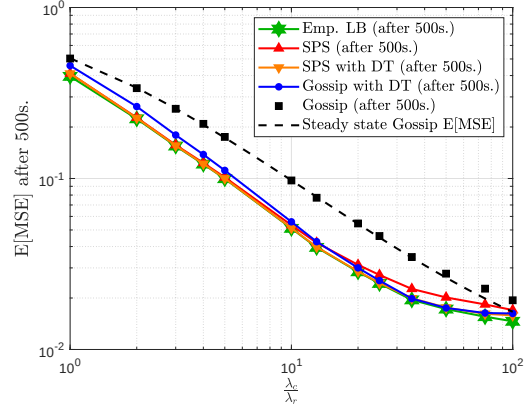
Figure 6.3 shows the performance (the expected mean squared error) achieved by the symmetric push-sum algorithm with difference tracking (without Assumption 5.1 and with $\beta = 2p_r$) after 500 seconds for different system configurations. This performance is compared to the one of the gossip averaging algorithm (from Chapter 3), the symmetric push-sum algorithm (from Chapter 4), the gossip averaging with difference tracking (from Chapter 5) and the best ideal algorithm (from Chapter 2).

Actually it is the same figure than Figure 5.14 with an additional algorithm (in orange): the symmetric push-sum algorithm with difference tracking that we have just described in the previous section. As a reminder, these figures show the impact of the system size n and the rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$ on the expected mean squared error of the algorithms after 500 seconds. The parameter scaling is done with constant $n\lambda_c = 10$, as explained in Section 1.3. Concretely, Figures 6.3a and 6.3b show the evolution with the rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$ respectively for a small and a large system ($n = 15$ and $n = 75$) and Figures 6.3c and 6.3d show the evolution with the system size n respectively for a small and a large rate ratio ($\rho = 4$ and $\rho = 50$).

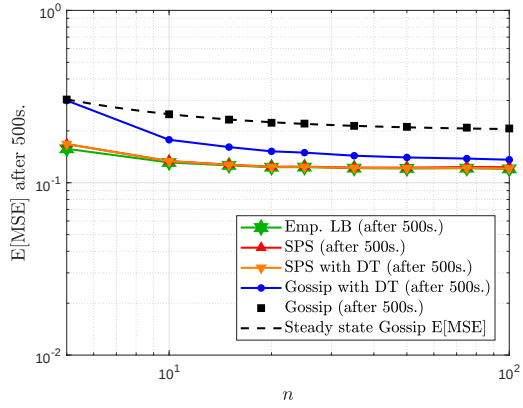
These figures show that the symmetric push-sum algorithm with difference tracking (in orange) combines the advantages of both the symmetric push-sum (in red) and the gossip averaging with difference tracking (in blue) since its expected mean squared error after 500 is systematically equal to the minimum



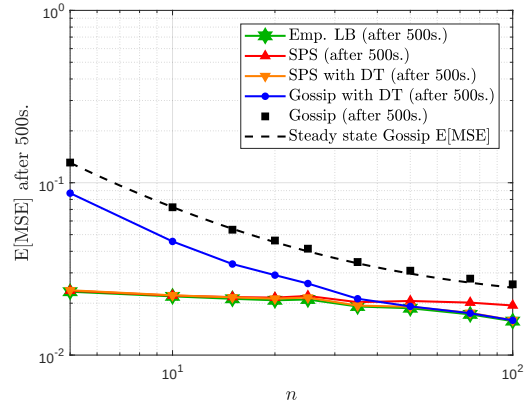
(a) Evolution with rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$, for $n = 15$.



(b) Evolution with rate ratio $\rho = \frac{\lambda_c}{\lambda_r}$, for $n = 75$.



(c) Evolution with system size n , for $\rho = \frac{\lambda_c}{\lambda_r} = 4$.



(d) Evolution with system size n , for $\rho = \frac{\lambda_c}{\lambda_r} = 50$.

Figure 6.3: Comparison of the expected mean squared error after 500 seconds of the symmetric push-sum algorithm with difference tracking (denoted SPS with DT) (using $\beta = 2p_r$ and without Assumption 5.1) with the one of other algorithms (the best ideal algorithm (denoted Emp LB.), the classical gossip algorithm, the symmetric push-sum (SPS) and the gossip with difference tracking (Gossip with DT)) for a variety of different systems (with constant $n\lambda_c = 10$). Expectations have been computed empirically with 1000 realization simulations. The theoretical expression of the steady state mean squared error of the gossip algorithm (see equation (3.5)) is also plotted (dashed black line).

between the expected mean squared error of both algorithms. After 500 seconds, the expected mean squared error of the algorithm therefore fits closely the empirical lower bound for almost all the different situations depicted and allows therefore solving almost perfectly the external averaging problem that was stated in Section 1.1.1. In average over all points of Figure 6.3, the relative difference between the expected mean squared error of the algorithm and the empirical lower bound is less than 2%. Actually, it is in general less than 1% but some extreme cases increase the average.

This is the case for (very) small systems with (very) small rate ratio ρ : see the leftmost orange point of Figures 6.3a and 6.3c that each achieves a relative difference with the empirical lower bound of about 6%. This is probably due to the decrease of the initial weight of an incoming agent which may be slightly further from the optimal decrease in situations with extremely low rate ratio (see Section 4.2). Moreover, a slight error in the initial weight of incoming agents impact more the small systems (with small n), which explains why this phenomenon is less visible in Figure 6.3b for which $n = 75$.

Another extreme case where the expected mean squared error of the algorithm is a bit further from the empirical lower bound (about 8% in relative difference) is for a large system with (very) large rate ratio ρ , such as the rightmost point of Figure 6.3b. This gap is due to the effect of the difference tracking on the expected squared bias. The incorporation rate β has been tuned in order to ensure it impact

almost not the expected squared bias, which seems true for most situations. However, having $\beta = 2p_r$ seems to impact slightly the expected squared variance for (very) large system with (very) large rate ratio (implying a very small value of p_r). For such situations, the value of β should be increased a bit in order to reach exactly the empirical lower bound.

Despite these extremes cases where the algorithm is slightly above the empirical lower bound, the symmetric push-sum algorithm with difference tracking is definitively the best algorithm that exists until now in order to solve the external averaging problem as stated in Section 1.1.1 and it greatly improves the performance of the existing gossip averaging algorithm. Moreover, the performance of the algorithm fits very closely the empirical lower bound in most situations, leaving only little room for future improvements.

