Synchronization problems in dynamic multi-agent networks

Louis Penet de Monterno

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2023IPPAX116

Thèse de doctorat

Synchronization problems in dynamic multi-agent networks

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à École polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Paris, le 19 octobre 2023, par

LOUIS PENET DE MONTERNO

Composition du Jury :

Benjamin Doerr Professeur, École polytechnique, IP Paris	Président
Arnaud Casteigts Professeur, Université de Genève	Rapporteur
Sébastien Tixeuil Professeur, Sorbonne Université	Rapporteur
Karine Altisen Maître de conférences, Grenoble INP	Examinatrice
George Giakkoupis Directeur de recherches, INRIA Rennes	Examinateur
Sergio Rajsbaum Directeur de recherches, Universidad Nacional Autónoma de México	Examinateur
Bernadette Charron-Bost Directrice de recherches, École Normale Supérieure	Directrice de thèse
Stephan Merz Directeur de recherches, INRIA Nancy	Co-directeur de thèse

Contents

Abstract

We study a network of agents connected by a dynamic communication graph and we consider a computing model consisting of a sequence of synchronous rounds. We study several synchronization and coordination problems in the context of arbitrary starts and initializations.

First, we study the Firing Squad synchronization problem in which the agents must "fire" simultaneously, i.e., in the same round, in spite of asynchronous starts. Solving this problem requires strong assumptions. We introduce a weakening of this problem in which the agents must fire, not in the same round, but rather in rounds that are congruent modulo some integer P. We present an algorithm solving this problem under weaker communication assumptions: while the exact synchronization for the Firing Squad requires a strong connectivity property, which is not granted in many contexts, the synchronization modulo P is achievable as long as there exists an agent which is connected to all other agents in any bounded time period of consecutive rounds.

Proving the correctness of this algorithm is hard due to a combinatorial explosion of the number of executions. That's why we give a formal proof of the correctness properties of our algorithm using the Isabelle proof assistant.

Then we consider the model of "self-stabilizing" algorithms that must reach a "correct" state regardless of the arbitrary initial states of the agents. We study the synchronization modulo P: each agent holds a clock. All those local clocks must be congruent to P from a certain round. This problem has previously been studied in static strongly connected networks. Moreover, agents were assumed to know some global information (e.g., the total number of agents).

In the second part of this thesis, our aim is finding a relaxation of those hypothesis. We first show that no algorithm can reconcile an absence of global knowledge and a finite set of states. Then we propose a self-stabilizing variant of the MinMax consensus algorithm for the synchronization modulo P. This algorithm tolerates a very high degree of dynamicity of the communication network but requires an infinite memory.

Then we introduce an algorithm that we named SAP (for Self Adaptive Period). Its state space is infinite. However, only a finite number of states is reached in each of its executions. This kind of algorithm is referred to as finite but unbounded memory. We prove the correctness of SAP in networks with an unknown but finite dynamic diameter. This result is then extended to a wider class of centered dynamic networks.

Finally, we study the behaviour of SAP in probabilistic communication models. Instead of proving a property on each executions of SAP, we prove a probabilistic hyperproperty – the synchronization modulo P with high probability –, that is, a property on the set of its executions. Proving an hyperproperty, instead of a simple property, is generally more tedious and requires different techniques. In our case, we define a hierarchy of probabilistic diameters. Using the first two probabilistic diameters, we show that SAP solves the synchronization modulo P with high probability. This proof covers a large number of probabilistic communication models, including the rumor spreading models such as PUSH, PULL and PULLPUSH.

Résumé

Nous considérons un réseau d'agents reliés par un graphe de communication dynamique et nous nous plaçons dans un modèle de calcul consistant en une succession de rounds synchrones. Nous étudions différents problèmes de synchronisation et coordination dans le contexte de départs et initialisations arbitraires.

Tout d'abord, nous étudions le problème de synchronisation du Firing Squad consistant à ce que les agents « fassent feu » simultanément, i.e., au même round, malgré des départs asynchrones. Ce problème nécessitant des hypothèses fortes pour être résoluble, nous introduisons le problème plus faible où les agents doivent faire feu, non pas au même round, mais à des rounds

congrus modulo un certain entier P. Nous présentons un algorithme qui résout ce problème sous des hypothèses de connectivité du graphe de communication bien plus faibles : alors que la synchronisation parfaite pour le Firing Squad exige une connexité forte entre les agents, non garantie dans de nombreux contextes, la synchronisation modulo P est réalisable lorsqu'il existe un agent connecté à tous les autres agents dans toute période de temps (rounds consécutifs) bornée.

La correction de l'algorithme que nous présentons est difficile à prouver du fait d'une explosion combinatoire du nombre de ses exécutions. C'est pourquoi, nous donnons ensuite la preuve formelle des propriétés de correction de notre algorithme avec l'assistant de preuve Isabelle.

Nous considérons ensuite le modèle des algorithmes « auto-stabilisants » qui doivent atteindre un état « correct », quels que soient les états initiaux arbitraires des agents. Nous étudions le problème de la synchronisation modulo P : chaque agent détient une horloge et toutes ces horloges locales doivent être congrues modulo P à partir d'un certain round. Ce problème a été étudié par le passé dans le cadre des réseaux statiques fortement connexes. De plus, les agents sont supposés disposer d'informations globales sur le réseau (e.g., le nombre d'agents).

Notre objectif, dans la seconde partie de cette thèse, est d'étudier s'il est possible de relaxer ces hypothèses. Nous commençons par établir l'impossibilité de concilier l'absence de connaissance globale avec l'utilisation d'algorithmes à états finis. Nous proposons alors une variante auto-stabilisante de l'algorithme de consensus MinMax pour la synchronisation modulo P. Cet algorithme tolère une très grande dynamicité des liens de communication mais requiert une mémoire infinie.

Nous introduisons ensuite un algorithme que nous appelons SAP (pour Self Adaptive Period). Son espace des états est infini mais seul un nombre fini d'états est atteint dans chacune des exécutions. On parle alors d'algorithme à mémoire finie mais non bornée. Nous démontrons la correction de SAP dans les réseaux dynamiques avec un diamètre dynamique fini mais inconnu. Nous étendons ensuite ce résultat à la classe plus générale des réseaux dynamiques centrés.

Pour finir, nous étudions le comportement de l'algorithme SAP dans les modèles de communications probabilistes. Il ne s'agit plus de prouver une propriété de chaque exécution de l'algorithme, mais une hyperpropriété probabiliste – ici la synchronisation modulo P avec haute probabilité –, c'est à dire une propriété de l'ensemble de ses exécutions. De façon générale, les preuves d'hyperpropriétés sont plus délicates que les preuves de simples propriétés, et nécessitent des techniques différentes. En l'occurrence, nous définissons une hiérarchie de diamètres probabilistes. À partir des deux premiers diamètres de cette hiérarchie, nous prouvons que SAP résout la synchronisation modulo P avec haute probabilité. Cette preuve s'applique à un grand nombre de modèles de communications probabilistes, notamment les modèles de propagation de rumeur comme les modèles PULL, PUSH et PULLPUSH.

Acknowledgements

First and foremost I am extremely grateful to my supervisors, Bernadette Charron-Bost and Stephan Merz for their invaluable advice, continuous support, and patience during my PhD study. Their immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life. I would also like to thank Guillaume Prémel and Patrick Lambein for the very useful discussions we had. I am also indebted to Paolo Boldi and Sebastiano Vigna for their deep work on self-stabilization, which inspired me throughout this journey. I would like to thank my friends, lab mates, colleagues and research team for a cherished time spent together in the lab, and in social settings. Finally, I would like to express my gratitude to my parents. Without their tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my study.

Chapter 1

Introduction

There is a considerable interest in distributed systems consisting of multiple, potentially mobile, agents. This is mainly motivated by the emergence of large scale networks, characterized by the lack of centralized control, the access to limited information and a time-varying connectivity. Control and optimization algorithms deployed in such networks should be completely distributed, relying only on local observations and information, and robust against unexpected changes in topology such as link or node[1] failures.

Broadly speaking, there are two main classes of models in the field of distributed computing. First, in *asynchronous models* [69], each agent is modeled by an automaton that is capable of performing three types of actions: internal transition, input action and output action. The sending and receiving actions are used to enable communication between agents and their environment. This is a very general model, suitable for describing almost any type of asynchronous concurrent system, including asynchronous shared memory [70] systems and asynchronous network systems. By itself, this automaton model has very little structure, which allows it to be used for modelling many different types of distributed systems. Additional structure must be added to the basic model to enable it to describe particular types of asynchronous systems. What the model does provide is a precise way of describing and reasoning about system components (e.g., processes or communication channels) that interact with each other and that operate at arbitrary relative speeds. In such models, it is often useful to add some additional assumption to put a limit on the arbitrary relative speeds of system components [48]. It specifies that all the components in a system get "fair" turns to perform steps every so often. It rules out the possibility that some components are permanently denied turns to take steps.

On the other hand, in the *synchronous models*, it is assumed that each agent repeatedly performs the following steps: (a) broadcasts messages determined by its state at the beginning of round t, (b) receives *some* of the messages sent to it, and finally (c) performs an internal transition to a successor state. The agents are assumed to progress in lock-step, as if they all receive the ticks from a global hardware clock. The combination of those three steps is called a *round*.

In essence, the purpose of asynchronous models is to closely mirror the physical execution of computations striving to maintain a direct connection to the real-world process. In stark contrast, synchronous models offer a high-level abstraction that simplifies reasoning about systems. From my perspective, while asynchronous models offer enhanced expressiveness, this advantage is often overshadowed by the complexity they introduce.

This opinion rests on two primary considerations. Firstly, in asynchronous models, the space of potential interleavings of actions is vast. To illustrate this phenomenon, consider synchronous models where, in each round, each node undergoes a state transition in function of a *multi-set* of messages received in the same round. In contrast, in asynchronous message-passing models, agents typically receive messages individually. Consequently, a single state transition in the synchronous model equates to multiple events in the asynchronous model, with numerous possible orderings. Proving properties in asynchronous models thus necessitates grappling with an exceedingly large space of execution possibilities, presenting a formidable challenge.

On the other hand, contrary to what the definition of synchronous models may suggest, agents need not progress in perfect lockstep physically. The sole requirement synchronous models impose is that messages sent in round t should not be received in rounds other than round t. A straightforward construction exists that can emulate synchronized rounds within a fully asynchronous context: each agent maintains an integer clock representing the current round number, and every message carries this round number as a tag. Consequently, when agents receive a message tagged with a particular integer t, they can enforce that it is only processed

[1] In this thesis, we indiscriminately use the terms "agent" and "node".

in round t. This construction can be further refined, particularly to enhance its fault tolerance and time complexity, as explored in the work of Awerbuch [7].

For those reasons, this thesis only covers models with synchronized rounds. Therefore, the previously-mentioned constructions of synchronized rounds are out of the scope of our work, although they provide the foundation on which this thesis relies. A major feature of synchronous models is the fact that the communication pattern at each round is captured by a directed graph that may change continually from one round to the next. This infinite sequence of directed graphs with a same set of nodes is called a *dynamic graph*. This notion is at the center of this thesis. To demonstrate the expressiveness of synchronous models with dynamic graphs, we can mention the Heard-Of model [24], in which the features of the system, such as the degree of synchrony and the (benign) failure model are solely captured by the properties of the dynamic communication graph. For example, some models contain a notion of *crash failure*. A node is said to crash if, at a certain point in time, it stops taking step. Our model contain no such notion of crash failure. However, a crashed node can be represented by a node which, from a certain round t_0, does not send any message, that is, it has no outgoing neighbors in the communication graphs of round t_0 and all subsequent rounds. Such a description defines a certain set of dynamic graph. In this thesis, we use the term *network class* to refer to a set of dynamic graphs.

This thesis is dedicated to the investigation of two distinct problems. Firstly, we delve into the firing squad problem, a problem with historical roots dating back to the early days of automata theory, as evidenced by works such as [71, 72]. The problem is relevant in the context of *asynchronous starts*, where no assurance exists that all agents will commence their operations at the same round. In the firing squad problem, each node is tasked with transmitting a signal at some round, often referred to as *firing*. The central requirement is that all nodes must fire synchronously, implying that they all transmit signals in the same round. Effectively, solving the firing squad problem equates to emulating synchronous starts within an environment characterized by asynchronous starts. Given that a substantial portion of the distributed algorithms literature assumes synchronous starts, the resolution of the firing squad problem becomes pivotal, facilitating the integration of such algorithms into settings of asynchronous starts.

As we will explain later, there are instances where solving the firing squad problem becomes infeasible. Consequently, it becomes necessary to modify the problem's specifications. Interestingly, the two problems examined in this thesis are relaxations of the firing squad problem.

One such modification is the *mod P-firing squad*. In this variant, where P is an integer, the nodes are not required to fire simultaneously but must instead fire in rounds that share the same congruence modulo P. While this relaxation may not directly serve the purpose of simulating synchronous starts, it offers valuable applications. For instance, it finds utility in scenarios where a coordinator needs to be elected through a round-robin strategy. In a system comprising n nodes, each assigned a unique identifier from the set $0, \ldots, n-1$, mod n-firing squad ensures that a unique coordinator is designated in each round. Another compelling use case lies in the deployment of distributed algorithms structured into synchronized *phases*. Examples of such algorithms include the *Two-Phase* and *Three-Phase Commit* algorithms [11] and various consensus algorithms [10, 48, 67, 30]. In these algorithms, a crucial requirement is the alignment of all nodes within the same phase, making mod P-firing squad a valuable tool for ensuring correct execution.

One of the principal contributions of this thesis lies in the development of an algorithm designed to solve this problem. Notably, our algorithm rests upon entirely original concepts and approaches. It leans heavily on combinatorial techniques, making the proof process a notably error-prone endeavor. In light of this, employing formal verification becomes especially relevant to ensure the correctness of our results. We started by establishing the computational model within the proof assistant Isabelle, building upon the foundational work of Merz [23]. Subse-

quently, we formalized our algorithm, crafting a formal proof that faithfully mirrors our initial pencil-and-paper proof.

The rest of the thesis tackles the mod P-synchronization problem, which further relaxes the mod P-firing squad. In this new problem, the nodes do not fire, but instead hold an integer clock between 0 and $P-1$. The objective is ensuring that all clocks are eventually congruent to P. That corresponds to synchronization in *phase*, as opposed to the problem of synchronization in *frequency* (e.g. for instance in [78, 50, 51, 68]). By contrast with the mod P-firing squad, agents do not have to "known" when the synchronization is achieved. Clock synchronization is a fundamental problem arising in a number of applications, both in engineering and natural systems. A synchronized clock is a fundamental basic block used in many engineering systems, e.g. in the universal self-stabilizing algorithm developed by Boldi and Vigna [16]. Clock synchronization also corresponds to a ubiquitous phenomenon in the natural world and finds numerous applications in physics and biology, e.g., the Kuramoto model for the synchronization of coupled oscillators [79], synchronous flashing fireflies, collective synchronization of pancreatic beta cells [63].

In this second segment of the thesis, dedicated to the field of synchronization, we transition to a distinct paradigm: that of *self-stabilizing* algorithms. In the preceding section, we encountered challenges rooted in asynchronous starts, where uncertainties revolved around the temporal dimension. In the self-stabilizing paradigm, however, agents are granted the liberty of initiating their operations from arbitrary initial states, shifting the locus of uncertainty to the spatial dimension.

The concept of self-stabilization has a long-standing history. A self-stabilizing algorithm possesses the remarkable capability to withstand *transient faults*. In practical terms, this means that the algorithm can recover to a correct state within a finite time span following any arbitrary modification of the system's state during its execution. A prominent example of transient faults arises from cosmic rays, which have the potential to flip bits in the memory of digital devices. Given the sheer diversity of behaviours that constitute transient faults, self-stabilizing algorithms exhibit exceptional resilience.

Notably, it's worth mentioning that self-stabilizing algorithms inherently accommodate asynchronous starts, requiring no additional effort to tolerate them. Consequently, our analysis of self-stabilizing algorithms primarily revolves around scenarios with synchronous starts. However, it's important to note that the reverse does not hold true in general: a synchronization algorithm capable of withstanding asynchronous starts doesn't necessarily qualify as self-stabilizing. The results presented in this thesis shed light on the intricate relationship between self-stabilizing algorithms and algorithms designed to endure asynchronous starts, unraveling their precise interplay.

A common shortcoming of many paper that deal with those two problems is that only static strongly connected network are considered. In light of the Heard-Of model, this is problematic: as explained above, to represent a network in which crash failures may occur, we use a certain network class which contains non-static dynamic graphs. We saw this shortcoming as an opportunity to provide a meaningful contribution. Therefore a major objective of this thesis is to provide results that tolerate dynamic networks.

In the context of static networks, the concept of strong connectivity naturally serves as a fundamental assumption. However, when dealing with dynamic networks, this notion requires redefinition to suit the evolving nature of the graph. Various approaches exist to make sense of strong connectivity in the context of dynamic graphs.

To do so, we define the notion of *temporal path*: For instance, if an arc (i, j) exists in round t and another arc (j, k) exists in round $t+1$, then the sequence i, j, k constitutes a *temporal path* within the dynamic graph. Intuitively, this concept of a temporal path provides

a means to describe the propagation of information between agents within a dynamic network. In traditional graph theory, the diameter is defined as the maximum length of the shortest path between any two nodes. By using the notion of temporal paths, we extend this definition to introduce the concept of a *dynamic diameter* within dynamic graphs. We believe that the most suitable extension of strong connectivity to dynamic graphs is to consider networks whose dynamic diameter remains finite. Similarly, we extend the conventional *radius* of a graph to introduce the concept of a *dynamic radius* in dynamic graphs. These concepts of dynamic diameter and dynamic radius hold pivotal roles within our proofs and analyses, providing a foundational framework for understanding and characterizing strong connectivity in dynamic networks.

A large chunk of the literature on distributed algorithms contains some form of probabilistic behaviour in the communications. The idea consists in choosing some probability distributions on the set of all dynamic graphs. In this thesis, such probability distributions are called *probabilistic communication networks*. Then, the notion of solvability that we used so far is replaced by a notion of *solvability with high probability*. In other works, instead of proving a property on each execution of some algorithm, the task consists in proving the following *hyperproperty* [34]: the probability measure of the set of executions of some algorithm in which some target property is achieved is greater than a chosen real $p \in [0, 1)$. It is also possible to introduce some probabilistic behaviour in the state transitions of the agents [10], but this possibility will be set aside in this thesis. Many possible probabilistic communication networks have already been studied: In one of the earliest contribution, given some integers n and N, Erdös and Rényi [49] focuses on the uniform distribution on the set of graphs[2] with n nodes and N arcs. Another example is the PUSH and PULL models [56, 39]: A "base" graph G is fixed and, in each round, each node sends (resp. receives) a message to some neighbor in G, sampled uniformly at random. A natural question is therefore whether the model and the tools that we presented until now behave nicely in setting in which the communications are probabilistic. In Chapter 6, we explain that the notion of dynamic diameter is no longer relevant in this context. The objective of this chapter is therefore to define some new tools to deal with probabilistic proofs while remaining similar to the ones used in the deterministic proofs, thus bridging the gap between both approaches.

This thesis is structured as follows. Chapter 2 contains all essential definitions, notably the computation model, the problems and the graph theory notions we rely on. Chapter 3 presents two impossibility results that are useful to understand the next chapters. Chapter 4 propose a construction that solves the mod P-synchronization problem based on the existing MINMAX algorithm. In Chapter 5, we study the mod P-firing squad problem by proposing a novel algorithm. Chapter 6 formally introduces a novel set of tools useful for probabilistic analysis of distributed algorithms. We then study the mod P-synchronization problem and we introduce an algorithm called SAP. We prove the correctness of this algorithm using those tools. Chapter 7 is an extension of the previous chapter: it bridges the gap between probabilistic tools we introduce with the existing literature on rumor spreading. Finally, Chapter 8 presents a deterministic study of SAP. Our three publications [38, 28, 29] contain the results that are presented in Chapters 5, 8 and 6, respectively.

[2]Given a probability distribution on the set of graphs with n nodes, it is possible to construct a probability distribution on the set of dynamic graphs, by independently sampling a graph for each integer t.

Chapter 2

Model

2.1 The computational model

We consider a networked system of n agents (nodes), denoted $1, 2, \cdots, n$. We assume a round-based computational model in the spirit of the Heard-Of model [30], in which point-to-point communications are organized into *synchronized rounds*: each node sends messages to all nodes and receives messages sent by *some* of the nodes. Rounds are communication closed in the sense that no node receives messages in round t ($t = 1, 2, \ldots$) that are sent in a round different from t. The collection of communications (which nodes receive messages from which nodes) at each round t is modelled by a directed graph (digraph, for short) $\mathbb{G}(t) = ([n], E_t)$, where $[n] = \{1, \ldots, n\}$. The digraph at round t is called the *communication graph at round* t. The set of i's incoming neighbors in the digraph $\mathbb{G}(t)$ is denoted by $In_i(t)$.

In all this thesis, we assume a self-loop at each node in all these digraphs since every node can communicate with itself instantaneously. The sequence of such digraphs $\mathbb{G} = (\mathbb{G}(t))_{t\geq 1}$ with a constant set of nodes is called a *dynamic graph* [19]. A dynamic graph is said to be *static* if it is composed of a single digraph repeated forever. A *network class* is any non-empty set of dynamic graphs. We denote $\mathcal{G}_n$ the set of all dynamic graphs of size n and containing a self-loops at each node.

In round t, each node i successively (a) broadcasts[1] messages determined by its state at the beginning of round t, (b) receives *some* of the messages sent to it, and finally (c) performs an internal transition to a successor state. We recall that the synchronous rounds model is an abstraction: it is not assumed that the internal state transitions of the nodes physically happen at the same time. Moreover, in each round, each node may choose a certain value as output. A *local algorithm* is therefore defined as follows.

Definition 1 (local algorithm)**.** *A local algorithm* ALG *is a tuple* $(\mathcal{Q}, \mathcal{I}, \mathcal{M}, \mathcal{O}, \sigma, \tau, \omega)$*, where*

- $\mathcal{Q}$ *is a non-empty set that is called the* set of states *of* ALG*,*
- $\mathcal{I}$ *is a non-empty subset of* $\mathcal{Q}$ *that is called the* set of initial states *of* ALG*,*
- $\mathcal{M}$ *is a non-empty set that is called the* set of messages *of* ALG*,*
- $\mathcal{O}$ *is a non-empty set that is called the* set of outputs *of* ALG*,*
- σ *is a function of type* $\mathcal{Q} \to \mathcal{M}$ *that is called the* sending function *of* ALG*,*
- τ *is a function of type* $\mathcal{Q} \times \mathcal{M}^{\oplus} \to \mathcal{Q}$ *that is called the* transition function *of* ALG*, where* $\mathcal{M}^{\oplus}$ *denotes the set of multisets of* $\mathcal{M}$*.*
- ω *is a function of type* $\mathcal{Q} \to \mathcal{O}$ *that is called the* output function *of* ALG*.*

In the rest of this thesis, we use the $\{\{\ldots\}\}$ notation to denote multisets. The *sending function* of ALG determines the messages to be sent in step (a), and its *transition function* determines its state updates in step (c). In each round, each node displays an output value, which is computed by applying the function ω to its current state. The notion of problem will be defined as a predicate on sequences of vectors of output values. A *global algorithm* (or *algorithm*, for short) for the set of nodes $[n]$ is a collection of local algorithms, one per node. We will only consider symmetric algorithms, in which all nodes run the same local algorithm.

Moreover, we say that an algorithm ALG *uses bounded memory* if its set of states $\mathcal{Q}$ if finite.

We now introduce a definition that is deeply connected to the notion of *self-stabilization* that is introduced later in Section 2.4.

[1] The system is anonymous and nodes have no knowledge about the dynamic graph. Therefore, the only valid way to send messages is "send to all". The communication graph describes which message will actually be received.

Definition 2 (non-initialized algorithm)**.** *An algorithm* ALG $= (\mathcal{Q}, \mathcal{I}, \mathcal{M}, V, \sigma, \tau, \omega)$ *is said to be* self-stabilizing *if its set of initial states is equal to its set of states, that is,* $\mathcal{I} = \mathcal{Q}$*.*

Basically, when $\mathcal{Q} \neq \mathcal{I}$, an agent must perform an initialization phase at some point, so that its internal state belongs to $\mathcal{I}$. There is no need for such procedure if $\mathcal{Q} = \mathcal{I}$, hence the choice of this terminology.

2.2 Executions and start schedules

Definition

A *start schedule* $\mathbb{S}$ is a collection $(s_i)_{i\in[n]}$, where each s_i is a positive integer or is equal to ∞. Given a dynamic graph $\mathbb{G}$ and a start schedule, $\mathbb{G}^a(t) = ([n], E_t^a)$ denotes the graph where $E_t^a \subseteq E_t$ is the set of arcs that are either self-loops or connect two nodes i and j such that $s_i \geq t$ and $s_j \geq t$. The set of i's incoming neighbors in the digraph $\mathbb{G}^a(t)$ is denoted by $In_i^a(t)$.

Definition 3 (execution)**.** *The execution of an algorithm* ALG *in a system of size* n *with a dynamic graph* $\mathbb{G}$*, a start schedule* $(s_i)_{i\in[n]}$ *and a collection of initial states* $(\mu_i)_{i\in[n]}$ *belonging to* $\mathcal{I}$ *is the sequence of vectors* $(q_i(t))_{i\in[n],t\in\mathbb{N}}$ *of elements in* $\mathcal{Q}$ *such that, for all node* i *and non-negative integer* t*,*

$$q_i(t) = \begin{cases} \mu_i & \text{if } t < s_i \\ \tau\left(q_i(t-1), \{\{\sigma(q_j(t-1)) \mid j \in In_i^a(t)\}\}\right) & \text{otherwise.} \end{cases} \tag{2.1}$$

Intuitively, it proceeds as follows: Each node i is initially *passive*. If $s_i = \infty$, then the node i remains passive forever. Otherwise, s_i is a positive integer, and i becomes *active* at the beginning of round s_i, setting up its initial state to μ_i. An active node sends messages and updates its state, as explained in Section 2.1. A passive node sends no message. Therefore, from the point of view of an active node, a passive incoming neighbor (belonging to $In_i(t) \setminus In_i^a(t)$) is indistinguishable from a node that is not an incoming neighbor (belonging to $[n] \setminus In_i(t)$). The state of an active node i at the end of round t is $q_i(t)$. By extension, $q_i(t)$ refers to the initial state of i if $t < s_i$.

Asynchronous starts with heartbeats

An additional assumption on the communication model may be necessary (e.g., in Chapter 5). It consists in assuming that, in round t ($t = 1, 2 \dots$), each passive node sends *heartbeats*, corresponding to a special *null* messages that does not belong to $\mathcal{M}$. An active node can now distinguish a passive incoming neighbor from a non-incoming neighbor. In this setting, the type of the transition function τ is $\mathcal{Q} \times (\mathcal{M} \uplus \{null\})^{\oplus} \to \mathcal{Q}$, instead of $\mathcal{Q} \times \mathcal{M}^{\oplus} \to \mathcal{Q}$. The definition of an execution has to be modified as follows.

Definition 4 (execution with heartbeats)**.** *The execution of an algorithm* ALG *in a system of size* n *with a dynamic graph* $\mathbb{G}$*, a start schedule* $(s_i)_{i\in[n]}$ *and a collection of initial states* $(\mu_i)_{i\in[n]}$ *belonging to* $\mathcal{I}$ *is the sequence of vectors* $(q_i(t))_{i\in[n],t\in\mathbb{N}}$ *of elements in* $\mathcal{Q}$ *such that, for all node* i *and non-negative integer* t*,*

$$q_i(t) = \begin{cases} \mu_i & \text{if } t < s_i \\ \tau\left(q_i(t-1), \{\{\sigma(q_j(t-1)) \mid j \in In_i^a(t)\}\} \cup \{\{null \mid j \in In_i(t) \setminus In_i^a(t)\}\}\right) & \text{otherwise.} \end{cases}$$

Note that the states "passive" and "active" do not refer to any physical notion, and are relative to the algorithm under consideration: as an example, if two algorithms ALG_1 and ALG_2 are sequentially executed according to the order "ALG_1 followed by ALG_2" by each node, then at some round, a node may be active w.r.t. ALG_1 while it is passive w.r.t. ALG_2. In such a situation, the node is integrally part of the system and can send messages, but these messages are empty with respect to the semantics of the algorithm ALG_2. Those messages are then interpreted as heartbeats by ALG_2. Moreover, the start schedule of ALG_2 is equal to the "termination schedule" of ALG_1.

Start schedules

A *start class* is a non-empty set of start schedules. Here, we define some start classes.

Definition 5 (complete start schedule)**.** *A start schedule* $\mathbb{S} = (s_i)_{i\in[n]}$ *is* complete *if every* s_i *is finite.*

Definition 6 (diffusive start schedule)**.** *A start schedule* $\mathbb{S} = (s_i)_{i\in[n]}$ *is* diffusive with respect to a dynamic graph $\mathbb{G}$ *with* n *nodes if at least one* s_i *is finite and the following condition is satisfied:*

$$\forall i \in [n], \forall t \in \mathbb{N}, \forall j \in In_i(t),\ \ s_j \leq t \Rightarrow s_i \leq t+1.$$

In diffusive start schedules, nodes start either spontaneously or upon the receipt of a message from an active node. If the communication network is static and strongly connected, then a diffusive start schedule is complete. Finally, we define the class of synchronous start schedules. In this class, all the clock synchronization problems presented in Section 2.3 become trivial.

Definition 7 (synchronous start schedule)**.** *A start schedule* $\mathbb{S} = (s_i)_{i\in[n]}$ *is* synchronous *if all* s_i *are finite and equal.*

When running a non-initialized algorithm, all nodes are assumed to start at round 1. Indeed, if a non-initialized algorithm is executed with a complete start schedule, then in order to prove that this execution eventually satisfies a certain property, it is sufficient to consider its suffix starting at round $s^{max} = \max_{i\in[n]} s_i$: As the algorithm is non-initialized, the state of the system in round s^{max} does not matter. Therefore, any solvability result for certain problem on a certain non-initialized algorithm that assumes a synchronous start schedule can immediately be transposed to an analog result with a complete start schedule. In the context of non-initialized, the dynamic graphs $\mathbb{G}$ and $\mathbb{G}^a$ are therefore equal.

Finite memory

An algorithm is said to *use finite memory* if, in all of its possible executions (in a certain network class), the memory usage by each node eventually stops growing. More formally,

Definition 8 (finite memory)**.** *An algorithm* ALG uses finite memory in a network class $\mathcal{C}$ *if, for all integers* $n > 0$*, in all its executions* $(q_i(t))_{i\in[n],t\in\mathbb{N}}$ *in this network class, the set* $\{q_i(t) \mid i \in [n], t \in \mathbb{N}\}$ *is finite.*

By contrast with the notion of bounded memory (see Section 2.1), this notion is relative to a specific network class. Also notice that an algorithm using bounded memory (i.e., $\mathcal{Q}$ is finite) also uses finite memory.

problem type	equality	congruence modulo P
firing	firing squad	mod P-firing squad
stabilizing	synchronization	mod P-synchronization

Table 2.1: Summary of the names of the problems that will be presented in this document.

2.3 Clock synchronization problems

Given an algorithm ALG, the output function can be applied to all elements of an execution $(q_i(t))_{i\in[n],t\in\mathbb{N}}$ of ALG. We obtain a sequence of vectors of elements from the output set $\mathcal{O}$. A *problem* is a predicate on such sequences.

Definition 9 (solving a problem)**.** *An algorithm* ALG *solves a problem $\mathcal{P}$ in a network class and start class if, in any execution $(q_i(t))_{i\in[n],t\in\mathbb{N}}$ in this network class and start class, the sequence of vector $(\omega(q_i(t)))_{i\in[n],t\in\mathbb{N}}$ satisfies the predicate $\mathcal{P}$.*

We say that an algorithm ALG *tolerates asynchronous starts* if ALG solves a certain problem in the class of complete start schedules.

Several variants of clock synchronization

This thesis tackles four variants of the clock synchronization problem. Among those four variants, two are *stabilizing* problems and the other two are *firing* problems. In each case, our definitions contains a relation between two integers. This relation is either the equality or the congruence modulo some integer P. The names of those problems are provided in Table 2.1.

First, we define the *(stabilizing) synchronization problem*, which uses the equality relation.

Definition 10 (synchronization)**.** *The* synchronization *problem consists of the following predicate on sequences of vectors of integers $(x_i(t))_{i\in[n],t\in\mathbb{N}}$:*

$$\exists c\in\mathbb{Z}, \exists t_0\in\mathbb{N}, \forall t\geq t_0, \forall i\in[n],\ x_i(t)=c+t.$$

An execution $(q_i(t))_{i\in[n],t\in\mathbb{N}}$ of an algorithm ALG in a system of size n is said to be *synchronized* from round t_0 if,

$$\exists c\in\mathbb{Z}, \forall t\geq t_0, \forall i\in[n],\ \omega(q_i(t))=c+t.$$

The mod P-synchronization problem is a weakening of the synchronization problem: the equality relation is replaced by an equality modulo some period $P>1$. In the rest of this document, this relation is denoted $\equiv_P$.

Definition 11 (mod P-synchronization)**.** *The* mod P-synchronization *problem consists of the following predicate on sequences of vectors of integers $(x_i(t))_{i\in[n],t\in\mathbb{N}}$:*

$$\exists c\in\mathbb{Z}, \exists t_0\in\mathbb{N}, \forall t\geq t_0, \forall i\in[n],\ x_i(t)\equiv_P c+t.$$

An execution $(q_i(t))_{i\in[n],t\in\mathbb{N}}$ of an algorithm ALG in a system of size n is said to be *mod P-synchronized* from round t_0 if,

$$\exists c\in\mathbb{Z}, \exists t_0\in\mathbb{N}, \forall t\geq t_0, \forall i\in[n],\ \omega(q_i(t))\equiv_P c+t.$$

When no confusion can arise, we simply say that this execution is *synchronized* from round t_0.