Chapter 7

Weighted gossip averaging algorithm using the knowledge sets

This chapter presents the design of a gossip-based algorithm that weights the agents during communication according to the amount of knowledge they have access to. This category of algorithms uses identifiers, a growing memory and more computations than the other algorithms but it does not require any information about the system as it is the case for implementing the symmetric push-sum algorithm or the difference tracking in previous chapters. This has been explored in the hope that an agent could calculate the best ideal estimate from Chapter 2, without having to store the initial data from all the agents in its knowledge scope. It has not succeeded, but it results into an algorithm that achieves a very low expected variance, with some compromises about the expected squared bias when the system is large. This over memory and computation costs are therefore not always worth it but the development of the algorithm remains interesting and opens up other perspectives.

7.1 Assumption and motivation

Let us consider the following assumption:

Assumption 7.1. Each agent $i \in \mathcal{N}(t)$ stores and updates its knowledge set $\mathcal{K}_i$, i.e. the set of agent with which it has been in contact (directly or not) (see Definition 2.1).

Memorizing these knowledge sets requires a growing memory. Therefore, contrarily to the other designed algorithms, we do not respect the fixed bounded memory assumption and agents are supposed to use their indices identifiers which are needed for the knowledge sets. All other assumptions of the problem statement from Section 1.1.1 are still satisfied.

If it is allowed for an agent to access to a such memory, it should in general also retain the set of measurements z_k of each agent k in its knowledge set in order to be able to perform the best ideal algorithm developed in Chapter 2. However, the knowledge set always contains indices of agents, while the data z held by each agent may be highly multi-dimensional (e.g. image or video). Heavy data are of course more difficult to store in a such scale. In this situation it seems comprehensible that agent could store its knowledge set (containing only indices) but not the initial data of each agent of the set, which would allow performing the best ideal algorithm.

The next section presents a gossip-based algorithm for which the weight of each agent for a specific communication depends on their knowledge sets. It tries to recover the best possible estimates from the best ideal algorithm (see Definition 2.2), without remembering the data of the agents. As it is indicated in the title, this is not possible using a such algorithm and Section 7.3 presents a variation of the algorithm that aims to minimize the variance after each communication.

7.2 Impossibility of recovering the best estimates using a weighted gossip with knowledge sets

Our purpose is to use the knowledge sets of the agents in order to design a gossip-based algorithm of the following form:

$$\begin{aligned}
&\text{Initialization for all } i \in \mathcal{N}(t): & x_i(0) = z_i \quad \text{and} \quad \mathcal{K}_i(0) = \{i\}, \\
&\text{Interaction between agent } i \in \mathcal{N}(t) \text{ and } j \in \mathcal{N}(t): & x_i^+ = x_j^+ = \alpha(\mathcal{K}_i, \mathcal{K}_j) x_i + (1 - \alpha(\mathcal{K}_i, \mathcal{K}_j)) x_j, \\
& & \mathcal{K}_i^+ = \mathcal{K}_j^+ = \mathcal{K}_i \cup \mathcal{K}_j, \\
&\text{Estimate of agent } i \in \mathcal{N}(t): & y_i(t) = x_i(t),
\end{aligned}$$

where $\alpha \in [0, 1]$. The expression of this weighting α in the communication update (between $i \in \mathcal{N}(t)$ and $j \in \mathcal{N}(t)$) should be designed such that the new estimates (after the communication) are as close as possible to the average of the initial measurements of all the agents in the union of both knowledge sets ($\mathcal{K}_i \cup \mathcal{K}_j$), which corresponds to the best ideal estimate for these agents. The expression of α should therefore depends on both knowledge sets of the communicating agents ($\mathcal{K}_i$ and $\mathcal{K}_j$) but the notation of this dependence is omitted for the following reasonings.

To find the best communication update possible, let us analyze the situation where 2 agents $i \in \mathcal{N}(t)$ and $j \in \mathcal{N}(t)$ exchange their respective information. Let us suppose that i and j each holds (a good approximation of) the average of all the initial measurements of the agents contained in their knowledge set

$$x_i = \frac{1}{|\mathcal{K}_i|} \sum_{k \in \mathcal{K}_i} z_k, \quad \text{and} \quad x_j = \frac{1}{|\mathcal{K}_j|} \sum_{k \in \mathcal{K}_j} z_k. \quad (7.1)$$

This is case at the beginning since $x_i(0) = z_i$ and $\mathcal{K}_i(0) = \{i\}$ for any agent $i \in \mathcal{N}(t)$. Let us see if there exists a choice of weighted communication such that it remains true (or almost true) after a communication, in other words, we would like to update x_i and x_j such that

$$x_i^+ = x_j^+ = \alpha x_i + (1 - \alpha) x_j = \frac{1}{|\mathcal{K}_i \cup \mathcal{K}_j|} \sum_{k \in \mathcal{K}_i \cup \mathcal{K}_j} z_k$$

where $\alpha \in [0, 1]$. In order to ensure that this equation is verified after the communication, the variance of x_i^+ and x_j^+ should be equal to

$$\text{Var} \left(\frac{1}{|\mathcal{K}_i \cup \mathcal{K}_j|} \sum_{k \in \mathcal{K}_i \cup \mathcal{K}_j} z_k \right) = \frac{\sigma^2}{|\mathcal{K}_i \cup \mathcal{K}_j|}$$

We can set α such that the resulting variance of the new x_i^+ is minimized and see if it achieves a such variance.

The variance of x_i^+ can be computed as follows:

$$\begin{aligned}
\text{Var}(x_i^+) &= \text{Var}(\alpha x_i + (1 - \alpha) x_j) = \alpha^2 \text{Var}(x_i) + (1 - \alpha)^2 \text{Var}(x_j) + 2\alpha(1 - \alpha) \text{Cov}(x_i, x_j) \\
&= \alpha^2 \frac{\sigma^2}{|\mathcal{K}_i|} + (1 - \alpha)^2 \frac{\sigma^2}{|\mathcal{K}_j|} + 2\alpha(1 - \alpha) \sigma^2 \frac{|\mathcal{K}_i \cap \mathcal{K}_j|}{|\mathcal{K}_i| |\mathcal{K}_j|}
\end{aligned} \quad (7.2)$$

This variance is minimum when

$$\alpha = \frac{|\mathcal{K}_i| - |\mathcal{K}_i \cap \mathcal{K}_j|}{|\mathcal{K}_i| + |\mathcal{K}_j| - 2|\mathcal{K}_i \cap \mathcal{K}_j|} = \frac{|\tilde{\mathcal{K}}_{i,j}|}{|\tilde{\mathcal{K}}_{i,j}| + |\tilde{\mathcal{K}}_{j,i}|}, \quad (7.3)$$

where we define $\tilde{\mathcal{K}}_{i,j}$ as the set of agent indices that are known at i but not at j , i.e : $\tilde{\mathcal{K}}_{i,j} = \mathcal{K}_i \setminus (\mathcal{K}_i \cap \mathcal{K}_j)$. We have well that $\alpha \in [0, 1]$. This expression of α can be computed easily by the communicating agents i and j since they share their knowledge set with each other and can then compute the size and the intersection of these knowledge sets.

This α given by equation (7.3), that minimizes the resulting variance after the communication, does unfortunately not reach the variance of the best ideal estimate $\frac{\sigma^2}{|\mathcal{K}_i \cup \mathcal{K}_j|}$ in general (except for specific cases of $\mathcal{K}_i$ and $\mathcal{K}_j$), as it should be the case in order to keep initial hypothesis (7.1) true after a communication. This shows that this hypothesis of achieving exactly the best ideal estimate is too strong. Next section states a lighter design hypothesis that remains easily true after a communication.

7.3 Algorithm using the knowledge sets and the knowledge weights

To tackle the problem encountered in previous section, one could change the initial assumption related to the x estimates of the agents (see equation (7.1)) and only suppose that the agents hold an estimate that corresponds to a weighted average of the measurements of all the agents in their knowledge set:

$$x_i(t) = \sum_{k \in \mathcal{K}_i} \phi_{ik}(t) z_k \quad \text{for all } i \in \mathcal{N}(t), \quad (7.4)$$

where $\phi_{ik}(t)$ denotes the nonnegative weight that agent i attributes to the measurement of agent k in its estimates at time t . These $\phi_{ik}(t)$ are called the knowledge weight. We should have that $\sum_{k \in \mathcal{K}_i} \phi_{ik}(t) = 1$ for all $i \in \mathcal{N}(t)$.

Initially, this assumption is obviously respected with only one nonzero knowledge weight: $\phi_{ii}(0) = 1$ for any agent $i \in \mathcal{N}(t)$ since $x_i(0) = z_i$ and $\mathcal{K}_i(0) = \{i\}$ for any agent $i \in \mathcal{N}(t)$. For the following, the notation of the time dependence of either the estimates x or the knowledge weights ϕ is omitted.

The tuning of the communication update of the algorithm should allow this hypothesis (equation (7.4)) to remain true after any communication, while enforcing the estimates of agents to get close to the best ideal estimate, meaning that all knowledge weights ϕ_{ki} of a given agent i should be close to each other. We have seen in previous section that imposing these knowledge weights to be always equal to each other (which amounts to imposing an exact average of the measurements) is not always possible.

For designing a such communication update, we suppose that each agent $i \in \mathcal{N}(t)$ memorizes the knowledge weight ϕ_{ik} it attributes to each agent k in its knowledge set. These knowledge weights are updated during the communication to ensure that this hypothesis about the x estimates (equation (7.4)) remains valid. They can also be used in order to design a weighting α of the agents involved in a communication that minimizes the resulting variance or any other quantities right after the communication. This can be seen as a greedy optimization of some general performance of the system.

The communication scheme between $i \in \mathcal{N}(t)$ and $j \in \mathcal{N}(t)$ can be written as follows:

$$x_i^+ = x_j^+ = \alpha x_i + (1 - \alpha) x_j \quad \left(\text{which is equal to } \sum_{k \in \mathcal{K}_i \cup \mathcal{K}_j} \phi_{ik}^+ z_k \right), \quad (7.5)$$

$$\phi_{ik}^+ = \phi_{jk}^+ = \alpha \phi_{ik} \quad \text{for } k \in \mathcal{K}_i \setminus \mathcal{K}_j, \quad (7.6)$$

$$\phi_{ik}^+ = \phi_{jk}^+ = (1 - \alpha) \phi_{jk} \quad \text{for } k \in \mathcal{K}_j \setminus \mathcal{K}_i, \quad (7.7)$$

$$\phi_{ik}^+ = \phi_{jk}^+ = \alpha \phi_{ik} + (1 - \alpha) \phi_{jk} \quad \text{for } k \in \mathcal{K}_i \cap \mathcal{K}_j, \quad (7.8)$$

$$\mathcal{K}_i^+ = \mathcal{K}_j^+ = \mathcal{K}_i \cup \mathcal{K}_j, \quad (7.9)$$

where $\alpha \in [0, 1]$. It may depends on the knowledge sets $\mathcal{K}_i$, $\mathcal{K}_j$ and on the knowledge weight ϕ_{ik} and ϕ_{jk} (for any k respectively in $\mathcal{K}_i$ or $\mathcal{K}_j$) of the communicating agents $i \in \mathcal{N}(t)$ and $j \in \mathcal{N}(t)$. This communication update, by definition, keeps the initial hypothesis from equation (7.4) true (with $\sum_{k \in \mathcal{K}_i} \phi_{ik}(t) = 1$ for all $i \in \mathcal{N}(t)$).

7.3.1 Minimizing the resulting variance

This weighting α can be set up in order to minimize the resulting variance of x_i^+ and x_j^+ right after a communication between agents $i \in \mathcal{N}(t)$ and $j \in \mathcal{N}(t)$:

$$\begin{aligned} \min_{\alpha} \quad & \text{Var}(x_i^+) \quad \text{given the values of } x_i, \phi_{ik}, x_j \text{ and } \phi_{jk}. \\ \text{such that} \quad & \alpha \in [0, 1]. \end{aligned} \quad (7.10)$$

It corresponds to a kind of greedy optimization of the variance in the entire system.

The variance after a communication between agents $i \in \mathcal{N}(t)$ and $j \in \mathcal{N}(t)$ can be written using equation (7.2). Supposing that the value of $\text{Var}(x_i)$, $\text{Var}(x_j)$ and $\text{Cov}(x_i, x_j)$ (before the communication) can be computed using the knowledge weights ϕ_{ik} and ϕ_{jk} of each communicating agents (see below for how), the optimal solution of problem (7.10) can be obtained by considering the solution of the relaxed problem, without constraint $\alpha \in [0, 1]$. This relaxed solution is computed by imposing the derivative of $\text{Var}(x_i^+)$ with respect to α to 0: $\frac{\partial \text{Var}(x_i^+)}{\partial \alpha} = 0$, which gives:

$$\alpha = \frac{\text{Var}(x_j) - \text{Cov}(x_i, x_j)}{\text{Var}(x_i) + \text{Var}(x_j) - 2\text{Cov}(x_i, x_j)}. \quad (7.11)$$

We suppose that the denominator is nonzero: $\text{Var}(x_i) + \text{Var}(x_j) - 2\text{Cov}(x_i, x_j) \neq 0$. It will be explained below what does it mean if the denominator is zero, depending on the expressions of $\text{Var}(x_i)$, $\text{Var}(x_j)$ and $\text{Cov}(x_i, x_j)$ that are found. This relaxed solution for α is not always between 0 and 1 (as required by problem 7.10), but since the objective function is a univariate quadratic function in α , if the relaxed solution (given by equation (7.11)) is not in the admissible domain (i.e. between 0 and 1), we can simply project it onto the domain to obtain a feasible optimal solution. In other words, if equation (7.11) gives a value of α larger than one (respectively lower than zero), it is projected on one (respectively zero).