We now deal with the firing problems. Given an execution of an algorithm whose set of outputs is $\{\bot, \top\}$, a node i is said to *fire* in round t if it outputs $\top$ in round t whereas it outputs $\bot$ in all earlier rounds. If i eventually fires, the round in which it fires is denoted t_i. Otherwise, $t_i = \infty$.

$$t_i \stackrel{\text{def}}{=} \inf\{t \in \mathbb{N} \mid \omega(q_i(t)) = \top\}. \tag{2.2}$$

Definition 12 (firing squad)**.** *The* firing squad *problem consists of the conjunction of the following predicates on sequences $(x_i(t))_{i\in[n],t\in\mathbb{N}}$ of elements from the set $\{\top, \bot\}$:*

- *Termination:* $\forall i \in [n],\ s_i < \infty \Rightarrow t_i < \infty$,
- *Simultaneity:* $\exists c \in \mathbb{N}, \forall i \in [n],\ t_i < \infty \Rightarrow t_i = c$.

Similarly, this problem can be relaxed by replacing the equality relation by the congruence relation.

Definition 13 (mod P-firing squad)**.** *The* mod P-firing squad *problem consists of the conjunction of the following predicates on sequences $(x_i(t))_{i\in[n],t\in\mathbb{N}}$ of elements from the set $\{\top, \bot\}$:*

- *Termination:* $\forall i \in [n],\ s_i < \infty \Rightarrow t_i < \infty$,
- *Simultaneity:* $\exists c \in \mathbb{N}, \forall i \in [n],\ t_i < \infty \Rightarrow t_i \equiv_P c$.

The following theorem shows that solving a stabilizing synchronization problem is strictly easier than solving the firing squad problem.

Theorem 14. *If the firing squad problem is solvable in a certain network class, then the synchronization problem is also solvable in the same network class.*

Proof. We fix a network class and an algorithm $\text{ALG}_1 = (\mathcal{Q}, \mathcal{I}, \mathcal{M}, \{\top, \bot\}, \sigma, \tau, \omega)$ solving the firing squad problem. We construct an algorithm

$$\text{ALG}_2 \stackrel{\text{def}}{=} (\mathcal{Q} \times \mathbb{N}, \mathcal{I} \times \{0\}, \mathcal{M}, \mathbb{N}, \sigma \circ p_1, \tau', p_2),$$

where p_1 and p_2 are the first and second projections, and τ' is defined as

$$\tau'((q,x),\{m_1,m_2,\dots\}) \stackrel{\text{def}}{=} \begin{cases} (\tau(q,\{m_1,m_2,\dots\}),0) & \text{if } \omega(q) = \bot \text{ and } \omega(\tau(q,\dots)) = \top \\ (\tau(q,\{m_1,m_2,\dots\}),x+1) & \text{otherwise.} \end{cases}$$

The intuition behind this definition is as follows: we add an integer clock variable x_i to each node. This clock is incremented in each round, and is reset to 0 upon firing. We now show that if ALG_1 satisfies the two clauses of the firing squad problem, then in any execution of ALG_2, the clocks are synchronized once all nodes have fired. We fix an execution of ALG_1 and we study the corresponding execution of ALG_2, that is, the execution of ALG_2 with the same dynamic graph, start schedule and initial state. Because of the termination clause, there exists a round t_0 in which all nodes have fired. By applying the transition function, from the round in which i fires, until round t_0, we obtain $x_i(t_0) = t_0 - t_i$. Because of the simultaneity clause, all t_i are equal, and thus, the system is synchronized in round t_0. □

By replacing all equalities of this proof by the congruence modulo P relation, we obtain that if the mod P-firing squad is solvable in a certain network class, then the mod P-synchronization is also solvable in the same network class.

Motivation for mod P-firing squad

The firing squad problem is a classical problem which is typically used to simulate synchronous starts in the context of asynchronous starts. The mod P-firing squad is a weakening of the firing squad problem. We present some use cases in which this weakening is sufficient.

Let ALG be an algorithm structured in regular *phases* consisting of a fixed number P of consecutive rounds: the behaviour of each node (i.e., the update rule of its state and the message it sends) at round t is determined by the value of t modulo P. Moreover, assume that ALG has been proved correct with respect to some specification when all nodes start ALG synchronously, but with any dynamic graph in a network class that is stable under the addition of arbitrary finite prefixes. For instance, the *ThreePhaseCommit* algorithm for non-blocking atomic commitment [11], as well as the consensus algorithms in [48] or the *LastVoting* algorithm [30] – corresponding to the consensus core of *Paxos* [67] – fulfill all the above requirements for phases of length $P = 3$ and $P = 4$, respectively, and the family of dynamic graphs in which there exists an infinite number of "good" communication patterns (e.g., a sequence of $2P$ consecutive communication graphs in which a majority of nodes is heard by all nodes in each graph). The use of a mod P-firing squad algorithm prior to the algorithm ALG yields a new algorithm that executes exactly like ALG does, after a finite preliminary period during which every node becomes active and fires. This variant of ALG is therefore guaranteed to be correct with asynchronous starts and dynamic graphs in the network class mentioned above.

Another typical example for which the perfect synchronization requirement in the firing squad problem can be weakened into mod P-firing squad is the development of the basic *rotating coordinator* strategy for a given algorithm ALG in the context of asynchronous starts. Roughly speaking, this strategy consists in the following: each node i has unique identifiers in $[n]$, and maintains a local counter c_i whose current value is the number of rounds elapsed since the node i started executing ALG. At each round, the coordinator of i is the node with the identifier that is equal to the current value of c_i modulo n. Since there may be only one coordinator per round, such a selection rule requires synchronized counters. Clearly, with the use of a mod n-firing squad algorithm in a preliminary phase, the above scheme implements the rotating coordinator strategy.

Consensus problems

Another type of problems in distributed control are the consensus problems. Similarly to clock synchronization problems, there exists several variants of the consensus problem. In this section, we define two of them: stabilizing consensus and irrevocable consensus. We fix an arbitrary set of output $\mathcal{O}$. Each node i holds an initial value $\mu_i \in \mathcal{O}$ and their goal is to agree on one of the initial values.

Definition 15 (stabilizing consensus)**.** *The* stabilizing consensus *problem consists of the conjunction of the following predicates on sequences of vectors* $(x_i(t))_{i\in[n],t\in\mathbb{N}}$ *of elements from the set* $\mathcal{O}$*:*

- *Agreement:* $\exists c \in \mathcal{O}, \exists t_0 \in \mathbb{N}, \forall t \geq t_0, \forall i \in [n],\ x_i(t) = c.$
- *Validity:* $\forall t \in \mathbb{N}, \forall i \in [n], \exists j \in [n],\ x_i(t) = x_j(0).$

In the irrevocable consensus, the output set is $\mathcal{O}' = \mathcal{O} \times \{\top, \bot\}$. We denote p_1 and p_2 the first and second projections. A node decides when it adopts $\top$ as the second component of its output, the first component of its output becomes irrevocable.

Definition 16 (irrevocable consensus)**.** *The* irrevocable consensus *problem consists of the conjunction of the following predicates on sequences of vectors* $(x_i(t))_{i\in[n],t\in\mathbb{N}}$ *of elements of* $\mathcal{O}\times\{\top,\bot\}$*:*

- *Agreement:* $\exists c\in\mathcal{O},\exists t_0\in\mathbb{N},\forall t\geq t_0,\forall i\in[n],\ p_1(x_i(t))=c$.
- *Validity:* $\forall t\in\mathbb{N},\forall i\in[n],\exists j\in[n],\ p_1(x_i(t))=p_1(x_j(0))$.
- *Termination:* $\forall i\in[n],\exists t_0\in\mathbb{N},\forall t\in\mathbb{N},\ t>t_0\Leftrightarrow p_2(x_i(t))=\top$.
- *Irrevocability:* $\forall i\in[n],\forall t_1,t_2\in\mathbb{N},\ p_2(x_i(t_1))=p_2(x_i(t_2))=\top\Rightarrow p_1(x_i(t_1))=p_1(x_i(t_2))$.

2.4 Self-stabilization

The point of the notion of self-stabilization is to tolerate *transient faults*. A *transient fault* is an arbitrary modification of the state of a system at a certain instant. As an example, a possible source of transient faults in real-world systems is cosmic rays, that can randomly flip a bit in the memory of a digital device. A self-stabilizing algorithm is an algorithm that is able to recover from a transient fault in finite time. In other words, in any execution, a certain correctness property is eventually achieved regardless of the initial state of the nodes.

Definition 17 (self-stabilizing synchronization algorithm)**.** *A* self-stabilizing (resp. mod P) synchronization algorithm *is a non-initialized algorithm that solves the (resp. mod* P*) synchronization problem in a certain network class.*

This definition highlights the fact that an algorithm is self-stabilization *with respect to a certain specification*, in this case, synchronization. By contrast, an algorithm is non-initialized outside any specification. The notion of self-stabilization has been further developed by Nesterenko and Tixeuil [74]. They use a more general definitions of self-stabilization, and Definition 17 corresponds to their notion of *ideal stabilization*, which refines self-stabilization. By contrast

2.5 Network classes

In this section, we define all network classes that will be useful in the rest of this document.

Graph product

Let us first recall the notion of *product* of two digraphs $G_1=([n],E_1)$ and $G_2=([n],E_2)$, denoted by $G_1\circ G_2$ and defined as follows [27]: $G_1\circ G_2=([n],E_{12})$, where an arc (i,j) belongs to E_{12} if and only if there exists $k\in[n]$ such that $(i,k)\in E_1$ and $(k,j)\in E_2$. This notion of product coincides with the product of the incidence matrices. Assuming that G_1 and G_2 contain a self-loop at each node, we have

$$E_1\cup E_2\subseteq E_{12}.$$

This inclusion is strict, in general. As a consequence, the diameter of the graph $G_1\circ G_2$ is lower than or equal to the diameter of the graph $([n],E_{12})$. However, the diameter or $G_1\circ G_2$ is finite if and only if the diameter of $([n],E_{12})$ is finite. For any dynamic graph $\mathbb{G}$ and any integers t and t' such that $t'>t\geq 1$, we let

$$\mathbb{G}(t:t')\stackrel{\text{def}}{=}\mathbb{G}(t)\circ\mathbb{G}(t+1)\circ\cdots\circ\mathbb{G}(t').$$

By extension, we let $\mathbb{G}(t:t)=\mathbb{G}(t)$.

Each arc (i, j) in the digraph $\mathbb{G}(t : t')$ corresponds to a (not necessarily unique) *temporal path* $i \triangleright j$ *in the interval* $[t, t']$, i.e., a finite sequence of nodes $i = k_{t-1}, k_t, \ldots, k_{t'} = j$ such that each pair $(k_{\ell-1}, k_\ell)$ is an arc of $\mathbb{G}(\ell)$. We simply say *path*, for short. The set of incoming neighbors of j in $\mathbb{G}(t : t')$ and $\mathbb{G}^a(t : t')$ are denoted by $In_j(t : t')$ and $In_j^a(t : t')$, respectively. A path $k_{t-1}, k_t, \ldots, k_{t'}$ is said to be *active* if the nodes $k_{t-1}, k_t, \ldots, k_{t'}$ are active in rounds $t-1, t, \ldots t'$, respectively.

Eccentricity, dynamic diameter, dynamic radius

In the study of distributed algorithms in static networks, the notions of strong connectivity and diameter often come up. Those notions may be extended to dynamic networks, as detailed in this section. We define a family of network classes, which will be extensively used throughout this document. Given a non-negative integer t, a positive integer ℓ and a node $i \in [n]$, we let

$$\Gamma_i^{t,\ell} \stackrel{\text{def}}{=} \{\mathbb{G} \in \mathcal{G}_n \mid \forall j \in [n],\ (i, j) \text{ is an arc of } \mathbb{G}(t+1 : t+\ell)\}).$$

To be rigorous, those sets should be indexed by n. We omit this extra parameter as we will always use the expression $\Gamma_i^{t,d}$ in a context where no confusion can arise. To give an intuitive understanding of those classes, we provide the following scenario. We fix a dynamic graph $\mathbb{G}$, a node i and two integers t and d. Imagine that, at the end of round t, node i is colored yellow (see Fig. 2.1). In each round between rounds $t+1$ and $t+d$, all nodes that have a yellow node as incoming neither also become yellow (see Fig. 2.2 and Fig. 2.3). The yellow color therefore propagates in the network during d rounds. Then $\mathbb{G}$ belongs to $\Gamma_i^{t,d}$ if and only if all nodes are yellow at the end of round $t+d$ (see Fig. 2.4).

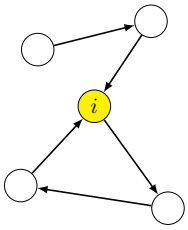


Figure 2.1: Round t

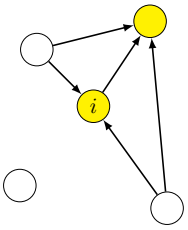


Figure 2.2: Round $t+1$

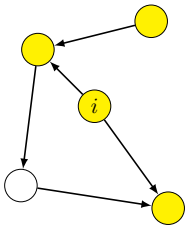


Figure 2.3: Round $t+d-1$

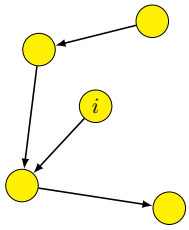


Figure 2.4: Round $t+d$

Figure 2.5: Representation

Because of self-loops, it holds that if an interval $[t_1, t_1 + \ell_1]$ is contained in another interval $[t_2, t_2 + \ell_2]$, then

$$\Gamma_i^{t_1,\ell_1} \subseteq \Gamma_i^{t_2,\ell_2}. \tag{2.3}$$

Definition 18 (dynamic eccentricity)**.** *The* dynamic eccentricity of a node i *in a dynamic graph* $\mathbb{G}$ *is defined as*

$$\mathrm{e}_{\mathbb{G}}(i) \stackrel{def}{=} \inf\{d \in \mathbb{N}^+ \mid \forall t \in \mathbb{N},\ \mathbb{G} \in \Gamma_i^{t,d}\}.$$

The dynamic eccentricity is named in analogy with the notion of eccentricity in a graph. Similarly, the definitions of *dynamic diameter* and *dynamic radius* are named in analogy to the diameter and radius in a graph. For short, we will say *eccentricity* to refer to dynamic eccentricity. The node i is said to be *central* in $\mathbb{G}$ if its eccentricity is finite.

Definition 19 (center)**.** *The* center *of* $\mathbb{G}$*, denoted by* $Z(\mathbb{G})$*, is defined as the set of* $\mathbb{G}$*'s central nodes.*

$$Z(\mathbb{G}) \stackrel{def}{=} \{i \in [n] \mid \mathrm{e}_{\mathbb{G}}(i) < \infty\}.$$

(a) graph G_1 (b) graph G_2

Figure 2.6: Two graphs.

Definition 20 (dynamic diameter)**.** *The* dynamic diameter *of* $\mathbb{G}$*, denoted* $D(\mathbb{G})$*, is defined as*

$$D(\mathbb{G}) \stackrel{def}{=} \sup_{i\in[n]} \mathrm{e}_{\mathbb{G}}(i),$$

Definition 21 (dynamic radius)**.** *The* dynamic radius *of* $\mathbb{G}$*, denoted* $\mathrm{rad}(\mathbb{G})$*, is defined as*

$$\mathrm{rad}(\mathbb{G}) \stackrel{def}{=} \inf_{i\in[n]} \mathrm{e}_{\mathbb{G}}(i),$$

If $\mathbb{G}$ is an infinite repetition of some digraph G, then the dynamic diameter and the dynamic radius of $\mathbb{G}$ are equal to the diameter and the radius of G, respectively.

Strongly centered networks

The definition of strongly centered networks relies on the notion of *kernel*. Intuitively, a node i belongs to the kernel of $\mathbb{G}$, denoted $K(\mathbb{G})$, if and only if i can infinitely often communicate with any node of the dynamic graph $\mathbb{G}$.

Definition 22 (kernel)**.** *The* kernel *of a dynamic graph* $\mathbb{G}$ *of size* n *is defined as*

$$K(\mathbb{G}) \stackrel{def}{=} \{i \in [n] \mid \forall j \in [n], \forall t \in \mathbb{N}, \exists t' > t,\ i \in In_j(t+1:t')\}.$$

Clearly, it holds that $Z(\mathbb{G}) \subseteq K(\mathbb{G})$. The inclusion is strict in general. To illustrate this, let G_1 and G_2 be the two graphs defined in Fig. (2.6). Let $\mathbb{G}$ be the dynamic graph such that $\mathbb{G}(t) = G_1$ if t is a prime number. Otherwise, $\mathbb{G}(t) = G_2$. In this example, both nodes belong to $K(\mathbb{G})$, as there is an infinite number of prime numbers and non-prime numbers. Moreover, node 2 is central as there is no sequence of three consecutive prime numbers. More precisely, the eccentricity of node 2 is equal to 3. However, node 1 is not central, as there exists arbitrarily long sequences of consecutive non-prime numbers.

Definition 23 (stronly centered)**.** *A dynamic graph* $\mathbb{G}$ *is said to be* strongly centered *if the following properties are satisfied.*

$$Z(\mathbb{G}) \neq \emptyset \quad \textit{and} \quad K(\mathbb{G}) = Z(\mathbb{G}). \tag{2.4}$$

Predicate transformers

Predicate transformers are constructions that transforms predicates on graphs (characterized by a set $\mathcal{C}$ of static graphs) into predicates on dynamic graphs. We now define three predicate transformers.

Definition 24 ($\mathcal{C}$ with delay Δ)**.** *A dynamic graph* $\mathbb{G}$ belongs to $\mathcal{C}$ with delay Δ *if, for each non-negative integer* t*, the graph* $\mathbb{G}(t+1:t+\Delta)$ *belongs to* $\mathcal{C}$*.*

Definition 25 ($\mathcal{C}$ with bounded delay)**.** *A dynamic graph* $\mathbb{G}$ belongs to $\mathcal{C}$ with bounded delay *if there exists an integer* Δ *such that, for each non-negative integer* t*, the graph* $\mathbb{G}(t+1:t+\Delta)$ *belongs to* $\mathcal{C}$*.*

Definition 26 ($\mathcal{C}$ with finite delay)**.** *A dynamic graph* $\mathbb{G}$ belongs to $\mathcal{C}$ with finite delay *if, for each non-negative integer* t*, there exists an integer* Δ *such that the graph* $\mathbb{G}(t+1:t+\Delta)$ *belongs to* $\mathcal{C}$*.*

If $\mathbb{G}$ belongs to $\mathcal{C}$ with bounded delay, it also belongs to $\mathcal{C}$ with finite delay. Moreover, if $\mathbb{G}$ belongs to $\mathcal{C}$ with delay Δ, it also belongs to $\mathcal{C}$ with bounded delay.

There exists some clear relations between those definitions and the definitions from the previous sections. A network that is strongly connected with delay 1 is a network that is strongly connected in each round. It is easy to show that its dynamic diameter is at most $n-1$. Similarly, if $\mathbb{G}$ is strongly connected with delay Δ, then its dynamic diameter is bounded by $(n-1)\Delta$. The class of networks with finite dynamic diameter is equal to the class of networks that are strongly connected with bounded delay. Moreover, the class of strongly connected networks with finite delay is equal to the class of networks in which all nodes belong to the kernel.

Centered networks with bounded delay

A graph is said to be *centered* if it contains a spanning star, that is, there exists a node i which belongs to the set of incoming neighbors of all nodes. Composing this notion and Definition 25, we obtain the class of centered network with bounded delay. The class of dynamic graphs that are centered with delay Δ can be written as:

$$\bigcap_{t\in\mathbb{N}} \left(\bigcup_{i\in[n]} \Gamma_i^{t,\Delta} \right).$$

A dynamic graph $\mathbb{G}$ which is centered with delay 1 is only composed of digraphs that contain a spanning star. The center of each star may be different from one round to another.

It is easy to see that a dynamic graph with a finite dynamic radius is also centered with delay $\mathrm{rad}(\mathbb{G})$. However, the converse is not true. Indeed, consider a network of size n. Let G_i be the star digraph centered in i, that is, for all $i \in [n]$,

$$G_i \stackrel{\text{def}}{=} ([n], \bigcup_{j\in[n]} \{(j,j),(i,j)\}.$$

The dynamic graph defined as

$$\mathbb{G} \stackrel{\text{def}}{=} G_1, G_2, G_2, G_1, G_1, G_1, G_2, G_2, G_2, G_2, G_1, G_1, G_1, G_1, G_1, \ldots$$

is centered with delay 1. However, its dynamic radius is infinite: neither 1 nor 2 are central nodes, as $\mathbb{G}$ contains arbitrarily long periods containing only G_2 (or G_1, respectively).

A connection exists between the notion of kernel and the class of centered networks with bounded delay. Indeed, if $\mathbb{G}$ is centered with delay Δ, then there exists a sequence of nodes $(i_\ell)_{\ell\in\mathbb{N}}$ such that $\mathbb{G}$ belongs to

$$\Gamma_{i_0}^{0,\Delta} \cap \Gamma_{i_1}^{1,\Delta} \cap \Gamma_{i_2}^{2,\Delta} \cap \Gamma_{i_3}^{3,\Delta} \cap \ldots$$

By the pigeonhole principle, there exists a node that occurs infinitely often in this sequence of nodes. Therefore, the kernel of $\mathbb{G}$ is non-empty.

2.6 In a nutshell

In this chapter, we introduced many notions of various flavours, such as problems, network classes, start schedules ... Those notions are grouped under the term of *features*. This section sums up the different features. The four problems that we introduced are listed in Table 2.1.

We introduced the notion of bounded memory algorithm and finite memory algorithm for a given network class. An algorithm which is neither bounded nor finite memory will be qualified as *infinite* memory, in a given network class.

We also defined several start classes of start schedules: diffusive, complete and synchronous. Recall that the diffusivity of a start schedule is relative to a dynamic graph. In particular, if the dynamic diameter of $\mathbb{G}$ is finite, then diffusive start schedules are complete. We also introduced the notion of self-stabilization. As previously explained, it is easy to show that self-stabilizing synchronization algorithms tolerate asynchronous complete start schedule. An additional possible feature is the assumption of heartbeat messages.

Finally, this thesis studies many network classes, in particular, the classes of:

1. centered networks with bounded delay;
2. dynamic networks with a finite dynamic radius;
3. dynamic networks with a dynamic radius bounded by some integer B;
4. strongly centered dynamic networks;
5. dynamic networks with a finite dynamic diameter;
6. dynamic networks with a dynamic diameter bounded by some integer B;
7. static strongly connected networks.

This list is ordered by decreasing order of inclusion.

Chapter 3

Impossibility results for mod P-synchronization

3.1 Introduction

In this chapter, we present some impossibility results that puts into perspective the results of the next chapters. These results address the issue of algorithms with *bounded memory*. In this context, is easy to notice that solving the synchronization (in $\mathbb{Z}$) problem is impossible, as the output value is an integer clock that should tend to infinity. This chapter therefore tackles the mod P-synchronization problem. In this chapter, $[n]$ denotes $\{0, 1, \ldots, n-1\}$, instead of $\{1, 2, \ldots, n\}$.

3.2 The mod P-synchronization problem in static networks

In this section, we prove that the mod P-synchronization problem cannot be solved in the class of static strongly connected networks by a self-stabilizing algorithm, with bounded memory. We reason by contradiction: we fix an algorithm ALG, with a finite set of states, and we construct an execution of ALG that does not synchronize.

To do so, we arbitrarily fix two states q^0 and q^1 of ALG and we construct a sequence of states $(q^r)_{r\in\mathbb{N}}$. Using the fact that the set of states is finite and our computational model is purely deterministic, we show that this sequence is eventually periodic. We denote L its period. The execution we build contains L nodes and the network is a directed ring, whose length is equal to L. We show that, in each round, the global state always follows a same pattern, and the system is thus never synchronized.

Theorem 27. *For any period $P > 1$, no self-stabilizing algorithm solves the mod P-synchronization problem using bounded memory in the class of static strongly connected networks.*

Proof. We fix a period $P > 1$ and a self-stabilizing algorithm $\text{ALG} = (\mathcal{Q}, \mathcal{Q}, \mathcal{M}, [P], \sigma, \tau, \omega)$. We fix two states $q^0, q^1 \in \mathcal{Q}$ and we inductively construct the sequence $(q^r)_{r\in\mathbb{N}}$. For any $r > 1$, we let

$$q^r \stackrel{\text{def}}{=} \tau\left(q^{r-1}, \left\{\left\{\sigma(q^{r-1}), \sigma(q^{r-2})\right\}\right\}\right).$$

Since $\mathcal{Q}$ is finite, this sequence is ultimately periodic. Let ℓ be a positive integers such that this sequence is periodic starting from element $\ell - 1$ and L be its period.

We construct an execution of ALG in a system of size L. We first choose the following initial state: for any node $i \in [L]$, we let

$$x_i(0) \stackrel{\text{def}}{=} q^{\ell+i}.$$

The (static) communication graph of the execution is the directed cycle:

$$G \stackrel{\text{def}}{=} \Big([L], \bigcup_{i\in[L]} \{(i,i), (i, [i+1]_L)\}\Big),$$

where $[i+1]_L$ denotes the remainder of the Euclidean division of $i+1$ by L. The global state of the system is as follows: for each node $i \in [L]$, for each round $t \in \mathbb{N}$,

$$x_i(t) = q^{\ell+i+t}. \tag{3.1}$$

The proof of this claim is by induction on t.

1. The base case follows from the definition of the initial state of the system.

2. Assume that, in a certain round t, the state of the system is given by Eq. (3.1). In round $t+1$, each node i receives a message from itself and the node $[i-1]_L$. The state of this latter node in round t is $q^{\ell+t+i-1}$. This is true even in the case $i=0$, because of the periodicity of $(q^r)_{r\in\mathbb{N}}$. Therefore

$$\begin{aligned} x_i(t+1) &= \tau\left(x_i(t), \{\{\sigma(x_i(t)), \sigma(x_{[i-1]_L}(t))\}\}\right) \\ &= \tau\left(q^{\ell+i+t}, \{\{\sigma(q^{\ell+i+t}), \sigma(q^{\ell+i+t-1})\}\}\right) \\ &= q^{\ell+i+t+1}. \end{aligned}$$

Finally, we show that the system is never synchronized. In each round t,

$$x_1(t) = q^{\ell+t+1} = x_0(t+1).$$

Therefore,

$$\omega(x_1(t)) + 1 \not\equiv_P \omega(x_0(t+1)).$$

The execution constructed in this proof is thus never synchronized. □

3.3 Extension to non-self-stabilizing algorithms

The previous section denies the existence of of a *self-stabilizing* algorithm that solves mod P-synchronization with bounded memory. A natural question is whether this results also holds for non-self-stabilizing algorithms. In this section, we provide a second impossibility result that covers diffusive start schedules (see Definition 6). Moreover, our results holds even if heartbeats (see Section 2.2) are added to the model. This new result covers dynamic networks: on this aspect, the theorem of the previous section is stronger. Overall both results are complementary.

The intuition of the proof is as follows. We consider an arbitrary algorithm ALG with a finite set of states, and we construct an execution of ALG in which the dynamic diameter is finite, but mod P-synchronization is never achieved. We choose an initial (local) state of ALG, denoted q_0^0. A single node, starting in this state and receiving no message, would update its state, according to a certain sequence $q_0^1, q_0^2, \ldots$. Using the fact that the computational model is purely deterministic, we show that this sequence is ultimately periodic. It is represented in Fig. 3.1 by the leftmost cherry-shaped subgraph. In any execution of ALG, an isolated node starting in state q_0^0 therefore remains trapped in this cycle.

We also define a sequence of states $q_1^0, q_2^0, \ldots$. The exact construction of those states is detailed later. Each of them is the starting point of an ultimately periodic sequence, similar to the one starting at q_0^0.

By default, all the "cherry tails" and the "cherry bodies" represented in Fig. 3.1 can have any length between 0 and the total number of states, denoted M. However, notice that a sequence that is periodic with a period P is also periodic with a period of any multiple of P. Similarly, a sequence that is periodic from some integer t is also periodic from any integer greater than t. We therefore define the integer H as the least common multiple of all integers between 1 and M. This integer can be used as the common length of all "cherry tails" and "cherry bodies".

We construct an execution of ALG, in which the global state of the system periodically follows a certain pattern. Fig. 3.2 represents this pattern, once the normalization process detailed above is applied[1]. In this execution, the system is composed of $2H$ nodes, gathered in two groups. In each round t which is a multiple of $2H$, the system is in a state where the nodes in first group

[1] Fig. 3.1 implies that $M \geq 10$, and hence, $H \geq \mathrm{lcm}(10, 9, \ldots, 2) = 2520$. For the sake of readability, Fig. 3.2 is drawn as if $H = 3$.

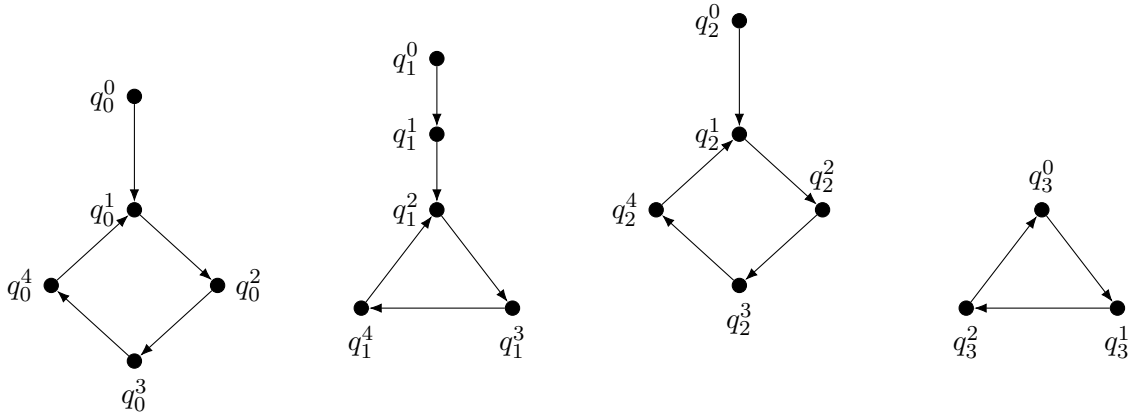


Figure 3.1: Representation of a portion of the state space of algorithm ALG. Arcs are constructed by the transition function when no outside message is received. Some states may be duplicated.

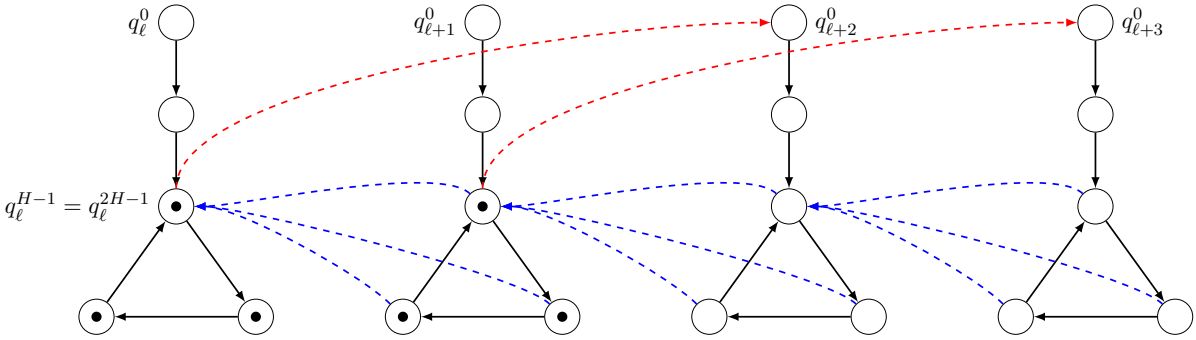


Figure 3.2: Representation of the global state that is reached every $2H$ rounds, in the execution of ALG. Each vertex represents a state, and each token represents a node. Black arcs represent the application of the transition function, similarly to Fig. 3.1. Blue arcs represent a possible set of messages that a node, in a certain state, may receive. Upon the receipt of such messages, a node updates its state according to the corresponding red arc. Some states may be duplicated.

are trapped in the loop generated by the state q_ℓ^0, for some integer ℓ, while the other nodes are trapped in the loop generated by the state $q_{\ell+1}^0$. In round $t+1$, the node 0, which begins in state q_ℓ^{2H-1}, receives a message from all the nodes belonging to the second group. Those messages are represented in Fig. 3.2 by the three leftmost blue arcs. The state transition upon the receipt of those messages is represented by the leftmost red arc. The state $q_{\ell+2}^0$ denotes the state reached by node 0 in round $t+1$. One by one, all the nodes of the first group will follow the same state transition, and will end up trapped in the loop generated by the state $q_{\ell+2}^0$. In the meantime, the nodes of the second group remain in the loop generated by $q_{\ell+1}^0$. The pattern formed by the global state in round t occurs again in round $t+2H$.

By posing $q_1^0 = q_0^0$, it is easy to construct a start schedule such that $2H$ nodes starting in state q_0^0 form the pattern in round $2H$. Moreover, the dynamic diameter of the dynamic graph is clearly finite, as all nodes in the first group periodically send a message to all nodes in the second group, and vice versa. Moreover, assuming by contradiction that ALG solves the mod P-synchronization problem, the states composing each cherry body contains all possible output values, from 0 to $P-1$. Therefore, the execution defined is never synchronized.

Theorem 28. *For any period $P > 1$, no algorithm solves the mod P-synchronization problem using bounded memory in the class of networks with a finite dynamic diameter and diffusive start*

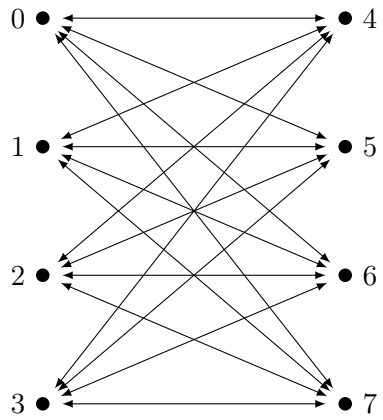


Figure 3.3: For any t, this figure represents the graph obtained by taking the union of the set of edges of the graphs $\mathbb{G}(t+1), \dots, \mathbb{G}(t+4H)$. This graph is drawn with $H = 4$, omitting self-loops. In the general case, the diameter of this graph is 2. Moreover, the set of edges of $\mathbb{G}(t+1 : t+4H)$ is a superset of the set of edges in this figure. Therefore, the dynamic diameter of $\mathbb{G}$ is at most $8H$.

schedule.