Let us compute $\text{Var}(x_i)$, $\text{Var}(x_j)$ and $\text{Cov}(x_i, x_j)$ in terms of the knowledge weights ϕ_{ik} and ϕ_{jk} such that the communicating agents can compute the optimal value of α easily. Using our new hypothesis from equation (7.4), we have that

$$\text{Var}(x_i) = \text{Var}\left(\sum_{k \in \mathcal{K}_i} \phi_{ik} z_k\right) = \sum_{k \in \mathcal{K}_i} \phi_{ik}^2 \text{Var}(z_k) = \sum_{k \in \mathcal{K}_i} \phi_{ik}^2 \sigma^2. \quad (7.12)$$

We also used the fact that the measurements z_k are independent and identically distributed (i.i.d.). The expression of $\text{Var}(y_j)$ can be obtained similarly. For its part, the covariance can be expressed as:

$$\text{Cov}(x_i, x_j) = \text{Cov}\left(\sum_{k \in \mathcal{K}_i} \phi_{ik} z_k, \sum_{k \in \mathcal{K}_j} \phi_{jk} z_k\right) = \sum_{k \in \mathcal{K}_i} \sum_{l \in \mathcal{K}_i} \phi_{ik} \phi_{jl} \text{Cov}(z_k, z_l) = \sum_{k \in \mathcal{K}_i \cap \mathcal{K}_j} \phi_{ik} \phi_{jk} \sigma^2, \quad (7.13)$$

since for i.i.d. measurements z , we have

$$\text{Cov}(z_k, z_l) = \begin{cases} \text{Var}(z_k) = \sigma^2 & \text{for } k = l, \\ 0 & \text{otherwise.} \end{cases}$$

Having an optimal value of α (from equation (7.11)) with a zero denominator therefore means that the knowledge sets and the knowledge weights of both agents i and j are identical (for nonzero σ):

$$\begin{aligned} \text{Var}(x_i) + \text{Var}(x_j) - 2\text{Cov}(x_i, x_j) &= \sum_{k \in \mathcal{K}_i} \phi_{ik}^2 \sigma^2 + \sum_{k \in \mathcal{K}_j} \phi_{jk}^2 \sigma^2 - 2 \sum_{k \in \mathcal{K}_i \cap \mathcal{K}_j} \phi_{ik} \phi_{jk} \sigma^2 = 0 \\ \Leftrightarrow \sum_{k \in \mathcal{K}_i \cap \mathcal{K}_j} (\phi_{ik} - \phi_{jk})^2 + \sum_{k \in \mathcal{K}_i \setminus \{\mathcal{K}_i \cap \mathcal{K}_j\}} \phi_{ik}^2 + \sum_{k \in \mathcal{K}_j \setminus \{\mathcal{K}_i \cap \mathcal{K}_j\}} \phi_{jk}^2 &= 0 \\ \Leftrightarrow \mathcal{K}_i \cap \mathcal{K}_j = \mathcal{K}_i = \mathcal{K}_j \quad \text{and} \quad (\phi_{ik} - \phi_{jk})^2 = 0 \quad \text{for all } k \in \mathcal{K}_i \cap \mathcal{K}_j. \end{aligned}$$

The last equivalence is true because we have that $\sum_{k \in \mathcal{K}_i} \phi_{ik}(t) = 1$ for all $i \in \mathcal{N}(t)$. In that situation, any value α can be chosen. In fact, the communication is useless.

Knowing relations (7.12) and (7.13) allows the agents to use the α weighting from equation (7.11), with potential projections to have $\alpha \in [0, 1]$, in the communications scheme described by equations (7.5) to (7.9) in order to minimize the resulting variance. This is called the knowledge weighted gossip algorithm in the following analysis. Implementing this algorithm requires each agent i to memorize an

index and a weight for each agent k in its knowledge sets. This communication updates are in general time-consuming (in comparison with other algorithms such as the gossip averaging) since it requires to calculate the union of the knowledge sets of both communicating agents (equation (7.9)) and to update all their knowledge weights (ϕ) (equations (7.6), (7.7) and (7.8)).

Let us see if these memory and time costs are worth it by analyzing briefly the performance achieved by the knowledge weighted gossip algorithm.

Performance of the algorithm

Figure 7.1 shows the time evolution of expected variance and squared bias of the knowledge weighted gossip (in cyan) in comparison with the classical gossip algorithm (in black) and the best ideal algorithm (in green) for four different system with small and large size and rate ratio. As expected, the expected variance of the algorithm totally and directly fits the one of the best ideal algorithm meaning that our greedy minimization of the variance after each communication seems well leading to a low variance in the entire system. In terms of expected variance, this algorithm therefore beats the classical gossip averaging algorithm and even the efficient symmetric push-sum with difference tracking which does not always fit directly the expected variance of the best ideal algorithm as explained and illustrated in Section 6.2.

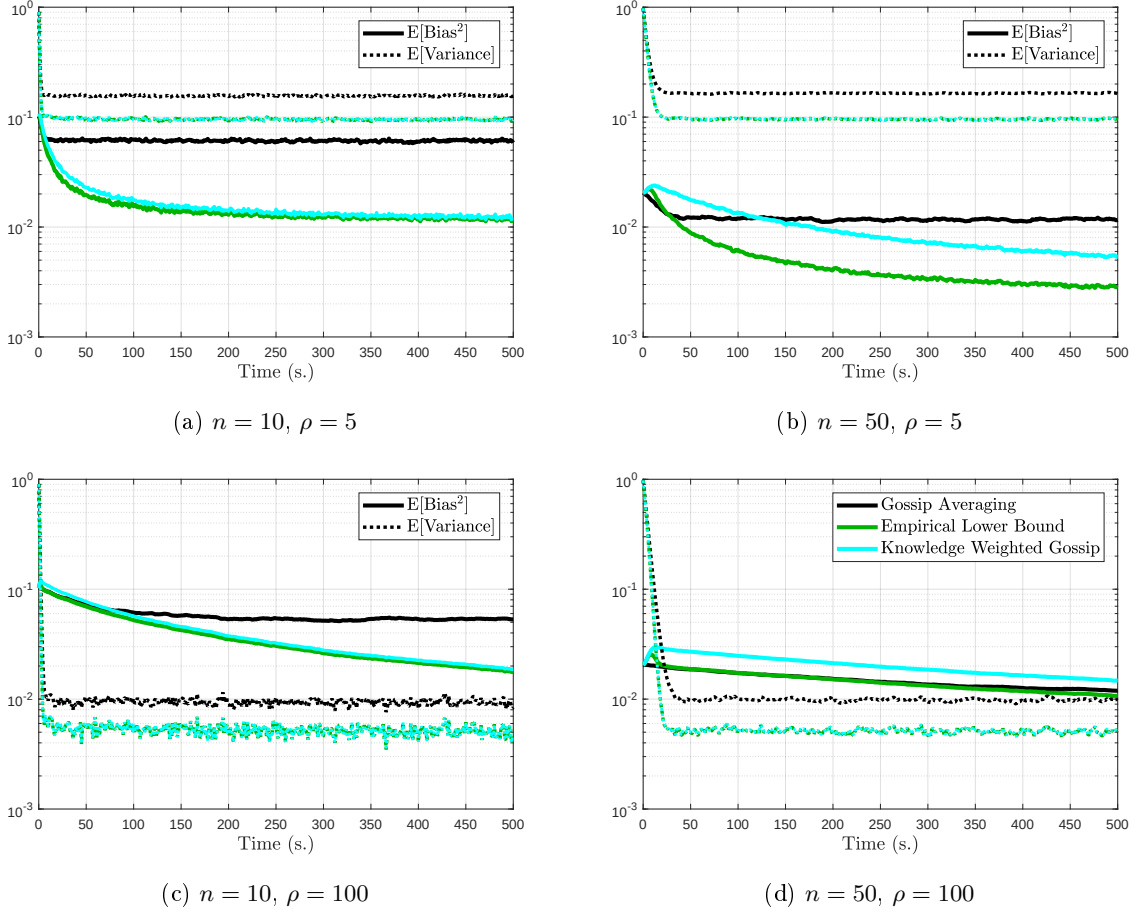


Figure 7.1: Comparison between the expected squared bias (solid line) and the expected variance (dotted line) time evolution of the weighted gossip using the knowledge sets (in cyan), the gossip averaging algorithm (in black) and the best ideal algorithm (in green) (that provides the empirical lower bound on $\mathbb{E}[\text{MSE}(t)]$) for four different system configurations (n and $\rho = \frac{\lambda_c}{\lambda_r}$) with scaling $n\lambda_c = 10$. Expectations have been computed empirically with 5000 realization simulations.

As shown in Figure 7.1, the expected squared bias of the algorithm generally decreases well when the system is small (see Figures 7.1a and 7.1c) and is quite close to the one of the best ideal algorithm. But when the size of the system increases, the expected squared bias of the algorithm starts increasing during a while before decreasing slowly, as shown in Figures 7.1b and 7.1d. It can even be worse than the one of the gossip averaging for more or less long time depending on the configuration of the system (it seems longer when the rate ratio is large). In Figure 7.1d for example, the expected squared bias of the algorithm remains above the one of the gossip (at least) during the 500 first seconds.

The resulting expected mean squared errors achieved by the algorithm in these four situations are shown in Figure 7.2. For small systems its expected mean squared errors is very good and approach closely the empirical lower bound (see Figures 7.2a and 7.2c). It is also the case for small rate ratio ρ such as on Figure 7.2b where the expected mean squared error is dominated by the expected variance. However, for large system with large rate ratio (as in Figure 7.2d), the algorithm performs poorly, and is even poorer than the gossip averaging algorithm for a while (at least the first 200 seconds in Figure 7.2d).

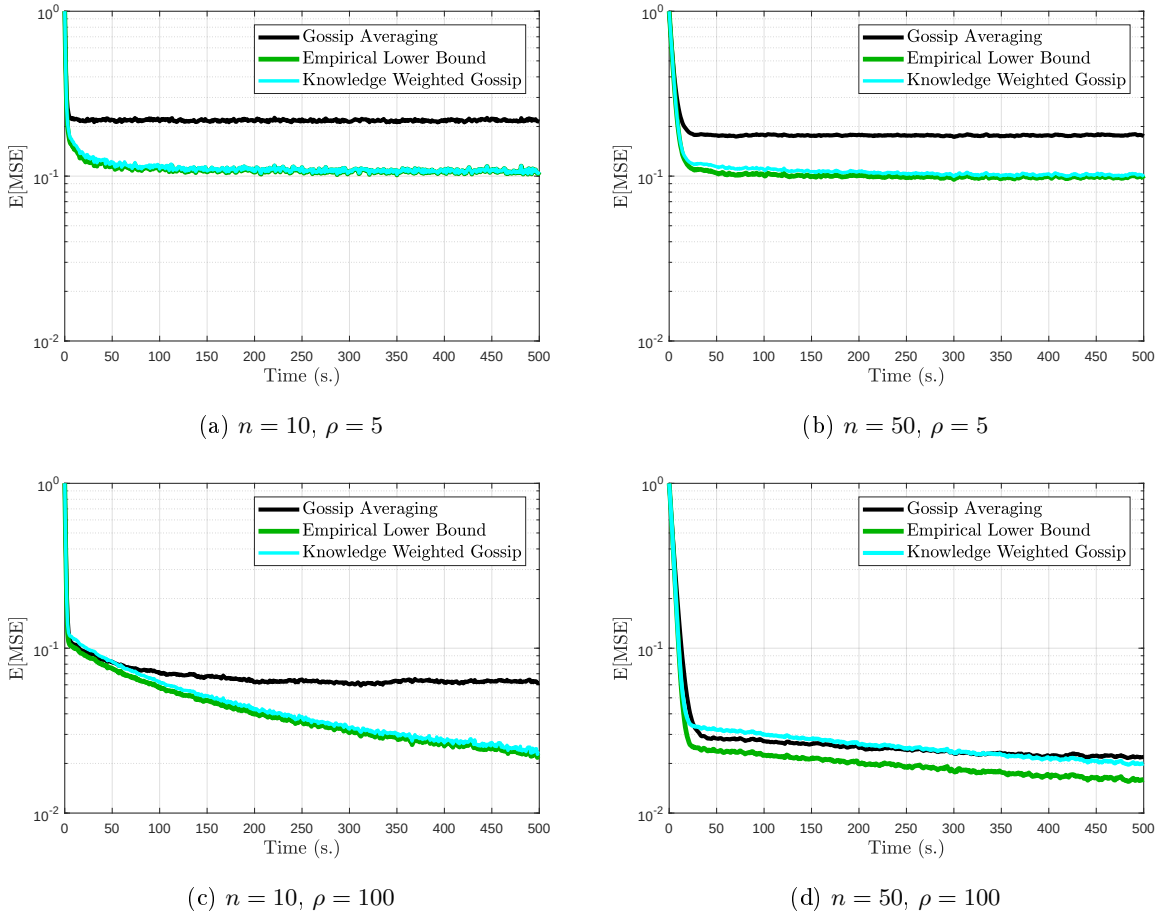


Figure 7.2: Comparison between the expected mean squared error time evolution ($\mathbb{E}[\text{MSE}(t)]$) of the weighted gossip using the knowledge sets, the gossip averaging algorithm and the best ideal algorithm (that provides the empirical lower bound on $\mathbb{E}[\text{MSE}(t)]$) for four different system configurations (n and $\rho = \frac{\lambda_c}{\lambda_r}$) with scaling $n\lambda_c = 10$. Expectations have been computed empirically with 5000 realization simulations.