Proof. Recall that $[x]_H$ denotes the remainder of the euclidean division of x by H. We fix a period $P > 1$ and an algorithm $\text{ALG} = (\mathcal{Q}, \mathcal{I}, \mathcal{M}, [P], \sigma, \tau, \omega)$. We assume that $\mathcal{Q}$ is finite and we denote M its cardinality. We let

$$H = \operatorname{lcm}(2, 3, \dots, M), \tag{3.2}$$

where lcm is the least common multiple operator. We fix an initial state $q_0^0 \in \mathcal{I}$ and we inductively construct the family of states $(q_\ell^r)_{\ell,r\in\mathbb{N}}$.

$$q_\ell^r \stackrel{\text{def}}{=} \begin{cases} q_0^r & \text{if } \ell = 1 \\ \tau\left(q_{\ell-2}^{2H-1}, \left\{\left\{\sigma(q_{\ell-2}^{H-1}), \sigma(q_{\ell-1}^{H}), \sigma(q_{\ell-1}^{H+1}), \dots, \sigma(q_{\ell-1}^{2H-1})\right\}\right\}\right) & \text{if } \ell > 1 \text{ and } r = 0 \\ \tau\left(q_\ell^{r-1}, \left\{\left\{\sigma(q_\ell^{r-1})\right\}\right\}\right) & \text{if } r > 0. \end{cases}$$

Since $\mathcal{Q}$ is finite, for any ℓ, the sequence $q_\ell^0, q_\ell^1, \dots$ is ultimately periodic. More precisely, the sequence $q_\ell^{H-1}, q_\ell^H, q_\ell^{H+1}, \dots$ is periodic, and the length of its period is a divisor or H.

We now construct a system of $n = 2H$ nodes, and an execution of ALG in this system by choosing a start schedule and a dynamic graph. The initial state of all nodes is q_0^0. For each $\nu \in [n]$, the state of node ν in round t is denoted $x_\nu(t)$. The start schedule of the system is as follows: for each $\nu \in [n]$, we let

$$s_i \stackrel{\text{def}}{=} [i]_H + 2.$$

We now construct a dynamic graph. For each non-negative integer t, we denote ρ the quotient of the euclidean division of t by $2H$. We let

$$E_{t+1} \stackrel{\text{def}}{=} \begin{cases} \{(i, [t]_H), \mid i \in \{H, H+1, \dots 2H-1\}\} & \text{if } [\rho]_2 = 1 \text{ and } [t]_{2H} < H \\ \{(i, H + [t]_H), \mid i \in \{0, 1, \dots H-1\}\} & \text{if } [\rho]_2 = 0 \text{ and } [t]_{2H} < H \\ \emptyset & \text{otherwise.} \end{cases}$$

and we pose $\mathbb{G}(t) = ([n], \{(i,i) \mid i \in [n]\} \cup E_t)$. This dynamic graph is clearly periodic, and its period is $4H$. The dynamic diameter of this dynamic graph is less than or equal to $8H$ (see

Fig. 3.3). Moreover, the start schedule defined here is diffusive, and active nodes never have any passive incoming neighbor. Therefore, our proof holds, even when assuming that passive nodes send heartbeats.

Given this dynamic graph and the start schedule, the state of the system in each round can be computed. We show by induction that, in each round $t > 0$ that is a multiple of $2H$, it holds that, for each node $i \in [n]$,

$$x_i(t) = q_{\rho+\lambda}^{2H-1-[i]_H} \quad \text{where} \quad \rho = \frac{t}{2H} \quad \text{and} \quad \lambda = \begin{cases} -1 & \text{if } i \geq H \text{ xor } [\rho]_2 = 1 \\ 0 & \text{otherwise.} \end{cases}$$

1. Base case: $t = 2H$. During the first $2H$ rounds, no message is received by active nodes, besides their own message. The start schedule ensures that, for all nodes $i \in [n]$,

$$x_i(2H) = q_0^{2H-1-[i]_H} = q_1^{2H-1-[i]_H}.$$

2. Inductive case: we fix some t that is a multiple of $2H$, and we pose ρ and λ, similarly to the above. Without loss of generality, we assume that $[\rho]_2 = 1$: a proof in the case $[\rho]_2 = 0$ can be obtained by switching the roles of each node $i < H$ with the node $i + H$.

   By definition of $\mathbb{G}$, node 0 receives in round $t+1$ messages from the nodes $H, H+1, \ldots, 2H-1$, in addition of its own message. By the induction hypothesis, the state of node 0 in round t is $q_{\rho-1}^{2H-1}$. Moreover, the state of the nodes $H, \ldots, 2H-1$ is $q_\rho^{2H-1}, \ldots, q_\rho^H$. We obtain

$$x_0(t+1) = \tau\left(q_{\rho-1}^{2H-1}, \left\{\left\{\sigma(q_{\rho-1}^{2H-1}), \sigma(q_\rho^H), \sigma(q_\rho^{H+1}), \ldots, \sigma(q_\rho^{2H-1})\right\}\right\}\right) = q_{\rho+1}^0.$$

   We now consider a node $i \in \{1, \ldots, H-1\}$. In round $t+i$, this node reaches the state $q_{\rho-1}^{2H-1}$. Similarly to node 0, we obtain

$$x_i(t+i+1) = q_{\rho+1}^0.$$

   In round $t+H$, the state of the nodes $H, \ldots, 2H-1$ is unchanged compared to round t, because of the periodicity of the sequence $(q_\rho^r)_{r \geq 0}$. The state of each node $i \in [n]$ is therefore

$$x_i(t+H) = \begin{cases} q_{\rho+1}^{H-1-[i]_H} & \text{if } i < H \\ q_\rho^{2H-1-[i]_H} & \text{otherwise.} \end{cases}$$

   Between rounds $t+H+1$ and $t+2H$, no messages are received besides the messages sent by the nodes to themselves. We finally obtain the global state of the system in round $t+2H$, which concludes the inductive proof.

$$x_i(t+2H) = q_{\rho'+\lambda'}^{2H-1-[i]_H} \quad \text{where} \quad \rho' = \frac{t}{2H} + 1 \quad \text{and} \quad \lambda' = \begin{cases} -1 & \text{if } i \geq H \\ 0 & \text{otherwise.} \end{cases}$$

For each t that is a multiple of $2H$, in the execution that have been built, we have, for some integers ρ and λ,

$$x_0(t) = q_{\rho+\lambda}^{2H-1} = x_1(t+1).$$

Therefore,

$$\omega(x_0(t)) + 1 \not\equiv_P \omega(x_1(t+1)).$$

The execution constructed in this proof is thus never synchronized. □

3.4 Conclusion

The relative scopes of Theorems 27 and 28 is summarized in Table 3.4. This table also mentions a result by Feldmann et al. [53], which provides a mod P-synchronization algorithm in a narrower context, namely, static bidirectional connected networks. Between this result and our impossibility results, a chasm of open questions remains.

start schedule	communication graph: $D(\mathbb{G}) < \infty$ and		
	static, bidirectional	static	dynamic
synchronous start schedule	✓ (trivial)		
diffusive start schedule	✓ [53]	?	✗ (Thm. 28)
complete start schedule	?	?	✗ (Thm. 28)
self-stabilizing algorithm	?	✗ (Thm. 27)	✗ (Thm. 27 and 28)

Figure 3.4: Solvability results of the mod P-synchronization problem with bounded memory, when the dynamic diameter is finite. The cells containing a ✗ sign are covered by one of the impossibility results of this section (Theorems 27 and 28).

This leaves us with a few ways to circumvent these impossibility results. First, giving up the requirement of bounded memory. Second, assuming the knowledge of a bound on the dynamic diameter, that is, considering the class of networks with a dynamic diameter less than or equal to a certain integer B, instead of the class of networks with a finite dynamic diameter. Most existing mod P-synchronization algorithm follow this path [6, 17]. Our SAP_g algorithm, studied in Chapter 8, can follows both strategies, as explained in Section 8.3. Third, using a probabilistic framework: if we only require to solve the mod P-synchronization problem *with high probability*, then it can be solved with bounded memory. This is the case of some of the algorithms proposed by Bastide et al. [8]. Finally, notice that by Theorem 14, the mod P-firing squad problem is also covered by Theorems 27 and 28, *a fortiori*.

Chapter 4

The synchronization problem

4.1 Introduction

The objective of this chapter is the construction of a self-stabilizing algorithm solving the synchronization problem. To solve this problem, we first study the closely-related problem of stabilizing consensus. In networks with a finite dynamic diameter, both problems are easily solvable: a simple algorithm based on the update rule below is sufficient to solve the stabilizing consensus problem.

$$x_i \leftarrow \min_{j \in In} x_j \tag{4.1}$$

Using the max function instead of min would work just as well. A similar algorithm solves the synchronization problem in the same network class.

$$x_i \leftarrow 1 + \min_{j \in In} x_j \tag{4.2}$$

Therefore, a natural question is whether the synchronization problem is solvable when the dynamic diameter is infinite. This chapter relies on the MINMAX algorithm introduced by Charron-Bost et al. [27], whose objective is solving the stabilizing consensus problem while tolerating asynchronous starts. It proposes a solution in a very broad network class, that contains all networks with a finite dynamic radius, but also contains many networks with an infinite radius. Moreover, MINMAX solves the stabilizing consensus without requiring any global knowledge on the network.

Our contribution is as follows: we first present an extension of MINMAX, named BMINMAX, which mitigates an issue of memory usage growth[1]. We then introduce a general construction scheme that bridges the gap between the stabilizing consensus problem and the synchronization problem. Combining BMINMAX with this construction, we obtain a synchronization algorithm, denoted SMINMAX. This algorithm is self-stabilizing and inherits all the above-mentioned advantages of MINMAX.

4.2 The MINMAX algorithm

We recall that the definition of stabilizing consensus is provided in Definition 15. We fix a totally-ordered set $\mathcal{O}$, and each node i holds an initial value in the set $\mathcal{O}$, denoted m_i. The intuition behind the MINMAX algorithm can be explained using the notion of *view*. The *view* of a node i in round t is a tree of depth t that gathers all information that i can possibly obtain in round t. The leaves of this tree contains the initial states of the incoming neighbors of i in $\mathbb{G}(1:t)$. From this tree and those initial values, the current state of i in round t can be inductively reconstructed. More formally, the view of i in round t can be defined as the data structure constructed by the *full-information protocol*, defined by the following update rule:

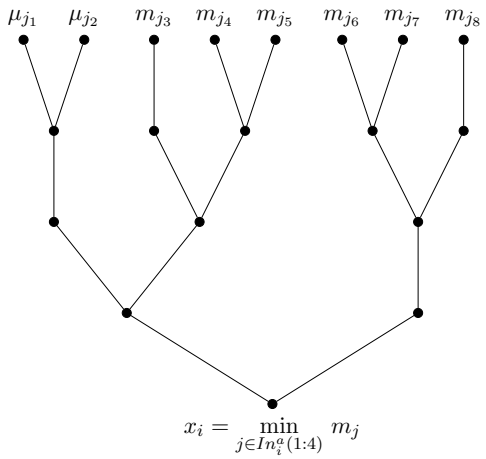
$$\mathcal{V}_i \leftarrow \mathrm{Node}(\mathcal{V}_{i_1}, \mathcal{V}_{i_2}, \dots),$$

where $i_1, i_2, \dots$ are i's incoming neighbors. Here $\mathcal{V}_i$ is a variable containing a tree that is initially reduced to a single node containing a certain initial value, and Node constructs a new tree from a collection of subtrees.

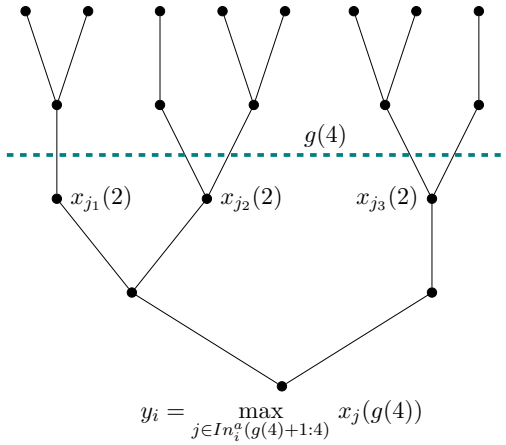
The MINMAX algorithm works as follows. Each node uses a AGE_i variable, which can be interpreted as a flattened view of i (see Fig. 4.1). Denoting

$$x_i(t) \stackrel{\text{def}}{=} \begin{cases} m_i & \text{if } i \text{ is passive in round } t \\ \min\{\lambda \in \mathcal{O} \mid AGE_i(t)[\lambda] < \infty\} & \text{otherwise,} \end{cases}$$

[1]The memory usage tends to infinity in many executions of MINMAX. This topic is further detailed in Section 19.



(a) Computation of the x_i value, in function of the initial values m_i.



(b) Update rule of the y_i variable, in function of the x_i variables and the cutoff function g.

Figure 4.1: Representation of the update rule of MINMAX in round 4, using the notion of view. In both trees, each node represents some node in a certain round. The root represents some node i in round 4, and the children of a node representing j at depth t are j's incoming neighbors in round $4 - t$.

it holds that $x_i(t)$ is equal to the minimum initial value that node i has ever heard of in round t (see[2] Lemma 30 and Fig. 4.1a). Each node also hold a variable y_i that contains the maximum x_j that i has recently heard of (see Fig. 4.1b). MINMAX is parameterized by a *cutoff function* g of type $\mathbb{N} \to \mathbb{N}^+$. The algorithms presented in this chapter rely on the following hypothesis on g.

$$\lim_{t \to \infty} g(t) = \lim_{t \to \infty} t - g(t) = \infty. \tag{4.3}$$

The role of this function is to prune the oldest values recorded in AGE_i, using an auxiliary clock h_i that measures the elapsed time. Algorithm 1 provides the pseudo-code[3] for the MINMAX$_g$ algorithm. Even if $\mathcal{O}$ is infinite, It is easy to notice that in each execution of MINMAX, only a finite number of values have a non-infinite image in each AGE_i. Therefore, the variable AGE_i can always be stored using a finite amount of memory.

Theorem 29 (Theorem 9 in [27])**.** *Assuming a complete activation schedule and a non-decreasing cutoff function g satisfying Eq. (4.3), the MINMAX$_g$ algorithm solves the stabilizing consensus problem in the class of centered networks with bounded delay.*

4.3 Avoiding infinite memory usage

The MINMAX$_g$ algorithm provides a solution to the stabilizing consensus problem in a wide range of networks. Unfortunately, both time and space complexities are problematic. On the one hand, due to the reliance on the notion of kernel, there is no bound on stabilization time. This is a fundamental limitation of the MINMAX$_g$ algorithm: the only way to tackle this issue is restricting the network class. On the other hand, in its current form, space complexity is also problematic: in a network of size n, each variable AGE_i contains up to n non-infinite entries. In

[2] the lemma is provided for BMINMAX. However, the phenomenon that is illustrated here is similar in MINMAX.
[3] As suggested in the pseudo-code, the output function of MINMAX$_g$ is simply a projection that returns y_i.

Algorithm 1: Pseudo-code of node i in MINMAX$_g$, in function of i's initial value m_i.

Initialization:
  $y_i \in \mathcal{O}$, initially m_i `/* output variable */`
  $h_i \in \mathbb{N}$, initially 0
  $AGE_i \in \mathcal{O} \to \mathbb{N} \cup \{\infty\}$, initially $AGE_i[\lambda] = 0$ if $\lambda = m_i$. Otherwise, $AGE_i[\lambda] = \infty$.
At each round:
  send $\langle AGE_i, h_i \rangle$ to all
  receive messages from a subset of nodes $In \subseteq [n]$
  for $\lambda \in \mathcal{O}$ **do**
    $AGE_i[\lambda] \leftarrow 1 + \min_{j \in In} AGE_j[j]$
  end
  $y_i \leftarrow \max\{\lambda \in \mathcal{O} \mid AGE_i[\lambda] \leq g(h_i)\}$
  $AGE_i[\min\{\lambda \in \mathcal{O} \mid AGE_i[\lambda] < \infty\}] \leftarrow 0$
  $h_i \leftarrow h_i + 1$

the worst case, in each node, $n-1$ of those entries are incremented forever. Therefore, MINMAX uses infinite memory (see Definition 8).

To alleviate this problem, we introduce the BMINMAX$_g$ algorithm (see Algorithm 2). This algorithm uses an adaptive mechanism, which increments h_i only a finite number of times. The entries of each AGE_i are pruned based on a threshold equal to $g(h_i)$. This mechanism therefore prevents the infinite growth of memory usage. We show that this algorithm solves the stabilizing consensus problem under the same assumptions as MINMAX$_g$. Although the proof scheme of BMINMAX$_g$ follows the same pattern as the proof of MINMAX$_g$, provided in [27], none of the lemmas in [27] are reusable as is. We therefore provide a complete proof of the correctness of BMINMAX$_g$.

Algorithm 2: Pseudo-code of node i in BMINMAX$_g$, in function of i's initial value m_i.

Initialization:
  $y_i \in \mathcal{O}$, initially m_i `/* output variable */`
  $h_i \in \mathbb{N}$, initially 0
  $AGE_i \in \mathcal{O} \to \mathbb{N} \cup \{\infty\}$, initially $AGE_i[\lambda] = 0$ if $\lambda = m_i$. Otherwise, $AGE_i[\lambda] = \infty$.
At each round:
  send $\langle AGE_i, h_i \rangle$ to all
  receive messages from a subset of nodes $In \subseteq [n]$
  for $\lambda \in \mathcal{O}$ **do**
    $AGE_i[\lambda] \leftarrow 1 + \min_{j \in In} AGE_j[\lambda]$
    if $AGE_i[\lambda] > g(h_i)$ **then**
      $AGE_i[\lambda] \leftarrow \infty$
    end
  end
  $y_i \leftarrow \max\{\lambda \in \mathcal{O} \mid AGE_i[\lambda] < \infty\}$
  $AGE_i[\min\{\lambda \in \mathcal{O} \mid AGE_i[\lambda] < \infty\}] \leftarrow 0$
  if $\exists j \in In, y_j > y_i$ **then**
    $h_i \leftarrow h_i + 1$
  end

We fix an execution ϵ of BMINMAX$_g$. We denote $\mathbb{G}$ its communication graph, and we assume that $\mathbb{G}$ is centered with delay Δ, for some positive integer Δ. We denote $y_i(t), AGE_i(t), \dots$ the value of the variables $y, AGE, \dots$ of node i at the end of round t, if i is active in round t. We let

$$x_i(t) \stackrel{\text{def}}{=} \begin{cases} m_i & \text{if } i \text{ is passive in round } t \\ \min\{\lambda \in \mathcal{O} \mid AGE_i(t)[\lambda] < \infty\} & \text{otherwise.} \end{cases}$$

Lemma 30. *For all rounds t, for all nodes $i \in [n]$, we have $x_i(t) = \min_{j \in In_i^a(1:t)} m_j$.*

Proof. We prove this lemma by induction over t.

1. Base case: by convention, for each node i, $In_i^a(1:0) = \{i\}$. Then $x_i(0) = m_i = \min_{j \in In_i^a(1:0)} m_j$.

2. Inductive case: we fix some node i. If i is passive in round $t+1$, then $In_i^a(1:t+1) = \{i\}$ and
$$x_i(t+1) = m_i = \min_{j \in In_i^a(1:t+1)} m_j.$$
Otherwise, i is active in round $t+1$. By the induction hypothesis, we have
$$\min_{j \in In_i^a(1:t+1)} m_j = \min_{j \in In_i^a(t+1)} \left(\min_{k \in In_j^a(1:t)} m_k \right) = \min_{j \in In_i^a(t+1)} x_j(t).$$
Let us show by two-sided inequality that $x_i(t+1) = \min_{j \in In_i^a(t+1)} x_j(t)$.

   a) We consider the node $j \in In_i^a(t+1)$ which has the minimum $x_j(t)$. By line 15, $AGE_j(t)[x_j(t)]$ is null. Then, from line 9, we obtain $AGE_i(t+1)[x_j(t)] \leq 1$. Finally, from the definition of $x_i(t+1)$,
   $$x_i(t+1) \leq x_j(t) = \min_{k \in In_i^a(t+1)} x_k(t).$$

   b) By definition of $x_i(t+1)$, we have $AGE_i(t+1)[x_i(t+1)] < \infty$. Then, using line 9, there exists some node $j \in In_i^a(t+1)$ such that $AGE_j(t)[x_i(t+1)] < \infty$. By definition of $x_i(t)$, we obtain:
   $$x_i(t+1) \geq x_j(t) \geq \min_{k \in In_i^a(t+1)} x_k(t).$$

□

By Lemma 30, for each node i, the sequence $(x_i(t))_{t \in \mathbb{N}}$ is non-increasing and lower-bounded by $\min\{m_j, j \in [n]\}$. Then this sequence stabilises in finite time. We denote x_i^* its final value. We also denote

$$x^* = \max_{i \in [n]} x_i^*.$$

We partition $[n]$ into two subsets: $\mathcal{N}_{fin}$ contains all nodes whose h variable ultimately stabilizes, whereas $\mathcal{N}_{inf}$ contains all nodes whose h variable goes to infinity as t tends to infinity. Assuming a complete activation schedule, it is easy to notice that the following four propositions, parameterized by t, are ultimately satisfied forever in the execution ϵ:

1. every node is active in round t,
2. the value of $x_j(t)$ for each node j has reached its final value x_j^*,
3. the value of $h_j(t)$ for each node $j \in \mathcal{N}_{fin}$ has reached its final value,
4. the value of $h_j(t)$ for each node $j \in \mathcal{N}_{inf}$ satisfies $g(h_j(t)) > \Delta$.

We fix some round t^0 from which each of those propositions hold forever.

Lemma 31. *Every node $\gamma \in K(\mathbb{G})$ satisfies $x_\gamma^* = x^*$*

Proof. We denote i some node satisfying $x_i^* = x^*$. We fix some node $\gamma \in K(\mathbb{G})$. By definition of K, there exists a round $t^1 > t^0$ such that $\gamma \in In_i(t^0 + 1 : t^1)$. This implies $In_\gamma^a(1 : t^0) \subseteq In_i^a(1 : t^1)$. By Lemma 30:

$$x_i(t^1) = \min_{j \in In_i^a(1:t^1)} m_j \leq \min_{j \in In_\gamma^a(1:t^0)} m_j = x_\gamma(t^0).$$

Finally, we show that $x^* = x_\gamma^*$ using a two-sided inequality proof. First,

$$x^* = x_i^* = x_i(t^1) \leq x_\gamma(t^0) = x_\gamma^*$$

holds, by Lemma 30. Second, $x^* \geq x_\gamma^*$ holds, by definition of x^*. □

Lemma 32. *If $\mathbb{G}$ is centered with delay Δ, then there exists some integer t^0 such that $\mathbb{G}$ belongs to*

$$\bigcap_{t \geq t^0} \left(\bigcup_{i \in K(\mathbb{G})} \Gamma_i^{t,\Delta} \right). \tag{4.4}$$

Proof. If $\mathbb{G}$ is centered with delay Δ, then there exists a sequence of nodes $(i_\ell)_{\ell \in \mathbb{N}}$ such that $\mathbb{G}$ belongs to

$$\Gamma_{i_0}^{0,\Delta} \cap \Gamma_{i_1}^{1,\Delta} \cap \Gamma_{i_2}^{2,\Delta} \cap \Gamma_{i_3}^{3,\Delta} \cap \dots$$

For any node i, this sequence contains an infinite number of occurrence of i if and only if i belongs to the kernel of $\mathbb{G}$. By the pigeonhole principle, the kernel of $K(\mathbb{G})$ is non-empty. We denote t^0 the round from which $(i_\ell)_{\ell \in \mathbb{N}}$ only contains elements of $K(\mathbb{G})$. The dynamic graph $\mathbb{G}$ therefore belongs to the class defined in Eq. (4.4). □

Lemma 33. *There exists an integer t^2 such that, for all integer $t \geq t^2$, all nodes i satisfy $y_i(t) \leq x^*$.*

Proof. The proof consists in fixing some value $\lambda > x^*$ and proving that, from a certain round, $AGE_i(t)[\lambda]$ is infinite in each node i. By line 14, this is a sufficient condition for $y_i(t) \leq x^*$. To do so, we first we prove that, for any integer $\ell \in \mathbb{N}$, for any node i,

$$AGE_i(t^0 + \ell)[\lambda] \geq \ell.$$

The proof is by induction over ℓ:

1. The base case is trivial as each value in each AGE_i variable is non-negative.
2. For the induction case, we fix some node i. By definition of x^*, $\lambda > x^* \geq x_i^*$, therefore, by Proposition 2, line 15 does not set $AGE_i(t^0 + \ell + 1)[\lambda]$ to 0. We consider the node $j \in In_i(t + 1)$ which holds the lowest $AGE_j(t^0 + \ell)[\lambda]$ value in round $t^0 + \ell$. By induction hypothesis, we obtain

$$AGE_i(t^0 + \ell + 1)[\lambda] \geq 1 + AGE_j(t^0 + \ell)[\lambda] \geq 1 + \ell.$$

We obtain, for all integer t,

$$AGE_i(t)[\lambda] - g(h_i(t)) \geq t - t^0 - g(t).$$

Since g satisfies $\lim_{t \leftarrow \infty} \ t - g(t) = \infty$, there exists a round from which $AGE_i(t)[\lambda] > g(h_i(t))$ holds in each node i. From this point, using line 11, $AGE_i(t)[\lambda]$ is infinite, as required. □

Lemma 34. *There exists an integer t^3 such that, for all integer $t \geq t^3$, all nodes i satisfy $y_i(t) \geq x^*$.*

Proof. We denote t^1 the integer from which Lemma 33 is satisfied forever and such that Lemma 32 holds. We fix some node i and some integer $t \geq \max(t^0, t^1)$. By Lemma 32, there exists a node $\gamma \in K(\mathbb{G})$ such that $\mathbb{G} \in \Gamma_\gamma^{t,\Delta}$.

Then, there exists a $\gamma \triangleright i$ path in the interval $[t+1, t+\Delta]$. We denote this path $\gamma = k_t, k_{t+1}, \ldots, k_{t+\Delta} = i$. We prove by induction on this path that all nodes $k_{t+\ell}$ satisfy

$$AGE_{k_{t+\ell}}(t+\ell)[x^*] \leq \ell.$$

1. By Proposition 2 and Lemma 31, we have $x_\gamma(t) = x_\gamma^* = x^*$. The base case results from line 15.

2. We fix some integer $\ell < \Delta$. The induction hypothesis implies that $AGE_{k_{t+\ell}}(t+\ell)[x^*]$ is finite. We first show that $AGE_{k_{t+\ell+1}}(t+\ell+1)[x^*]$ is finite. To do so, we only need to show that $AGE_{k_{t+\ell+1}}(t+\ell+1)[x^*]$ is not set to infinity in round $t+\ell+1$ by line 11. We reason by case distinction.

   a) If $k_{t+\ell+1} \in \mathcal{N}_{inf}$, then, using Proposition 4 and induction hypothesis, the condition line 10 is false when $\lambda = x^*$.

   b) Otherwise $k_{t+\ell+1} \in \mathcal{N}_{fin}$. By contradiction, we assume that line 11 sets the variable $AGE_{k_{t+\ell+1}}(t+\ell+1)[x^*]$ to infinity. Using this and Lemma 33, we obtain

   $$y_{k_{t+\ell+1}}(t+\ell+1) < x^*.$$

   On the other hand, the induction hypothesis implies $AGE_{k_{t+\ell}}(t+\ell)[x^*]$ is finite, and hence, by line 14,

   $$y_{k_{t+\ell}}(t+\ell) \geq x^*.$$

   Then the condition line 16 is true in round $t+\ell+1$ in node $k_{t+\ell+1}$, and line 17 is executed. We get a contradiction using Proposition 3.

   We now obtain that $AGE_{k_{t+\ell+1}}(t+\ell+1)[x^*]$ is finite, and we conclude the inductive case using line 9 and the induction hypothesis.

   $$AGE_{k_{t+\ell+1}}(t+\ell+1)[x^*] \leq 1 + AGE_{k_{t+\ell}}(t+\ell)[x^*] \leq 1 + \ell.$$

We ultimately have $AGE_i(t+\Delta)[x^*] < \infty$ for any $t > \max(t^0, t^1)$. The lemma then results from line 14. □

Theorem 35. *Assuming a complete activation schedule and a non-decreasing cutoff function g satisfying Eq. (4.3), the* BMINMAX$_g$ *algorithm solves the stabilizing consensus problem in the class of centered networks with bounded delay. Moreover,* BMINMAX$_g$ *uses finite memory in this network class.*

Proof. We fix an execution of BMINMAX$_g$ with a centered network with delay Δ. It is easy to notice that the following invariant is satisfied.

$$\forall \lambda \in \mathcal{O}, \forall t \in \mathbb{N}, \forall i \in [n], \quad AGE_i(t)[\lambda] \neq \infty \;\Rightarrow\; \exists j \in [n], m_j = \lambda. \tag{4.5}$$

The validity clause immediately follows. The agreement clause comes from Lemmas 33 and 34. Once every node has reached $y_i(t) = x^*$, the conditional statement line 16 is always false. From that point, the memory usage of each node is bounded. □

4.4 Self-stabilizing clock synchronization

The BMINMAX algorithm is not self-stabilizing: the proof of the validity clause of the stabilizing consensus relies on the fact that the system is initialized according to the pseudo-code of BMINMAX. By contrast, having a closer look at the proof of the agreement clause (Lemmas 33 and 34), it remains valid with any initial state, as long as the AGE_i variables initially contain a finite number of non-infinite values (this point is tackled later, in Section 4.4).

In this section, we take advantage of the similarity between the agreement clause of the stabilizing consensus problem and the synchronization problem. We propose a variation of BMINMAX, named SMINMAX, that solves the synchronization problem. Since there is no equivalent to the validity clause in the specification of the synchronization problem, no assumption on the initial state is needed: this new algorithm is self-stabilizing.

A general transformation mechanism

In this section, we fix a self-stabilizing algorithm ALG $= (\mathcal{Q}, \mathcal{Q}, \mathcal{M}, \mathcal{O}, \sigma, \tau, \omega)$. Our objective is as follows: assuming that ALG satisfies the agreement clause of the stabilizing consensus problem, we are going to construct a self-stabilizing algorithm SALG that solves the synchronization problem. The idea is as follows: recall the two algorithms, defined in Eq.(4.1) and (4.2). They respectively solve the stabilizing consensus problem and the synchronization problem in any network with a finite dynamic diameter. This two algorithms provide an example in which adding a $+1$ operator into a algorithm solving the stabilizing consensus problem transforms it into a synchronization algorithm. The fundamental reason explaining why such a transformation works in the case of this algorithm is as follows: for all integers a and b,

$$\min(1 + a, 1 + b) = 1 + \min(a, b). \tag{4.6}$$

In the general case, ALG needs to satisfy a certain property, that is analog to Eq. (4.6) in order to be transformed into a synchronization algorithm. More precisely, we need the existence of an *increment function*, defined as follows.

Increment function

From now, we choose $\mathcal{O} = \mathbb{N}$. A function $\iota : \mathcal{Q} \to \mathcal{Q}$ is said to be a *increment function* of ALG if, for any $q_0, q_1, q_2, \dots \in \mathcal{Q}$, the following commutativity properties are satisfied:

$$\tau(\iota(q_0), \{\sigma(\iota(q_1)), \sigma(\iota(q_2)), \dots\}) = \iota(\tau(q_0, \{\sigma(q_1), \sigma(q_2), \dots\})), \tag{C1}$$

$$\omega(\iota(q_0)) = 1 + \omega(q_0). \tag{C2}$$

Construction of a synchronization algorithm

We fix an increment function ι of ALG. We define the algorithm SALG as

$$\text{SALG} = (\mathcal{Q}, \mathcal{Q}, \mathcal{M}, \mathbb{N}, \sigma, \iota \circ \tau, \omega). \tag{4.7}$$

For any execution $\bar{\epsilon}$ of SALG, we denote ϵ a execution of ALG having the same global initial state and communication graph as $\bar{\epsilon}$. We denote $q_i(t)$ and $\bar{q}_i(t)$ the state of node i in the round t of executions ϵ and $\bar{\epsilon}$ respectively. We denote ι^t the self-composition of ι.

Lemma 36. *For any execution ϵ, for any round t and node i, we have $\bar{q}_i(t) = \iota^t(q_i(t))$*

Proof. We show this lemma by induction on t.

1. By convention, ι^0 is the identity function on $\mathcal{Q}$. Therefore, the base case results from the definition of ϵ.

2. For the inductive case, we consider a node $i \in [n]$ and a round t. Using successively the definition of $\bar{\epsilon}$, then the induction hypothesis, and property C1, we have:

$$\begin{aligned}
\bar{q}_i(t+1) &= \iota(\tau(\bar{q}_i(t), \{\sigma(\bar{q}_j(t)), j \in In_i(t+1)\})) \\
&= \iota(\tau(\iota^t(q_i(t)), \{\sigma(\iota^t(q_j(t))), j \in In_i(t+1)\})) \\
&= \iota \circ \iota^t(\tau(q_i(t), \{\sigma(q_j(t)), j \in In_i(t+1)\})) \\
&= \iota^{t+1}(q_i(t+1))
\end{aligned}$$

□

Theorem 37. *Assuming that an algorithm ALG, that outputs integers, satisfies the agreement clause of the stabilizing consensus problem in a certain network class $\mathcal{G}$, if ι is an increment function of ALG, then the algorithm SALG defined in Eq. (4.7) solves the synchronization problem in the network class $\mathcal{G}$.*

Proof. We fix an execution $\bar{\epsilon}$ of SALG. We consider the corresponding execution ϵ of ALG. By the agreement property of ALG, there exists a round t^0 from which $\omega(q_i(t))$ is stabilized on some value $v \in \mathbb{N}$. Then, using Lemma 36 and property C2, we obtain: for all nodes i, for all round $t \geq t^0$,

$$\omega(\bar{q}_i(t)) = \omega(\iota^t(q_i(t))) = t + \omega(q_i(t)) = t + v$$

The system is thus synchronized after round t^0. □

A natural question is whether the assumption of the existence of an increment function is a strong hypothesis. First, notice that an algorithm such that ω is a constant function trivially solves the agreement clause of the stabilizing consensus problem. However, no increment function exists for such an algorithm, as Proposition C2 cannot be satisfied. By contrast, our point was turning existing stabilizing consensus algorithm into synchronization algorithm. Among the stabilizing consensus algorithms we know, all of them can be equipped with an increment function. As a general rule of thumb, if an algorithm ALG solves the stabilizing consensus problem on $\mathbb{N}$ and "uses" comparison operators ($\leq$, min, ...), but does not manipulate output values with arithmetic operators ($+$, $\times$, ...), then we can reasonably expect ALG to have an increment function.