As it has been observed, this algorithm is therefore attractive only for situations where the expected variance has a great impact on the expected mean squared error in comparison with the expected squared bias (i.e. for small ρ) or where the expected squared bias is not (too) badly impacted by the algorithm (i.e. for small system size n). These situations also correspond to the ones where the chance of observing

a *restart* in the system are not very low. Fortunately, on the contrary of the other algorithm (in their implementable version), this algorithm reacts as well to a *restart* in the system than in any other situation. Indeed, to work properly, it only requires the agents to know their current knowledge sets and knowledge weights, whatever they are.

7.3.2 Perspectives

This knowledge weighted gossip algorithm presents a memory and computations overcost, without performing perfectly in most situations (due to a not so good expected squared bias for larger systems). Its advantages in comparison with the symmetric push-sum algorithm with difference tracking is its expected variance which is exactly the same as the one of the best ideal algorithm from the very beginning in any situation, the management of the potential *restart* of the system and the fact that agents does not need any prior information about the system, such as its size n or its rate ratio ρ . These overcosts are therefore only worth it if we care about the benefits the algorithm provides, which do not include having very good performance in general.

However, this path is not closed since the weighting α used in the communication scheme (equations (7.5) to (7.9)) can be design to optimize other quantities than the variance (depending on the knowledge sets and the knowledge weights), which could maybe improve the performance of the algorithm. For example, it could minimize the difference between the resulting knowledge weights after the communication, in order to have an estimate that is closer to the exact average of the measurements of the agents in the resulting knowledge set (which is the best ideal estimate). This could be another future work.

Conclusion

The multi-agent systems and the decentralized paradigm they represent have become more and more prominent due to their many applications in various fields and relevant due to the growing size of the studied systems that makes centralized solution impossible. Indeed, it turns out that the decentralized approaches provide numerous advantages, starting with the scalability of the system. It therefore allows working with large systems thanks to the distribution of the information among the agents. Moreover, the multi-agent systems are considered to be more robust since they do not rely on any central unit, on which the success of an operation would depend, and each agent keep its own approximation of the solution.

Most of the existing work concerning multi-agent system consider a fixed composition of agents (i.e. closed system). However, considering such systems to be open to arrivals and departures of agents has shown to be a major and necessary improvement of the decentralized methods. Indeed, having larger systems implies a higher probability of observing arrivals or departures of agents. Furthermore, some systems may be inherently open. In such open multi-agent systems, many problems have not yet been addressed.

For example, this is the case of aggregating different data held by different agents in order to estimate, in a decentralized way, the true distribution mean of these measurements. This problem is called the external averaging problem and is properly stated in Chapter 1. It is the focus of this master thesis. The main contributions are the development of performance limitations and of new efficient algorithms for solving this problem in open multi-agent systems.

Throughout this work, the open nature of the system is supposed to manifest itself only through replacements of agents, meaning that each departure is immediately followed by an arrival. This assumption allows the system to keep a constant size. These replacements are supposed to happen according to a Poisson clock. Extension of the results to separated departures and arrivals should constitute one of the important future works.

The open character of the system allows it to continuously receive new agents with new data, which should make possible the improvement of the estimates of the agents in the system. Making these improvements from the possible to the real is challenging because the system should incorporate information from new agents without forgetting information from agents having left in the past, without using a growing memory and without being affected too much by noise.

With the help of simulations and based on existing results, new algorithms have been designed to answer these challenges.

Chapter 2 also plays a major role in the design of new algorithms since it establishes limitations about the performance that any realistic algorithm can achieve. The performance of an algorithm is measured using the expected mean squared error metric, that can be decomposed into expected squared bias expected variance. This chapter presents an ideal algorithm that achieves the best expected mean squared error among all the algorithm that generate linear and unbiased estimates. Its performance provides therefore a lower bound on the performance of any such algorithm and can be computed empirically. Its performance has also been lower bounded theoretically but the two resulting bounds are not tight at all.

Chapter 3 presents an existing well-known algorithm which is the gossip averaging algorithm. Its analysis in open system is explained and allows giving theoretical guarantees about the performance of this algorithm for solving the external averaging problem. It turns out that the algorithm does not

perform well. Two main issues have been explained: firstly, the information of the new arriving agents has too much importance into the system, which leads the system to progressively forget the information of the older agents and to achieve a large expected squared bias. Secondly, each arrival disturbs largely the estimates of the agents which leads to a large expected variance, especially when they are frequent. Two new algorithms are designed in Chapters 4 and 5 with the intention of solving respectively the first and the second issue of this gossip averaging algorithm.

Chapter 4 presents the symmetric push-sum algorithm that allows giving a weight to each measurement of the agents. Setting correctly the initial weight of each agent, especially the new arriving one is a key aspect for resolving the first issue of the gossip averaging algorithm and achieve a good expected squared bias. Using the assumption that the replacements are extremely infrequent, we have established an expression of these initial weights for each arriving agent that guarantees the agents to converge toward the external average of the system between each replacement. As expected, these initial weights are decreasing at each new replacement. It figures that this weight expression seems to be nearly optimal in any situation, even when the replacements are very frequent. These decreasing initial weights allow achieving an expected squared bias that totally fits the one of the best ideal algorithm in any situation. It also improves the expected variance by reducing progressively the disturbance of each new agent in the system but there remains a progression margin, especially concerning the speed at which it decreases, which may be slow in some situations. Theoretical analysis of this algorithm has been undertaken, without leading to any guarantees on its performance. Implementing this algorithm in practice requires the agents to know the size of the system and the frequency of the replacements (in comparison with the communication frequency).

Chapter 5 presents a variation of the gossip averaging algorithm allowing to reduce rapidly the expected variance of a system. It is based on the idea of difference tracking which spreads out the arrival of new information: a new arriving agent initially pretends to have the same estimate as another agent already in the system and keeps track of the difference with its true initial estimate in order to incorporate it progressively into the system. The rate at which the difference variables of the agents are incorporated into their estimates influences largely the expected variance of the system and may, when the rate is too low, influence the expected squared bias of the system. A theoretical analysis of the algorithm has been carried out under the assumption that each arriving agent directly communicates. This allows having guarantees about the performance of the algorithm under this assumption which gives better steady state performance guarantees than the gossip averaging (under the same assumption). This theoretical analysis also helps to tune correctly the value of the incorporation rate. A well tuned fixed value of this rate allows improving quickly the expected variance without modifying the expected squared bias. Therefore, it does not often reach the empirical lower bound for the expected mean squared error but it does in the only situations where the symmetric push-sum does not. In this regard, both algorithms complement each other and it seems natural to combine them.

The combination of both algorithms in one has been presented in Chapter 6. The symmetric push-sum with difference tracking benefits from the advantages of both algorithms and is therefore the best algorithm that has been designed. It resolves both issues of the gossip averaging algorithm and it fits closely the performance limitations in any situation (within 1% for most situations). Improvements are still possible by setting up the detection and management of potential natural restarts of the system, which are generally extremely rare.

Finally, Chapter 7 starts exploring another category of algorithms where agents can use a growing memory and identifiers in order to choose wisely a weighting of the two communicating agents, depending on the state of their knowledge that is aggregated into their estimate. These algorithms make sense when the data held by each agent is particularly heavy. This chapter results into an algorithm that reduces particularly well the expected variance, with some compromise on the expected squared bias that may be badly impacted, especially for large system. Other perspectives are still possible with this approach and can be explored in the future.

This master thesis gives a better understanding of this external averaging problem as stated in Chapter 1. The new algorithms that have been designed really improve the performance of the gossip

averaging algorithm. The best of those new algorithms is surely the symmetric push-sum algorithm with difference tracking which succeeds to solve both issues of the gossip averaging algorithm at the same time and to reach very closely the performance limitations. Further works are required in order to deduce general performance guarantees about this algorithm, and its simpler form the symmetric push-sum. But the algorithm can already been used in practice for estimating some value of interest in a decentralized way in multi-agent systems subject to replacements of agents, for which no efficient algorithm did not exist before. Future work should adapt the results and the algorithm to the case where the arrivals and departures of the agents can be separated. Thanks to this master thesis, I think I have made my contribution to the study of the open multi-agent systems and I have presented efficient solutions to a new problem it raises: how to aggregate efficiently information of arriving and leaving agents. Many other problems are still waiting to be considered in order to drive the decentralized methods further, from a research point of view but also from the point of view of real-life applications such as interacting autonomous vehicles, sensor networks (e.g. in smart cities) or many other things. Expanding our general knowledge and the scope of possibilities it provides is very important for me, but we should not forget to think again about the utility and the soundness of each application before implementing it in the real world.

Bibliography

- [A02] Deshmukh A. *Knowledge Management, Organizational Intelligence and Learning, and Complexity*. Vol. I - Distributed Artificial Intelligence. 2002.
- [AHH17] Mahmoud Abdelrahim, Julien M. Hendrickx, and W. P. M. H. Heemels. “MAX-consensus in open multi-agent systems with gossip interactions”. In: *CoRR* abs/1709.05793 (2017).
- [Ava19] Anand Avati. *Bias-Variance Analysis:Theory and Practice*. <http://cs229.stanford.edu/summer2019/BiasVarianceAnalysis.pdf>. Stanford University - Machine Learning. 2019.
- [Blo+05] V. D. Blondel, J. M. Hendrickx, A. Olshevsky, and J. N. Tsitsiklis. “Convergence in Multiagent Coordination, Consensus, and Flocking”. In: *Proceedings of the 44th IEEE Conference on Decision and Control*. 2005, pp. 2996–3000.
- [Boy+06] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. “Randomized Gossip Algorithms”. In: *IEEE/ACM Trans. Netw.* 14.SI (June 2006), pp. 2508–2530.
- [Bra18] Baptiste Brasseur. “Algorithms in open multi-agent systems : gossiping with random replacements”. Anglais. MA thesis. UCL - Ecole polytechnique de Louvain, 2018. URL: <http://hdl.handle.net/2078.1/thesis:16470>.
- [Cao+13] Y. Cao, W. Yu, W. Ren, and G. Chen. “An Overview of Recent Progress in the Study of Distributed Multi-Agent Coordination”. In: *IEEE Transactions on Industrial Informatics* 9.1 (2013), pp. 427–438.
- [CHW11] Rutger Claes, Tom Holvoet, and Danny Weyns. “A Decentralized Approach for Anticipatory Vehicle Routing Using Delegate Multiagent Systems”. In: *IEEE Transactions on Intelligent Transportation Systems* 12 (June 2011), pp. 364–373. DOI: 10.1109/TITS.2011.2105867.
- [CI12] Kai Cai and Hideaki Ishii. “Average consensus on general strongly connected digraphs”. In: *Automatica* 48.11 (2012), pp. 2750–2761. URL: <http://www.sciencedirect.com/science/article/pii/S0005109812004049>.
- [CI14] K. Cai and H. Ishii. “Average Consensus on Arbitrary Strongly Connected Digraphs With Time-Varying Topologies”. In: *IEEE Transactions on Automatic Control* 59.4 (2014), pp. 1066–1071.
- [DBB14] M. Dorigo, M. Birattari, and M. Brambilla. “Swarm robotics”. In: *Scholarpedia* 9.1 (2014). revision #138643, p. 1463. DOI: 10.4249/scholarpedia.1463.
- [DCX12] Y. Duan, B.X. Cui, and X.H. Xu. “A multi-agent reinforcement learning approach to robot soccer”. In: *Artif Intell Rev* 38 (2012), pp. 193–211. DOI: <https://doi.org/10.1007/s10462-011-9244-8>.
- [Der05] Konstantinos G. Derpanis. *Jensen’s Inequality*. http://www.cs.yorku.ca/~kosta/CompVis_Notes/jensen.pdf. York University. 2005.
- [DKJ18] Ali Dorri, Salil Kanhere, and Raja Jurdak. “Multi-Agent Systems: A survey”. In: *IEEE Access* (Apr. 2018), pp. 1–1. DOI: 10.1109/ACCESS.2018.2831228.
- [FF18] M. Franceschelli and P. Frasca. “Proportional Dynamic Consensus in Open Multi-Agent Systems”. In: *2018 IEEE Conference on Decision and Control (CDC)*. 2018, pp. 900–905.
- [GH19] B. Gerencsér and J. M. Hendrickx. “Push-Sum With Transmission Failures”. In: *IEEE Transactions on Automatic Control* 64.3 (2019), pp. 1019–1033.