Application to BMINMAX

Let ι be the function defined as

$$\iota((y,h,AGE)) \stackrel{\text{def}}{=} (y+1,h,\lambda \mapsto AGE[\lambda-1]).$$

It is easy to notice that ι is an increment function of the BMINMAX algorithm.

In BMINMAX, the set of states contains some states in which the AGE_i variable contains an infinite number of non-infinite values. However, such a state is unreachable, thanks to initializations. In this section, we are constructing a self-stabilizing algorithm: its whole state space is reachable. Such a state would therefore be problematic, as the min or max operators would not be well-defined. To avoid this issue, we introduce the set of functions $\mathcal{F}_{fin}$.

$$\mathcal{F}_{fin} \stackrel{\text{def}}{=} \left\{ f : \mathbb{N} \to \mathbb{N} \cup \{\infty\} \mid f^{-1}(\mathbb{N}) \text{ is a finite set} \right\}.$$

The definition of SMINMAX$_g$ is given by the pseudo-code of Algorithm 3. From Theorems 35 and 37,

Algorithm 3: Pseudo-code of node i in the SMINMAX$_g$ algorithm.

Initialization:
  $x_i \in \mathbb{N}$
  $y_i \in \mathbb{N}$ `/* output variable */`
  $AGE_i \in \mathcal{F}_{fin}$
  $h_i \in \mathbb{N}$
At each round:
  send $\langle AGE_i, h_i \rangle$ to all
  receive messages from a subset of nodes $In \subseteq [n]$
  for $\lambda \in \mathbb{N}$ **do**
    $AGE_i[\lambda] \leftarrow 1 + \min_{j \in In} AGE_j[\lambda-1]$ `/* by convention,` $AGE_j[-1] = \infty$ `*/`
    if $AGE_i[\lambda] > g(h_i)$ **then**
      $AGE_i[\lambda] \leftarrow \infty$
    end
  end
  $y_i \leftarrow \max\{\lambda \in \mathcal{O} \mid AGE_i[\lambda] < \infty\}$
  $AGE_i[\min\{\lambda \in \mathcal{O} \mid AGE_i[\lambda] < \infty\}] \leftarrow 0$
  if $\exists j \in In, y_j > y_i$ **then**
    $h_i \leftarrow h_i + 1$
  end

we obtain the following corollary.

Corollary 38. *For any non-decreasing cutoff function g satisfying Eq. (4.3), the SMINMAX$_g$ algorithm solves the synchronization problem in the class of centered networks with bounded delay.*

Complexity analysis of SMINMAX

Similarly to MINMAX and BMINMAX, the stabilization time of SMINMAX cannot be bounded, as long as the network class we consider relies on the notion of kernel. Therefore, in this section, we focus on its space complexity. We fix an execution of SMINMAX, and we denote $\mathcal{V}$ the following set.

$$\mathcal{V} = \{v \in \mathbb{N} \mid \exists i \in [n], AGE_i(0)[v] \neq \infty\}.$$

Each AGE_i variable can be efficiently stored in the form of the following data structure.

$$(v_0, \{(v - v_0, AGE_i[v]) \mid AGE_i[v] \neq \infty\}) \quad \text{where} \quad v_0 = \min\{v \in \mathbb{N} \mid AGE_i[v] \neq \infty\}.$$

The first component of this tuple is an integer which is bounded by $t + \sup \mathcal{V}$. The second component of this tuple is a set, whose cardinality is bounded by n. Then, we consider any element belonging to this set. The first component is bounded by $\sup \mathcal{V} - \inf \mathcal{V}$, which is finite. The second component is bounded by

$$\sup_{i \in [n], t \in \mathbb{N}} h_i(t) \leq t_0 + \max_{i \in [n]} h_i(0),$$

where t_0 denotes the stabilization time of the considered execution. Altogether, for a fixed initial state, the space complexity of SMINMAX is $O(\log t + n \log t_0)$.

Two remarks can be made about this space complexity. First, the minimum space complexity a synchronization algorithm can possibly achieve is $\Omega(\log t)$, as each node must at least store its output clock. Second, if SMINMAX was based on MINMAX instead of BMINMAX, each $AGE_i[v]$ would have been bounded by $t + \max_{i \in [n]} h_i(0)$ instead of $t_0 + \max_{i \in [n]} h_i(0)$, and the resulting space complexity of SMINMAX would have been $O(n \log t)$.

4.5 Bibliographic notes

In this section, we present the solutions proposed for the different flavours of consensus problem. In each algorithm below, an increment function can easily be found.

Although there is a plethora of papers on agreement problems in multi-agent systems, few are specifically devoted to stabilizing consensus. To the best of our knowledge, the problem has been first investigated by Angluin, Fischer, and Jiang [5]. They studied solvability of stabilizing consensus in an asynchronous totally connected system where agents have distinct identifiers and may experience various type of faults, focusing on Byzantine faults. This problem has been studied later in [41, 9] in the synchronous gossip model. These papers propose randomized stabilizing consensus algorithms with convergence times that are functions of the number of possible input values.

The original consensus problem, with irrevocable decisions, has been the subject of much more study, specifically in the context of fault-tolerance and a fixed topology. There is also a large body of previous work on consensus in dynamic networks. In the latter works, agents are supposed to start synchronously, to share global informations on the network, and to have distinct identifiers [37]. Moreover, topology changes are dramatically restricted [12], or communication graphs are supposed to be permanently bidirectional and connected [66].

We can also mention the asymptotic consensus problem, in which the output set is $\mathbb{R}$, and the output values of all nodes must converge to a common limit. This problem has been also extensively studied as it arises in a large variety of applications in automatic control or for the modeling of natural phenomena [80]. Averaging algorithms, in which every agent repeatedly takes a weighted average of its own value and values received from its neighbors, are the natural and most widely studied algorithms for this problem. To some extend, averaging algorithms are continuous version of MINMAX. One central result by Cao, Morse, and Anderson [18] is that every safe averaging algorithm – that is, an averaging algorithm where positive weights are uniformly bounded away from zero – solves this problem with a continually rooted, time-varying topology, even if the set of roots and links change arbitrarily.

Chapter 5

Firing squad modulo P

5.1 Introduction

A classical problem in the context of asynchronous starts is the firing squad problem (see Definition 12). In some sense, solving this problem amounts to simulating synchronous starts. Unfortunately, the impossibility result in [26] demonstrates that the firing squad problem is not solvable without a strong connectivity property of the network, namely, the dynamic diameter is finite and an upper bound is known by all nodes.

Our contribution is the introduction of a weakening of the firing squad problem, named the mod P-firing squad problem (see Definition 13). We solve this problem using an algorithm, named TERMSYNCH$_P$, in the class of networks with a dynamic radius less than or equal to a bound Δ, that is, the bound Δ is known by all nodes. The main advantage of this new problem is therefore that it is solvable even in networks with an infinite dynamic diameter. In addition of the knowledge of the bound Δ, the correctness of the algorithm relies on the assumptions that $P \geq \Delta$, $P > 2$, and all passive nodes send heartbeats in each round. In Section 5.5, both assumptions on the period P are lifted. By contrast, we show that the reliance on heartbeats and the knowledge on the bound Δ cannot be avoided.

The TERMSYNCH$_P$ algorithm is quite sophisticated, and we witnessed a "combinatorial explosion" in the complexity of its proof. To increase our confidence in the correctness of our arguments, a formal proof verified by the Isabelle proof assistant has been developed. This proof closely follows the structure of our pencil-and-paper proof. An overview on the formal proof is provided at the end of the chapter.

Denoting s^{max} the earliest round in which all nodes are active, TERMSYNCH$_P$ guarantees that all nodes fire within $s^{max} + 6nP$ rounds, where n is the size of the system. However, TERMSYNCH$_P$ suffers from the same weakness as MINMAX, that is, in all of its executions, the memory usage tends to infinity. For this reason, we propose an extension of TERMSYNCH$_P$, named BTERMSYNCH$_P$, that eventually stops the memory usage growth. We show that the memory usage of BTERMSYNCH$_P$ does not exceed $\log_2(s^{max} + 6nP) + 6$.

5.2 Definition and informal description of the algorithm

We fix some $P > 2$. We define the TERMSYNCH$_P$ algorithm, which appears as Algorithm 4. Each node holds a *level* variable. When the node becomes active, it enters a level 0 state. It later moves to level 1, then to level 2. Each time a node moves from some level to the next, this constitutes a *level-up event*. From now on, the level reached during a given level-up event will be called the *strength* of this event. Reaching level 2 means firing. The conditional statements at lines 19 and 25 of Algorithm 4 are executed when the node reaches level 1 and 2 respectively. The intuition of the algorithm can be summarized by two simple ideas.

Firstly, *each node keeps track of the most recent strongest level-up event.* Only the strongest level-up events are considered: if some node "knows" about a level-up event from level 1 to level 2, it will not record any level-up event from level 0 to level 1, nor any level-up event from passive state to level 0. Among the strongest level-up events, the nodes keep track of the age of the most recent one. This defines an ordering on the set of level-up events. For that purpose, they hold two variables c_i and $force_i$. At any round, node i knows that c_i rounds ago, some node reached a level equal to $force_i$ from the previous level (as will be proved in Lemma 44). If $z_i(t)$ denotes the level-up event that node i "remembers" in round t, then Lemma 45 shows that i only remembers the strongest most recent level-up event.

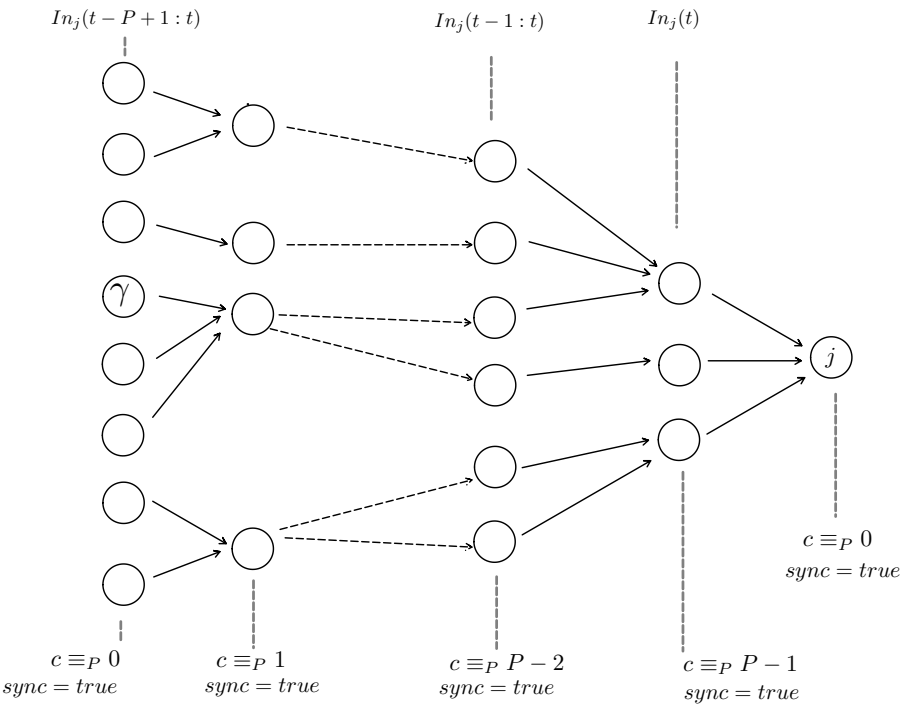
Secondly, let γ denote any central node, whose eccentricity is less than or equal to P. *A node may level up in round t only if its counter c_i is congruent to zero, and the counter of γ was also congruent to zero, P rounds ago.*

Algorithm 4: Pseudo-code of node i in the TERMSYNCH$_P$ algorithm.

Initialization:
  $c_i \in \mathbb{N}$, initially 0
  $synch_i \leftarrow false$
  $ready_i \leftarrow false$
  $force_i \in \{0, 1, 2\}$, initially 0
  $level_i \in \{0, 1, 2\}$, initially 0

At each round:
  send $\langle c_i, synch_i, force_i, ready_i \rangle$ to all
  receive incoming messages: let In^a be the set of nodes from which a non-null message is received.
  if *all received messages are non-null* **then**
    $synch_i \leftarrow \bigwedge_{j \in In^a} synch_j \wedge c_j \equiv_P c_i$
  else
    $synch_i \leftarrow false$
  end
  $ready_i \leftarrow \bigwedge_{j \in In^a} ready_j$
  $force_i \leftarrow \max_{j \in In^a} force_j$
  $c_i \leftarrow 1 + \min_{\substack{j \in In^a \\ force_j = force_i}} c_j$
  if $c_i \equiv_P 0$ **then**
    if $level_i = 0 \wedge synch_i$ **then**
      $level_i \leftarrow 1$
      if $force_i < 2$ **then**
        $force_i \leftarrow 1$
        $c_i \leftarrow 0$
      end
    else if $level_i = 1 \wedge ready_i \wedge synch_i$ **then**
      $level_i \leftarrow 2$ `/* the node` i `fires */`
      $force_i \leftarrow 2$
      $c_i \leftarrow 0$
    end
    $synch_i \leftarrow true$
    $ready_i \leftarrow level_i > 0$
  end

Figure 5.1: Impact of the state of incoming neighbors of j between round $t-P$ and t on the decision of j in round t: case where every c_i is congruent to 0 in round $t-P$.



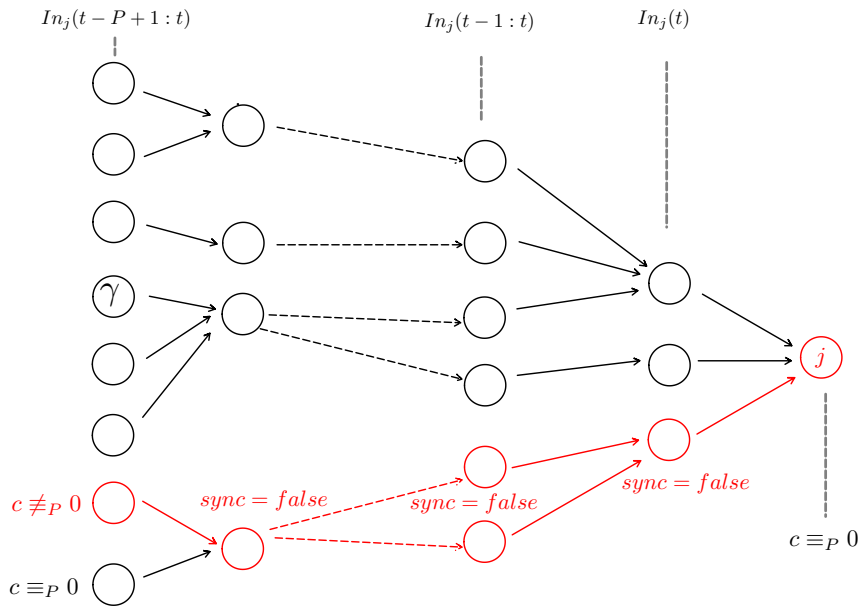
Since the nodes do not know a fixed central node, they conservatively level up only if all of their incoming neighbors $j \in In_i(t-P+1:t)$ were congruent to zero P rounds ago. By definition, γ is one of these incoming neighbors. For that purpose, they use a Boolean variable *synch*. When the counter of some node j becomes congruent to zero in some round $t-P$, it sets its $synch_j$ variable to *true* in line 30. During the next $P-1$ rounds, it will check whether the counters of its incoming neighbors are all congruent to its own counter (line 11). In case they are not, the node will set its $synch_i$ variable to *false*. This *false* value will disseminate to its outgoing neighbors (also line 11). Any node whose *synch* variable is false cannot move to the next level during the next P rounds. In contrast, if in round t, the $synch_i$ variable is still true, node i knows that no non-congruence was detected between round $t-P$ and round t. This means that every central node was congruent with zero in round $t-P$ (as will be proved in Lemma 41.b). In that case, a level-up event will take place (see Fig. 5.1). In contrast, if some node $j \in In_i(t-P+1:t)$ is not congruent to zero in round $t-P$, then the line 11 guarantees that $synch_i$ will ultimately be *false* at the beginning of round t (see Fig. 5.2). In addition to *synch*, the *ready* variable makes sure that a node i can move to level 2 only if, P rounds ago, γ was already in level 1 (as will be proved in Lemma 42). Intuitively, the round t_γ in which γ reaches level 1 is used as a landmark for the synchronization: Lemma 46 shows that nodes fire in rounds which are congruent to t_γ modulo P.

Observe that the presence of self-loops in each communication graph implies that, in the pseudo-code of Algorithm 4, the minima and maxima are well-defined.

5.3 Notation and preliminary lemmas

In the rest of this section, we fix an execution ϵ of the TERMSYNCH$_P$ algorithm over a network of size n, for a complete activation schedule $\mathbb{S}$ and a dynamic graph $\mathbb{G}$ whose radius is less than

Figure 5.2: Impact of the state of incoming neighbors of j between round $t-P$ and t on the decision of j in round t: case where some c_i are not congruent to 0 in round $t-P$.



or equal to some integer Δ, namely,

$$\exists \gamma \in [n],\, \forall t \in \mathbb{N},\, \forall i \in [n], \gamma \in In_i(t+1:t+\Delta).$$

We assume that $\Delta \leq P$. Let $s^{\max}$ be the earliest round in which all nodes are active, that is,

$$s^{max} \stackrel{\text{def}}{=} \max_{i \in [n]} s_i.$$

Note that $s^{\max} < \infty$ as the activation schedule is complete. Let γ denote some central node of $\mathbb{G}$ satisfying $\mathrm{e}_{\mathbb{G}}(\gamma) \leq \Delta$.

If the node i is active in round t, for every variable x of the algorithm, we denote the value of x_i just before i executes line 18 at round t and at the very end of round t by $x_i^{pre}(t)$ and $x_i(t)$ respectively. By extension, $x_i(t)$ refers to the initial state if $t = s_i - 1$. We now prove that this execution satisfies both properties of the mod P-firing squad problem.

We denote $In_i^a(t)$ the subset of nodes in $In_i(t)$ which are active in round $t-1$ in this execution. Some simple claims follow immediately from the definition of the transition function, regardless of the connectivity properties of $\mathbb{G}$. We consider some node $i \in [n]$ and some round t in which i is active (i.e., $t \geq s_i$).

Lemma 39.

(a) $level_i(t+1) \in \{level_i(t), level_i(t)+1\}$

(b) *If* $c_i(t) \neq 0$, *then* $force_i(t) = force_i^{pre}(t)$ *and* $c_i(t) = c_i^{pre}(t)$.

(c) $c_i(t) \equiv_P c_i^{pre}(t)$.

(d) *If* $synch_i^{pre}(t) = true$ *holds, then each node* $j \in In_i(t)$ *is active in round* $t-1$ *and satisfies* $c_j^{pre}(t-1) + 1 \equiv_P c_i^{pre}(t)$.

(e) *If* $c_i^{pre}(t) \not\equiv_P 1$ *and* $synch_i^{pre}(t)$ *holds, then each node* $j \in In_i(t)$ *is active in round* $t-1$ *and* $synch_j^{pre}(t-1)$ *holds.*

(f) *If* $c_i^{pre}(t) \not\equiv_P 1$ *and* $synch_i^{pre}(t) = ready_i^{pre}(t) = true$, *then for every node* $j \in In_i^a(t)$, *it holds that* $ready_j^{pre}(t-1) = true$.

(g) *For every* $j \in In_i^a(t)$, *we have*
$force_j^{pre}(t-1) \leq force_j(t-1) \leq force_i^{pre}(t) \leq force_i(t)$.

(h) *For every* $j \in In_i^a(t)$, *if* $force_j^{pre}(t-1) = force_i^{pre}(t)$ *then*
$c_i^{pre}(t) \leq 1 + c_j(t-1) \leq 1 + c_j^{pre}(t-1)$.

(i) $level_i(t) \leq force_i(t)$.

Proof.

(a) The value of $level_i(t+1)$ is equal to $level_i(t)$, unless line 20 or 26 is executed in round $t+1$. In that case, $level_i(t+1) = level_i(t) + 1$.

(b) If c_i is nonzero at the end of round t, then lines 23 and 28 cannot be executed during round t. Therefore, lines 22 and 27 are not executed either. Since no other lines starting at line 18 modify the variables $force_i$ or c_i, it follows that $force_i(t) = force_i^{pre}(t)$ and $c_i(t) = c_i^{pre}(t)$.

(c) The assignments in lines 21 and 25 ensure that $c_i(t)$ is 0, and they are executed only if $c_i^{pre}(t) \equiv_P 0$.

(d) Firstly, $c_j^{pre}(t-1)$ is well-defined because $In_i(t) = In_i^a(t)$ (see line 10). Moreover, the set $\{c_j(t-1), j \in In_i(t)\}$ contains integers which are mutually congruent modulo P (see line 11). Using claim 39.c and line 17

$$c_j^{pre}(t-1) + 1 \equiv_P c_j(t-1) + 1 \equiv_P c_i^{pre}(t).$$

(e) By claim 39.d, every incoming neighbor j of i is active in round $t-1$ and satisfies $c_j^{pre}(t-1) \not\equiv_P c_i^{pre}(t) - 1 \equiv_P 0$. Then the conditional statement starting in line 18 is not executed by j in round $t-1$. Then, both $synch_j(t-1)$ and $synch_j^{pre}(t-1)$ hold.

(f) Assume that $c_i(t) \not\equiv_P 1 \wedge ready_i^{pre}(t) \wedge synch_i^{pre}(t)$. Using the previous proof, every incoming neighbor j is active in round $t-1$ and the conditional statement starting in line 18 is not executed by j in round $t-1$. Finally, by line 15, j satisfies $ready_j^{pre}(t-1)$.

(g) This property follows directly from lines 16, 22 and 27.

(h) If $force_j^{pre}(t) = force_j^{pre}(t-1)$, then $force_j(t) = force_i^{pre}(t-1)$ by Lemma 39.g. Then $c_i^{pre}(t) \leq 1 + c_j(t-1) \leq 1 + c_j^{pre}(t-1)$ by line 17.

(i) We prove by induction on $t \geq s_i - 1$ that $\forall t \geq s_i - 1$, $level_i(t) \leq force_i(t)$.

a) If $t = s_i - 1$, then i is in the initial state in round t. Then $level_i(t) = force_i(t) = 0$.

b) Assume now that $level_i(t) \leq force_i(t)$. If i levels up in round $t+1$, then $level_i(t+1) \leq force_i(t+1)$ by lines 20, 22, 26. Otherwise, by Lemma 39.g and by induction hypothesis, $level_i(t+1) = level_i(t) \leq force_i(t) \leq force_i(t+1)$. □

Lemma 40. *No node can perform a level-up event action in round $P-1$ or earlier.*

Proof. We prove by induction on t that:

$$\forall t < P, \forall i \in [n], t \geq s_i - 1 \Rightarrow c_i(t) \leq t \wedge force_i(t) = 0 \wedge \neg synch_i(t).$$

1. For the base case, any node active from the first round is in initial state in round 0:

$$c_i(0) = 0 \wedge force_i(0) = 0 \wedge \neg synch_i(0).$$

2. Let t be some integer in $\{0, \ldots, P-2\}$. Let i be some node which is active in round $t+2$. Either i is in its initial state in round $t+1$, and then clearly

$$c_i(t+1) = 0 \wedge force_i(t+1) = 0 \wedge \neg synch_i(t+1).$$

Or i is active in round $t+1$. By induction hypothesis, every active incoming neighbor j of i in round $t+1$ has $force_j(t) = 0 \wedge \neg synch_j(t)$. Then

$$force_i^{pre}(t+1) = 0 \wedge \neg synch_i^{pre}(t+1).$$

Using the induction hypothesis and line 17, we have

$$t+1 \geq c_i(t) + 1 \geq c_i^{pre}(t+1).$$

Then $c_i^{pre}(t+1) \in \{1, \ldots, P-1\}$. By line 18, the variables of i are not modified in round $t+1$ after line 18. From previous claims, we obtain that

$$c_i(t+1) \leq t+1 \wedge force_i(t+1) = 0 \wedge \neg synch_i(t+1).$$

Using $\neg synch_i(t)$ and line 30, we obtain that a level-up event is impossible for any i, for $t < P$. □

We now show a few properties on the incoming neighbors of nodes that reach level 1 or 2. This situation is illustrated by Fig. 5.1.

Lemma 41. *Let ℓ be an integer, $0 \leq \ell < P$, and let i and j be two nodes such that $i \in In_j(t-P+\ell+1 : t)$. If j is active and moves to level 1 or 2 in round t, then*

(a) *i is active in round $t-P+\ell$.*

(b) *$c_i^{pre}(t-P+\ell) \equiv_P \ell$.*

(c) *If $ready_j^{pre}(t)$ is true and $i > 0$, then $ready_i^{pre}(t-P+\ell)$ is true as well.*

Proof. By Lemma 40, $t \geq P$. Let $i = k_{t-P+\ell}, \ldots, k_t = j$ denote some $i \triangleright j$ path in the interval $[t-P+\ell+1, t]$. By a backward induction, we show that, for any $\ell' \in \{\ell, \ldots, P\}$, the node $k_{t-P+\ell'}$ is active at round $t-P+\ell'$ and

$$\begin{aligned} & c^{pre}_{k_{t-P+\ell'}}(t-P+\ell') \equiv_P \ell' \\ \wedge \quad & \ell' > 0 \Rightarrow synch^{pre}_{k_{t-P+\ell'}}(t-P+\ell') \\ \wedge \quad & \ell' > 0 \wedge ready^{pre}_{\ell'}(t) \Rightarrow ready^{pre}_{k_{t-P+\ell'}}(t-P+\ell'). \end{aligned}$$

1. The base case (i.e., $\ell' = P$ and $k_{t-P+\ell'} = \ell'$) comes from lines 18, 19, and 25.

2. For the inductive case, we assume that

$$\begin{aligned} & c^{pre}_{k_{t-P+\ell'+1}}(t-P+\ell'+1) \equiv_P \ell'+1 \\ \wedge \quad & synch^{pre}_{k_{t-P+\ell'+1}}(t-P+\ell'+1) \\ \wedge \quad & ready^{pre}_{\ell'+1}(t) \Rightarrow ready^{pre}_{k_{t-P+\ell'+1}}(t-P+\ell'+1). \end{aligned}$$

By Lemma 39.d, $k_{t-P+\ell'}$ is active in round $t-P+\ell'$ and

$$c^{pre}_{k_{t-P+\ell'}}(t-P+\ell') \equiv_P \ell'.$$

Moreover, if $\ell' > 0$, we obtain $synch^{pre}_{k_{t-P+\ell'}}(t-P+\ell')$ by Lemma 39.e, and by Lemma 39.f, $ready^{pre}_{\ell'}(t)$ implies $ready^{pre}_{k_{t-P+\ell'}}(t-P+\ell')$. □

Lemma 42. *If some node j reaches level 2 in round t_j, then γ is already in level 1 in round t_j.*

Proof. Let j be some node which reaches level 2 in round t_j. By Lemma 40, $t_j \geq P$. By line 25, we have

$$c^{pre}_j(t) \equiv_P 0 \wedge synch^{pre}_j(t) \wedge ready^{pre}_j(t).$$

By Eq. 2.3, $\mathbb{G} \in \Gamma^{t_j-P,\Delta}_\gamma \subseteq \Gamma^{t_j-P,P}_\gamma$, therefore there exists a $\gamma \triangleright i$ path in the interval $[t_j-P+1, t_j]$, that we denote

$$\gamma = k_{t_j-P}, k_{t_j-P+1}, \cdots k_{t_j} = j.$$

Applying Lemma 41 with $\ell = 1$ and $i = k_{t_j-P+1}$, we obtain that k_{t_j-P+1} is active in round t_j-P+1 and $ready^{pre}_{k_{t_j-P+1}}(t_j-P+1)$ holds. Then $ready_\gamma(t_j-P)$ is true using line 15. Applying Lemma 41 a second time, with $\ell = 0$ and $i = \gamma$, we obtain that γ is active in round $t_j - P$ and $c^{pre}_\gamma(t_j - P) \equiv_P 0$. Finally, $level_\gamma(t_j - P) > 0$ using line 18 and 31. □

Lemma 43. *If γ reaches level 1 in round t_γ, no node can reach level 1 or 2 in any of the rounds $t_\gamma + 1, \ldots, t_\gamma + P - 1$.*

Proof. By Lemma 40, $t_\gamma \geq P$. We assume that some node j levels up in round $t_\gamma + h$ where $h \in \{1, \ldots, P-1\}$. By Eq. 2.3, $\mathbb{G} \in \Gamma^{t_\gamma-P+h,\Delta}_\gamma \subseteq \Gamma^{t_\gamma-P+h,P}_\gamma$, and

$$\gamma \in In_j(t_\gamma - P + h + 1 : t_\gamma + h).$$

Applying Lemma 41.b with $\ell = 0$, we get $c^{pre}_\gamma(t_\gamma - P + h) \equiv_P 0$. The presence of the self-loops implies the existence of a $\gamma \triangleright \gamma$ path in the interval $[t_\gamma - P + h + 1, t_\gamma]$. Applying Lemma 41.b with this path and $\ell = h$, we get $c^{pre}_\gamma(t_\gamma - P + h) \equiv_P h$. We get a contradiction from $h \equiv_P 0$. □

Lemma 44. *Let i be some node, and t be some round in which i is active. There exists some node w which reached a level equal to $force^{pre}_i(t)$ in round $t - c^{pre}_i(t)$. Moreover, an active $w \triangleright i$ path exists in the interval $[t - c^{pre}_i(t) + 1, t]$.*

Proof. We show this lemma by induction on $c^{pre}_i(t)$.

1. As $c^{pre}_i(t) \geq 1$ by line 17, the induction begins at $c^{pre}_i(t) = 1$. In that case, i received a message $\langle 0, *, force^{pre}_i(t), * \rangle$ from some node j (see lines 16 and 17). Then j reached a level equal to $force^{pre}_i(t)$ in round $t-1$. Because $j \in In_i(t)$, an active $j \triangleright i$ path exists in the interval $[t, t]$.

2. Let us fix some $c_i^{pre}(t) > 1$. Then, i received from some node j a message

$$\langle c_i^{pre}(t) - 1, *, force_i^{pre}(t), * \rangle.$$

From Lemma 39.b, $c_i^{pre}(t) - 1 = c_j^{pre}(t-1)$ and $force_i^{pre}(t) = force_j^{pre}(t-1)$. Applying the induction hypothesis to j in round $t-1$, we obtain some node w which reaches a level equal to $force_i^{pre}(t)$ in round $t - c_i^{pre}(t)$. We also obtain an active $w \triangleright i$ path in the interval $[t - c_i^{pre}(t) + 1, t]$. □

We define the set

$$Z \stackrel{\text{def}}{=} \{(f,t) \mid \exists i \in [n], level_i(t) = f \wedge level_i(t-1) \neq f\}. \tag{5.1}$$

This set represents the finite set of level-up events. Using Lemma 44, the tuple $z_i(t)$ defined below belongs to Z: for all node i,

$$z_i(t) \stackrel{\text{def}}{=} (force_i^{pre}(t), t - c_i^{pre}(t)) \in Z$$

in every round $t \geq s_i$ in which i is active. We order Z lexicographically. The following lemma proves that i records the most recent strongest level-up event of its view.

Lemma 45. *If there exists an active $i \triangleright j$ path between two nodes i and j in the interval $[t+1, t']$, then $z_i(t) \leq z_j(t')$. Moreover, if i reached a level equal to f in round t, then $(f,t) \leq z_j(t')$.*

Proof. Using claims 39.g and 39.h, we have, for any integer t and any node w:

$$z_w(t+1) \geq \max_{w' \in In_w^a(t+1)} z_{w'}(t). \tag{5.2}$$

Given an active $i \triangleright j$ path between i and j in the interval $[t+1, t']$, the main claim of the lemma follows from Eq. 5.2, applied to each node in this path. In the special case where i reached a level equal to f in round t, any outgoing neighbor w of i satisfies

$$z_w(t+1) \geq (f,t).$$

This inequality also comes from claims 39.g and 39.h. By Eq. 5.2, we obtain that $(f,t) \leq z_j(t')$, as required. □

Lemma 46. *If γ reaches level 1 in some round t_γ, whereas some i reaches level 1 or 2 in some round $t_i \geq t_\gamma$, then $t_i \equiv_P t_\gamma$.*

Proof. By contradiction, we consider the earliest node i which levels up in some round $t_i \geq t_\gamma$ with $t_i \not\equiv_P t_\gamma$. By Lemma 43, $t_i \geq t_\gamma + P$. There exists a $\gamma \triangleright i$ path in the interval $[t_i - P + 1, t_i]$, and this path is active by Lemma 41.a. Using Lemma 45, the self-loop of γ, and this active path, we obtain

$$(1, t_\gamma) \leq z_\gamma(t_i - P) \leq z_i(t_i). \tag{5.3}$$

Lemma 44 implies the existence of a node j which reached a level equal to $force_i^{pre}(t_i)$ in some round $t_j = t_i - c_i^{pre}(t_i)$. In the case $force_i^{pre}(t_i) = 2$, from Lemma 42, we obtain $t_j \geq t_\gamma$. Otherwise, using $(1, t_\gamma) \leq z_i(t_i)$, we also have $t_j \geq t_\gamma$.

By line 18, we have $c_i^{pre}(t_i) \equiv_P 0$. Recalling $t_j = t_i - c_i^{pre}(t_i)$, we obtain $t_j \equiv_P t_i \not\equiv_P t_\gamma$. This contradicts the fact that i was the earliest node satisfying $t_i \geq t_\gamma$ and $t_i \not\equiv_P t_\gamma$. □

Lemma 47. *If every node is active in round t, and $z_\gamma(t) = z_\gamma(t+3P)$, then γ is in level 1 in round $t+3P$.*

Proof. Let t^0 be a round in which every node is active. By Lemma 45, the sequence $(z_\gamma(t))_{t>s(\gamma)}$ is non-decreasing. By assumption, it remains constant between the rounds t and $t+3P$. Then, there exists some round $t^1 \in \{t^0, t^0+1, \dots t^0+P-1\}$ such that $c_\gamma^{pre}(t^1) \equiv_P 0$. Then we prove by induction on ℓ the following invariant:

$$\forall \ell < P, \forall i \in In_\gamma(t^1+P+\ell+1 : t^1+2P),\ c_i(t^1+P+\ell) \equiv_P \ell \\ \text{and } synch_i(t^1+P+\ell) \text{ holds}$$

1. Base case: we fix some node $i \in In_\gamma(t^1+P+1 : t^1+2P)$. By Lemma 45,

   $$z_\gamma(t^1) \geq z_i(t^1+P) \geq z_\gamma(t^1+2P).$$

   Moreover, by assumption, $z_\gamma(t^1) = z_\gamma(t^1+2P)$. Using successively Claim 39.c, the equality $z_\gamma(t^1) = z_\gamma(t^1+2P)$ and the definition of t^1, we obtain

   $$c_i(t^1+P) \equiv_P c_i^{pre}(t^1+P) \equiv_P c_\gamma^{pre}(t^1) \equiv_P 0. \tag{5.4}$$

   Moreover, $synch_i(t^1+P)$ holds by line 30.

2. Induction case: we fix some integer $\ell < P-1$ and some node i such that $i \in In_\gamma(t^1+P+\ell+2 : t^1+2P)$. In round $t^1+P+\ell+1$, every incoming neighbor j of i belongs to $In_\gamma(t^1+P+\ell+1 : t^1+2P)$, and hence, by induction hypothesis, satisfies $c_j \equiv_P \ell$ and $synch_j$ in round $t^1+P+\ell$. By lines 17 and 11, $c_i(t^1+P+\ell+1) \equiv_P \ell+1$ and $synch_i(t^1+P+\ell+1)$ holds, as required.

Finally, by choosing $\ell = P-1$, the previous invariant, lines 17 and 11 imply that $c_\gamma^{pre}(t^1+2P) \equiv_P 0$ and $synch_\gamma^{pre}(t^1+2P)$ is true. By lines 18 and 19, γ moves to level 1 in round t^1+2P at the latest. □

5.4 Correctness proof

Lemma 48. *Assuming a dynamic graph satisfying* $\mathrm{rad}(\mathbb{G}) \leq P$*, any execution of the* TERMSYNCH$_P$ *algorithm satisfies simultaneity.*

Proof. We fix some node i, and we assume that i reaches level 2 in round t_i. Let γ be a central node whose eccentricity is at most P. From Lemma 42, we obtain $t_i \geq t_\gamma$, where t_γ is the round in which γ reaches level 1. By Lemma 46, $t_i \equiv_P t_\gamma$. That proves simultaneity. □

Lemma 49. *Under the assumptions of a complete activation schedule and of a dynamic graph satisfying* $\mathrm{rad}(\mathbb{G}) \leq P$*, any execution of the* TERMSYNCH$_P$ *algorithm terminates. Moreover, every node fires* $6nP$ *rounds after the activation of all nodes, at the latest, where* n *is the size of the system.*

Proof. Recall that s^{max} denotes the round from which every node is active. Let γ be a central node whose eccentricity is at most P. Let t_γ be the round, if any, in which γ moves to level 1. The proof consists in two parts: First, we show that t_γ exists and is bounded by $t^{max} = s^{max} + 3P(2n-1)$. Then we deduce the termination property of the TERMSYNCH$_P$ algorithm.

We assume by contradiction that γ is still at level 0 in round t^{max}. By Lemma 42, each node other than γ is at most at level 1 in round t^{max}. Then, at most $2n-1$ level-up events occurred in round t^{max} and beforehand. We consider the sequence $(\hat{z}_\ell)_{\ell \in \mathbb{N}}$ such that

$$\hat{z}_\ell \stackrel{\text{def}}{=} z_\gamma(s^{max} + 3P \times \ell).$$

This sequence is non-decreasing by Lemma 45. In addition, by Lemma 47, this sequence is strictly increasing as long as γ is at level 0. We obtain a contradiction by the pigeonhole principle: the first $2n$ elements of the sequence $(\hat{z}_\ell)_{\ell\in\mathbb{N}}$ are distinct, whereas only $2n-1$ level-up events happened before round t^{max}. This ends the first part of the proof.

We achieve the second part of the proof using two invariants. First, we prove the following invariant by induction over ℓ.

$$\forall \ell \in \mathbb{N}, \forall i \in [n], c_i(t_\gamma + P + \ell) \equiv_P \ell \text{ and } \mathit{synch}_i(t_\gamma + P + \ell) \text{ holds} \tag{5.5}$$

1. Base case. By Claim 39.c and Lemma 46, as $z_\ell(t_\gamma + P) \geq z_\gamma(t_\gamma)$, we obtain

$$c_i(t_\gamma + P) \equiv_P c_i^{pre}(t_\gamma + P) \equiv_P c_\gamma^{pre}(t_\gamma) \equiv_P 0.$$

   Moreover, $\mathit{synch}_i(t_\gamma + P)$ holds by line 30.

2. The inductive case holds by lines 17 and 11, using the induction hypothesis.

Given a node i, we apply Eq. 5.5 to each of i's incoming neighbors in round $t_\gamma + 2P$. We obtain $c_i^{pre}(t_\gamma + 2P) \equiv_P 0$ and $\mathit{synch}_i^{pre}(t_\gamma + 2P)$, and hence i reaches level 1 in round $t_\gamma + 2P$ at the latest. We prove another invariant by induction over ℓ.

$$\forall \ell \in \mathbb{N}, \forall i \in [n], \mathit{ready}_i(t_\gamma + 2P + \ell) \text{ holds} \tag{5.6}$$

The base case holds by line 31, and the inductive case holds by line 15. Finally, using Eq. 5.5 and 5.6, every node fires in round $t_\gamma + 3P \leq s^{max} + 6Pn$ at the latest. □

The previous two lemmas yield the following correctness theorem:

Theorem 50. *Under the assumptions of a complete activation schedule and of a dynamic graph satisfying* $\mathrm{rad}(\mathbb{G}) \leq P$*, the* TERMSYNCH$_P$ *algorithm solves the* mod P*-firing squad problem for any integer P greater than 2. Moreover, every node fires $6nP$ rounds after the activation of all nodes, at the latest.*

5.5 Solvability results

We show that the mod P-firing squad problem is always solvable, regardless of the value of P, if the bound Δ on the delay is known: for each possible Δ, we can exhibit an algorithm which solves mod P-firing squad in any dynamic graph satisfying $\mathrm{rad}(\mathbb{G}) \leq \Delta$.

Corollary 51. *For any positive integers P and Δ, the* mod P*-firing squad problem is solvable in the class of networks with a radius less that or equal to Δ, in any complete activation schedule.*

Proof. Depending on the relative values of P and Δ, we consider the following cases:

1. $P = 1$. The simultaneity property is a tautology in this case. The problem is trivially solvable in any network class, in particular the network class considered here.

2. $\Delta \leq P$ and $P > 2$. By Theorem 50, the TERMSYNCH$_P$ algorithm solves the mod P-firing squad problem in the class of networks with a dynamic radius less that or equal to Δ.

3. $\Delta \leq P = 2$. Theorem 50 shows that the TERMSYNCH$_4$ algorithm achieves mod 4-firing squad in the class of networks with a dynamic radius less that or equal to 2, and hence achieves mod 2-firing squad in the same network class.

4. $\Delta > P$. We have $\Delta \leq \lceil \frac{\Delta}{P} \rceil \cdot P$. By Theorem 50, the mod $\lceil \frac{\Delta}{P} \rceil \cdot P$-synchronization problem is solvable in the class of networks with a dynamic radius less that or equal to Δ, using $\text{TERMSYNCH}_{\lceil \frac{\Delta}{P} \rceil \cdot P}$. The mod P-firing squad problem is also solvable in the same networks class, *a fortiori*. □

We show that the mod P-firing squad problem is not solvable if the delay Δ is unknown to the nodes. By contrast with Theorems 28 and 27, the following results also excludes algorithms with an infinite memory usage.

Theorem 52. *For any period $P > 1$, then the* mod P*-firing squad problem is not solvable in the class of networks with a finite dynamic radius and a complete activation schedule.*

Proof. By contradiction, assume that an algorithm ALG solves the problem in the above-mentioned network class. We consider any system and we fix two nodes i and j in this system. We denote I the digraph only containing self-loops. We denote C_i and C_j the digraphs only containing self-loops and a star centered in i and j respectively. We construct four executions of ALG:

1. Every node starts in round 1. The dynamic graph is equal to C_i at each round. Its radius is hence equal to 1. Using the termination of ALG, i fires in some round f_i.

2. Every node starts in round 1. The dynamic graph is equal to C_j at each round. Its radius is hence equal to 1. Using the termination of ALG, j fires in some round f_j.

3. Every node starts in round 1. During the first $f_i + f_j$ rounds, the communication graph is equal to I. In every subsequent round, the communication graph is equal to C_i. Its radius is hence equal to $1 + f_i + f_j$.

4. The node i starts in round 1, whereas every other node starts in round 2. During the first $f_i + f_j$ rounds, the communication graph is equal to I. In every subsequent round, the communication graph is equal to C_i. Its radius is hence equal to $1 + f_i + f_j$.

From the point of view of i, the third execution is indistinguishable from the first execution. Therefore, i fires in round f_i in the third execution. From the point of view of j, the third execution is indistinguishable from the second execution during the first f_j rounds. Thus, j fires in round f_j in the third execution. Using the simultaneity of ALG in the third execution, we obtain:

$$f_i \equiv_P f_j.$$

Similarly, i fires in round f_i and j fires in round $1 + f_j$ in the fourth execution. Using the simultaneity of ALG in the fourth execution, we obtain:

$$f_i \equiv_P f_j + 1.$$

Since we assumed $P > 1$, a contradiction is reached. □

Moreover, we show that the mod P-firing squad problem is not solvable if passive nodes do not send heartbeats.

Theorem 53. *In the model in which passive nodes do not send heartbeats, for any integers $P > 1$ and Δ, the* mod P*-firing squad problem is not solvable in the class of networks with a dynamic radius less than or equal to Δ and a complete activation schedule.*

Proof. By contradiction, we assume that an algorithm ALG solves the mod P-firing squad problem is the above-mentioned network class. We execute this algorithm on a network containing a simple node that starts in round 1. In this execution, this node fires in a certain round t_0. We now consider the system of size $n \geq 2$. Let G be the star graph centered in node 0, that is,

$$G \stackrel{\text{def}}{=} ([n], \bigcup_{i \in [n]} \{(i,i),(0,i)\}).$$

We construct an execution of ALG in which the communication graph is equal to G in all rounds. We choose to start node 0 in round $t_0 + 2$, node 1 in round 1 and all other nodes in round 2. As node 0 does not send heartbeats, this execution is indistinguishable from the single-node execution, from the point of view all nodes, until round $t_0 + 2$. Therefore node 1 fires in round t_0 while node 2 fires in round $t_0 + 1$, and simultaneity is violated. □

5.6 Avoiding infinite memory usage

For all nodes i and all rounds t, we have $(force_i^{pre}(t), t - c_i^{pre}(t)) \in Z$ by Lemma 44. Since Z is finite, $c_i^{pre}(t)$ tends to infinity as t tends to infinity. We present below an idea (inspired by [16]) which can alleviate this issue: in each execution of Algorithm 4, total memory usage increases forever, whereas in each execution of Algorithm 5, total memory usage grows during some arbitrarily long initial period, and then drops and remains bounded forever.

The idea is as follows: as soon as $force_i(t) = 2$, the node i knows that some node j fired in round $t - c_i(t)$ (see Lemma 44). Then i may fire in any round $t' \equiv_P t - c_i(t)$. At this point, the transition function can thus be simplified as in Algorithm 5.

Algorithm 5: Pseudo-code of node i in the BTERMSYNCH$_P$ algorithm

```
1  Initialization:
2      initialize with TERMSYNCH_P's initial state
3  At each round:
4      if force_i = 2 then
5          send ⟨c_i, true, 2, true⟩ to all
6          c_i ← [1 + c_i]_P
7          if level_i < 2 ∧ c_i = 0 then
8              level_i ← 2
9          end
10     end
11     else
12         apply TERMSYNCH_P's transition function
13     end
```

Theorem 54. *Under the assumptions of a complete activation schedule and of a dynamic graph with a dynamic radius less than or equal to P, Algorithm 5 solves the* mod P*-firing squad problem. Moreover, in each execution of Algorithm 5, the memory usage of each node is bounded by* $\log_2(s^{max} + 6nP) + 6$ *bits.*

The $\log_2(s^{max} + 6nP)$ corresponds to the amount of memory needed to store $c_i(t)$: as i fires in round $s^{max} + 6nP$ at the latest, the value of $c_i(t)$ does not exceed $s^{max} + 6nP$. All other variables can be stored using 6 bits.

5.7 Some extra formal definition

This section presents the formal proof of our algorithm. The whole formalization is available at https://github.com/louisdm31/asynchronous_starts_HO_model/tree/master.

Formalisation of the computational model

In our model, in any execution of some algorithm ALG, in each round, each node is either passive or active. In the latter case, it holds a local state in the set of state of ALG. We therefore introduce the *locState* datatype to represent the local state of a node.

datatype *'s locState = Passive | Active s*

Moreover, given a round t and two nodes i and j, from the point of view of j, there are three possibilities. Either i is active and j is one of its outgoing neighbor in round t and j receives some payload from i in round t, or i is passive in round t and sends a heartbeat, denoted *Bot* in our formalisation, or j does not belong to the set of i's outgoing neighbor. We therefore define a datatype corresponding to the data received by a node from an other node, in a given round.

datatype *'m message = Content m | Bot | Void*

An algorithm can be defined as follows, where s denotes the set of states of ALG and m its set of messages. The set of nodes is denoted *proc*.

record *('proc, 's, 'm) algorithm =*
initState :: s ⇒ bool
sendMsg :: s ⇒ m
nextState :: s ⇒ (proc ⇒ m message) ⇒ s ⇒ bool

Our formalisation was designed to handle non-anonymous algorithms (i.e., algorithms in which different nodes have different local algorithms) and coordinated algorithms (i.e., algorithms in which the local algorithms depend on a common coordinator). For the sake of readability, the code snippets presented in this section do not handle those possibilities, as the TERMSYNCH$_P$ algorithm does not take advantage of them. However, the above snippet clearly shows that our formalisation handles non-deterministic state transitions, since *nextState* is a transition relation, instead of a transition function.

An execution of an algorithm ALG is represented by a variable *rho :: nat ⇒ proc ⇒ s locState*. Moreover, given a finite set of nodes *proc*, a dynamic graph is represented by a variable *HO :: nat ⇒ proc ⇒ proc set*. The communication model is encapsulated on a *HORun* predicate: the proposition *HORun alg rho HO* if and only if *rho* is a valid execution of ALG with a dynamic graph *HO*. Notice that *HORun* does not need to take the start schedule as a separate input, as it is contained in *rho*.

Formalisation of the algorithm

As the TERMSYNCH$_P$ algorithm uses five variables, the set of states of TERMSYNCH$_P$ is defined by the following object.

record *locState =*
x :: nat
synch :: bool
ready :: bool
force :: nat — $force \in \{0, 1, 2\}$
level :: nat — $level \in \{0, 1, 2\}$

Then we provide the initial state and the transition function of the TERMSYNCH$_P$ algorithm. The argument *msgs* is a function of type *proc* ⇒ *m message* that maps each node to the message received from this node.

```
definition initState where
  initState state ≡ state = (| x = 0, synch = False, ready = False, force = 0, level = 0 |)

definition nextState :: s ⇒ (proc ⇒ s message) ⇒ s ⇒ bool where
  nextState s msgs s' ≡ s' =
    let synch_pre = (∀ p. msgs p ≠ Void ⟶
      (∃ m. msgs p = Content m ∧ synch m ∧ c m mod P = c s mod P)) in
    let ready_pre = (∀ p m. msgs p = Content m ⟶ ready m) in
    let force_pre = (Max (P_mod.forceMsgs ' range msgs)) in
    let c_pre = Suc (LEAST v. ∃ m p. msgs p = Content m ∧ force m = force_pre ∧ c m =
v) in
    if c_pre mod P = 0 then
      if level s = 0 ∧ synch_pre then
        if force_pre ≤ 1 then
          (| c = 0, synch = True, ready = True,
            force = 1, level = 1 |)
        else
          (| c = c_pre, synch = True, ready = True,
            force = force_pre, level = 1 |)
      else
      if level s = 1 ∧ synch_pre ∧ ready_pre then
        (| c = 0, synch = True, ready = True,
          force = 2, level = 2 |)
      else
        (| c = c_pre, synch = True, ready = level s > 0,
          force = force_pre, level = level s |)
    else
      (| c = c_pre, synch = synch_pre, ready = ready_pre,
        force = force_pre, level = level s |)
```

Formalisation of our results

The properties that have been formally established are as follows.

```
definition liveness where      — termination
  liveness rho t ≡ ∀ u. ∃ s. rho t u = Active s ∧ level s = 2
definition safety where      — simultaneity
  safety rho ≡ ∃ c. ∀ u t s ss.
    rho t u = Active s ⟶ level s < 2 ⟶
    rho (Suc t) u = Active ss ⟶ level ss = 2 ⟶ t mod P = c
```

Their correctness is proved under the following assumptions:

```
assumes ∀ u t. path HO gamma u t P      — gamma's eccentricity is at most P
  and ∀ u t. u ∈ HO t u                 — the graph contains self-loops
  and HORun (HOMachine P) rho HO        — rho is an execution
  and ∀ p. ∃ t. rho t p ≠ Asleep        — the schedule is complete
  and P > 2
  and OFCLASS(proc, finite_class)       — finite set of nodes
```

5.8 Bibliographic notes

On the first hand, the mod P-*firing squad* problem is clearly related to the stabilizing mod P-*synchronization* problem that has been extensively studied (e.g., see [6, 61, 17]). In the latter problem, only *eventual* synchronization is required, and nodes are not aware of the round at which synchronization is achieved (no "firing event").

On the other hand, the mod P-*firing squad* problem is a weakening of the firing squad problem, which was originally studied in the context of automata theory (e.g., [71, 72]). This model considers a finite but unknown number n of nodes which are connected in a line (or in some other specific topologies in more recent works – see e.g., [75, 36]). Nodes are identical *finite* state machines whose number of states is independent of n, and at each time unit each node changes its state according to the states of its neighbors on the line. A start signal is given to a node located at one end of the line – the "general" – and then is propagated to the rest of the nodes so that all nodes have eventually to fire simultaneously. Thus the above model assumes diffusive start signals (see Definition 6). The most recent work by Charron-Bost and Moran [26] lifts the assumption of diffusive starts and tolerates dynamic topologies. However, it requires a finite dynamic diameter, and all nodes must[1] know a bound on this diameter.

[1]This second assumption may be weakened when nodes have identifiers.

Chapter 6

The SAP algorithm: a probabilistic analysis

6.1 Introduction

In chapter 4, we introduced the SMINMAX algorithm and we showed that this algorithm solves the synchronization problem in a certain network class. However, the very specification of the synchronization problem is problematic in memory-constrained systems, as nodes must store a variable whose memory size increases forever. To address this drawback, we consider the mod P-synchronization problem.

The previous chapters rely on a *deterministic* communication model, in which each execution of a certain algorithm is considered individually, and the *property* corresponding to the mod P-synchronization problem must be satisfied by each possible execution in this model. By contrast, in a *probabilistic* model, the set of executions of an algorithm is considered as a whole, and the correctness then corresponds to the following *hyperproperty* [34]: the probability of executions of the considered algorithm in which mod P-synchronization is achieved is greater than a chosen real $p \in [0, 1)$.

For probabilistic communication models, the mod P-synchronization problem has been addressed by Boczkowski et al. [14] and later on by Bastide et al. [8], both in the particular framework of the PULL model [65] through the fully-connected graph: In each round each agent receives a message from an agent sampled uniformly at random. Their focus is on minimizing message size and both obtain a stabilization time of $O(\log n)$ in a network of size n. Unfortunately, the algorithms in both papers are specific to the PULL model, and their good performances highly rely on the assumption of a fully-connected network.

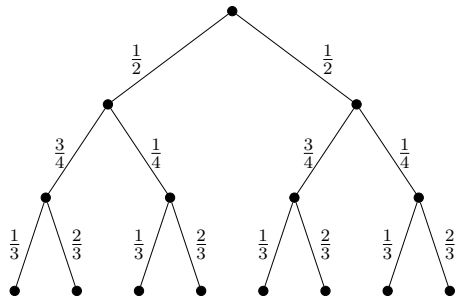
To obtain a solution that solves the mod P-synchronization problem in a more general class of networks, we developed novel proof strategies and new analysis tools to bridge the gap between deterministic and probabilistic proofs. Our goal is using an algorithm and a proof scheme developed in the deterministic framework, and transposing it in the probabilistic framework. Unfortunately, verifying probabilistic hyperproperties is technical. In particular, the notion of eccentricity, which is central in the previous chapters, is no more relevant in a probabilistic framework. Indeed, we may see that in most of probabilistic networks, each eccentricity is almost surely infinite (cf. Section 6.3). As a substitute of dynamic diameters, we devise new parameters for probabilistic dynamic graphs, namely, a hierarchy of *probabilistic diameters*, and prove several basic properties on these diameters that are interesting on their own. Using those probabilistic diameters, we define the notion of *strong connectivity with high probability*.

To validate the efficiency of our approach, we introduced an algorithm, denoted SAP, which is an extension of an algorithm introduced by Boldi and Vigna [16]. Their algorithm was presented for static strongly connected networks. Our main contribution is the proof that SAP solves the mod P-synchronization problem with high probability, assuming that the network is strongly connected w.h.p.

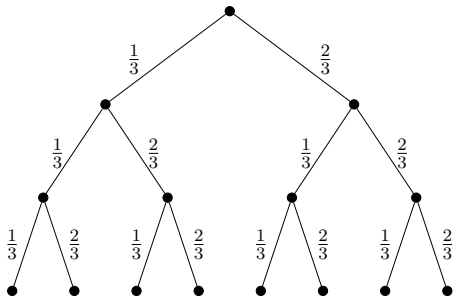
As the SAP solves the mod P-synchronization problem in both deterministic and probabilistic framework, SAP is covered by Theorem 27, and its memory usage is therefore unbounded. We only guarantee that the memory usage of SAP is finite in the class of networks that are studied in this chapter. By contrast, using a purely probabilistic approach, Bastide et al. [14, 8] introduce some mod P-synchronization algorithms with bounded memory.

6.2 Probabilistic framework

If Σ_n denotes the Borel σ-algebra on $\mathcal{G}_n$, then $(\mathcal{G}_n, \Sigma_n)$ is a measurable space. Then, we consider a probability measure on $(\mathcal{G}_n, \Sigma_n)$, denoted $\Pr_n$, or simply Pr. The pair $(\mathcal{G}_n, \Pr_n)$ is called a *probabilistic communication network* of size n.



(a) An example of memoryless probability measure.



(b) An example of memoryless and time-independent probability measure.

Figure 6.1: Two trees representing two probability measures, truncated at height 3. For the sake of readability, only two children per node are represented, instead of $2^{n(n-1)}$.

In previous chapters, $\mathbb{G}(\ell)$ used to denote the ℓ-th graph of a specific dynamic graph $\mathbb{G}$. In this chapter, $\mathbb{G}(\ell)$ can also be viewed as a random variable, which maps each dynamic graph of $\mathcal{G}_n$ to its ℓ-th component. By analogy with the terminology used in game theory (e.g., see [13]), we say that Pr is *memoryless* if the random variables $\mathbb{G}(1), \mathbb{G}(2), \mathbb{G}(3), \ldots$ are mutually independent. We say that Pr is *time-independent*[1] if those random variables are identically distributed.

Given a deterministic algorithm ALG, a positive integer n, an execution of ALG is characterized by an initial global state in $\mathcal{Q}^n$ and a communication graph in $\mathcal{G}_n$. Therefore, given a global state $q \in \mathcal{Q}^n$ of ALG, the set of executions of ALG starting at q, denoted $\mathcal{E}_q(\text{ALG})$, is isomorphic to $\mathcal{G}_n$. Hence, Pr induces a probabilistic measure on $\mathcal{E}_q(\text{ALG})$, which will also be denoted by Pr as no confusion can arise.

The set $\mathcal{G}_n$ can be represented by an infinite tree in which the nodes of depth ℓ are the sequences of length ℓ of dynamic graphs of size n. Each edge therefore corresponds to one of the $2^{n(n-1)}$ possible digraph of size n and each node has $2^{n(n-1)}$ children. A probability measure Pr on $\mathcal{G}_n$ can be represented by a labeling[2] of this tree, defined as follows: each edge between a node $G_1, \ldots, G_\ell$ and one of its children $G_1, \ldots, G_\ell, G_{\ell+1}$ is labeled by the real number

$$\Pr(\ \mathbb{G}(\ell+1) = G_{\ell+1} \mid \mathbb{G}(1) = G_1 \wedge \cdots \wedge \mathbb{G}(\ell) = G_\ell\ ).$$

Giving such a labeling amounts to providing a probability distribution on each branching of the tree representing $\mathcal{G}_n$. Figure 6.1 gives two example of such representations.

6.3 Probabilistic diameters

Let us fix a global state $q \in \mathcal{Q}^n$. For any real $p \in [0,1]$ and any integer $k \in [n]$, we define the *probabilistic order k diameter* as the minimum number of rounds required for k arbitrary nodes to communicate with all the nodes in the network with probability at least p. Formally, we let

$$\hat{D}^{(k)}(p) \stackrel{\text{def}}{=} \inf\{\delta \in \mathbb{N}^+ \mid \inf_{i_1,\cdots,i_k \in [n], t \in \mathbb{N}} \Pr(\Gamma^{t,\delta}_{i_1} \cap \cdots \cap \Gamma^{t,\delta}_{i_k}) \geq p\}.$$

Clearly, we have that $\hat{D}^{(1)}(p) \leq \cdots \leq \hat{D}^{(n)}(p)$, with equalities when $p = 1$. As a matter of fact, the probabilistic proof of the SAP algorithm that we develop in the following sections only involves the probabilistic diameters $\hat{D}^{(1)}(p)$ and $\hat{D}^{(2)}(p)$.

[1] is this terminology relevant?

[2] should I further detail the relation between Pr and the set of possible labelings?

A toy example

As an example, let us consider the memoryless probability measure Pr on $\mathcal{G}_2$ defined by

$$\Pr(\mathbb{G}(t) = G_1) = \Pr(\mathbb{G}(t) = G_2) = \frac{1}{2},$$

where G_1 and G_2 are the two-node digraphs defined in Figure 6.3.

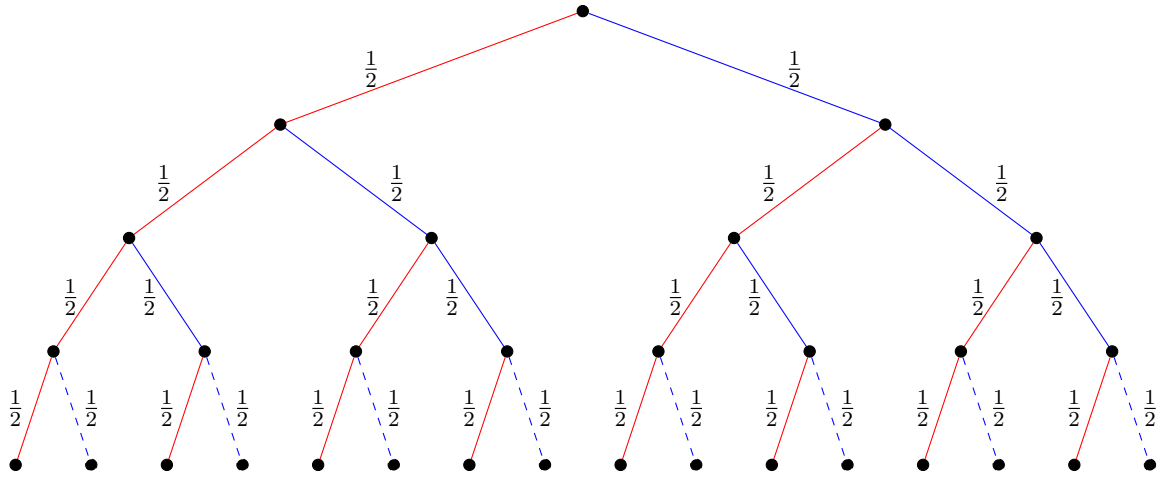


Figure 6.2: Representation of the probability measure Pr defined above, truncated at height 4. The graph G_1 is represented by a red arc, and G_2 by a blue one. Other possible graphs are not represented. The event $\Gamma_i^{2,4}$ is the set of dynamic graphs containing no dashed arc.

For any round t and any positive integer δ, we have:

$$\Pr(\Gamma_1^{t,\delta}) = 1 - \Pr\left(\bigcap_{d=1}^{\delta} (\mathbb{G}(t+d) = G_2)\right) = 1 - \prod_{\ell=1}^{\delta} \Pr(\mathbb{G}(t+d) = G_2) = 1 - 2^{-\delta}.$$

Similarly, $\Pr(\Gamma_2^{t,\delta}) = 1 - 2^{-\delta}$. Moreover,

$$\begin{aligned}
\Pr(\Gamma_1^{t,\delta} \cap \Gamma_2^{t,\delta}) &= 1 - \Pr(\overline{\Gamma_1^{t,\delta}} \cup \overline{\Gamma_2^{t,\delta}}) \\
&= 1 - \Pr\left(\bigcap_{d=1}^{\delta} (\mathbb{G}(t+d) = G_2) \cup \bigcap_{d=1}^{\delta} (\mathbb{G}(t+d) = G_1)\right) \\
&= 1 - 2^{-\delta+1}.
\end{aligned}$$

Using the definition of the probabilistic diameters and the two equations above, we obtain $\hat{D}^{(1)}(p)$ and $\hat{D}^{(2)}(p)$ in our example:

$$\begin{aligned}
\hat{D}^{(1)}(p) &= \inf\{\delta \in \mathbb{N}^+ \mid 1 - 2^{-\delta} \geq p\} = \lceil -\log_2(1-p) \rceil \text{ and} \\
\hat{D}^{(2)}(p) &= \inf\{\delta \in \mathbb{N}^+ \mid 1 - 2^{-\delta+1} \geq p\} = 1 + \lceil -\log_2(1-p) \rceil.
\end{aligned}$$

This simple example shows why it is not appropriate to use the parameter $\hat{D}(p)$ simply defined by:

$$\hat{D}(p) \stackrel{\text{def}}{=} \inf\{\delta \in \mathbb{N}^+ \mid \Pr(D(\mathbb{G}) \leq \delta) \geq p\}, \tag{6.1}$$

where $D(\mathbb{G})$ is the dynamic diameter of $\mathbb{G}$, defined in Eq. (20). Indeed, for each node $i \in \{1, 2\}$, we have:

$$\Pr(e_{\mathbb{G}}(i) \leq \delta) \leq \Pr\left(\bigcap_{\ell=0}^{\infty} \Gamma_i^{\ell\delta,\delta}\right) = \prod_{\ell=0}^{\infty} \Pr(\Gamma_i^{\ell\delta,\delta}) = 0,$$

(a) digraph G_1 (b) digraph G_2

Figure 6.3: Two digraphs.

and thus $\hat{D}(p)$ is infinite if p is positive. In other words, the dynamic diameter of almost all dynamic graphs is infinite in this example, while $\hat{D}^{(1)}(p)$ is finite when $p < 1$.

Relations between probabilistic diameters

We now state some general properties on the probabilistic diameters.

Lemma 55. *For any memoryless probability measure and all real numbers $p \in \left[\frac{1}{2}, 1\right]$, if $\hat{D}^{(1)}(p)$ is finite, then $\hat{D}^{(2)}(p)$ is finite and $\hat{D}^{(2)}(p) \leq 2\hat{D}^{(1)}(p)$.*

Proof. Because of the self-loops, the digraphs $\mathbb{G}(t+1 : t+\delta)$ and $\mathbb{G}(t+\delta+1 : t+2\delta)$ are both subgraphs of $\mathbb{G}(t+1 : t+2\delta)$, and hence $\Gamma_i^{t,\delta} \cup \Gamma_i^{t+\delta,\delta} \subseteq \Gamma_i^{t,2\delta}$. It follows that:

$$\begin{aligned}
\Pr\left(\Gamma_i^{t,2\hat{D}^{(1)}(p)} \cap \Gamma_j^{t,2\hat{D}^{(1)}(p)}\right) &\geq 1 - \Pr\left(\overline{\Gamma_i^{t,2\hat{D}^{(1)}(p)}}\right) - \Pr\left(\overline{\Gamma_j^{t,2\hat{D}^{(1)}(p)}}\right) \\
&\geq 1 - \Pr\left(\overline{\Gamma_i^{t,\hat{D}^{(1)}(p)}} \cap \overline{\Gamma_i^{t+\hat{D}^{(1)}(p),\hat{D}^{(1)}(p)}}\right) \\
&\quad - \Pr\left(\overline{\Gamma_j^{t,\hat{D}^{(1)}(p)}} \cap \overline{\Gamma_j^{t+\hat{D}^{(1)}(p),\hat{D}^{(1)}(p)}}\right) \\
&\geq 1 - 2(1-p)^2.
\end{aligned}$$

The second inequality holds because of the above-proved inclusion, and the third one because of the memoryless assumption. If $p \in \left[\frac{1}{2}, 1\right]$, then $1 - 2(1-p)^2 \geq p$ and $\hat{D}^{(2)}(p) \leq 2\hat{D}^{(1)}(p)$. □

Using a similar proof, it is possible to show the following lemma.

Lemma 56. *For any memoryless probability measure and all real numbers $p \in \left[\frac{1}{2}, 1\right]$, for all positive integers ℓ_1 and $\ell_2 \leq \ell_1$ such that $\ell_1 + \ell_2 \leq n$, if $\hat{D}^{(\ell_1)}(p)$ and $\hat{D}^{(\ell_2)}(p)$ are finite, then $\hat{D}^{(\ell_1+\ell_2)}(p)$ is finite and $\hat{D}^{(\ell_1+\ell_2)}(p) \leq 2\hat{D}^{(\ell_1)}(p)$.*

Therefore, by induction on $\ell \leq n$, if $\hat{D}^{(1)}(p)$ is finite, then all $\hat{D}^{(\ell)}(p)$ are finite and

$$\hat{D}^{(\ell)}(p) \leq 2^{\lceil \log_2 \ell \rceil} \hat{D}^{(1)}(p).$$

Finally, we prove the following finiteness result for $\hat{D}^{(1)}(p)$.

Lemma 57. *For any memoryless probability measure, if $\hat{D}^{(1)}(p_0)$ is finite for some $p_0 \in (0, 1]$, then $\hat{D}^{(1)}(p)$ is finite for all real numbers $p \in [0, 1)$.*

Proof. For every node i, for every integers $t \geq 0$ and $\ell > 0$, we have

$$\begin{aligned}
\Pr\left(\Gamma_i^{t,\ell\hat{D}^{(1)}(p_0)}\right) &\geq \Pr\left(\bigcup_{h=0}^{\ell-1} \Gamma_i^{t+h\hat{D}^{(1)}(p_0),\hat{D}^{(1)}(p_0)}\right) \\
&= 1 - \prod_{h=0}^{\ell-1} \Pr\left(\overline{\Gamma_i^{t+h\hat{D}^{(1)}(p_0),\hat{D}^{(1)}(p_0)}}\right) \\
&\geq 1 - (1-p_0)^{\ell}.
\end{aligned}$$

The first inequality holds because of the self-loops, as explained in the proof of Lemma 55. The second one is due to the memoryless assumption on the Pr probability measure.

If p_0 is positive, then $\lim_{\ell\to\infty} 1-(1-p_0)^\ell = 1$. Thus, for any real number p less than one, there exists some integer ℓ_0 such that $\Pr(\Gamma_i^{t,\ell_0\hat{D}^{(1)}(p_0)}) \geq p$, which implies that $\hat{D}^{(1)}(p)$ is finite and $\hat{D}^{(1)}(p) \leq \ell_0\hat{D}^{(1)}(p)$. □

Since all $\hat{D}^{(\ell)}$ are non-decreasing, Lemmas 55 and 57 imply that if $\hat{D}^{(1)}(p_0)$ is finite for some $p_0 \in (0,1]$, then all probabilistic diameters are finite for all $p \in [0,1)$, in which case the network $(\mathcal{G}_n, \Pr_n)$ is said to be *strongly connected with high probability*. This notion is linked to the notion of dynamic diameter. Assume that a dynamic graph $\mathbb{G}$ has a finite dynamic diameter $D_{\mathbb{G}}$. Let Pr be the probability measure such that $\Pr(\{\mathbb{G}\}) = 1$, then for all real numbers $p \in (0,1]$,

$$D_{\mathbb{G}} = \hat{D}^{(1)}(p) = \hat{D}^{(2)}(p) = \cdots = \hat{D}^{(n)}(p).$$

6.4 The SAP algorithm

We now present the self-stabilizing SAP algorithm and state its basic properties.

Description of the algorithm

A typical approach to solve the mod P-synchronization problem consists in the following algorithm: at each round, each node sends its own variable $C_i \in \{0,\ldots,P-1\}$ and applies the following update rule:

$$C_i \leftarrow \big[\min_{j\in \mathrm{In}_i} C_j + 1\big]_P,$$

where In_i denotes the current set of i's incoming neighbors, and $[c]_P$ is the remainder of the Euclidean division of c by P. Unfortunately, this naive algorithm does not work[3] when $\hat{D}^{(1)}(p)$ is too large compared to the period P. To overcome this problem, the SAP algorithm uses self-adaptive periods and the basic fact that for any positive integer M, we have

$$\big[\,[c]_{PM}\,\big]_P = [c]_P.$$

More precisely, each node i uses two *integer* variables M_i and C_i, and computes the clock value C_i not modulo P, but rather modulo the time-varying period PM_i. The variable M_i is used as a guess to find a large enough multiple of P so to make the clocks eventually stabilized. Until synchronization, the variables M_i increase so that the shortest period PM_i eventually becomes large enough compared to the largest clock value in the network. In the rest of this chapter, $\mathcal{S}_t$ denotes the set of executions in which the system is synchronized in round t. Once all clocks are congruent modulo P, they remain congruent forever, meaning that $\mathcal{S}_t \subseteq \mathcal{S}_{t+1}$. The update rule for M_i is parametrized by a function $g : \mathbb{N} \to \mathbb{N}$. The corresponding algorithm is denoted SAP_g, and its code is given below. Line 7 in the pseudo-code implies that $C_i(t) < PM_i(t)$, and for the sake of simplicity, we assume that this inequality also holds initially, that is, $C_i(0) < PM_i(0)$.

In the rest of the chapter, the function g is supposed to be a non-decreasing and *inflationary* function, i.e., $x < g(x)$ for every positive integer x. Therefore, each M_i variable is non-decreasing. If ℓ is a positive integer, g^ℓ denotes the ℓ-th iterate of g. For every positive real number x, we let

$$g^*(x) \stackrel{\text{def}}{=} \inf\{\ell \in \mathbb{N}^+ \mid g^\ell(1) \geq x\}.$$

Since g is inflationary, $g^*(x)$ is finite for all integers x, and $g^*(x) \leq x$.

[3] see Theorem 4.13 in [2].

Algorithm 6: Pseudo-code of node i in the SAP_g algorithm

1 **Variables:**
2 $C_i \in \mathbb{N}$
3 $M_i \in \mathbb{N}^+$
4 **At each round:**
5 send $\langle C_i, M_i \rangle$ to all
6 receive $\langle C_{j_1}, M_{j_1} \rangle, \langle C_{j_2}, M_{j_2} \rangle, \dots$ from the set In_i of incoming neighbours
7 $C_i \leftarrow \left[\min_{j \in \text{In}_i} C_j + 1\right]_{PM_i}$
8 $M_i \leftarrow \max_{j \in \text{In}_i} M_j$
9 **if** $C_j \not\equiv_P C_{j'}$ *for some* $j, j' \in \text{In}_i$ **then**
10 $M_i \leftarrow g(M_i)$
11 **end**

Properties of SAP_g's executions

Let us consider an execution ϵ of the SAP_g algorithm over a network of size n, with the dynamic graph $\mathbb{G}$. We start with five basic properties of ϵ which directly come from the pseudo-code.

Lemma 58. *If (i, j) is an arc in $\mathbb{G}(s:t)$, then $C_j(t) \leq C_i(s-1) + t - s + 1$.*

Lemma 59. *If (i, j) is an arc in $\mathbb{G}(s:t)$, then one of the following statements is true:*

1. $C_j(t) \equiv_P C_i(s-1) + t - s + 1$;
2. $M_j(t) \geq g(M_i(s-1))$.

Lemma 60. *Let d be a positive integer. If $C_i(t) + d \leq PM_i(t)$ holds for all nodes i, then all the clocks C_i are greater than 0 in the round interval $[t+1, t+d-1]$.*

Proof. Let i be any node, and let $\ell \in [d-1]$. We have

$$1 + \min_{j \in \text{In}_i(t+\ell)} C_j(t+\ell-1) \leq 1 + C_i(t+\ell-1) \leq \ell + C_i(t) < PM_i(t) \leq PM_i(t+\ell-1).$$

The first inequality is due to the self-loop at node i in $\mathbb{G}(t+\ell)$, the second one is a consequence of the self-loop and Lemma 58, the third inequality is the assumption of the lemma. The fourth one comes from the fact that M_i is non-decreasing. It follows from line 7 that $C_i(t+\ell) \neq 0$. □

Lemma 61. *If the clock value of the agent i is non-null at round t, then it is equal to*

$$C_i(t) = 1 + \min_{j \in \text{In}_i(t)} C_j(t-1).$$

Proof. Let i^+ be a node satisfying $\min_{j \in \text{In}_i(t)} C_j(t-1) = C_{i^+}(t-1)$. The lemma just relies on the following series of inequalities:

$$C_{i^+}(t-1) \leq C_i(t-1) \leq PM_i(t-1) - 1.$$

The last inequality is clear for $t = 1$, and for $t \geq 2$, it is a consequence of $C_i(t-1) \leq PM_i(t-2) - 1$ and of the fact that M_i is non-decreasing. Moreover, by line 7, one of those inequalities is strict, since $C_i(t)$ is non-null. Thus, by line 7, Eq. (61) holds. □

For the probabilistic correctness proof of SAP_g, we will use another property of its executions, stated in the lemma below.

Lemma 62. *Let d be any positive integer, and k be a node such that $C_k(t) = \min_{j\in[n]} C_j(t)$. If the execution ϵ belongs to $\Gamma_k^{t,d}$ and all the clocks C_i are greater than 0 in the round interval $[t+1, t+d-1]$, then the network is synchronized in round $t+d$.*

Proof. We fix an execution σ and a positive integer d. First, we prove by induction on $\ell \in [d-1]$ that

$$\forall i \in [n], \quad C_i(t+\ell) = \ell + \min_{j\in \text{In}_i(t+1:t+\ell)} C_j(t). \tag{6.2}$$

1. The base case $\ell = 1$ is an immediate consequence of Lemma 61.

2. Inductive step: let us assume that Eq. (6.2) holds for some ℓ with $1 \leq \ell < d-1$. For every node i in $[n]$, we have

$$\begin{aligned} C_i(t+\ell+1) &= 1 + \min_{j\in \text{In}_i(t+\ell+1)} C_j(t+\ell) \\ &= 1 + \ell + \min_{j\in \text{In}_i(t+\ell+1)} \left(\min_{j'\in \text{In}_i(t+1:t+\ell)} C_{j'}(t) \right) \\ &= 1 + \ell + \min_{j\in \text{In}_i(t+1:t+\ell+1)} C_j(t). \end{aligned}$$

   The first equality is a direct consequence of Lemma 61, the second one is by inductive hypothesis, and the third one is due to the fact that $\mathbb{G}(t+1 : t+\ell+1) = \mathbb{G}(t+1 : t+\ell) \circ \mathbb{G}(t+\ell+1)$.

This completes the proof of Eq (6.2) for every integer $\ell \in [d-1]$.

Then for each node i, we get

$$\begin{aligned} C_i(t+d) &= \left[1 + \min_{j\in \text{In}_i(t+d)} C_j(t+d-1) \right]_{PM_i(t+d-1)} \\ &= \left[d + \min_{j\in \text{In}_i(t+1:t+d)} C_j(t) \right]_{PM_i(t+d-1)} \\ &= \left[d + C_k(t) \right]_{PM_i(t+d-1)}. \end{aligned}$$

The second equality comes from a reasoning similar to the inductive case above, using Eq (6.2) at round $t+d-1$. It follows that all the counters $C_i(t+d)$ are equal modulo P, i.e., the system is synchronized in round $t+d$. □

6.5 Probabilistic correctness of SAP_g

Our approach for the correctness proof of the SAP_g algorithm relies on a fundamental probabilistic hyperproperty relating the adaptive mechanism for the periods in SAP_g to the order one probabilistic diameter of the network.

We fix some integer n, some real $p \in (0,1)$ and some initial state $q \in \mathcal{Q}^n$ of SAP_g. We consider a memoryless probability measure Pr on $(\mathcal{G}_n, \Sigma_n)$, and so on the set $\mathcal{E}_q$ of SAP_g's executions starting in q. We assume that the probabilistic network $(\mathcal{G}_n, \Pr_n)$ is strongly connected w.h.p., and we let

$$t_0 \stackrel{\text{def}}{=} \hat{D}^{(2)}(p) \left\lceil \frac{\log(1-p)^{-1}}{p} \left(\sqrt{g^*\Big(\frac{2\hat{D}^{(1)}(p)}{P}\Big)} + \sqrt{2} \right)^2 \right\rceil \tag{6.3}$$

which is finite since g is inflationary. For all positive integers t, we consider the random variable $M(t)$ defined as

$$M(t) \stackrel{\text{def}}{=} \min_{i \in [n]} M_i(t).$$

Lemma 63. *For every real number $p \in (0,1)$, we have*

$$\Pr\left(\left(M(t_0) \geq \frac{2\hat{D}^{(1)}(p)}{P}\right) \cup \mathcal{S}_{t_0}\right) \geq p. \tag{6.4}$$

Proof. For ease of notation, we let $\bar{g} = g^*\left(\frac{2\hat{D}^{(1)}(p)}{P}\right)$ and $\ell_0 = t_0/\hat{D}^{(2)}(p)$. In the first part of the proof, we construct a family of independent random variables B_t that all follow a Bernoulli distribution. In each execution in $\mathcal{E}_q$ that is not synchronized at round t, there exist two nodes $i_1(t)$ and $i_2(t)$ such that

$$C_{i_1(t)}(t) \not\equiv_P C_{i_2(t)}(t). \tag{6.5}$$

Then i_1 and i_2 can be viewed as two random variables that map any execution of SAP_g to a sequence of type $\mathbb{N} \to [n]$. Let B_t be the random variable equal to 1 on $\Gamma_{i_1(t)}^{t,\hat{D}^{(2)}(p)} \cap \Gamma_{i_2(t)}^{t,\hat{D}^{(2)}(p)}$ and equal to 0, otherwise. By definition of $\hat{D}^{(2)}(p)$, each B_t follows a Bernoulli distribution whose parameter $\Pr(B_t = 1)$ is greater than or equal to p. Since Pr is memoryless, the random variable

$$B \stackrel{\text{def}}{=} \sum_{\ell=0}^{\ell_0 - 1} B_{\ell\hat{D}^{(2)}(p)}$$

is a sum of independent Bernoulli variables.

We now show that in all executions in $\mathcal{E}_q$ that are not synchronized in round $\ell_0\hat{D}^{(2)}(p)$, it holds that

$$M(\ell_0\hat{D}^{(2)}(p)) \geq g^B(1). \tag{6.6}$$

For that, we fix such an execution and prove by induction on ℓ_0 that Eq. (6.6) holds in this execution.

1. Base case: $\ell_0 = 0$. Then we have $B = 0$, and so $M(\ell_0\hat{D}^{(2)}(p)) \geq 1 = g^B(1)$ as needed.

2. Inductive case: Assume that

$$M(\ell_0\hat{D}^{(2)}(p)) \geq g^{\sum_{t=0}^{\ell_0-1} B_{t\hat{D}^{(2)}(p)}}(1)$$

holds for for some $\ell_0 \in \mathbb{N}$ and that the system is not synchronized in round $(\ell_0+1)\hat{D}^{(2)}(p)$. Then, the nodes $i_1 = i_1(\ell_0\hat{D}^{(2)}(p))$ and $i_2 = i_2(\ell_0\hat{D}^{(2)}(p))$ satisfy Eq. (6.5). Therefore, for every node i, there exists some $x \in \{1,2\}$ such that

$$C_i((\ell_0+1)\hat{D}^{(2)}(p)) \not\equiv_P C_{i_x}(\ell_0\hat{D}^{(2)}(p)) + \hat{D}^{(2)}(p). \tag{6.7}$$

If $B_{\ell_0\hat{D}^{(2)}(p)} = 0$, then the inductive case immediately follows. Otherwise, $B_{\ell_0\hat{D}^{(2)}(p)} = 1$, and so the digraph $\mathbb{G}(\ell_0\hat{D}^{(2)}(p) + 1 : (\ell_0+1)\hat{D}^{(2)}(p))$ contains all the arcs of the form (i_1, i) and (i_2, i). Then for every node i, it holds that

$$M_i((\ell_0+1)\hat{D}^{(2)}(p)) \geq g(M_{i_x}(\ell_0\hat{D}^{(2)}(p))) \geq g(M(\ell_0\hat{D}^{(2)}(p))). \tag{6.8}$$

The first inequality holds by Lemma 59 and Eq. (6.7), and the second one because g is non-decreasing. Using the induction hypothesis, we get

$$M((\ell_0+1)\hat{D}^{(2)}(p)) \geq g\left(g^{\sum_{t=0}^{\ell_0-1} B_{t\hat{D}^{(2)}(p)}}(1)\right) = g^{\sum_{t=0}^{\ell_0} B_{t\hat{D}^{(2)}(p)}}(1).$$

We now let $x_0 = \frac{1}{p}\Big(\bar{g} + \log(1-p)^{-1} + \sqrt{2\bar{g}\log(1-p)^{-1} + \log^2(1-p)^{-1}}\Big)$, and easily check that

$$\ell_0 \geq \frac{1}{p}(\sqrt{\bar{g}} + \sqrt{2\log(1-p)^{-1}})^2 \geq \frac{1}{p}(\bar{g} + 2\log(1-p)^{-1} + 2\sqrt{2\bar{g}\log(1-p)^{-1}}) \geq x_0 > 0. \quad (6.9)$$

Moreover, x_0 satisfies

$$-\frac{x_0 p}{2}\left(1 - \frac{\bar{g}}{x_0 p}\right)^2 = \log(1-p). \quad (6.10)$$

We obtain

$$\begin{aligned}
\Pr\left(\left(M(t_0) \geq \frac{2\hat{D}^{(1)}(p)}{P}\right) \cup \mathcal{S}_{t_0}\right) &\geq \Pr\left(B \geq g^*\Big(\frac{2\hat{D}^{(1)}(p)}{P}\Big)\right) \\
&\geq 1 - \Pr\left(B \leq \frac{\bar{g}}{x_0 p}\,\mathbb{E}(B)\right) \\
&\geq 1 - e^{-\frac{\mathbb{E}(B)}{2}\left(1-\frac{\bar{g}}{x_0 p}\right)^2} \\
&\geq 1 - e^{-\frac{x_0 p}{2}\left(1-\frac{\bar{g}}{x_0 p}\right)^2} \\
&= p.
\end{aligned}$$

The first inequalities comes from Eq. (6.6). The second and the fourth inequalities hold because, by definition of B and Eq. (6.9), we have $\mathbb{E}(B) \geq \ell_0 p \geq x_0 p$. The third inequality is a Chernoff bound [31] applied to B, which is a sum of independent Bernoulli variables. The last equality is by Eq. (6.10). □

Combined with the basic properties of the SAP$_g$'s executions stated in the previous section, Lemma 63 allows us to show the main result of this chapter.

Theorem 64. *If g is a non-decreasing and inflationary function, then the SAP$_g$ algorithm solves the mod P-synchronization problem in any probabilistic network that is strongly connected w.h.p. More precisely, for all $p \in (0,1)$, nodes synchronize within*

$$\hat{D}^{(2)}(p)\left\lceil \frac{\log(1-p)^{-1}}{p}\left(\sqrt{g^*\Big(\frac{2\hat{D}^{(1)}(p)}{P}\Big)} + \sqrt{2}\right)^2\right\rceil + 3\hat{D}^{(1)}(p)$$

rounds with probability p^4, if $\hat{D}^{(1)}$ and $\hat{D}^{(2)}$ denote the order one and two probabilistic diameters of the network.

Proof. For ease of notation, we let $\hat{D}^{(1)} = \hat{D}^{(1)}(p)$. We first define four random variables:

1. Let i_0 be any node satisfying $C_{i_0}(t_0) = \min_{i\in[n]} C_i(t_0)$.

2. Let t_1 be the smallest integer greater than or equal to t_0, such that at least one node holds a clock equal to 0 in round t_1 if it exists, or is equal to infinity otherwise.

3. If t_1 is finite, let i_1 be any node such that $C_{i_1}(t_1) = 0$. Otherwise, let i_1 be an arbitrary node.

4. If t_1 is finite, let i_2 be any node satisfying $C_{i_2}(t_1 + \hat{D}^{(1)}) = \min_{i\in[n]} C_i(t_1 + \hat{D}^{(1)})$. Otherwise, let i_2 be an arbitrary node.

Then we define the events E and E' as

$$E \stackrel{\text{def}}{=} \{\epsilon \in \mathcal{G}_n \mid t_1 < t_0 + \hat{D}^{(1)}\} \qquad \text{and} \qquad E' \stackrel{\text{def}}{=} \{\epsilon \in \mathcal{G}_n \mid M(t_0) \geq \tfrac{2\hat{D}^{(1)}}{P}\}.$$

By Lemma 62, we have $\Gamma_{i_0}^{t_0,\hat{D}^{(1)}} \cap \overline{E} \subseteq \mathcal{S}_{t_0+\hat{D}^{(1)}} \subseteq \mathcal{S}_{t_0+3\hat{D}^{(1)}}$, and hence

$$\Pr\left(\mathcal{S}_{t_0+3\hat{D}^{(1)}} \mid \Gamma_{i_0}^{t_0,\hat{D}^{(1)}} \cap \overline{E} \cap (E' \cup \mathcal{S}_{t_0})\right) = 1. \tag{6.11}$$

In any execution belonging to E, t_1 is finite and $C_{i_1}(t_1) = 0$. Therefore, in any execution in $E' \cap E \cap \Gamma_{i_1}^{t_1,\hat{D}^{(1)}}$, every variable M_i satisfies

$$PM_i(t_1 + \hat{D}^{(1)}) \geq PM(t_1 + \hat{D}^{(1)}) \geq PM(t_0) \geq 2\hat{D}^{(1)} = C_{i_1}(t_1) + 2\hat{D}^{(1)} \geq C_i(t_1 + \hat{D}^{(1)}) + \hat{D}^{(1)}.$$

The first and third inequalities above are by definition of M and E', respectively. The second one holds because M is non-decreasing, and the last one comes from Lemma 58 and the fact that the execution is in $\Gamma_{i_1}^{t_1,\hat{D}^{(1)}}$. Lemma 60 then applies, and Lemma 62 shows that

$$E' \cap E \cap \Gamma_{i_1}^{t_1,\hat{D}^{(1)}} \cap \Gamma_{i_2}^{t_1+\hat{D}^{(1)},\hat{D}^{(1)}} \subseteq \mathcal{S}_{t_1+2\hat{D}^{(1)}}.$$

Since the random variable t_1 is greater than t_0, we get $\mathcal{S}_{t_0} \subseteq \mathcal{S}_{t_1+2\hat{D}^{(1)}}$, and so

$$\Pr\left(\mathcal{S}_{t_1+2\hat{D}^{(1)}} \mid (E' \cup \mathcal{S}_{t_0}) \cap E \cap \Gamma_{i_1}^{t_1,\hat{D}^{(1)}} \cap \Gamma_{i_2}^{t_1+\hat{D}^{(1)},2\hat{D}^{(1)}} \cap \Gamma_{i_0}^{t_0,\hat{D}^{(1)}}\right) = 1. \tag{6.12}$$

We are now in position to bound the probability $\Pr(\mathcal{S}_{t_0+3\hat{D}^{(1)}})$ from below. For the sake of readability, the conditional probability given the event $(E' \cup \mathcal{S}_{t_0}) \cap \Gamma_{i_0}^{t_0,\hat{D}^{(1)}}$ is now denoted $\Pr'$. Then we have

$$\begin{aligned}
\Pr(\mathcal{S}_{t_0+3\hat{D}^{(1)}}) &\geq \Pr(\mathcal{S}_{t_0+3\hat{D}^{(1)}} \cap \Gamma_{i_0}^{t_0,\hat{D}^{(1)}} \cap (E' \cup \mathcal{S}_{t_0})) \\
&= \Pr'(\mathcal{S}_{t_0+3\hat{D}^{(1)}}) \times \Pr(\Gamma_{i_0}^{t_0,\hat{D}^{(1)}} \mid E' \cup \mathcal{S}_{t_0}) \times \Pr(E' \cup \mathcal{S}_{t_0}) \\
&\geq p^2 \Pr'(\mathcal{S}_{t_0+3\hat{D}^{(1)}}) \\
&= p^2 \Pr'(\mathcal{S}_{t_0+3\hat{D}^{(1)}} \mid E) \Pr'(E) + p^2 \Pr'(\mathcal{S}_{t_0+3\hat{D}^{(1)}} \mid \overline{E}) \Pr'(\overline{E}) \\
&\geq p^2 \Pr'(\mathcal{S}_{t_1+2\hat{D}^{(1)}} \mid E) \Pr'(E) + p^2 \Pr'(\overline{E}) \\
&\geq p^2 \Pr'(\mathcal{S}_{t_1+2\hat{D}^{(1)}} \cap \Gamma_{i_1}^{t_1,\hat{D}^{(1)}} \cap \Gamma_{i_2}^{t_1+\hat{D}^{(1)},\hat{D}^{(1)}} \mid E) \Pr'(E) + p^2 \Pr'(\overline{E}) \\
&\geq p^2 \Pr'(\mathcal{S}_{t_1+2\hat{D}^{(1)}} \mid E \cap \Gamma_{i_1}^{t_1,\hat{D}^{(1)}} \cap \Gamma_{i_2}^{t_1+\hat{D}^{(1)},\hat{D}^{(1)}}) \\
&\quad \times \Pr'(\Gamma_{i_2}^{t_1+\hat{D}^{(1)},\hat{D}^{(1)}} \mid E \cap \Gamma_{i_1}^{t_1,\hat{D}^{(1)}}) \\
&\quad \times \Pr'(\Gamma_{i_1}^{t_1,\hat{D}^{(1)}} \mid E) \Pr'(E) + p^2 \Pr'(\overline{E}) \\
&\geq p^4 \Pr'(E) + p^2 \Pr'(\overline{E}) \\
&\geq p^4
\end{aligned}$$

Lemma 63 and the fact that Pr is memoryless are used in the second inequality. Third inequality is based on Eq. (6.11) and the fact that, in any execution in E, it holds that $\mathcal{S}_{t_1+2\hat{D}^{(1)}} \subseteq \mathcal{S}_{t_0+3\hat{D}^{(1)}}$. Sixth inequality relies on Eq. (6.12) and the fact that Pr is memoryless. □

Memory complexity

We first state a theorem that will be used to bound the memory usage, measured in bits, of each node in any execution of SAP_g in which mod P-synchronization is achieved.

Theorem 65. *In any execution of SAP_g that achieves mod P-synchronization, if h is the round in which the system synchronizes, then the memory usage of each node is less than* $\log_2 P + 2\log_2\left(g^h\left(\max_{i\in[n]} M_i(0)\right)\right)$ *bits.*

Proof. We define, for each round t,

$$\overline{M}(t) \stackrel{\text{def}}{=} \max_{i\in[n]} M_i(t).$$

From the pseudo-code of SAP_g, we directly obtain, for each positive integer t,

$$\overline{M}(t) \leq g\left(\overline{M}(t-1)\right),$$

and thus,

$$\overline{M}(t) \leq g^t\left(\overline{M}(0)\right). \tag{6.13}$$

As $\overline{M}(t)$ is non-decreasing as long as $t \leq h$ and is stable afterwards, each M_i belongs to the interval $\{1,\ldots,\overline{M}(h)\}$, and each C_i belongs to $\{0,\ldots,P\overline{M}(h)-1\}$. The number of reachable states by any single node is at most equal to the cardinality of the product of these two sets, that is, $Pg^h\left(\overline{M}(0)\right)^2$. Then at most $\log_2 P + 2\log_2\left(g^h\left(\overline{M}(0)\right)\right)$ bits are needed to store the state of one node. □

The bound on the stabilization time SAP_g therefore provides an upper bound on its space complexity, namely each node uses at most

$$\log_2 P + 2\log_2\left(g^{t_0+3\hat{D}^{(1)}(p)}(\overline{M}_0)\right)$$

bits with probability p^4, if t_0 is defined by Eq. (6.3) and $\overline{M}_0 = \max_{i\in[n]} M_i(0)$. The time bound and the space bound thus depend respectively on the functions g^* and g, leading to a time-space trade-off for choosing g: the faster g grows, the lower the synchronization time is, and the higher its space complexity is.

6.6 Bibliogaphic notes

Self-stabilizing clocks have been extensively studied in different communication models and under different assumptions, In particular, clocks may be unbounded, in which case they are required to be eventually equal, instead of only congruent. The synchronization problem of unbounded clocks admits simple solutions in strongly connected networks, namely the *Min* and *Max* algorithms [50, 59].

Periodic clocks require more sophisticated synchronization mechanisms. In addition to strong connectivity and static networks, the pioneering papers on periodic clock synchronization [6, 62, 17, 2] all assume that a bound on the diameter is available. Then Boldi and Vigna [16] proposed a synchronization algorithm, based on a self-adaptive period mechanism, that dispenses with the latter assumption.

More recently, periodic clock synchronization has been studied in the *Beeping model* [35] in which agents have severely limited communication capabilities: given a connected bidirectional communication graph, in each round, each agent can either send a "beep" to all its neighbors

or stay silent. A self-stabilizing algorithm has been proposed by Feldmann et al. [53], which is optimal both in time and space, but which, unfortunately, requires that a bound on the network size is available for each agent.[4]

There are also numerous results for mod P-synchronization with faulty agents. The fault-tolerant solutions that have been proposed in various failure models, including the Byzantine failure model, use algorithmic schemes initially developed for consensus (e.g., see [45, 46]). They typically require a bidirectional connected (most of the time fully-connected) network.

For probabilistic communication models, the problem of clock synchronization has been addressed by Boczkowski et al. [14] and later on by Bastide et al. [8], both in the particular framework of the PULL model [65] through the fully-connected graph: In each round each agent receives a message from an agent sampled uniformly at random. Their focus is on minimizing message size and both obtain a stabilization time of $O(\log n)$ in a network of size n. Unfortunately, the algorithms in both papers are specific to the PULL model, and their good performances highly rely on the assumption of a fully-connected network.

Clock synchronization has been studied in another probabilistic communication model, namely, the model of *population protocols*, consisting of a set of agents, interacting in randomly chosen pairs. This is basically an asynchronous model, where the synchronization task is quite different from the one studied in this document since it resumes to implement the abstraction of rounds [4]. In other words, the point in the population protocol model is to achieve synchronization in *frequency* instead of synchronization in *phase*.

[4] In [53], Feldmann et al. also proposed an algorithm that does not use any bound on the network size, but that only tolerates asynchronous starts.

Chapter 7

Probabilistic diameters in PUSH and PULL models

7.1 Introduction

The previous chapter demonstrates that the notion of probabilistic diameter is a powerful tool for the probabilistic analysis of distributed algorithms. In this chapter, we pick some popular probabilistic communication networks, and we try to obtain a value of $\hat{D}^{(1)}(p)$ in such networks. There are two possible approaches: first, using a theoretical analysis, an upper bound on $\hat{D}^{(1)}(p)$ can be obtain. Such an upper bound is usually asymptotic, that is, it holds for sufficiently large networks. Second, $\hat{D}^{(1)}(p)$ may be computed exactly on any fixed probabilistic communication network. This chapter explores both approaches.

The PUSH and PULL models have been extensively studied with various base networks. In this section, we pick two particular cases of probabilistic networks and we provide upper bound on $\hat{D}^{(1)}(p)$ by taking advantage of the literature. First, Feige et al. [52] introduced the notion of *almost sure rumor coverage time* and provided a general upper bound on this parameter. They deal with several types of probabilistic models, including the PUSH model in bidirectional networks. Second, Doerr and Kostrygin [43] provide a far-reaching result that notably covers the case of fully-connected networks. Using those two papers as examples, we show that a bound on $\hat{D}^{(1)}(p)$ can easily be obtain from the literature on rumor spreading.

Finally, we compute the value of $\hat{D}^{(1)}(p)$ in the PUSH and PULL models in a fully-connected network for some values of p and for n up to 200. Unfortunately, those probabilistic communication models are one of the few cases in which the computation of $\hat{D}^{(1)}(p)$ is reasonably efficient: in this case, the time complexity of the computation of each $\hat{D}^{(1)}(p)$ is polynomial in the number of nodes. This is not the case for most probabilistic communication models. We compare those experimental results with the theoretical bounds previously presented.

7.2 The PUSH and PULL models in a general bidirectional network

In the PUSH communication model, Feige et al. showed that a rumor reaches the n nodes of a bidirectional and connected network within $12n \log_2 n$ rounds with probability $1-1/n$ (Theorem 2.1 in [52]). In other words, the order one probabilistic diameter satisfies:

$$\hat{D}^{(1)}(1 - 1/n) \leq 12n \log_2 n.$$

Having a close look at the proof provided by Feige et al., it appears that this bound also holds in the PULL model.

7.3 The PUSH and PULL models in fully-connected networks

In the particular case of fully-connected networks, Frieze and Grimmett [56] and later Pittel [77] provided a bound on the time complexity of rumor spreading in the PUSH model. The case of the PULL model in fully-connected networks has been studied by Doerr and Kostrygin [43]. All the previously-mentioned results are asymptotic. This is why the resulting bound on $\hat{D}^{(1)}(p)$ will be proved to hold only in sufficiently large networks.

Let us briefly recall the main result of Doerr and Kostrygin [43]: They define the random variable T that denotes the rumor spreading time in fully-connected networks. More precisely, fixing a node i_0, for each dynamic graph $\mathbb{G}$ of size n, we let

$$T_n(\mathbb{G}) \stackrel{\text{def}}{=} \inf\{\delta \in \mathbb{N} \mid \mathbb{G} \in \Gamma_{i_0}^{0,\delta}\}.$$

Since the network is fully-connected, all nodes play the same role and the probability distribution of T_n does not actually depend on the choice of the origin node i_0.

Theorem 66 (Table 1 in [43]). *In the* PUSH *model in fully-connected static networks, it holds that*

$$\mathbb{E}(T_n) = \log_2 n + \log n \pm O(1).$$

In the PUSH *model, it holds that*

$$\mathbb{E}(T_n) = \log_2 n + \log_2 \log n \pm O(1).$$

Moreover, in each case, there exists two positive real numbers A and α such that, for all positive real r, for all positive integer n,

$$Pr_n(|T_n - \mathbb{E}(T_n)| \geq r) \leq Ae^{-\alpha r}.$$

To obtain a bound on $\hat{D}^{(1)}(p)$, we choose any $\kappa : \mathbb{N} \to \mathbb{N}$ that tends to infinity and a real $p \in (0,1)$. We first consider the case of the PUSH model. For sufficiently large integers n, it holds that

$$Ae^{-\alpha\frac{\kappa(n)}{2}} \leq 1 - p \quad \text{and} \quad \mathbb{E}(T_n) \leq \log_2 n + \log n + \frac{\kappa(n)}{2}.$$

It follows that for every $p \in [0,1)$ and every such function κ, there exists a positive integer $N_\kappa(p)$ such that for all integers $n \geq N_\kappa(p)$, it holds that

$$\begin{aligned}
\Pr_n\big(T_n \leq \log n + \log_2 n + \kappa(n)\big) &\geq \Pr_n\Big(T_n \leq \mathbb{E}(T_n) + \frac{\kappa(n)}{2}\Big) \\
&\geq \Pr_n\Big(\Big|T_n - \mathbb{E}(T_n)\Big| \leq \frac{\kappa(n)}{2}\Big) \\
&\geq 1 - Ae^{-\alpha\frac{\kappa(n)}{2}} \\
&\geq p.
\end{aligned}$$

In the PUSH model, all random variables $\mathbb{G}(t)$ are identically distributed. Hence, for all nodes i, and all non-negative integers t and δ, we have $\Pr(\Gamma_i^{t,\delta}) = \Pr(\Gamma_{i_0}^{0,\delta})$, and thus for all integers $n \geq N_\kappa(p)$,

$$\hat{D}^{(1)}(p) = \inf\big\{\delta \in \mathbb{N} \mid \Pr(\Gamma_{i_0}^{0,\delta}) \geq p\big\} \leq \log_2 n + \log n + \kappa(n). \tag{7.1}$$

Similarly, in the PULL model, we have, under the same conditions,

$$\hat{D}^{(1)}(p) \leq \log_2 n + \log_2 \log n + \kappa(n). \tag{7.2}$$

7.4 Application to SAP

Using Lemma 55, choosing $\kappa = \log_2$ and using some simplifications based on the inequalities $(1 - 1/n)^4 \geq 1 - 4/n$, $(1 + \sqrt{2})^2 \leq 6$ and $g^* \geq 1$, Theorem 64 then yields the following result for bidirectional networks.

Corollary 67. *Let g be any non-decreasing and inflationary function. For the* PUSH *and* PULL *models in a general bidirectional connected network with n nodes, the SAP_g algorithm achieves mod P-synchronization within $348\, n\, (\log_2 n)^2 g^*(24\, P^{-1}\, n \log_2 n)$ rounds with probability $1 - \frac{4}{n}$.*

For fully-connected networks, we obtain the following corollary of Theorem 64 using Eq. (7.1).

Corollary 68. *Let g be a non-decreasing inflationary function. For any real number $p \in [\frac{1}{2}, 1)$ and any integer $n \geq N_{\log_2}(p)$, the SAP_g algorithm achieves mod P-synchronization within $81\, \log(1-p)^{-1}\, (\log_2 n)\, g^*\left(6\, P^{-1}\, \log_2 n\right)$ rounds with probability p^4 in the fully-connected graph of size n and the communication* PUSH *model.*

In the context of Corollaries 67 and 68, Tables 7.1 and 7.2 provide the probabilistic time and space complexities of SAP_g for two different choices of g, namely $g = x \mapsto x+1$ and $g = x \mapsto 2x$. Recall that $\overline{M}_0$ denotes $\max_{i\in[n]} M_i(0)$. Both tables illustrate the general space-time trade-off that we have just pointed out, at the end of Section 6.5.

g	stabilization time	space complexity
$g = x \mapsto x+1$	$O\left(\log(1-p)^{-1}\log^2 n\right)$	$O\Big(\log\left(\overline{M}_0 + \log(1-p)^{-1}\log n\right)\Big)$
$g = x \mapsto 2x$	$O\left(\log(1-p)^{-1}\log n \log\log n\right)$	$O\Big(\log \overline{M}_0 + \log(1-p)^{-1}\log n \log\log n\Big)$

Table 7.1: Complexity of the SAP_g algorithm for the PUSH and PULL models in a fully-connected network of size n, with probability p^4.

g	stabilization time	space complexity
$g = x \mapsto x+1$	$O\left(n^2\log^3 n\right)$	$O\Big(\log\left(\overline{M}_0 + n\right)\Big)$
$g = x \mapsto 2x$	$O\left(n\log^3 n\right)$	$O\Big(\log \overline{M}_0 + n\log^3 n\Big)$

Table 7.2: Complexity of the SAP_g algorithm for the PUSH and PULL models in a bidirectional network of size n, with probability $1 - \frac{4}{n}$.

7.5 Numerical calculation of $\hat{D}^{(1)}(p)$

As a complement to Eq. (7.1), we now compute the value of $\hat{D}^{(1)}(p)$ in small fully-connected networks, and thus obtain an approximation of $N_{\log_2}(p)$. The python code that we developed for this section is available at https://gitlab.com/bossuet/probabilistic_diameter.

In the PUSH model

We fix a node $i_0 \in [n]$ and we define the random variable $R_n(t)$ by

$$R_n(t) = \Big|\{j \in [n] \mid (i_0, j) \text{ is an arc of } \mathbb{G}(1:t)\}\Big|.$$

Observe that for the PUSH model in a fully-connected graph, $R_n(t)$ does not depend on the choice of i_0. The value of $\hat{D}^{(1)}(p)$ can easily be deduced from the distribution of the random variables $R_n(1), R_n(2), \ldots$.

$$\hat{D}^{(1)}(p) = \inf\{t \in \mathbb{N} \mid \Pr_n(R_n(t) = n) \geq p\}.$$

Moreover, this sequence of random variables is a Markov process and their probability distribution can be computed using the following lemma.

Lemma 69. *Let $a, b \in \{0, \ldots, n\}$. If $a \leq b \leq 2a$, then*

$$\Pr_n(R_n(t+1) = b \mid R_n(t) = a) = \frac{1}{n^a} \sum_{\ell=b-a}^{a} \binom{a}{\ell}\binom{n-a}{a-\ell} \left\{ \begin{matrix} \ell \\ b-a \end{matrix} \right\} a^{a-\ell}(b-a)!$$

where $\left\{ \begin{matrix} a \\ b \end{matrix} \right\}$ is the Stirling number of the second kind. Otherwise, $\Pr_n(R_n(t+1) = b \,|\, R_n(t) = a)$ is null.

Proof. We denote by A and B the two sets of nodes that are the targets of an arc whose source is i_0 in the digraphs $\mathbb{G}(1:t)$ and $\mathbb{G}(1:t+1)$, respectively. Thus a node j belongs to B if and only if there exists an arc from A to j in $\mathbb{G}(t+1)$.

In round $t+1$, each node in A picks one node uniformly, among all nodes. Then the total number of draws is n^a. Since each draw is equiprobable, we only have to count the number of favorable draws, that is, the draws such that $|B| = b$. Let ℓ_0 be the number of nodes in A that pick a node in $[n] \setminus A$ in round $t+1$; we have

$$\Pr_n(|B| = b \mid |A| = a) = \sum_{\ell=0}^{a} \Pr_n(|B| = b \cap \ell_0 = \ell \mid |A| = a).$$

We now fix some $\ell_0 \in \{0, \cdots, a\}$, and sample ℓ_0 nodes among the a nodes in A. For that, there are $\binom{a}{\ell_0}$ possibilities. Moreover, we partition the set $[n] \setminus A$ into two parts: $B \setminus A$, of size $b-a$ and $[n] \setminus B$. The number of possible partitioning is $\binom{n-a}{b-a}$. Then there are exactly $\left\{ {\ell_0 \atop b-a} \right\} (b-a)!$ surjective mappings from the previously chosen set of ℓ_0 nodes in A into the set $B \setminus A$ [81]. Finally, $a - \ell_0$ nodes in A pick a node belonging to A in round $t+1$. There are $a^{a-\ell_0}$ possibilities. Gathering all mentioned terms, and removing terms in which $\left\{ {\ell_0 \atop b-a} \right\} = 0$, we obtain the final expression of the lemma. □

Interestingly, a different expression of $\Pr_n(R_n(t+1) = b \mid R_n(t) = a)$ was used by Pittel in [77]. We computed $\hat{D}^{(1)}(p)$ for $n < 200$, our results are reported in Figure 7.1. For each $p \in \{0.5,\ 0.95,\ 0.99\}$, the values of $\hat{D}^{(1)}(p)$ provided by Lemma 69 are denoted by dots.

Figure 7.1 yields an estimation of $N_{\log_2}(p)$: Choosing $p = 0.5$, all the values of $\hat{D}^{(1)}(0.5)$ that we have computed are smaller that the bound provided by Eq. (7.1). This suggests that Corollary 68 holds for all n, that is, $N_{\log_2}(0.5) = 1$. Similarly, Figure 7.1 suggests that $N_{\log_2}(0.95) = 14$ and $N_{\log_2}(0.99) = 48$.

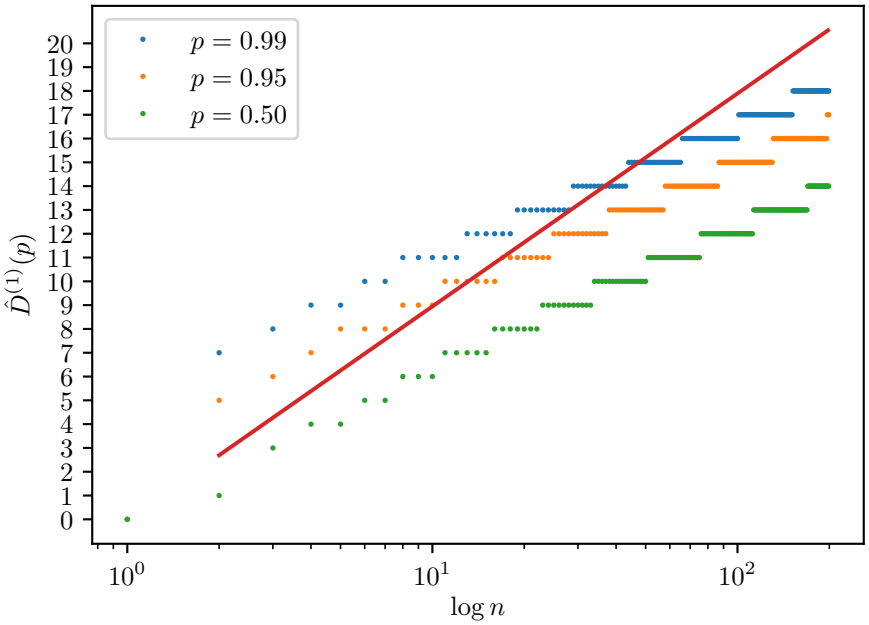


Figure 7.1: Each dot represents a value of $\hat{D}^{(1)}(p)$ in the PUSH model, in a fully-connected network, in function of p and the size of the network n. The straight line represents the theoretical bound provided by Eq. (7.1), choosing $\kappa = \log_2$.

7.6 In the PULL model

We consider the PULL model, and we define $R_n(t)$, similarity to the previous section.

Lemma 70. *Let $a, b \in \{0, \dots, n\}$. If $a \leq b$, then*

$$\Pr_n(R_n(t+1) = b \mid R_n(t) = a) = \binom{n-a}{b-a} \frac{(n-a)^{n-b} \cdot a^{b-a}}{n^{n-a}}.$$

Otherwise, $\Pr_n(R_n(t+1) = b \mid R_n(t) = a)$ is null.

Proof. We denote by A and B the two sets of nodes that are the targets of an arc whose source is i_0 in the digraphs $\mathbb{G}(1:t)$ and $\mathbb{G}(1:t+1)$, respectively. Thus a node j belongs to B if and only if there exists an arc from A to j in $\mathbb{G}(t+1)$.

In round $t+1$, each node in $[n] \setminus A$ picks one node uniformly, among all nodes. Then the total number of draws is n^{n-a}. Since each draw is equiprobable, we only have to count the number of favorable draws, that is, the draws such that $|B| = b$. Moreover, we partition the set $[n] \setminus A$ into two parts: $B \setminus A$, of size $b - a$ and $[n] \setminus B$. The number of possible partitionings is $\binom{n-a}{b-a}$. Finally, each node in $B \setminus A$ picks a node belonging to $[n] \setminus A$ in round $t+1$ and each node in $[n] \setminus B$ picks a node belonging to A in round $t+1$ There are $(n-a)^{n-b}$ and a^{b-a} possibilities, respectively. Gathering all mentioned terms, we obtain the final expression of the lemma. □

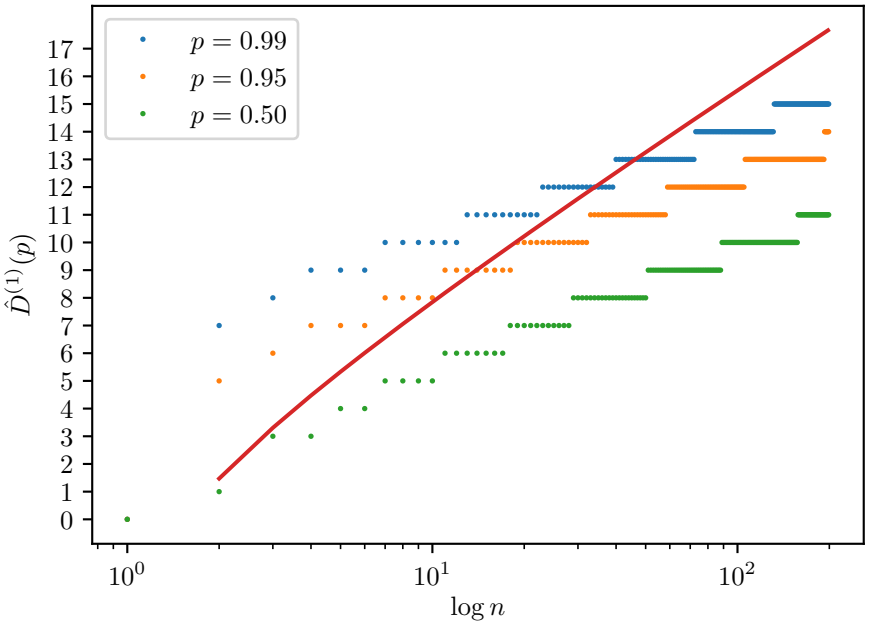


Figure 7.2: Some values of $\hat{D}^{(1)}(p)$ in the PULL model, in a fully-connected network of size n. The straight line represents the theoretical bound provided by Eq. (7.2), choosing $\kappa = \log_2$.

7.7 Bibliographic notes

Randomized rumor spreading is one of the core primitives to disseminate information in distributed networks. The importance of these processes not only has led to a huge body of experimental results, but, starting with the influential works of Frieze and Grimmett [56] and Karp, Shenker, Schindelhauer, and Vöcking [64] also to a large number of mathematical analyses of rumor spreading algorithms giving runtime or robustness guarantees for existing algorithms and, based on such findings, proposing new algorithms.

Roughly speaking, two types of results can be found in the literature, general bounds trying to give a performance guarantee based only on certain graph parameters and analyses for specific graphs or graph classes. In the domain of general bounds, [52] provide the classic maximum-degree-diameter bound, in addition of their $12n \log n$ bound. More recently, a number of works bounding the rumor spreading time in terms of conductance or other expansion properties [73, 32, 58, 57], which not only greatly helped our understanding of existing processes, but could also be exploited to design new dissemination algorithms [20, 21, 22, 60]. The natural downside of such general results is that they often do not give sharp bounds. It seems that among the known graph parameters, none captures very well how suitable this network structure is for randomized rumor spreading. Also, it has to be mentioned that these results mostly apply to the PUSH-PULL protocol.

The other research direction followed in the past is to try to prove sharper bounds for specific graph classes. This led, among others, to the results that the PUSH protocol spreads a rumor in a complete graph in time $\log_2 n + \log n \pm \omega(1)$ with high probability $1 - o(1)$ (whp.) [77], whereas the PUSH-PULL protocol does so in time $\log_3 n + O(\log \log n)$ [64]. The PUSH protocol spreads rumors in hypercubes in time $O(\log n)$ whp. [52], determining the leading constant is a major open problem. For Erdős-Rény random graphs with edge probability asymptotically larger than the connectivity threshold, a runtime of $\log_{2-p} n + \frac{1}{p} \log n \pm o(\log n)$ was shown for the PUSH

protocol allowing transmission errors with rate p [54]. For preferential attachment graphs, which are often used as model for real-world networks, it was proven that the PUSH protocol needs $\Omega(n^\alpha)$ rounds, $\alpha > 0$ some constant, whereas the PUSH-PULL protocol takes time $\Theta(\log n)$ and $\Theta((\log n)/\log\log n)$ when nodes avoid to call the same neighbor twice in a row [33, 40]. Even faster rumor spreading times were shown on Chung-Lu power-law random graphs[55].

One weakness of all these results on specific graphs is that they very much rely on the particular properties of the protocol under investigation. Even in fully connected networks (complete graphs), the existing analyses for the basic PUSH protocol [56, 77, 44], the PUSH protocol in the presence of transmission failures [42], the PUSH protocol with multiple calls [76], and the PUSH-PULL protocol [64] all uses highly specific arguments that cannot be used immediately for the other processes. To address this weakness, Doerr and Kostrygin [43] provide sharp bounds using a proof that fits a large range of rumor spreading models. In addition of the cases previously mentioned in this chapter, they deal with extensions of the classical PUSH, PULL and PULL-PUSH models: their results cover models with random transmission errors and also some models in which the communication graph is randomly sampled in each round.

Chapter 8

The SAP algorithm: a deterministic analysis

8.1 Introduction

This chapter further discuss the mod P-synchronization problem and the SAP$_g$ algorithm. Following an approach similar to Chapters 4 and 5, our goal is to establish that SAP$_g$ solves the mod P-synchronization problem without assuming a finite dynamic diameter. This chapter abandons the probabilistic approach previously used to study SAP$_g$. Indeed, the probabilistic approach provides a first relaxation of the assumption of a finite dynamic diameter. This chapter provides another possible relaxation. Combining those two orthogonal relaxations did not seem to be relevant.

As a preliminary, we study the behaviour of SAP$_g$ in networks with a finite dynamic diameter. The solvability result we obtain is a direct corollary of the main result of Chapter 6. Then we try to establish the correctness of SAP$_g$ in networks with an infinite dynamic diameter. A first idea consists in considering the class of dynamic networks with a finite dynamic radius. Unfortunately, section 8.4 presents a scenario that demonstrates that assuming a finite radius is insufficient to achieve mod P-synchronization using SAP$_g$. The solvability of mod P-synchronization in this network class remains an open problem. We therefore rely on a narrower network class, that is, the class of strongly centered networks (see Definition 23). The main contribution of this chapter is the proof that SAP$_g$ achieves mod P-synchronization in this network class.

memory usage	$D(\mathbb{G}) < \infty$	$Z(\mathbb{G}) = K(\mathbb{G}) \neq \emptyset$	$R(\mathbb{G}) < \infty$
bounded memory	✗ (see Thm. 27)		
finite memory	SAP$_g$ with g inflationary		? (see Sec. 8.4)
infinite memory	SMINMAX		

Figure 8.1: Solvability results for the mod P-synchronization problem by self-stabilizing algorithms in dynamic networks, in function of the network topology and the memory usage allowed.

8.2 The SAP algorithm in networks with finite dynamic diameter

The correctness proof of SAP$_g$ that will be developed in this chapter rely on the fact that SAP$_g$ achieves synchronization in network with finite dynamic diameter. To prove this preliminary result, we state Lemma 71 and Corollary 72, which are analog to Lemma 63 and Theorem 64, respectively. We fix an execution ϵ of SAP$_g$ with a finite dynamic diameter. In this section only, we let

$$t_0 \stackrel{\text{def}}{=} g^* \left(\frac{2D\left(\mathbb{G}\right)}{P} \right) D(\mathbb{G}).$$

Lemma 71. *Assuming that ϵ is not synchronized in round t_0 where g is a non-decreasing and inflationary function, it holds that*

$$M(t_0) \geq \frac{2D(\mathbb{G})}{P}. \tag{8.1}$$

Proof. For ease of notation, we let $\bar{g} = g^* \left(\frac{2D(\mathbb{G})}{P} \right)$. We assume that ϵ is not synchronized in round $\bar{g}D(\mathbb{G})$. We can show by induction on $\bar{g}$ that

$$M(t_0) \geq g^{\bar{g}}(0) = \frac{2D(\mathbb{G})}{P}.$$

The induction itself is similar to the one used in the proof of Lemma 63. □

The following theorem is a corollary of Theorem 64, choosing a specific probability measure Pr.

Corollary 72. *In any execution of* SAP_g *with a dynamic graph whose dynamic diameter* $D(\mathbb{G})$ *is finite, the* SAP_g *algorithm achieves mod* P*-synchronization for any non-decreasing and inflationary function* g*. Moreover, the stabilization time is bounded by* $\left(g^*\left(\frac{2D(\mathbb{G})}{P}\right)+3\right)D(\mathbb{G})$.

Proof. We fix an execution ϵ of SAP_g with a finite dynamic diameter. We pose the following probability measure: for any event E,

$$\Pr(E) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \epsilon \in E \\ 0 & \text{otherwise} \end{cases} \tag{8.2}$$

With such a probability measure, we can show that any two events E_1 and E_2 are independent.

$$\Pr(E_1 \cap E_2) = \begin{cases} 1 & \text{if } \epsilon \in E_1 \wedge \epsilon \in E_2 \\ 0 & \text{otherwise} \end{cases} = \Pr(E_1) \times \Pr(E_2).$$

Therefore Pr is memoryless. We fix some real $p \in (0,1)$. We easily verify that $\hat{D}_1(p) = \hat{D}_2(p) = D(\mathbb{G})$. By Lemma 71, we obtain

$$\Pr\left(\left(M(t_0) \geq \frac{2\hat{D}^{(1)}(p)}{P}\right) \cup \mathcal{S}_{t_0}\right) \geq p, \tag{8.3}$$

where $t_0 = g^*\left(\frac{2D(\mathbb{G})}{P}\right)D(\mathbb{G})$. Theorem 64 has been established for a specific value of t_0, which differs from the one used in this section. However, taking a closer look at the proof of this theorem, the proof remains true for any value of t_0 that satisfy Eq. (6.4). By Theorem 64, mod P-synchronization is achieved in round $t_0 + 3D(\mathbb{G})$ round with probability p^4, and hence, the execution ϵ is synchronized in round $t_0 + 3D(\mathbb{G})$. □

8.3 Specializations of the SAP Algorithm

In the rest of this chapter, we borrow the λ notation from lambda calculus: $\lambda x.x+1$ denotes the function that maps x to $x+1$. When some bound B on the dynamic diameter is given, we may choose g to be the constant function $g = \lambda x.M$ with $M = \lceil \frac{2B}{P} \rceil$. Then we get $g^*\left(\frac{2D(\mathbb{G})}{P}\right) = 1$ and the pseudo-code of SAP_g reduces to Algorithm 7.

Algorithm 7: Pseudo-code of node i in the $\text{SAP}_{\lambda x.M}$ algorithm

```
1 Variables:
2     C_i ∈ ℕ
3 At each round:
4     send ⟨C_i⟩ to all
5     receive ⟨C_{j1}⟩, ⟨C_{j2}⟩, ... from the set In_i of incoming neighbours
6     C_i ← [min_{j∈In_i} C_j + 1]_{PM}
```

Corollary 73. *The* $\text{SAP}_{\lambda x.M}$ *algorithm solves the mod* P*-synchronization problem in any dynamic graph with a dynamic diameter less than or equal to* $PM/2$*. Moreover, the stabilization time is bounded by* $1 + 3D(\mathbb{G})$.

The proof of this corollary is similar to the proof of Corollary 72: in this case, we choose $t_0 = 1$. Therefore, assuming that $D(\mathbb{G}) \leq PM/2$, Eq. (6.4) holds. Let us observe that Corollary 73 provides an upper bound of three times the dynamic diameter D on $\text{SAP}_{\lambda x.M}$'s stabilization time, which is independent on the bound B. The limit of $PM/2$ in Corollary 73 is tight, as proved by the following result.

Theorem 74 (Theorem 4.13 in [2])**.** *For any even integers P and D satisfying $P < 2D$, there exists an execution of $\text{SAP}_{\lambda x.1}$ with a dynamic graph $\mathbb{G}$ whose dynamic diameter is D in which mod P-synchronization is never achieved.*

The relation between Corollaries 73 and 72 is illustrated in Table 8.1.

memory usage	$D(\mathbb{G}) \leq B$	$D(\mathbb{G}) < \infty$
bounded memory	SAP_g with $g = \lambda x. \lceil \frac{2B}{P} \rceil$	✗ (see Thm. 27)
finite memory	SAP_g with g inflationary	

Table 8.1: Solvability results for the mod P-synchronization problem by self-stabilizing algorithms in dynamic networks, in function of the knowledge on a bound on the dynamic diameter and the memory usage allowed.

Interestingly, the self-stabilizing algorithm in [17], called *SS-MinSU* and developed for clock synchronization in a static and strongly connected network when a bound B on the dynamic diameter[1] is available, is actually an optimization of the $\text{SAP}_{\lambda x.M}$ algorithm.

As for the algorithm proposed in [6] for a static strongly connected digraph G, it corresponds to the $\text{SAP}_{\lambda x.1}$ algorithm, combined with a round-robin strategy which consists, for each node, to send one message per round according to this fixed cyclic order amongst the outgoing neighbors in G. This strategy thus translates the fixed digraph G into a dynamic graph $\mathbb{G}$. Using Proposition 24 in [25], $\mathbb{G}$'s dynamic diameter can be upper bounded by $3n$. Via Corollary 73, the interpretation of the algorithm in [6] for a fixed digraph G in terms of a run of $\text{SAP}_{\lambda x.1}$ over the corresponding dynamic graph $\mathbb{G}$ shows that this algorithm works when $P \geq 6n$, and its stabilization time is less than $9n$ (instead of the correctness condition $P \geq n^2$ and the stabilization bound of $\frac{3}{2}n^2$, given both in [6]).

8.4 The SAP algorithm with infinite dynamic diameter

The aim of this section is to study how the assumption of a finite dynamic diameter can be relaxed so that the SAP_g algorithm still achieves mod P-synchronization.

The SAP algorithm with a finite dynamic radius

We now study whether SAP_g can achieve mod P-synchronization in networks with an infinite dynamic diameter. For that, we first demonstrate that the sole assumption of a finite dynamic radius is not sufficient for SAP_g to achieve mod P-synchronization. We construct an execution of SAP_g with a central node i. The underlying idea of our scenario is that sporadic incoming neighbors disrupt the value of i's clock and hence preclude any alignment of the other clocks on C_i.[2]

[1] The bound B is denoted α in the *SS-MinSU* algorithm.

[2] We provide a Python script that may be helpful to verify the correctness of our construction: https://gitlab.com/bossuet/sap_execution.git.

Let G, H_j, H_k, I be the four digraphs defined in Figure 8.2 with three nodes i, j, k, and let Φ_k be the following predicate on the rounds of a SAP_g execution:

$$\big(M_i = M_j\big) \wedge \big(M_i \geq M_k\big) \wedge \big(C_i = C_j\big) \wedge \big(C_i \not\equiv_P 0\big) \wedge \big(C_i \leq PM_i - 2\big) \wedge \big(C_k = 0\big).$$

The predicate Φ_j is obtained by exchanging the roles of j and k. The proof of the following lemma follows from a step by step execution of the SAP_g algorithm between rounds t and $t + PM_i(t) - C_i(t)$.

Lemma 75. *Let t be a round of a SAP_g execution with a dynamic graph $\mathbb{G}$, and let m and c denote $M_i(t)$ and $C_i(t)$, respectively. Let $\mathbb{G}'$ be any dynamic graph that coincides with $\mathbb{G}$ up to t and such that:*

$$\mathbb{G}'(t+1) = \cdots = \mathbb{G}'(t + Pm - c - 2) = G,\ \ \mathbb{G}'(t + Pm - c - 1) = H_k,\ \ \mathbb{G}'(t + Pm - c) = I.$$

If Φ_k holds at round t of the SAP_g execution with $\mathbb{G}'$, then Φ_j holds at round $t + Pm - c$ of this execution.

Proof. For simplicity, we assume that $t = 0$. Given the pseudo-code of SAP_g and the dynamic graph $\mathbb{G}'$, the state of the system in each round can be computed (see Table 8.2).

Round number	$\mathbb{G}'(t)$	$C_i(t)$	$M_i(t)$	$C_j(t)$	$M_j(t)$	$C_k(t)$	$M_k(t)$
initially		c	m	c	m	0	$< m$
1	G	$c+1$	m	$c+1$	m	1	$g(m)$
$t \leq Pm - c - 2$	G	$c+t$	m	$c+t$	m	t	$g^t(m)$
$Pm - c - 2$	G	$Pm - 2$	m	$Pm - 2$	m	$Pm - c - 2$	$g^{Pm-c-2}(m)$
$Pm - c - 1$	H_k	$Pm - c - 1$	$g^{Pm-c-1}(m)$	$Pm - 1$	m	$Pm - c - 1$	$g^{Pm-c-1}(m)$
$Pm - c$	I	$Pm - c$	$g^{Pm-c-1}(m)$	0	m	$Pm - c$	$g^{Pm-c-1}(m)$

Table 8.2: State of each node up to round $Pm - c$.

Assuming that Φ_k holds in round 0, we easily verify that Φ_j is satisfied in round $Pm - c$. □

We now fix two positive integers m^0 and c^0 such that $c^0 \in \{1, \cdots, Pm^0 - 2\}$ and $c^0 \not\equiv_P 0$, and we consider the two sequences $(m^r)_{r\geq 0}$ and $(c^r)_{r\geq 0}$ defined by:

$$\begin{cases} m^{r+1} = g^{Pm^r - c^r - 1}(m^r) \\ c^{r+1} = Pm^r - c^r. \end{cases}$$

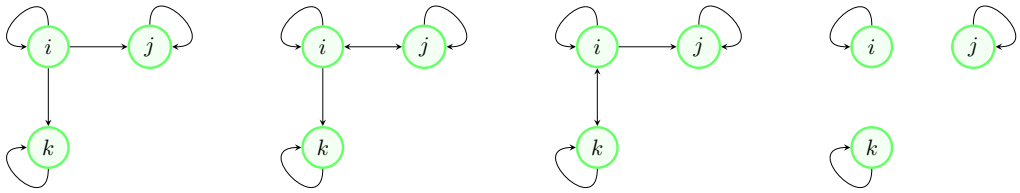
We let $m^{-1} = 0$. The dynamic graph $\mathbb{G}$ defined as:

$$\begin{aligned} &\mathbb{G}(Pm^{r-1} + 1) = \cdots = \mathbb{G}(Pm^r - c^r - 2) = G, \\ &\mathbb{G}(Pm^r - c^r - 1) = H_k \text{ or } H_j, \\ &\mathbb{G}(Pm^r - c^r - 1) = I, \end{aligned}$$

is rooted with delay two and i is its unique center. Lemma 75 shows that Φ_k holds infinitely often in the SAP_g execution with the dynamic graph $\mathbb{G}$ and starting with:

$$M_i(0) = M_j(0) = M_k(0) = m^0,\ \ C_i(0) = C_j(0) = c^0, \quad \text{and} \quad C_k(0) = 0.$$

Hence, the nodes are never synchronized.



(a) digraph G (b) digraph H_j (c) digraph H_k (d) digraph I

Figure 8.2: Four digraphs with three nodes.

The SAP algorithm in strongly centered network

That leads us to consider the stronger assumption that the network is strongly centered (see Definition 23), without requiring any global knowledge on $Z(\mathbb{G})$. However, the simple but typical scenario below shows that the simplified version of SAP_g with a fixed period, namely the $\text{SAP}\,\lambda x.M$ algorithm, does not achieve mod P-synchronization in the execution with the initial values $C_i(0) = C_j(0) = 1$ and $C_k(0) = 0$ and the static graph H defined in Figure 8.3, even for large value of M. Indeed, at each round t, it holds that $C_i(t) = [t+1]_{PM}$, $C_k(t) = [t]_{PM}$, and

$$C_j(t) = \begin{cases} 1 & \text{if } [t]_{PM} = 0 \\ [t]_{PM} & \text{otherwise.} \end{cases}$$

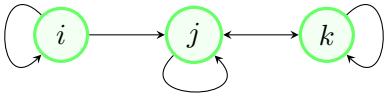


Figure 8.3: The digraph H with three nodes.

The striking point of increasing periods is precisely to overcome the above-mentioned limitation: we are going to prove that the SAP_g algorithm achieves mod-P synchronization in strongly centered networks under the sole condition of a non-decreasing and strictly inflationary function g. In other words, while Corollary 73 has no counterpart for dynamic graphs that are strongly centered with bounded delay, we will show that Corollary 72 extends to this latter class of dynamic graphs, with a synchronization phenomena quite different from that involved in the case of strong connectivity.

We fix a strongly centered dynamic graph $\mathbb{G}$, and an execution ϵ of SAP_g with $\mathbb{G}$. From now, Z, K, D and R denote $Z(\mathbb{G})$, $K(\mathbb{G})$, $D(\mathbb{G})$ and $\text{rad}(\mathbb{G})$, for short. We first prove the following preliminary lemma.

Lemma 76. *The center of any strongly centered dynamic graph $\mathbb{G}$ has no incoming arc from some index t_0.*

Proof. We denote $\mathbb{G}(\infty)$ a digraph whose set of nodes is $[n]$ that contains every arc that appears infinitely often in $\mathbb{G}$. By definition of K, each node $i \in K$ can infinitely often reach each node $j \in [n]$ in the dynamic graph $\mathbb{G}$, whereas there are finitely many paths between any two nodes. By the pigeonhole principle, each node in K is the root of a spanning tree in $\mathbb{G}(\infty)$. Using the definitions of K and $\mathbb{G}(\infty)$, the converse can also be proved. Then K, and hence Z have no incoming arc in $\mathbb{G}(\infty)$, since if i is the root of some spanning tree in $\mathbb{G}(\infty)$, then all i's incoming

neighbours are also roots of a spanning tree. Then, from a certain round, Z has no incoming arc in $\mathbb{G}$. □

In the self-stabilizing paradigm, any predicate that holds from a certain round can be assumed to hold from the beginning. We may then assume $t_0 = 0$ in the rest of the chapter: The nodes in Z receive no message from the nodes in $[n] \setminus Z$. From the viewpoint of every node in Z, the execution ϵ is thus indistinguishable from an execution with the set of nodes equal to Z and a dynamic graph with a finite dynamic diameter. Corollary 72 shows that mod P-synchronization is eventually achieved in Z. A closer look at the SAP_g algorithm yields the following more precise result: there exist two non-negative integers s and M such that

$$\forall t \geq s,\ \forall k, \ell \in Z:\ C_k(t) = C_\ell(t) \ \text{ and } \ M_k(t) = M. \tag{8.4}$$

The minimum integer s satisfying Eq. (8.4) is denoted by t_1, and $C(t)$ is the common value of all the counters $C_k(t)$ for $k \in Z$ and $t \geq t_1$. The node i is said to be *Z-synchronized at round t* if $C_i(t) \equiv_P C(t)$. The set of Z-synchronized nodes at round t is denoted by $S_Z(t)$. In the case the system is not synchronized in round t, i.e., $S_Z(t) \neq [n]$, we let

$$\tilde{M}(t) \stackrel{\text{def}}{=} \min_{i \notin S_Z(t)} M_i(t).$$

Using the existence of a self-loop at each node and the update rules of the variables M_i, we easily show that $\tilde{M}$ is non-decreasing:

Lemma 77. *For all $t \geq t_1$, it holds that $\tilde{M}(t+1) \geq \tilde{M}(t)$.*

Let i be a central node such that $\mathrm{e}_{\mathbb{G}}(i) = R$, and let j be an arbitrary node. There exists an arc (i, j) in each digraph $\mathbb{G}(t : t + R - 1)$. Since $C_i(t) < PM$, Lemma 58 implies the following upper bound on the clock C_j:

Lemma 78. *For all $t \geq t_1 + R$ and all nodes $j \in [n]$, it holds that $C_j(t) < PM + R$.*

Then Lemma 71 admits the following counterpart in strongly centered networks:

Lemma 79. *For every positive integer ℓ, one of the following statements is true:*

1. *the system is synchronized in round $t_1 + \ell R$;*
2. *$\tilde{M}(t_1 + \ell R) \geq g^{\ell-1}(M)$.*

Proof. We proceed by induction on ℓ. The base case $\ell = 1$ is due to the update rule for M_i. For the inductive step, assume that the lemma holds in round $t_1 + \ell R$ and that some node i is not Z-synchronized in round $t_1 + (\ell + 1)R$. By definition of R, the node i has an in-neighbor in Z in the directed graph $\mathbb{G}(t_1 + \ell R + 1 : t_1 + (\ell + 1)R)$, i.e., there exist a node $j \in Z$ and a path $j = j_0, j_1, \cdots, j_R = i$ in the round interval $[t_1 + \ell R + 1, t_1 + (\ell + 1)R]$. Since $j \in Z \subseteq S_Z(t_1 + \ell R)$ and $i \notin S_Z(t_1 + (\ell + 1)R)$, this path is not synchronized. Let $d \in \{1, \ldots, R\}$ be the first index such that $j_{d-1} \in S_Z(t_1 + \ell R + d - 1)$ and $j_d \notin S_Z(t_1 + \ell R + d)$. In round $t_1 + \ell R + d$, the node j_d has an incoming neighbor j^+ such that

$$C_{j^+}(t_1 + \ell R + d - 1) + 1 \equiv_P C_{j_d}(t_1 + \ell R + d),$$

that is, $j^+ \notin S_Z(t_1 + \ell R + d - 1)$. Then j_{d-1} and j^+ are two in-neighbors of j_d whose clocks are not congruent modulo P in round $t_1 + \ell R + d - 1$. It follows that:

$$M_i(t_1 + (\ell+1)R) \geq M_{j_d}(t_1 + \ell R + d) \geq g(M_{j^+}(t_1 + \ell R + d - 1)) \geq g(\tilde{M}(t_1 + \ell R + d - 1)) \geq g^\ell(M).$$

The first two inequalities are due to the update rules for M_i and M_{j_d}, the third one is by definition of $\tilde{M}$ and the fact that g is non-decreasing, and the last one is a consequence of the inductive assumption and Lemma 77. □

Theorem 80. *For any non-decreasing and inflationary function g, SAP_g solves the mod-P synchronization problem in any strongly centered dynamic graph.*

Proof. We let $\bar{g} = g^*\left(M + \frac{R+1}{P}\right)$ and $t_2 = t_1 + \bar{g}R$. The main part of the proof consists in showing, by induction on $t \geq t_2$, the following property:

$$\forall i \in [n] \setminus S_Z(t), \quad C_i(t) \geq t - t_2.$$

The base case $t = t_2$ is obvious. Suppose now that the above property holds at round $t \geq t_2$, and that $S_Z(t+1) \neq [n]$. Let us fix some node $j \notin S_Z(t+1)$; we are going to show that

$$C_j(t+1) \geq t + 1 - t_2. \tag{8.5}$$

By Lemma 61, one of the two propositions below is true.

(a) $C_j(t+1) = 1 + C_{j^+}(t)$ where j^+ is a node such that $C_{j^+}(t) = \min_{k \in In_j(t+1)} C_k(t)$,

(b) $C_j(t+1) = 0$ and $C_j(t) = PM_j(t) - 1$.

In case (a), the inequality (8.5) follows from the inductive assumption since $j^+ \notin S_Z(t)$. In case (b), Lemma 78 implies that

$$PM_j(t) = C_j(t) + 1 < PM + R + 1, \tag{8.6}$$

since $t \geq t_2 \geq t_1 + R$. Moreover, it holds that

$$M_j(t) \geq \tilde{M}(t) \geq \tilde{M}(t_2) \geq g^{\bar{g}}(M) \geq g^{\bar{g}}(0) \geq M + \frac{R+1}{P}. \tag{8.7}$$

The first inequality is due to the fact that $j \notin S_Z(t+1)$, which implies $j \notin S_Z(t)$. The second one holds by Lemma 77, and the third one is a consequence of Lemma 79. The forth inequality is due to the fact that g is non-decreasing, and the last one is by definition of $\bar{g}$. Thus Eq. (8.6) and (8.7) contradict each other and case (b) cannot occur, which completes the proof of (8.5).

To complete the proof of the theorem, we proceed by contradiction, and we assume that some node i is not Z-synchronized in round $t_3 = t_2 + PM + R$. Then, we would obtain both $C_i(t_3) \geq PM + R$ by the inequality (8.5) and $C_i(t_3) < PM + R$ by Lemma 78 since $t_3 \geq t_1 + R$. It follows that all the nodes are Z-synchronized in round t_3. □

8.5 Complexity analysis

In this section, we provide a complexity analysis of SAP_g in the case of a network with finite dynamic diameter, and then in the case of a strongly centered network. We discuss the choice of the g function and its impact on both stabilization time and space complexity.

Networks with finite dynamic diameter

Recall that $\lambda x.x+1$ denotes the function that maps x to $x+1$. Theorem 65 implies the following corollary in the case of a network with finite dynamic diameter.

Corollary 81. *In any execution of* SAP_g*, if the dynamic diameter* D *of the network is finite, then the memory usage of each node is bounded by* $\log_2 P + 2\log_2\left(g^{(g^*(2D/P+1)+2)D}\left(\max_{i\in[n]} M_i(0)\right)\right)$.

Theorem 64 and Corollary 81 demonstrate some trade-off between stabilization time and space complexity. The faster g grows, the lower the synchronization time is, and the higher its space complexity is. To further illustrate this trade-off, Table 8.3 provides the time and space complexity in three cases. First, when a bound B on the dynamic diameter is given, choosing $g = \lambda x.\lceil \frac{2B}{P} \rceil$ provides the best stabilization time, namely $3D$, which interestingly does not depend on the bound B. When no bound on the dynamic diameter is available, the overhead of $\text{SAP}_{\lambda x.2x}$ over $\text{SAP}_{\lambda x.\lceil 2B/P \rceil}$ is only logarithmic while $\text{SAP}_{\lambda x.x+1}$ results in an additional delay of $O(D^2)$ rounds for stabilization.

Regarding space complexity, $\text{SAP}_{\lambda x.\lceil 2B/P \rceil}$ and $\text{SAP}_{\lambda x.x+1}$ uses $O(\log B)$ and $O(\log D)$ bits, respectively. This illustrates how SAP_g may be more memory-efficient using its adaptive mechanism and a judicious choice of g. By contrast, the space complexity of $\text{SAP}_{\lambda x.2x}$ is only linear in D, which might be problematic for memory-constrained devices.

g	synchronization time	space complexity
$g = \lambda x.\lceil \frac{2B}{P} \rceil$	$3D$	$\log_2 P + 2\log_2 \lceil \frac{2B}{P} \rceil$
$g = \lambda x.x+1$	$\left(\frac{2D}{P}+3\right)D$	$\log_2 P + 2\log_2\left(\max_{i\in[n]} M_i(0) + \frac{2D^2}{P} + 3D\right)$
$g = \lambda x.2x$	$\left(\log_2\left(1+\frac{2D}{P}\right)+2\right)D$	$\log_2 P + 2\log_2\left(\max_{i\in[n]} M_i(0)\right)$ $+ 2D\log_2\left(1+\frac{2D}{P}\right) + 4D$

Table 8.3: Complexity bounds of SAP_g in networks with finite dynamic diameter D and $B \geq D$.

Strongly centered networks

In the case of a strongly centered network, Theorem 80 bounds the stabilization time by $t_3 = t_2 + PM + R$. Eq. (6.13) then provides the following upper bound for M, where t_2 is the minimum integer satisfying Eq.(8.4).

$$M \leq g^{t_2}(\max_{i\in[n]} M_i(0)),$$

and thus, we obtain:

$$\begin{aligned} t_3 &\leq t_2 + PM + R \\ &\leq t_1 + Rg^*\left(M + \frac{R+1}{P}\right) + PM + R \\ &\leq D\left(g^*\left(\frac{2D}{P}\right)+3\right) + Rg^*\left(M+\frac{R+1}{P}\right) + PM + R. \end{aligned}$$

where D is the dynamic diameter of the (dynamic) subgraph of $\mathbb{G}$ induced by Z. Theorem 65 then implies the following corollary in the case of a strongly centered network.

Corollary 82. *In any execution of* SAP_g *in which the network is strongly centered, the memory usage of each node is bounded by*

$$\log_2 P + 2\log_2\left(g^{Rg^*\left(M'+\frac{R+1}{P}+1\right)+PM'+R}\left(\max_{i\in[n]} M_i(0)\right)\right),$$

where $M' = g^{(g^*(2D/P+1)+2)D}(\max_{i\in[n]} M_i(0))$.

g	synchronization time	space complexity
$g = \lambda x.x+1$	$\left(t_2 + \max_{i\in[n]} M_i(0)\right)(P+R)$ $+R\left(2+\frac{R+1}{P}\right)$ where t_2 satisfies $t_2 \leq \left(\frac{2D}{P}+3\right)D$	$\log_2 P + 2\log_2\left(\left(t_2 + \max_{i\in[n]} M_i(0)\right)(P+R) + R\left(2+\frac{R+1}{P}\right) + \max_{i\in[n]} M_i(0)\right)$
$g = \lambda x.2x$	$P\left(\max_{i\in[n]} M_i(0)\right)2^{t_2} + R(1+t_2)$ $+R\log_2\left(\left(\max_{i\in[n]} M_i(0)\right)\left(1+\frac{R+1}{P}\right)\right)$ where $t_2 \leq \log_2\left(1+\frac{2D}{P}\right)D + 2D$	$\log_2 P + 2\log_2\left(\max_{i\in[n]} M_i(0)\right) + 2P\left(\max_{i\in[n]} M_i(0)\right)2^{t_2} + 2R\left(1+t_2+\log_2\left(\left(\max_{i\in[n]} M_i(0)\right)\left(1+\frac{R+1}{P}\right)\right)\right)$

Table 8.4: Complexity bounds of SAP_g in strongly centered networks. Here, R is the dynamic radius of $\mathbb{G}$ and D is the dynamic diameter of the dynamic subgraph induced by $Z(\mathbb{G})$.

Table 8.4 provides a bound on synchronization time and space complexity in the cases $g = \lambda x.x + 1$ and $g = \lambda x.2x$. It shows that the trade-off presented in the previous section no longer applies. In the case $g = \lambda x.2x$, both time and space complexity contain exponential terms. A real-world device would quickly run out of memory. The SAP_g algorithm remains practical only if g is a slowly growing function. Comparing with Table 8.3, we observe that SAP_g achieves better performance in networks with finite dynamic diameter than in strongly centered networks. Choosing $g = \lambda x.x + 1$, the time complexity is in $O\left(R\left(D^2 + R + \max_{i\in[n]} M_i(0)\right)\right)$ in the later case, compared to $O(D^2)$ in the earlier case. A similar overhead is added to space complexity. Overall, choosing $g = \lambda x.x + 1$ seems to provide the best performances, as it is the "least inflationary" function.

Chapter 9

Conclusion

We made a deliberate choice to present the impossibility results found in Chapter 3 at the outset of this thesis – an ironic twist given that we only stumbled upon these theorems in the waning days of our three-year journey. These results serve as a structural foundation for the entire work, permeating its various facets. They illuminate the reasons why the TERMSYNCH$_P$ algorithm necessitates a bound on the dynamic radius to function effectively and why the SAP$_g$ algorithm requires unbounded memory. Our impossibility results provide answers to questions that had lingered throughout the course of our research. They not only clarify critical constraints on algorithms but also shape the trajectory of our subsequent investigations, forming an integral part of the overarching narrative of this thesis.

The following paragraphs develop the main takeaways of the last three years. In the course of this study, we have delved into the realm of dynamic networks, employing concepts like dynamic diameter and dynamic radius, which draw their definitions from the notion of graph product. Our choice to utilize graph products was one of several approaches available for extending the fundamental notions of static graph theory to dynamic graphs. This naturally prompts the question: did we select the most suitable approach? Drawing from the insights and knowledge accumulated over the past three years, I am confident that we indeed made the right choice. In every scenario encountered, and in each argumentation that encompasses static networks, replacing the conventional "static" notions (such as paths within a single graph) with their "dynamic" counterparts (e.g., temporal paths) consistently upholds the integrity of the arguments. To the best of our knowledge, the only synchronization algorithm that does not follows this rule is from Feldmann et al. [53]. Consequently, it becomes evident that our dynamic definitions stand out as the most organic and intuitive generalization of their static counterparts.

Another notable takeaway from our journey is that what may appear as straightforward ideas can often be elusive to discover. A case in point can be found in Chapter 4, where we took the MINMAX algorithm as a foundation and derived two novel algorithms: BMINMAX and SMINMAX. To arrive at these results, we introduced a general methodology for constructing a synchronization algorithm employing a stabilizing consensus algorithm. The proof of correctness for this approach is encapsulated in Lemma 36 and Theorem 37, both of which primarily comprise succinct calculations. It's worth noting that the apparent simplicity of this construction belies the fact that its original proof was notably intricate and later refined. I personally find the ultimate version of this construction to be quite elegant, although, regrettably, my perspective on this matter did not fully resonate with my PhD supervisor.

In Chapter 5, we present our results for the mod P-firing squad problem, and we introduce a novel algorithm for this problem. The main takeaway of this chapter is the usefulness of proof assistants, which are not limited to toy algorithms. In my opinion, two essential elements make our formal proof practical. First, working with synchronized rounds is important to simplify the base model. By contrast, when the model is asynchronous [1], a significant amount of complexity is necessary, which makes formal proofs more challenging. Second, Isabelle provides powerful automated tactics. In contrast, Coq requires significantly more manual work. If we used Coq instead of Isabelle, the time spent to formally verify TERMSYNCH$_P$ would have increased by an order of magnitude.

This manuscript does not cover an aspect of our research that consumed a significant portion of our time. During this period, we conducted an in-depth exploration into the implications of the seminal work by Paolo Boldi and Sebastiano Vigna on self-stabilizing algorithms. Notably, their contribution included the introduction of a universal self-stabilizing algorithm [16]. Leveraging this framework, a multitude of problems, among them the mod P-synchronization problem, readily find solutions in static and strongly connected networks. The core idea underpinning the SAP$_g$ algorithm – namely, the adaptive mechanism – finds its roots in Protocol (6), a fundamental component of their universal algorithm. Furthermore, Boldi and Vigna's reliance on the notion

of *graph fibration*[15] has left a lasting impact on our research landscape. In brief, a fibration represents a mapping between two graphs, allowing multiple nodes to be mapped to a single node when their views[1] are indistinguishable. While none of the contributions within this thesis directly employ the concept of fibration, it is worth noting that our extensive discussions over the past three years have yielded valuable insights and ideas. In particular, we uncovered that Boldi and Vigna's primary result, originally formulated for static strongly connected graphs, could be extended to encompass static graphs containing a spanning tree. However, we chose not to delve into this extension within this thesis, as it only tangentially relates to the core subject of synchronization algorithms. It is also pertinent to highlight that the view-based interpretation of the MINMAX algorithm, as detailed in Section 4.2, draws inspiration from their approach.

In the course of a three-year research endeavor, it's only natural to encounter challenges and detours along the way. Discussing these moments of reflection in this conclusion can offer valuable insights: At one juncture, we embarked on an intriguing endeavor – to establish a transformation of any algorithm solving any kind of problem into a self-stabilizing one at no additional cost. The concept involved running multiple parallel instances of a non-self-stabilizing algorithm at each node. These instances would be continually replaced with new ones. Consequently, even if the oldest instances were affected by transient faults, the construction ensured that fresher instances would eventually supplant the older ones, ultimately producing a correct output. However, as we delved into implementing this concept, we encountered various technical intricacies that placed substantial constraints on the problem and the algorithm this construction could handle. Ultimately, what deterred us from pursuing this idea further was its suboptimal memory performance.

Chapters 6 and 8 collectively constitute an extensive exploration of the SAP_g problem, commencing with a probabilistic analysis in the former and culminating in a focus on strongly centered networks in the latter. It's essential to note that this sequence of chapters does not represent the chronological development of our intellectual journey. In fact, our initial foray into SAP_g began with a deterministic approach. Then we discovered all the papers studying rumor spreading in the PULL, PUSH and PULL-PUSH models. By the way, we were surprised to realize that the literature on this subject was somehow messy. For example, the PUSH model in fully-connected networks was studied by Pittel [77], in one of the earliest contribution. However, no analog result for the PULL model exists until the paper of Doerr and Kostrygin [43]. David Peleg told me privately that authors sometimes choose to study one model over another, with no obvious reason.

Discovering this topic prompted us to work on a probabilistic study of SAP_g. Section 6.3 sheds light on the thought process we followed. Our initial intention was to extend the deterministic proof of SAP_g to encompass the PULL and PUSH models through a simplified notion involving $\hat{D}(p)$, as defined in Eq. (6.1). This approach, had it succeeded, would have enabled us to readily adapt the deterministic proof to yield a direct corollary in the probabilistic context. Unfortunately, as detailed in Section 6.3, this approach turned out to be a dead-end.

The introduction of $\hat{D}^{(1)}(p)$ emerged as the solution to these challenges. Consequently, we found it necessary to redevelop the entire probabilistic proof of SAP_g from the ground up, as done in Chapter 6. This chapter underscores a key advantage of our probabilistic approach: its versatility. Our correctness proof for SAP_g extends across a broad spectrum of probabilistic communication models, a feature that distinguishes it from existing results.

Furthermore, we envision that this approach could hold value beyond SAP_g. For instance, consider population protocols [4], a model where agents interact, exchange information, and update their states asynchronously at each step. This model diverges from the synchronization problems examined in this thesis. Nevertheless, our probabilistic toolkit has the potential to offer

[1] for more details on *views*, refer to Section 4.2

solutions to other challenges within this domain, such as leader election [47] and the majority problem [3]. The large portion of existing literature in this area concentrates on fully connected networks, where any two nodes can interact. In future research, we may explore the application of our probabilistic tools to address these problems in more general communication networks.

In conclusion, we leave the reader with several intriguing open questions that beckon further exploration. Notably, while we have introduced BTERMSYNC$_P$ as a solution to the terminating mod P-synchronization problem with finite memory, the query of whether the same problem can be addressed with bounded memory, under the same assumptions as BTERMSYNC$_P$, remains unanswered. Additionally, as discussed in Section 8.4, we have established that the SAP$_g$ algorithm falls short in solving the mod P-synchronization problem when the network is solely assumed to have a finite dynamic radius. However, the existence of an algorithm capable of resolving the mod P-synchronization problem under these conditions remains an open question, waiting to be unraveled by future research endeavors.

Bibliography

[1] Karine Altisen, Pierre Corbineau, and Stéphane Devismes. A framework for certified self-stabilization. *Logical Methods in Computer Science*, 13, 2017.

[2] Karine Altisen, Stéphane Devismes, Swan Dubois, and Franck Petit. Introduction to distributed self-stabilizing algorithms. *Synthesis Lectures on Distributed Computing Theory*, 8(1):1–165, 2019.

[3] Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008.

[4] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

[5] Dana Angluin, Michael J Fischer, and Hong Jiang. Stabilizing consensus in mobile networks. In *Distributed Computing in Sensor Systems: Second IEEE International Conference, DCOSS 2006, San Francisco, CA, USA, June 18-20, 2006. Proceedings 2*, pages 37–50. Springer, 2006.

[6] Anish Arora, Shlomi Dolev, and Mohamed G. Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1:11–18, 1991.

[7] Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823, October 1985.

[8] Paul Bastide, George Giakkoupis, and Hayk Saribekyan. Self-stabilizing clock synchronization with 1-bit messages. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA, 2021*, pages 2154–2173, 2021.

[9] Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Stabilizing consensus with many opinions. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 620–635. SIAM, 2016.

[10] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the Second Symposium on Principles of Distributed Computing*, pages 27–30, 1983.

[11] Philip. A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[12] Martin Biely, Peter Robinson, and Ulrich Schmid. Agreement in directed dynamic networks. In *Structural Information and Communication Complexity: 19th International Colloquium, SIROCCO 2012, Reykjavik, Iceland, June 30-July 2, 2012, Revised Selected Papers 19*, pages 73–84. Springer, 2012.

[13] Henrik Björklund, Sven Sandberg, and Sergei Vorobyov. Memoryless determinacy of parity and mean payoff games: a simple proof. *Theoretical Computer Science*, 310(1-3):365–378, 2004.

[14] Lucas Boczkowski, Amos Korman, and Emanuele Natale. Minimizing message size in stochastic communication patterns: fast self-stabilizing protocols with 3 bits. *Distributed Comput.*, 32(3):173–191, 2019.

[15] Paolo Boldi and Sebastiano Vigna. Fibrations of graphs. *Discret. Math.*, 243(1-3):21–66, 2002.

[16] Paolo Boldi and Sebastiano Vigna. Universal dynamic synchronous self-stabilization. *Distributed Computing*, 15(3):137–153, 2002.

[17] Christian Boulinier, Franck Petit, and Vincent Villain. Synchronous vs. asynchronous unison. *Algorithmica*, 51(1):61–80, 2008.

[18] Ming Cao, A Stephen Morse, and Brian DO Anderson. Reaching a consensus in a dynamically changing environment: A graphical approach. *SIAM Journal on Control and Optimization*, 47(2):575–600, 2008.

[19] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. In *International Conference on Ad-Hoc Networks and Wireless*, volume 6811 of *Lecture Notes in Computer Science*, pages 346–359. Springer, 2011.

[20] Keren Censor-Hillel, Bernhard Haeupler, Jonathan Kelner, and Petar Maymounkov. Global computation in a poorly connected world: fast rumor spreading with no dependence on conductance. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 961–970, 2012.

[21] Keren Censor Hillel and Hadas Shachnai. Partial information spreading with application to distributed maximum coverage. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of Distributed Computing*, pages 161–170, 2010.

[22] Keren Censor-Hillel and Hadas Shachnai. Fast information spreading in graphs with large weak conductance. *SIAM Journal on Computing*, 41(6):1451–1465, 2012.

[23] B. Charron-Bost and S. Merz. Formal verification of a Consensus algorithm in the Heard-Of model. Technical Report LIX, LIX, École polytechnique, March 2009.

[24] B. Charron-Bost and A. Schiper. The heard-of model: computing in distributed systems with benign faults. In *3rd International Workshop on Development of Computational Models*, Wroclaw, Pologne, July 2007.

[25] Bernadette Charron-Bost. Geometric bounds for convergence rates of averaging algorithms. *Information and Computation*, 2022. To appear, available at https://arxiv.org/abs/2007.04837.

[26] Bernadette Charron-Bost and Shlomo Moran. The firing squad problem revisited. *Theoretical Computer Science*, 793:100–112, 2019.

[27] Bernadette Charron-Bost and Shlomo Moran. MinMax algorithms for stabilizing consensus. *Distributed Computing*, 34:195–206, 2021.

[28] Bernadette Charron-Bost and Louis Penet de Monterno. Self-Stabilizing Clock Synchronization in Dynamic Networks. In Eshcar Hillel, Roberto Palmieri, and Etienne Rivière, editors, *26th International Conference on Principles of Distributed Systems (OPODIS 2022)*, volume 253 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:17, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[29] Bernadette Charron-Bost and Louis Penet de Monterno. Self-Stabilizing Clock Synchronization in Probabilistic Networks. In *37th International Symposium on Distributed Computing (DISC 2023)*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

[30] Bernadette Charron-Bost and André Schiper. The Heard-Of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.

[31] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pages 493–507, 1952.

[32] Flavio Chierichetti, Silvio Lattanzi, and Alessandro Panconesi. Almost tight bounds for rumour spreading with conductance. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 399–408, 2010.

[33] Flavio Chierichetti, Silvio Lattanzi, and Alessandro Panconesi. Rumor spreading in social networks. *Theoretical Computer Science*, 412(24):2602–2610, 2011.

[34] Michael R Clarkson and Fred B Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[35] Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *24th International Symposium on Distributed Computing, DISC 2010*, volume 6343 of *Lecture Notes on Computer Science*, pages 148–162. Springer, 2010.

[36] Thiago Correa, Breno Gustavo, Lucas Lemos, and Amber Settle. An overview of recent solutions to and lower bounds for the firing synchronization problem. *arXiv preprint arXiv:1701.01045*, 2017.

[37] Étienne Coulouma and Emmanuel Godard. A characterization of dynamic networks where consensus is solvable. In *Structural Information and Communication Complexity: 20th International Colloquium, SIROCCO 2013, Ischia, Italy, July 1-3, 2013, Revised Selected Papers 20*, pages 24–35. Springer, 2013.

[38] Louis Penet de Monterno, Bernadette Charron-Bost, and Stephan Merz. Synchronization modulo p in dynamic networks. *Theoretical Computer Science*, 942:200–212, 2023.

[39] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, 1987.

[40] Benjamin Doerr, Mahmoud Fouz, and Tobias Friedrich. Social networks spread rumors in sublogarithmic time. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 21–30, 2011.

[41] Benjamin Doerr, Leslie Ann Goldberg, Lorenz Minder, Thomas Sauerwald, and Christian Scheideler. Stabilizing consensus with the power of two choices. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, pages 149–158, 2011.

[42] Benjamin Doerr, Anna Huber, and Ariel Levavi. Strong robustness of randomized rumor spreading protocols. *Discrete Applied Mathematics*, 161(6):778–793, 2013.

[43] Benjamin Doerr and Anatolii Kostrygin. Randomized rumor spreading revisited. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[44] Benjamin Doerr and Marvin Künnemann. Tight analysis of randomized rumor spreading in complete graphs. In *2014 Proceedings of the Eleventh Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 82–91. SIAM, 2014.

[45] Shlomi Dolev. Possible and impossible self-stabilizing digital clock synchronization in general graphs. *Real Time Syst.*, 12(1):95–107, 1997.

[46] Shlomi Dolev and Jennifer L. Welch. Self-stabilizing clock synchronization in the presence of byzantine faults. *J. ACM*, 51(5):780–799, 2004.

[47] David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, 31(4):257–271, 2018.

[48] Cynthia Dwork, Nancy A. Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[49] Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. math. inst. hung. acad. sci*, 5(1):17–60, 1960.

[50] Shimon Even and Sergio Rajsbaum. Unison, canon, and sluggish clocks in networks controlled by a synchronizer. *Mathematical systems theory*, 28(5):421–435, 1995.

[51] Rui Fan and Nancy Lynch. Gradient clock synchronization. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 320–327, 2004.

[52] Uriel Feige, David Peleg, Prabhakar Raghavan, and Eli Upfal. Randomized broadcast in networks. *Random Structures and Algorithms*, 1(4):447–460, 1990.

[53] Michael Feldmann, Ardalan Khazraei, and Christian Scheideler. Time- and space-optimal discrete clock synchronization in the beeping model. In *32nd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA*, pages 223–233. ACM, 2020.

[54] Nikolaos Fountoulakis, Anna Huber, and Konstantinos Panagiotou. Reliable broadcasting in random networks and the effect of density. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 2010.

[55] Nikolaos Fountoulakis, Konstantinos Panagiotou, and Thomas Sauerwald. Ultra-fast rumor spreading in social networks. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1642–1660. SIAM, 2012.

[56] Alan M Frieze and Geoffrey R Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10(1):57–77, 1985.

[57] George Giakkoupis. Tight bounds for rumor spreading in graphs of a given conductance. In *Symposium on theoretical aspects of computer science (STACS2011)*, volume 9, pages 57–68, 2011.

[58] George Giakkoupis. Tight bounds for rumor spreading with vertex expansion. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January, 2014*, pages 801–815. SIAM, 2014.

[59] Mohamed Gouda and Ted Herman. Stabilizing unison. *Inf. Process. Lett.*, 35(4):171–175, 1990.

[60] Bernhard Haeupler. Simple, fast and deterministic gossip and rumor spreading. *Journal of the ACM (JACM)*, 62(6):1–18, 2015.

[61] Ted Herman and Sukumar Ghosh. Stabilizing phase-clocks. *Inf. Process. Lett.*, 54(5):259–265, 1995.

[62] Ted Herman and Sukumar Ghosh. Stabilizing phase-clocks. *Information Processing Letters*, 54(5):259–265, 1995.

[63] Ali Jadbabaie. Natural algorithms in a networked world: technical perspective. *Commun. ACM*, 55(12):100, 2012.

[64] Richard Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vocking. Randomized rumor spreading. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 565–574. IEEE, 2000.

[65] Ronald Kempe, Joseph Y. Dobra, and Moshe Y. Gehrke. Gossip-based computation of aggregate information. In *Proceeding of the 44th IEEE Symposium on Foundations of Computer Science, FOCS*, pages 482–491, Cambridge, MA, USA, 2003.

[66] Fabian Kuhn, Yoram Moses, and Rotem Oshman. Coordinated consensus in dynamic networks. In *Proceedings 30th ACM Symposium Principles of Distributed Computing*, pages 1–10. ACM, 2011.

[67] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

[68] Christoph Lenzen, Thomas Locher, and Roger Wattenhofer. Tight bounds for clock synchronization. *Journal of the ACM (JACM)*, 57(2):1–42, 2010.

[69] N. A. Lynch and Tuttle M. R. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th Symposium on Principles of Distributed Computing*, pages 137–151, Vancouver, British Columbia, Canada, August 1987.

[70] Nancy A Lynch and Michael J Fischer. On describing the behavior and implementation of distributed systems. *Theoretical Computer Science*, 13(1):17–43, 1981.

[71] Edward F Moore. The firing squad synchronization problem. *Sequential machines, selected Papers*, pages 213–214, 1964.

[72] FR Moore and GG Langdon. A generalized firing squad problem. *Information and Control*, 12(3):212–220, 1968.

[73] Damon Mosk-Aoyama and Devavrat Shah. Fast distributed algorithms for computing separable functions. *IEEE Transactions on Information Theory*, 54(7):2997–3007, 2008.

[74] Mikhail Nesterenko and Sébastien Tixeuil. Ideal stabilisation. *International Journal of Grid and Utility Computing 25*, 4(4):219–230, 2013.

[75] Yasuaki Nishitani and Namio Honda. The firing squad synchronization problem for graphs. *Theoretical Computer Science*, 14(1):39–61, 1981.

[76] Konstantinos Panagiotou, Ali Pourmiri, and Thomas Sauerwald. Faster rumor spreading with multiple calls. In *Algorithms and Computation: 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings 24*, pages 446–456. Springer, 2013.

[77] Boris Pittel. On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, 1987.

[78] T. K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, 1987.

[79] Steven H. Strogatz. From kuramoto to crawford: exploring the onset of synchronization in populations of coupled oscillators. *Physica D*, 143(1-4):1–20, 2000.

[80] Tamás Vicsek, András Czirók, Eshel Ben-Jacob, Inon Cohen, and Ofer Shochet. Novel type of phase transition in a system of self-driven particles. *Physical review letters*, 75(6):1226, 1995.

[81] Horst Wegner. Stirling numbers of the second kind and bonferroni's inequalities. *Elemente der Mathematik*, 60(3):124–129, 2005.

ECOLE
DOCTORALE

Titre : Problèmes de synchronisation dans les réseaux dynamiques multi-agents

Mots clés : graphes dynamique, synchronisation, firing squad

Résumé : Cette thèse explore les problèmes de synchronisation et de coordination dans un réseau d'agents avec un graphe de communication dynamique utilisant une séquence de rounds synchrones. Dans la première partie, le problème du Firing Squad est examiné, où les agents doivent ≪ faire feu ≫ simultanément. Une version modifiée est introduite, permettant de faire feu en rounds congrues modulo P. Un algorithme est présenté pour ce problème modifié sous des hypothèses de communication plus faibles. Une preuve formelle de la correction de l'algorithme est fournie à l'aide de l'assistant de preuve Isabelle.
La thèse se penche ensuite sur les algorithmes auto-stabilisants pour la synchronisation modulo P, où les horloges des agents doivent être congruentes à P à partir d'un certain round. Le défi consiste à relâcher les hypothèses sur la connaissance globale. Un algorithme de consensus MinMax auto-stabilisant est proposé, s'adaptant aux réseaux de communication dynamiques mais nécessitant une mémoire infinie. L'algorithme SAP, avec une mémoire finie mais non bornée, est présenté et prouvé correct dans les réseaux avec un diamètre dynamique inconnu mais fini. Ce résultat est ensuite étendu à une classe plus large de réseaux dynamiques centrés.
La dernière partie explore le comportement de SAP dans les modèles de communication probabilistes. Au lieu de prouver des propriétés pour chaque exécution, une hyperpropriété probabiliste est établie - la synchronisation modulo P avec haute probabilité - couvrant les modèles de communication probabilistes. Cela implique la définition d'une hiérarchie de diamètres probabilistes et la démonstration de l'efficacité de SAP dans la résolution des problèmes de synchronisation avec haute probabilité dans différents modèles de communication.

Title : Synchronization problems in dynamic multi-agent networks

Keywords : Dynamic graphs, synchronisation, firing squad

Abstract : This thesis explores synchronization and coordination problems in a network of agents with a dynamic communication graph using a sequence of synchronous rounds. In the first part, the Firing Squad problem is examined, where agents need to fire simultaneously. A modified version is introduced, allowing firing in rounds congruent modulo P. An algorithm is presented for this modified problem under weaker communication assumptions. Formal proof of correctness is provided using the Isabelle proof assistant.
The thesis then delves into self-stabilizing algorithms for synchronization modulo P, where agents' clocks must be congruent to P from a certain round. The challenge is to relax assumptions about global knowledge. A self-stabilizing MinMax consensus algorithm is proposed, accommodating dynamic communication networks but requiring infinite memory. The SAP algorithm, with finite but unbounded memory, is introduced and proven correct in networks with an unknown but finite dynamic diameter. This result is then extended to a broader class of centered dynamic networks.
The final part explores SAP's behavior in probabilistic communication models. Instead of proving properties for each execution, a probabilistic hyperproperty is established — the synchronization modulo P with high probability — covering probabilistic communication models. This involves defining a hierarchy of probabilistic diameters and demonstrating SAP's efficacy in solving the synchronization problem with high probability across different communication models.

Institut Polytechnique de Paris
91120 Palaiseau, France