- [GS06] C. Grinstead and J. Snell. *Introduction to probability, 2n edition*. Ed. by American Mathematical Society. 2006.
- [GY13] Federica Garin and Ye Yuan. *Distributed privacy-preserving network size computation: A system-identification based method*. 2013.
- [Hei] Jeffery Hein. *Wald’s Identity*. <https://math.dartmouth.edu/~pw/math100w13/hein.pdf>. Dartmouth College.
- [Hen14] J. M. Hendrickx. “Views in a Graph: To Which Depth Must Equality Be Checked?” In: *IEEE Transactions on Parallel and Distributed Systems* 25.7 (2014), pp. 1907–1912.
- [HM05] Yuko Hatano and Mehran Mesbahi. “Agreement Over Random Networks”. In: *Automatic Control, IEEE Transactions on* 50 (Dec. 2005), pp. 1867–1872. DOI: 10.1109/TAC.2005.858670.
- [HM16] J. M. Hendrickx and S. Martin. “Open multi-agent systems: Gossiping with deterministic arrivals and departures”. In: *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. 2016, pp. 1094–1101.
- [HM17] Julien M. Hendrickx and Samuel Martin. *Open Multi-Agent Systems: Gossiping with Random Arrivals and Departures*. Sept. 2017.
- [KDG03] D. Kempe, A. Dobra, and J. Gehrke. “Gossip-based computation of aggregate information”. In: *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings*. 2003, pp. 482–491.
- [Kia+19] S. S. Kia, B. Van Scoy, J. Cortes, R. A. Freeman, K. M. Lynch, and S. Martinez. “Tutorial on Dynamic Average Consensus: The Problem, Its Applications, and the Algorithms”. In: *IEEE Control Systems Magazine* 39.3 (2019), pp. 40–72.
- [KKL15] Marcin Kardas, Mirosław Kutyłowski, and Jakub Lemiesz. “On Distributed Cardinality Estimation: Random Arcs Recycled”. In: Jan. 2015.
- [Leo13] Naomi Ehrich Leonard. “Multi-Agent System Dynamics: Bifurcation and Behavior of Animal Groups”. In: *IFAC Proceedings Volumes* 46.23 (2013). 9th IFAC Symposium on Nonlinear Control Systems, pp. 307–317. DOI: <https://doi.org/10.3182/20130904-3-FR-2041.00211>. URL: <http://www.sciencedirect.com/science/article/pii/S1474667016316779>.
- [Luc+15] R. Lucchese, D. Varagnolo, J. Delvenne, and J. Hendrickx. “Network cardinality estimation using max consensus: The case of Bernoulli trials”. In: *2015 54th IEEE Conference on Decision and Control (CDC)*. 2015, pp. 895–901.
- [MH20] Charles Monnuyer de Galland de Carnières and Julien M. Hendrickx. *Fundamental Performance Limitations for Average Consensus in Open Multi-Agent Systems*. 2020.
- [Mon18] Charles Monnuyer de Galland de Carnières. “Efficiency bounds in computations on open multi-agent systems”. Anglais. MA thesis. UCL - Ecole polytechnique de Louvain, 2018. URL: <http://hdl.handle.net/2078.1/thesis:14719>.
- [NOR18] A. Nedić, A. Olshevsky, and M. G. Rabbat. “Network Topology and Communication-Computation Tradeoffs in Decentralized Optimization”. In: *Proceedings of the IEEE* 106.5 (2018), pp. 953–976.
- [Oli18] Alexandre Olikier. “Open multi-agent systems with fixed-sized and possibly not complete topologies”. Anglais. MA thesis. UCL - Ecole polytechnique de Louvain, 2018. URL: <http://hdl.handle.net/2078.1/thesis:14693>.
- [Ota06] Jun Ota. “Multi-agent robot systems as distributed autonomous systems”. In: *Advanced Engineering Informatics* 20.1 (2006), pp. 59–70. DOI: <https://doi.org/10.1016/j.aei.2005.06.002>. URL: <http://www.sciencedirect.com/science/article/pii/S1474034605000509>.
- [Rao11] Anup Rao. *CSE599s: Extremal Combinatorics - Lecture 2 Jensen’s Inequality and Probabilistic Magic*. <https://homes.cs.washington.edu/~anuprao/pubs/CSE599sExtremal/lecture2.pdf>. University of Washington. 2011.

- [RN04] Michael Rabbat and Robert Nowak. “Distributed Optimization in Sensor Networks”. In: *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*. IPSN '04. Berkeley, California, USA: Association for Computing Machinery, 2004, pp. 20–27. ISBN: 1581138466. URL: <https://doi.org/10.1145/984622.984626>.
- [Sha+12] I. Shames, T. Charalambous, C. N. Hadjicostis, and M. Johansson. “Distributed network size estimation and average degree estimation and control in networks isomorphic to directed graphs”. In: *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. 2012, pp. 1885–1892.
- [Sie] Kyle Siegrist. *Point Estimation - Best Unbiased Estimators*. <https://www.randomservices.org/random/point/Unbiased.html>. University of Alabama in Huntsville.
- [SR12] Wu Shaochuan and Michael G. Rabbat. *Broadcast Gossip Algorithms for Consensus on Strongly Connected Digraphs*. 2012.
- [SSF19] Z. A. Z. Sanai Dashti, C. Seatzu, and M. Franceschelli. “Dynamic Consensus on the Median Value in Open Multi-Agent Systems”. In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. 2019, pp. 3691–3697.
- [TJ08] A. Tahbaz-Salehi and A. Jadbabaie. “A Necessary and Sufficient Condition for Consensus Over Random Networks”. In: *IEEE Transactions on Automatic Control* 53.3 (2008), pp. 791–795.
- [Van+16] Corentin Vande Kerckhove, Samuel Martin, Pascal Gend, Peter J. Rentfrow, Julien M. Hendrickx, and Vincent D. Blondel. “Modelling Influence and Opinion Evolution in Online Collective Behaviour”. In: *PLOS ONE* 11.6 (June 2016), pp. 1–25. URL: <https://doi.org/10.1371/journal.pone.0157685>.
- [VMH18] Vineeth Varma, Irinel-Constantin Morarescu, and Yezekael Hayel. “Continuous time opinion dynamics of agents with multi-leveled opinions and binary actions”. In: *IEEE Conference on Computer Communications, INFOCOM 2018*. Honolulu, United States, Apr. 2018. URL: <https://hal.archives-ouvertes.fr/hal-01756831>.
- [WBA05] Wei Ren, R. W. Beard, and E. M. Atkins. “A survey of consensus problems in multi-agent coordination”. In: *Proceedings of the 2005, American Control Conference, 2005*. 2005, 1859–1864 vol. 3.
- [Woo09] Michael Wooldridge. *An Introduction to MultiAgent Systems*. 2nd. Wiley Publishing, 2009.
- [Ye+18a] Mengbin Ye, Yuzhen Qin, Alain Govaert, Brian D. O. Anderson, and Ming Cao. *An Influence Network Model to Study Discrepancies in Expressed and Private Opinions*. 2018.
- [Ye+18b] Mengbin Ye, Minh Hoang Trinh, Young-Hun Lim, Brian D. O. Anderson, and Hyo-Sung Ahn. *Continuous-time Opinion Dynamics on Multiple Interdependent Topics*. 2018.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl