

# MinMax Algorithms for Stabilizing Consensus

Bernadette Charron-Bost<sup>1</sup>

Shlomo Moran<sup>2</sup>

<sup>1</sup> CNRS, École polytechnique, 91128 Palaiseau, France

<sup>2</sup> Department of Computer Science, Technion, Haifa, Israel 32000

June 24, 2019

## Abstract

In the *stabilizing consensus* problem, each agent of a networked system has an input value and is repeatedly writing an output value; it is required that eventually all the output values stabilize to the same value which, moreover, must be one of the input values. We study this problem for a synchronous model with identical and anonymous agents that are connected by a time-varying topology. Our main result is a generic MinMax algorithm that solves the stabilizing consensus problem in this model when, in each sufficiently long but bounded period of time, there is an agent, called a root, that can send messages, possibly indirectly, to all the agents. Such topologies are highly dynamic (in particular, roots may change arbitrarily over time) and enforce no strong connectivity property (an agent may be never a root). Our distributed MinMax algorithms require neither central control (e.g., synchronous starts) nor any global information (e.g., on the size of the network), and are quite efficient in terms of message size and storage requirements.

## 1 Introduction

There has been much recent interest in distributed control and coordination of networks consisting of multiple mobile agents. This is motivated by the emergence of large scale networks with no central control and time-varying topology. The algorithms deployed in such networks ought to be completely distributed, using only local information, and robust against unexpected changes in topology, despite the lack of global coordination like synchronous starts.

A canonical problem in distributed control is the *stabilizing consensus* problem [2, 13, 3]: each agent  $u$  starts with some initial value and repeatedly updates an output variable  $y_u$  which eventually stabilizes on the same input value. The stabilizing consensus problem arises in a number of applications including eventual consistency in replicated databases (see eg., [20]), motion of autonomous agents [19], and blockchain agreement [17, 6]. Similarly, the stable computation of a predicate in the model of population protocols [1] may be seen as a variant of stabilizing consensus, in which the stabilized output value is the truth value of some predicate of the multiset of initial values.

A stronger form of agreement is captured by the classical *consensus* problem which differs from stabilizing consensus in the fact that all the output variables  $y_u$  are write-once: when the agent  $u$  is aware that agreement has been reached on some initial value  $\mu$ , it writes  $\mu$  in  $y_u$ , in which case  $u$  is

said to *decide on  $\mu$* . Hence the discrepancy between stabilizing consensus and consensus typically lies in this additional requirement of irrevocable decisions.

On the other side, *asymptotic consensus* is a classical weakening of stabilizing consensus: in the case of initial values that are real numbers, agents are only required to compute the same outcome *asymptotically*. In other words, the condition of eventual stability on the variables  $y_u$  is replaced by the weaker one of convergence. Moreover the limit value is only required to be in the range of the initial values, which prevents the applicability of asymptotic consensus to the class of problems where the limit value must be one of the initial values.

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 [2]. 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 [13, 3] 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 [12]. Moreover, topology changes are dramatically restricted [4], or communication graphs are supposed to be permanently bidirectional and connected [15].

The asymptotic consensus 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 [19]. 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 widely studied algorithms for this problem. One central result by Cao, Morse, and Anderson [7] 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.

**Contribution.** The primary goal of this paper is the design of stabilizing consensus algorithms for synchronous, fault free networks of identical and anonymous agents connected by a time-varying topology without any guarantee of strong connectivity. It should be noted that while stabilizing consensus is trivially solved by a gossip algorithm when the time-varying topology is eventually strongly connected, in the sense that for every pair of agents  $u$  and  $v$  there always exists a time consistent path from  $u$  to  $v$ , there is no obvious solution in the case some nodes cannot receive information from part of the network. In the absence of such a connectivity property, synchronous starts cannot be simulated [10], and hence tolerating asynchronous starts makes the problem even more challenging.

We start by introducing the notion of *kernel* that models the set of *root agents* able at any time to send messages, possibly indirectly, to all other agents. If this can be achieved in at most  $T$  communication steps, then the topology is said to be *rooted with delay  $T$* . A time-varying topology with a non-empty kernel is thus rooted with finite but *a priori* unbounded delays. We first prove that stabilizing consensus is not solvable in the case of an empty kernel. Then we show that in the case of a time-varying topology that is rooted with bounded delay, the stabilizing consensus

problem is solvable, even if the bound is unknown.

For that, we introduce the *MinMax* update rules for the output variables  $y_u$ , and then provide a distributed implementation of these update rules that is efficient, both in terms of message size and storage requirements. The resulting distributed algorithms, called *MinMax* algorithms, require no leader, no agent identifiers, and assume no global knowledge of the network structure or size. Moreover, they tolerate that agents join the system asynchronously. We define the subclass of *safe* MinMax algorithms, and show that any such algorithm achieves stabilizing consensus if the topology is rooted with bounded delay. As a corollary, we get that stabilizing consensus is solvable in any asynchronous and completely connected network and a minority of faulty agents that crash or commit send omissions. Finally, we show that using safe MinMax algorithms, stabilizing consensus is not solvable under the sole assumption of a non-empty kernel, i.e., the topology is rooted with finite but unbounded delays.

Another contribution of this work is the introduction of new notions that capture global properties of dynamic graphs, like the *kernel*, the *integral*, the *limit superior* of a dynamic graph, which we believe to be useful for investigating other distributed problems in networked systems with time-varying topologies.

## 2 Preliminaries

### 2.1 The computational model

We consider a networked system with a *fixed* set  $V$  of  $n$  agents. Our algorithms assume anonymous networks in which agents have no identifiers and do not know the network size  $n$ .

We assume a round-based computational model in the spirit of the Heard-Of model [11]. Point-to-point communications are organized into *synchronized rounds*: each agent can send messages to all agents and can receive messages sent by some of the agents. Rounds are communication closed in the sense that no agent receives messages in round  $t$  that are sent in a round different from  $t$ . The collection of *possible* communications (which agents can communicate to which agents) at each round  $t$  is modelled by a directed graph (digraph) with one node per agent. The digraph at round  $t$  is denoted  $\mathbb{G}(t) = (V, E_t)$ , and is called the *communication graph et round  $t$* . When dealing with just graph notions, we will use the term node rather than the one of agent for an element of  $V$ . We assume a self-loop at each node in all these digraphs since every agent can communicate with itself instantaneously. The sequence of digraphs  $\mathbb{G} = (\mathbb{G}(t))_{t \in \mathbb{N}}$  is called a *dynamic graph* [8]. A *network model* is any non-empty set of dynamic graphs.

In every *run* of an algorithm, each agent  $u$  is initially *passive*: it neither sends nor receives messages, and do not change its state. Then it either becomes *active* at the beginning of some round  $s_u \geq 1$ , or remains passive forever – in which case we let  $s_u = \infty$ . A run is *active* if all agents are eventually active.

At the beginning of its starting round  $s_u$ , the agent  $u$  sets up its local variables and starts executing its program. In round  $t \geq s_u$ ,  $u$  sends messages to all agents, receives messages from all its incoming neighbors in the digraph  $\mathbb{G}(t)$  that are active, and finally goes to its next state applying a deterministic transition rule. Then the agent  $u$  proceeds to round  $t + 1$ . The number of the current round is not assumed to be provided to the agents.

The value of a local variable  $x_u$  of  $u$  at the *end* of round  $t \geq s_u$  is denoted by  $x_u(t)$ . By convention, the value of  $x_u(t)$  for  $t < s_u$  is defined as the initial value of  $x_u$ .

Since each agent is deterministic, a run is entirely determined by the initial state of the network, the dynamic graph  $\mathbb{G}$ , and the collection of the starting rounds. For each run,  $\mathbb{G}^a(t) = (V, E_t^a)$  denotes the digraph where  $E_t^a \subseteq E_t$  is the set of edges that are either self-loops<sup>1</sup> or connecting two agents that are active in round  $t$ . The sets of  $u$ 's incoming neighbors (in-neighbors) in the digraphs  $\mathbb{G}(t)$  and  $\mathbb{G}^a(t)$  are denoted by  $\text{In}_u(t)$  and  $\text{In}_u^a(t)$ , respectively.

## 2.2 Limits and integrals of dynamic graphs

Let us first recall that the *product* of two digraphs  $G_1 = (V, E_1)$  and  $G_2 = (V, E_2)$ , denoted  $G_1 \circ G_2$ , is the digraph with the set of nodes  $V$  and with an edge  $(u, v)$  if there exists  $w \in V$  such that  $(u, w) \in E_1$  and  $(w, v) \in E_2$ . For any dynamic graph  $\mathbb{G}$  and any integers  $t' > t \geq 1$ , we let  $\mathbb{G}(t : t') = \mathbb{G}(t) \circ \dots \circ \mathbb{G}(t')$ . By convention,  $\mathbb{G}(t : t) = \mathbb{G}(t)$ , and when  $0 \leq t' < t$ ,  $\mathbb{G}(t : t')$  is the digraph with only a self-loop at each node.

Given any dynamic graph  $\mathbb{G}$  and any scheduling of starts, the sets of  $u$ 's incoming neighbors (or in-neighbors for short) in  $\mathbb{G}(t : t')$  and in  $\mathbb{G}^a(t : t')$  are denoted by  $\text{In}_u(t : t')$  and  $\text{In}_u^a(t : t')$ , respectively, and simply by  $\text{In}_u(t)$  and  $\text{In}_u^a(t)$  when  $t' = t$ . Because of the self-loops, all these sets contain the node  $u$ . If  $t' < t$ , then  $\text{In}_u(t : t') = \text{In}_u^a(t : t') = \{u\}$ .

If  $t \leq t'$ , then a  $v \sim u$  path in the interval  $[t, t']$  is any finite sequence  $w_0 = v, w_1, \dots, w_m = u$  with  $m = t' - t + 1$  and  $(w_k, w_{k+1})$  is an edge of  $\mathbb{G}(t + k)$  for each  $k = 0, \dots, m - 1$ . Hence there exists a  $v \sim u$  path in the interval  $[t, t']$  if and only if  $(v, u)$  is an edge of  $\mathbb{G}(t : t')$ , or equivalently  $v \in \text{In}_u(t : t')$ .

By extension over the infinite interval  $[t, \infty)$ , we define the digraphs

$$\mathbb{G}(t : \infty) = (V, \cup_{t' \geq t} E(\mathbb{G}(t : t'))), \quad \mathbb{G}^a(t : \infty) = (V, \cup_{t' \geq t} E(\mathbb{G}^a(t : t'))),$$

and denote by  $\text{In}_u(t : \infty)$  and  $\text{In}_u^a(t : \infty)$  the sets of  $u$ 's in-neighbors in these two digraphs, i.e.,

$$\text{In}_u(t : \infty) = \cup_{t' \geq t} \text{In}_u(t : t'), \quad \text{In}_u^a(t : \infty) = \cup_{t' \geq t} \text{In}_u^a(t : t').$$

The dynamic graph  $\overline{\mathbb{G}}$ , defined by  $\overline{\mathbb{G}}(t) = \mathbb{G}(t : \infty)$ , is called the *integral* of  $\mathbb{G}$ .

The *limit superior* of  $\mathbb{G}$ , denoted by  $\mathbb{G}(\infty)$ , is defined as the digraph  $\mathbb{G}(\infty) = (V, E_\infty)$ , where  $E_\infty$  is the set of edges that appear in an infinite number of digraphs  $\mathbb{G}(t)$ , namely,

$$E_\infty = \{(u, v) \in V \times V : \forall t, \exists t' \geq t, (u, v) \in E(\mathbb{G}(t'))\}.$$

In particular, the digraph  $\overline{\mathbb{G}}(\infty)$  is the limit superior of  $\overline{\mathbb{G}}$ .

**Proposition 1.** *If  $\mathbb{G}$  is a dynamic graph with a permanent self-loop at each node, then  $\overline{\mathbb{G}}$  eventually stabilizes to  $\overline{\mathbb{G}}(\infty)$ , i.e., there is a positive integer  $s$  such that*

$$\forall t \geq s, \quad \overline{\mathbb{G}}(t) = \overline{\mathbb{G}}(\infty).$$

*Proof.* Because of the self-loops, every edge of  $\overline{\mathbb{G}}(t + 1)$  is an edge of  $\overline{\mathbb{G}}(t)$ . Hence the dynamic graph  $\overline{\mathbb{G}}$  eventually stabilizes to some digraph  $\overline{\mathbb{G}}(s)$ , i.e., there is a positive integer  $s$  such that

$$\forall t \geq s, \quad \overline{\mathbb{G}}(t) = \overline{\mathbb{G}}(s).$$

Hence all edges in  $\overline{\mathbb{G}}(s)$  are edges of  $\overline{\mathbb{G}}(\infty)$ .

Conversely, by definition of the limit superior, any edge of  $\overline{\mathbb{G}}(\infty)$  appears in some digraph  $\overline{\mathbb{G}}(t)$  with  $t \geq s$ , and since  $\overline{\mathbb{G}}(t) = \overline{\mathbb{G}}(s)$ , is also an edge of  $\overline{\mathbb{G}}(s)$ .  $\square$

<sup>1</sup>To allow for simple notation, there are self-loops at all nodes of  $\mathbb{G}^a(t)$ , including those corresponding to the passive agents at round  $t$ .

Let us recall that a digraph  $G$  is transitively closed if any edge of  $G \circ G$  is an edge of  $G$ . In the case  $G$  has a self-loop at each node, this is equivalent to  $G \circ G = G$ . The transitive closure of  $G$ , denoted by  $G^+$ , is the minimal transitively closed digraph that contains all edges of  $G$ .

**Theorem 2.** *If  $\mathbb{G}$  is a dynamic graph with permanently a self-loop at each node, then  $\overline{\mathbb{G}}(\infty)$  is the transitive closure of  $\mathbb{G}(\infty)$ , namely,*

$$\overline{\mathbb{G}}(\infty) = [\mathbb{G}(\infty)]^+.$$

*Proof.* First we prove that  $\overline{\mathbb{G}}(\infty)$  is transitively closed. Since every edge of  $\mathbb{G}(\infty)$  is also an edge of  $\overline{\mathbb{G}}(\infty)$ , this will show that  $[\mathbb{G}(\infty)]^+ \subseteq \overline{\mathbb{G}}(\infty)$ . For that, let  $s$  be the index from which  $\overline{\mathbb{G}}$  stabilizes (cf. Proposition 1), and let  $(u, v)$  and  $(v, w)$  be two edges of  $\overline{\mathbb{G}}(\infty)$ . Since  $\overline{\mathbb{G}}(\infty) = \overline{\mathbb{G}}(s)$ , there exists an index  $t \geq s$  such that  $(u, v)$  is an edge in  $\mathbb{G}(s : t)$ . Since  $\overline{\mathbb{G}}(\infty) = \overline{\mathbb{G}}(t+1)$ , there exists an index  $t' > t$  such that  $(v, w)$  is an edge in  $\mathbb{G}(t+1 : t')$ . It follows that  $(u, w)$  is an edge in  $\mathbb{G}(s : t')$ , and hence  $(u, w)$  is an edge in  $\overline{\mathbb{G}}(s) = \overline{\mathbb{G}}(\infty)$ .

We now prove the reverse inclusion; let  $(u, v)$  be an edge of  $\overline{\mathbb{G}}(\infty)$ . Since there are finitely many edges that appear finitely many times in  $\mathbb{G}$ , there is an index  $r$  such that for all  $t \geq r$ , any edge in  $\mathbb{G}(t)$  is an edge in  $\mathbb{G}(\infty)$ . Let  $t = \max(s, r)$ . By Proposition 1,  $(u, v)$  is an edge of  $\overline{\mathbb{G}}(t) = \overline{\mathbb{G}}(\infty)$ , i.e., there exists an index  $t' \geq t$  such that  $(u, v)$  is an edge of  $\mathbb{G}(t : t')$ . In other words, there is a  $u \sim v$  path in the interval  $[t, t']$ ; let  $w_0 = u, w_1, w_2, \dots, w_{t'-t+1} = v$  be such a path. Since  $t \geq r$ , each edge in this path is an edge in  $\mathbb{G}(\infty)$ , which shows that  $(u, v)$  is an edge of the transitive closure of  $\mathbb{G}(\infty)$ , namely  $[\mathbb{G}(\infty)]^+$ .  $\square$

### 2.3 Roots, central roots, and kernels

A node  $u$  is a *root* of the digraph  $G = (V, E)$  if for every node  $v \in V$ , there is a path from  $u$  to  $v$  in  $G$ , and  $G$  is said to be *rooted* if it has at least one root. Node  $u$  is a *central root* of  $G$  if for every node  $v \in V$ ,  $(u, v)$  is an edge of  $G$ . The set of  $G$ 's roots and the set of  $G$ 's central roots are denoted by  $\text{Roots}(G)$  and  $\text{CRoots}(G)$ , respectively.

The *kernel* of a dynamic graph  $\mathbb{G}$ , denoted by  $\text{Ker}(\mathbb{G})$ , is defined as

$$\text{Ker}(\mathbb{G}) = \{u \in V \mid \forall t \geq 1, \forall v \in V, \exists t' \geq t : (u, v) \in E(\mathbb{G}(t : t'))\}$$

or equivalently,

$$\text{Ker}(\mathbb{G}) = \bigcap_{t \geq 1} \text{CRoots}(\overline{\mathbb{G}}(t)).$$

**Proposition 3.** *If  $\mathbb{G}$  is a dynamic graph with permanently a self-loop at each node, then*

$$\text{Ker}(\mathbb{G}) = \text{CRoots}(\overline{\mathbb{G}}(\infty)) = \text{Roots}(\mathbb{G}(\infty)).$$

*Proof.* Because of the self-loops,  $\text{CRoots}(\overline{\mathbb{G}}(t+1)) \subseteq \text{CRoots}(\overline{\mathbb{G}}(t))$ , which by Proposition 1 implies that

$$\text{Ker}(\mathbb{G}) = \bigcap_{t \geq 1} \text{CRoots}(\overline{\mathbb{G}}(t)) = \text{CRoots}(\overline{\mathbb{G}}(\infty)).$$

By Theorem 2, the digraph  $\overline{\mathbb{G}}(\infty)$  is the transitive closure of  $\mathbb{G}(\infty)$ , and so

$$\text{Roots}(\mathbb{G}(\infty)) = \text{CRoots}(\overline{\mathbb{G}}(\infty))$$

which completes the proof.  $\square$

The dynamic graph  $\mathbb{G}$  is said to be *infinitely connected* if  $\text{Ker}(\mathbb{G}) = V$ , or equivalently, by Proposition 3, if  $\mathbb{G}(\infty)$  is strongly connected.

## 2.4 Bounded delay rootedness

A dynamic graph  $\mathbb{G}$  is said to be *permanently rooted* if all the digraphs  $\mathbb{G}(t)$  are rooted. This notion naturally extends as follows:

**Definition 4.** A dynamic graph  $\mathbb{G}$  is rooted with delay  $T$  if for every positive integer  $t$ ,  $\mathbb{G}(t : t + T - 1)$  is rooted.  $\mathbb{G}$  is rooted with a bounded delay if it is rooted with delay  $T$  for some fixed positive integer  $T$ .

Any dynamic graph that is rooted with a bounded delay has a non-empty kernel, i.e., there are nodes that are central roots of all the digraphs  $\overline{\mathbb{G}}(t)$ . Proposition 5 below shows that these nodes are actually central roots over bounded length intervals.

**Proposition 5.** If  $\mathbb{G}$  is a dynamic graph that is rooted with delay  $T$ , then there exists a positive integer  $s$  such that

$$\forall t \geq s, \forall u \in V, \text{In}_u(t : t + T(n - |\text{Ker}(\mathbb{G})|)) \cap \text{Ker}(\mathbb{G}) \neq \emptyset.$$

*Proof.* For simplicity, we assume that  $T = 1$ ; the general case can be easily reduced to the case  $T = 1$  by considering the dynamic graph  $\mathbb{G}_T$  defined by  $\mathbb{G}_T(t) = \mathbb{G}((t-1)T + 1 : tT)$  that is rooted with delay one.

Let  $s$  be a positive integer such that for all  $t \geq s$ , every edge of  $\mathbb{G}(t)$  is also an edge of the digraph  $\mathbb{G}(\infty)$ , i.e.,  $E_t \subseteq E_\infty$ . Then we have that

$$\forall t \geq s, \text{Roots}(\mathbb{G}(t)) \subseteq \text{Ker}(\mathbb{G}). \quad (1)$$

For any non-negative integer  $i$ , let us now introduce the set  $U_i$  of nodes that are outgoing neighbors in the digraph  $\mathbb{G}(t : t + i)$  of some of the nodes in  $\text{Ker}(\mathbb{G})$ . Because of the self-loops, we have that  $\text{Ker}(\mathbb{G}) \subseteq U_0$  and  $U_i \subseteq U_{i+1}$ . We now show that either  $U_i = V$  or  $U_i \subsetneq U_{i+1}$ .

For that, assume that  $U_i \neq V$ . Let  $u \notin U_i$ , and let  $v$  be a root of the digraph  $\mathbb{G}(t + i + 1)$ ; hence there exists a path from  $v$  to  $u$  in  $\mathbb{G}(t + i + 1)$ . From (1) and the above inclusions, we derive that

$$v \in \text{Roots}(\mathbb{G}(t + i + 1)) \subseteq \text{Ker}(\mathbb{G}) \subseteq U_0 \subseteq U_i.$$

Thereby, there are two consecutive nodes  $x$  and  $y$  in the  $v \sim u$  path such that  $x \in U_i$  and  $y \notin U_i$ . By construction,  $y$  is an outgoing neighbor of  $v \in \text{Ker}(\mathbb{G})$  in the digraph  $\mathbb{G}(t : t + i + 1)$ , and thus  $y \in U_{i+1}$ . In conclusion,  $y \in U_{i+1} \setminus U_i$ , which shows that  $U_i \neq U_{i+1}$ .

It follows that  $|U_i| \geq \min(n, k + i)$  where  $k = |\text{Ker}(\mathbb{G})|$ , and hence  $U_{n-k} = V$ . Thus for every node  $u \in V$ , it holds that  $\text{In}_u(t : t + T(n - k)) \cap \text{Ker}(\mathbb{G}) \neq \emptyset$ , as required.  $\square$

## 3 The Stabilizing Consensus Problem

Let  $\mathcal{V}$  be a totally ordered set and let  $A$  be an algorithm in which each agent  $u$  has an input value  $\mu_u \in \mathcal{V}$  and an output variable  $y_u$  initialized to  $\mu_u$ . The algorithm  $A$  achieves stabilizing consensus in an active run with the initial values  $(\mu_u)_{u \in V}$  if the following properties hold:

**Validity.** At every round  $t$  and for each agent  $u$ , there exists some agent  $v$  such that  $y_u(t) = \mu_v$ .

**Eventual agreement.** There exists some round  $s$  such that

$$\forall t \geq s, \forall u, v \in V, y_v(t) = y_u(s).$$

The common limit value of the variables  $y_u$  is called the *consensus value*. The algorithm  $A$  is said to solve the stabilizing consensus problem in a network model  $\mathcal{G}$  if it achieves stabilizing consensus in each of its active runs with a dynamic graph in  $\mathcal{G}$ .

## 4 A Necessary Condition for Stabilizing Consensus

Next we show a necessary condition for the stabilizing consensus problem to be solvable.

**Theorem 6.** *There is no algorithm that solves stabilizing consensus in a network model containing a dynamic graph with an empty kernel.*

*Proof.* Let  $\mathbb{G}$  be any dynamic graph with an empty kernel, and let  $s$  be an index such that for every  $t \geq s$ , we have  $E_t \subseteq E_\infty$ . Let us consider the acyclic digraph formed with the strongly connected components of  $\mathbb{G}(\infty)$ , called the condensation graph of  $\mathbb{G}(\infty)$ , and let us recall that the condensation graph of a non-rooted digraph contains at least two source nodes, i.e., two nodes with no incoming edges (see e.g. [14]). From Propositions 3, we derive that the condensation graph of  $\mathbb{G}(\infty)$  has at least two source nodes  $U_0$  and  $U_1$ . Hence from round  $s$ , none of the agents corresponding to the nodes in  $U_0$  (resp.  $U_1$ ) are reachable from the agents corresponding to the nodes in  $U_1$  (resp.  $U_0$ ).

For the sake of contradiction, assume that there exists an algorithm  $A$  that achieves stabilizing consensus in all the runs with the dynamic graph  $\mathbb{G}$ . Consider now any run of the algorithm  $A$  in which all agents start at round  $s$ , and all agents of the strongly connected components  $U_0$  and  $U_1$  have the input values 0 and 1, respectively. Because of the validity property, all the agents in  $U_0$  must set their output values permanently to 0. Similarly, all the agents in  $U_1$  must set their outputs permanently to 1, which shows that the eventual agreement property is violated in this run.  $\square$

## 5 MinMax Algorithms

In this section, we define the class of *MinMax* algorithms by the type of update rules for the variables  $y_u$ . The way MinMax algorithms can be implemented in our computing model will be addressed in Section 7.

We start with an informal description of these algorithms. As a first step, consider the *Min algorithm*, in which each agent  $u$  has an output variable  $x_u$  which is repeatedly set to the minimum input value  $u$  has heard of. It is easy to see that on dynamic graphs that are infinitely connected, the values of all  $x_u$  variables eventually stabilize on the minimum input value.

When the Min algorithm is applied on an arbitrary dynamic graph,  $x_u$  eventually stabilizes on the minimum input value received by  $u$ , to be denoted by  $m_u^*$ :

$$m_u^* \stackrel{\text{def}}{=} \min_{v \in \text{In}_u^a(1:\infty)} (\mu_v).$$

Hence there is an integer  $\theta$  such that for every round  $t \geq \theta$  and every agent  $u$ , it holds that  $x_u(t) = m_u^*$ . As shown below in Lemma 7, if  $u$  is in  $\text{Ker}(\mathbb{G})$ , then  $m_u^* = m^*$ , where

$$m^* \stackrel{\text{def}}{=} \max_{v \in V} (m_v^*).$$

If the integer  $\theta$  is given, then the following simple two phase scheme can solve stabilizing consensus on a dynamic graph with a non-empty kernel: The first phase consists of the first  $\theta$  rounds in which the Min algorithm is applied. This allows for each agent  $u$  to compute the value  $m_u^*$  in the variable  $x_u$ . In the second phase starting at round  $\theta + 1$ , for each agent  $u$  the variable  $y_u$  is repeatedly set to the maximal  $m_v^*$  value  $u$  has heard of.

Since  $\theta$  is not given, we implement the above scheme by assigning to each agent  $u$  at each round  $t$  an integer  $\theta_u(t) \leq t$ , and by computing the value  $y_u(t)$ , with the first phase consisting of the interval  $[1, \theta_u(t)]$ .

For this procedure to be correct, we need that eventually  $\theta_u(t) \geq \theta$  for each agent  $u \in V$ . This is the case if  $\lim_{t \rightarrow \infty} \theta_u(t) = \infty$ . Assuming that each agent  $v$  has computed the value of  $m_v^*$  by round  $\theta_u(t)$ , we also need that each agent  $u$  hears of some agent in the kernel during the round interval  $[\theta_u(t) + 1, t]$ . In conclusion,  $\theta_u(t)$  must be chosen (1) large enough to ensure that each agent  $v$  has computed  $m_v^*$  by round  $\theta_u(t)$  and (2) small enough to guarantee that  $u$  hears of some agent in the kernel during the period  $[\theta_u(t) + 1, t]$ .

A *MinMax rule* for the variable  $y_u$  is an update rule of the form

$$y_u(t) = \max_{v \in \text{In}_u^a(\theta_u(t)+1:t)} \left( \min_{w \in \text{In}_v^a(1:\theta_u(t))} (y_w(0)) \right) \quad (2)$$

where  $\theta_u(t)$  is any integer in the interval  $[0, t]$ . A *MinMax algorithm* is an algorithm in which for each agent  $u$  and each round  $t$ , the value of  $y_u$  is updated by a MinMax rule. It is determined by the way the integer-valued functions  $\theta_u$ , called *cut-off functions*, are chosen.

We now prove the basic property on which our strategy relies.

**Lemma 7.** *In any active run, if  $u$  is in  $\text{Ker}(\mathbb{G})$ , then for every agent  $v$  it holds that  $\text{In}_u^a(1 : \infty) \subseteq \text{In}_v^a(1 : \infty)$ , and  $m_u^* = m^*$ .*

*Proof.* Let  $w$  be an arbitrary agent in  $\text{In}_u^a(1 : \infty)$ , and let  $t_0 \in \mathbb{N}$  be such that  $w \in \text{In}_u^a(1 : t_0)$  and all agents are active at round  $t_0$ . Since  $u$  is in  $\text{Ker}(\mathbb{G})$ , there is some round  $t_1 > t_0$  such that  $u \in \text{In}_v(t_0 + 1 : t_1) = \text{In}_v^a(t_0 + 1 : t_1)$ . This implies that  $(w, v)$  is an edge of  $\mathbb{G}^a(1 : t_1)$ , and hence  $w \in \text{In}_v^a(1 : \infty)$ .

From the definition of  $m_u^*$ , it follows that  $m_u^* \geq m_v^*$  for every agent  $v$ , and hence  $m_u^* \geq m^*$ . By definition of  $m^*$ , it holds that  $m_u^* \leq m^*$ . Therefore we have that  $m_u^* = m^*$  as required.  $\square$

## 6 Safe MinMax Algorithms for Stabilizing Consensus

We now define the subclass of *safe MinMax algorithms*, and present properties of dynamic graphs guaranteeing that safe MinMax algorithms always stabilize on the value  $m^*$ .

## 6.1 Definition of safe MinMax algorithms

Let  $m_u(t)$  be the minimal input value that  $u$  has heard of by round  $t$ , i.e.,

$$m_u(t) \stackrel{\text{def}}{=} \min_{v \in \text{In}_u^a(1:t)} (\mu_v).$$

Using this notation, the update rule (2) can then be rewritten into

$$y_u(t) = \max_{v \in \text{In}_u^a(\theta_u(t)+1:t)} (m_v(\theta_u(t))). \quad (3)$$

By definition, the sequence  $(m_u(t))_{t \geq 1}$  is non-increasing and lower-bounded by  $m_u^*$ . Thus it stabilizes to some limit value at some round denoted  $t_u$ . We let  $t^* = \max\{t_v : v \in V\}$ .

**Lemma 8.** *For each agent  $u$ ,  $\lim_{t \rightarrow \infty} m_u(t) = m_u^*$ .*

*Proof.* By definition of  $m_u^*$ , there exist some agent  $v$  and some round  $t$  such that  $v \in \text{In}_u^a(1:t)$  and  $m_u^* = \mu_v$ . Hence, we get  $m_u(t) \leq \mu_v$ , and so  $m_u(t) \leq m_u^*$ . Since  $\text{In}_u^a(1:t) \subseteq \text{In}_u^a(1:\infty)$ , we have  $m_u(t) \geq m_u^*$ , and the lemma follows.  $\square$

Now consider an arbitrary agent  $u$ . Our goal is to set restrictions on the cut-off function  $\theta_u$  enforcing that eventually  $y_u(t) = m^*$ . The first restriction is that for all large enough  $t$ ,

$$\forall v \in V, \quad m_v(\theta_u(t)) = m_v^*. \quad (4)$$

Because the sequence  $(m_u(t))_{t \geq 1}$  is stationary and by Lemma 8, the condition (4) is satisfied for all large enough  $t$  if  $\lim_{t \rightarrow \infty} \theta_u(t) = \infty$ .

Assuming that (4) holds for some  $t \in \mathbb{N}$ , we use Lemma 7 to show that if  $\text{In}_u^a(\theta_u(t) + 1 : t)$  contains an agent from  $\text{Ker}(\mathbb{G})$  then  $y_u(t) = m^*$  as needed. In a large class of dynamic graphs with non-empty kernels, the latter condition is satisfied whenever  $t - \theta(t)$  is larger than some constant (which may depend on the given dynamic graph). So our second restriction is that  $\lim_{t \rightarrow \infty} t - \theta_u(t) = \infty$ .

The above discussion leads to the following definition: A MinMax algorithm is *safe* if in each of its active runs, it holds that

$$\forall u \in V, \quad \lim_{t \rightarrow \infty} \theta_u(t) = \lim_{t \rightarrow \infty} t - \theta_u(t) = \infty. \quad (5)$$

**Theorem 9.** *Any active run of a safe MinMax algorithm on a dynamic graph  $\mathbb{G}$  achieves stabilizing consensus if there is a positive integer  $s$  such that*

$$\forall t \geq s, \forall u \in V, \quad \text{In}_u(\theta_u(t) + 1 : t) \cap \text{Ker}(\mathbb{G}) \neq \emptyset. \quad (6)$$

*Proof.* Without loss of generality, assume that  $s \geq t^*$  and  $s \geq \max_{v \in V} (s_v)$ . Let us consider an arbitrary agent  $u$ . Since the algorithm is safe, there is a positive integer  $t_0$  such that  $\theta_u(t) \geq s$  for all  $t \geq t_0$ . Then for  $t \geq t_0$ , Equation (3) can be rewritten into

$$y_u(t) = \max_{v \in \text{In}_u(\theta_u(t)+1:t)} (m_v^*). \quad (7)$$

This immediately implies that  $y_u(t) \leq m^*$ .

We also obtain from (7) that for every agent  $v \in \text{In}_u(\theta_u(t) + 1 : t)$ , it holds that  $y_u(t) \geq m_v^*$ . By (6), the set  $\text{In}_u(\theta_u(t) + 1 : t)$  contains at least one agent  $v$  in  $\text{Ker}(\mathbf{G})$ . Lemma 7 implies that  $m_v^* = m^*$ , and so  $y_u(t) \geq m^*$ .

We conclude that for all  $t > t_0$ , it holds that  $y_u(t) = m^*$ , i.e., the run achieves stabilizing consensus on  $m^*$ .  $\square$

As observed, the Min (or Max) algorithm solves stabilizing consensus in any infinitely connected dynamic graph. Simple examples show that this is not the case for some MinMax algorithms. However, as a direct consequence of Theorem 9, we obtain the following result.

**Corollary 10.** *Every safe MinMax algorithm solves the stabilizing consensus problem in the network model of infinitely connected dynamic graphs.*

As for dynamic graphs that are rooted with a bounded delay, the combination of Proposition 5 and Theorem 9 yields the following corollary.

**Corollary 11.** *Every safe MinMax algorithm solves the stabilizing consensus problem in the network model of dynamic graphs that are rooted with a bounded delay.*

Interestingly, Corollaries 10 and 11 are the analogs for stabilizing consensus and MinMax algorithms of the fundamental solvability results by Moreau [16] and by Cao, Morse, and Anderson [7] for asymptotic consensus and averaging algorithms. Observe, however, that Corollary 10 holds for all infinitely connected dynamic graphs while the Moreau's theorem requires the communication graph to be bidirectional at every round.

## 6.2 A limitation of safe MinMax algorithms

A natural question raised by Theorem 9 is whether safe MinMax algorithms solve stabilizing consensus for every dynamic graph with a non-empty kernel. We show that this is not the case, and first establish the following property of safe MinMax algorithms.

**Lemma 12.** *In any run of a safe MinMax algorithm in which stabilizing consensus is achieved, the stabilizing consensus value is equal to  $m^*$ .*

*Proof.* Let  $\tilde{y}$  be the stabilizing consensus value, and consider a agent  $u$  such that  $m_u^* = m^*$ . Observe that for all rounds  $t$  it holds that  $y_u(t) \geq m_u(t)$ , and for all large enough  $t$  we have  $y_u(t) = \tilde{y}$  and  $m_u(t) = m^*$ . Thus we get that  $\tilde{y} = y_u(t) \geq m_u(t) = m^*$  for all large enough  $t$ , and hence  $\tilde{y} \geq m^*$ .

Conversely, let us consider an arbitrary agent  $u$ . Since the algorithm is safe,  $\theta_u(t) > t^*$  if  $t$  is large enough. For each such  $t$ , by (3), we have  $y_u(t) = \max_{v \in \text{In}_u(\theta_u(t)+1:t)} (m_v^*)$ , which implies that  $y_u(t) \leq m^*$ . The value of  $y_u$  stabilizes to  $\tilde{y}$ , which shows that  $\tilde{y} \leq m^*$ .

It follows that  $\tilde{y} = m^*$ , as claimed.  $\square$

**Theorem 13.** *There is no safe MinMax algorithm that solves stabilizing consensus in the network model of dynamic graphs with non-empty kernels.*

*Proof.* The argument is by contradiction: suppose that there is a safe MinMax algorithm  $A$  that solves stabilizing consensus in the network model  $\mathcal{G}_{nek}$  of dynamic graphs over a fixed set  $V$  of  $n \geq 2$  nodes and with non-empty kernels. Let us denote  $V = \{u, v_1, \dots, v_{n-1}\}$ , and consider the following two digraphs  $G$  and  $H$  with the set of nodes  $V$ :

1.  $G$  is the directed chain  $u, v_1, \dots, v_{n-1}$ ;
2.  $H$  is the directed chain  $v_1, \dots, v_{n-1}, u$ .

We consider the active runs of  $A$  in which all the nodes start at round 1 and where all the input values are equal to 0, except the input value of the node  $u$  that is equal to 1.

First, we consider the dynamic graph  $\mathbb{G}_0$  in which  $\mathbb{G}_0(t) = G$  at all rounds  $t$ . Clearly, we have that  $\text{Ker}(\mathbb{G}_0) = \{u\}$ , and the corresponding maximal value  $m_0^*$  is 1. By Lemma 12 the consensus value in this run is equal to 1, i.e., there exists some positive integer  $t_0$  such that for each round  $t \geq t_0$  and each node  $w$ , it holds that  $y_w(t) = 1$ . In particular,  $y_{v_1}(t_0) = 1$ .

We now consider the dynamic graph  $\mathbb{G}_1$  such that  $\mathbb{G}_1(t) = G$  for  $1 \leq t \leq t_0$  and  $\mathbb{G}_1(t) = H$  for  $t > t_0$ . Clearly  $\text{Ker}(\mathbb{G}_1) = \{v_1\}$ , and for the corresponding run of  $A$ , we have  $m_1^* = 0$ . By Lemma 12, the consensus value in this run is 0, i.e., there exists some positive integer  $t_1$  such that for each round  $t \geq t_0 + t_1$  and each node  $w$ , it holds that  $y_w(t) = 0$ . In particular,  $y_{v_1}(t_0 + t_1) = 0$ .

By repeating the above construction, we determine an infinite sequence of positive integers  $(t_k)_{k \in \mathbb{N}}$  and the dynamic graph  $\mathbb{G}_\infty$  defined by

$$\mathbb{G}_\infty(t) = \begin{cases} G & \text{if } 1 \leq t \leq t_0 \text{ or } t_0 + \dots + t_{2k-1} + 1 \leq t \leq t_0 + \dots + t_{2k} \\ H & \text{if } t_0 + \dots + t_{2k} + 1 \leq t \leq t_0 + \dots + t_{2k+1}. \end{cases}$$

We easily check that  $\text{Ker}(\mathbb{G}_\infty) = \{u, v_1\}$ , and for the corresponding run of  $A$ , it holds that

$$y_{v_1}(t) = \begin{cases} 1 & \text{if } t = t_0 + \dots + t_{2k} \\ 0 & \text{if } t = t_0 + \dots + t_{2k+1}. \end{cases}$$

In this run with a non-empty kernel, the sequence  $(y_{v_1}(t))$  is not convergent, which violates the eventual agreement property.  $\square$

Extending the analogy above pointed out, we may observe that a similar impossibility result for averaging algorithms and asymptotic consensus is proved in [5] with a different collection of three node dynamic graphs.

### 6.3 Convergence time of safe MinMax algorithms

Contrary to safe averaging algorithms that converge in at most an exponential (in the size  $n$  of the network) number of rounds with dynamic graphs that are permanently rooted [7, 9], the convergence time of safe MinMax algorithms with such dynamic graphs may be arbitrarily large: For instance, inserting the complete digraph at one round  $t > t^*$  may result in changing the value  $m^*$ , and so may require the MinMax algorithm to stabilize again.

Thus MinMax algorithms are highly unstable with respect to – even sporadic – topology changes. However, if we restrict our analysis to a dynamic graph  $\mathbb{G}$  formed with a fixed rooted digraph, safe MinMax algorithms converge much faster than safe averaging algorithms. To see that, assume all nodes start at round one, and let  $K = \text{Ker}(\mathbb{G})$ . It is not hard to see that within less than  $|K|$  rounds, each  $m_v$  with  $v \in K$  stabilizes to  $m^*$ . Proposition 5 states that for each node  $u$  and each round  $t$  it holds that

$$K \cap \text{In}_u(t : t + n - |K|) \neq \emptyset.$$

It follows that if  $|K| \leq \theta_u(t) + 1 \leq t - (n - |K|)$ , then  $y_u(t) = m^*$ .

Since  $\lim_{t \rightarrow \infty} \theta_u(t) = \lim_{t \rightarrow \infty} t - \theta_u(t) = \infty$ , there exists a positive integer  $\alpha_u$  such that if  $t > \alpha_u$ , then it holds that

$$\theta_u(t) \geq |K| - 1 \text{ and } t - \theta_u(t) \geq n - |K| + 1.$$

We conclude that the algorithm stabilizes by round  $\max_{u \in V}(\alpha_u)$  rounds. In particular, setting  $\theta_u(t) = \lfloor t/2 \rfloor$  guarantees convergence within  $2n$  rounds.

The latter result can be interestingly compared with the exponential lower bound proved in [18, 9]: the convergence time of any safe averaging algorithm is exponential in  $n$  on the fixed rooted topology of a *Butterfly* digraph.

## 7 Efficient Distributed Implementation of MinMax Algorithms

In this section, we discuss distributed implementations of MinMax algorithms in our computing model. Figure 1 presents a general, efficient distributed scheme for this implementation, which is applicable whenever the *difference functions* defined by

$$\delta_u = t - \theta_u$$

are locally computable. The exact nature of the  $\delta_u$  functions is left unspecified (line 9).

Observe that the cut-off function  $\theta_u$  satisfies the inequalities  $0 \leq \theta_u(t) \leq t$  if and only if  $\delta_u$  satisfies the same inequalities. The inequalities  $0 \leq \delta_u(t) \leq t$  can be easily enforced by having the agent  $u$  implement the simple round counter  $C_u$  defined by  $C_u(t) = t - s_u$ . Indeed, the difference function  $\delta_u(t) = f(C_u(t))$  satisfies these two inequalities when  $f$  is any integer-valued function such that  $0 \leq f(t) \leq t$ . Besides we can choose  $f$  so that the difference function  $\delta_u(t) = f(C_u(t))$  provides a *safe* MinMax algorithm: for instance, we may set  $f(k) = \lfloor k/2 \rfloor$  or  $f(k) = \lfloor \log k \rfloor$ .

A possible, but quite inefficient way for implementing MinMax algorithms consists in using a *full information* protocol, in which at each round  $t$  each active agent sends its *local view* at round  $t - 1$  to all other agents; the local view of  $u$  at round  $t$  for  $t \geq s_u - 1$  is a rooted tree with labeled leaves, denoted  $T_u(t)$ , defined inductively as follows: First,  $T_u(s_u - 1)$  is a single vertex labelled by  $\mu_u$ . Assume now that at round  $t$ , the agent  $u$  receives  $k$  messages with the trees  $T_1, \dots, T_k$ . Then  $T_u(t)$  is the tree consisting of a root with  $k$  children on which the trees  $T_1, \dots, T_k$  are hanged. Using  $T_u(t)$ , the agent  $u$  can then easily compute  $y_u(t)$  corresponding to the cut-off point  $\theta_u(t) = t - \delta_u(t)$ .

The point of our implementation is precisely to avoid the construction of the trees  $T_u(t)$ . For that, each agent  $u$  maintains, in addition to  $y_u$  and  $\delta_u$ , a variable  $x_u$  with values in  $\mathcal{V}$ . At each round  $t$ , the agent  $u$  sets  $x_u$  to the minimal input value it has heard of, i.e.,  $x_u(t) = m_u(t)$ .

We say that an input value  $\mu$  is *relevant for agent  $u$  at round  $t$*  if there is an agent  $v \in \text{In}_u^a(t - \delta_u(t) + 1 : t)$  such that  $x_v(t - \delta_u(t)) = \mu$ . Thus, the agent  $u$  needs to set  $y_u$  to its maximal relevant value at each round, which is done as explained below.

Just to simplify notation, we assume that the set  $\mathcal{V}$  of all the possible initial values is finite and given. To determine the set of its relevant input values, the agent  $u$  maintains a vector of integers  $\text{AGE}_u$  such that for each  $\mu \in \mathcal{V}$ ,  $\text{AGE}_u[\mu](t)$  is the minimal number of rounds, by  $u$ 's local view at round  $t$ , that have passed since the last time some agent  $v$  had set  $x_v$  to  $\mu$ . Thus  $\mu$  is relevant for  $u$  at round  $t$  if and only if  $\text{AGE}_u[\mu](t) \leq \delta_u(t)$ .

Now we show that the algorithm corresponding to the difference functions  $\delta_u$  is a MinMax algorithm with the cut-off functions  $\theta_u = t - \delta_u$ . We start by two preliminary lemmas.

---

**Initialization:**

- 1:  $x_u \in \mathcal{V}$ , initially  $\mu_u$ ;  $y_u \in \mathcal{V}$ , initially  $\mu_u$ ;  $\delta_u \in \mathbb{N}$ , initially 0;  $\text{AGE}_u \in (\mathbb{N} \cup \infty)^m$ , initially  $\infty^m$
- 2:  $\text{AGE}_u[x_u] \leftarrow 0$
- 3: **while** TRUE **do**
- 4:   send  $\text{AGE}_u$  to all agents
- 5:   receive  $\text{AGE}_{v_1} \dots, \text{AGE}_{v_\ell}$
- 6:   for all  $\mu \in \mathcal{V}$ ,  $\text{AGE}_u[\mu] \leftarrow 1 + \min_{1 \leq i \leq \ell} (\text{AGE}_{v_i}[\mu])$
- 7:    $x_u \leftarrow \min\{\mu : \text{AGE}_u[\mu] < \infty\}$
- 8:    $\text{AGE}_u[x_u] \leftarrow 0$
- 9:    $\delta_u \leftarrow \dots$
- 10:    $y_u \leftarrow \max\{\mu : \text{AGE}_u[\mu] \leq \delta_u\}$
- 11: **end while**

Figure 1: Distributed implementation of a MinMax algorithm with the cut-off functions  $\theta_u = t - \delta_u$ .

---

**Lemma 14.** *For any agent  $u$  and any round  $t \geq 1$ ,  $x_u(t) = m_u(t)$ .*

*Proof.* This is an immediate consequence of the initialization of the variable  $x_u$  (line 1), of its update rule (line 7), and the fact that if  $t < s_u$  then  $x_u(t) = x_u(s_u - 1)$ .  $\square$

**Lemma 15.** *If the agent  $u$  is active at round  $t$ , then for each integer  $k \in \{0, \dots, t\}$ ,*

$$\text{AGE}_u[\mu](t) \leq k \Leftrightarrow \exists v \in \text{In}_u^a(t - k + 1 : t), x_v(t - k) = \mu.$$

*Proof.* First, assume that there is an agent  $v \in \text{In}_u^a(t - k + 1 : t)$  such that  $x_v(t - k) = \mu$ , and let  $t_0 = \max\{t - k, s_v - 1\}$ . Since for  $t < s_v$ , we have set  $x_v(t) = x_v(s_v - 1)$ , it always holds that  $x_v(t - k) = x_v(t_0) = \mu$ . Moreover, we easily check that  $v$  is in  $\text{In}_u^a(t_0 + 1 : t)$ . Hence there exists a  $v \sim u$  path in the interval  $[t_0 + 1, t]$  that we denote by  $v_0 = v, v_1, \dots, v_{t-t_0} = u$ . Because of the update rule of the vectors  $\text{AGE}_w$ , we deduce step by step that

$$\text{AGE}_{v_1}[\mu](t_0 + 1) \leq 1, \text{AGE}_{v_2}[\mu](t_0 + 2) \leq 2, \dots, \text{AGE}_u[\mu](t) \leq t - t_0.$$

The claim follows by observing that  $t - t_0 \leq k$ .

We now show by induction on  $k$  the following implication:

$$\text{AGE}_u[\mu](t) \leq k \Rightarrow \exists v \in \text{In}_u^a(t - k + 1 : t), x_v(t - k) = \mu.$$

*Base case:*  $\text{AGE}_u[\mu](t) = 0$ . By lines 6 and 8, we deduce that  $x_u(t) = \mu$ . Then the agent  $v = u$  is in  $\text{In}_u^a(t + 1 : t)$  with  $x_v(t) = \mu$ , as required.

*Inductive step:* Assume that the above implication holds for some non-negative integer  $k$ , and let  $\text{AGE}_u[\mu](t) \leq k + 1$ . Then either (a)  $\text{AGE}_u[\mu](t) \leq k$  or (b)  $\text{AGE}_u[\mu](t) = k + 1$ .

- (a) By inductive assumption, there is some agent  $w \in \text{In}_u^a(t - k + 1 : t)$  such that  $x_w(t - k) = \mu$ . Then we consider the two following cases:
  1. If  $x_w(t - k - 1) = x_w(t - k)$ , then we let  $v = w$ .

2. Otherwise,  $x_w(t-k-1) \neq x_w(t-k)$ , which means that in round  $t-k$ ,  $x_w$  was set to  $x_v(t-k)$  for some agent  $v$  in  $\text{In}_w^a(t-k)$  by executing line 7. Thus we have that  $x_w(t-k) = x_v(t-k-1) = \mu$ .

In both cases, the proof of the claim in case (a) is completed by noting that since  $\mathbb{G}^a(t-k:t) = \mathbb{G}^a(t-k) \circ \mathbb{G}^a(t-k+1:t)$ , we have that  $v \in \text{In}_u^a(t-k:t)$ .

- (b) By line 6, there is some agent  $w$  in  $\text{In}_u^a(t)$  such that  $\text{AGE}_w[\mu](t-1) = k$ . The inductive hypothesis implies that there exists some agent  $v$  in  $\text{In}_w^a(t-k:t-1)$  such that

$$x_v(t-1-k) = \mu.$$

Since  $\mathbb{G}^a(t-k:t) = \mathbb{G}^a(t-k:t-1) \circ \mathbb{G}^a(t)$ , it follows that  $v \in \text{In}_u^a(t-k:t)$  as required. □

**Theorem 16.** *Any instance of the scheme in Figure 1 is a MinMax algorithm, with cut-off functions given at each round  $t$  by  $\theta_u(t) = t - \delta_u(t)$ .*

*Proof.* If the agent  $u$  is active at round  $t$ , then we have  $y_u(t) = \max \{ \mu \in \mathcal{V} : \text{AGE}_u[\mu] \leq \delta_u(t) \}$  (cf. line 10). From Lemma 15, it follows that

$$y_u(t) = \max \{ \mu \in \mathcal{V} : \exists v \in \text{In}_u^a(\theta_u + 1 : t), x_v(\theta_u) = \mu \} \quad (8)$$

where  $\theta_u = t - \delta_u(t)$ . By Lemma 14, it holds that

$$x_v(\theta_u) = \min_{w \in \text{In}_v^a(1:\theta_u)} (\mu_w). \quad (9)$$

By Equations (8) and (9), we get that, with  $\theta_u = t - \delta_u(t)$ ,

$$y_u(t) = \max_{v \in \text{In}_u^a(\theta_u+1:t)} \left( \min_{w \in \text{In}_v^a(1:\theta_u)} \mu_w \right). \quad \square$$

The resulting MinMax algorithms share the same key features as averaging algorithms, namely they assume no leader, do not use identifiers, and tolerate asynchronous starts. Unlike averaging algorithms, they are not memoryless, but are space efficient in the sense that except the  $\text{AGE}_u$  counters, which are bounded by  $\log(t)$ , all other local variables are of bounded size. Actually, the unbounded counters  $\text{AGE}_u[\mu]$  – which imply unbounded storage capacities and unbounded bandwidth – are the discrete counterpart of the infinite precision required in averaging algorithms.

### Stabilizing consensus with failures

In the light of Corollary 11 and Theorem 16, we now revisit the problem of stabilizing consensus in the context of benign failures. In particular, we consider completely connected systems and the failure model of crashes or the one of send omissions. Basically, the resulting communication graphs are not strongly connected, and thus naive approaches (e.g., the Min algorithm) do not work.

To tackle this problem, we propose a strategy consisting first in emulating synchronized rounds and then in using a safe MinMax algorithm on top of this emulation. Indeed, as demonstrated

in [11], synchronized rounds with a dynamic communication graph can be easily emulated in any such distributed system, be it synchronous or asynchronous, when the network size is given: synchrony assumptions and failures are captured as a whole just by the connectivity properties of the dynamic graph. Typically, synchronized rounds with a dynamic graph that is *non-split*<sup>2</sup> at each round can be emulated if a minority of agents may crash or fail by send omissions. Since a non-split digraph is rooted, any safe MinMax algorithm on top of this emulation solves stabilizing consensus despite asynchrony and agent failures.

**Corollary 17.** *The stabilizing consensus problem is solvable in an asynchronous system with a complete topology, reliable links, and a minority of agents that crash or commit send omissions.*

## 8 Concluding Remarks

In this paper, we studied the stabilizing consensus problem for dynamic networks with very few restrictions on the computing model and the network. In particular, we did not restrict link changes, except for retaining a weak connectivity property, namely rootedness over sufficiently long periods of time, captured by the condition of a non-empty kernel. First we showed that this property is necessary for solving stabilizing consensus, and then proved that it is nearly a sufficient property, in the sense that every safe MinMax algorithm solves stabilizing consensus if the dynamic graph is rooted with a bounded delay. Our solvability results for stabilizing consensus and MinMax algorithms are actually the analogs of the ones for asymptotic consensus and averaging algorithms.

Our work leaves open several questions. First, it would be interesting to study whether the stabilizing consensus problem remains solvable when the dynamic graph is rooted with finite but unbounded delays. That may lead to the design of algorithms, other than MinMax algorithms, that solve stabilizing consensus with no strong connectivity. Another related question concerns convergence time. As demonstrated in Section 6.3, the convergence time of any safe MinMax algorithm is unbounded even for a dynamic graph that is permanently rooted, i.e., rooted with delay one. This raises the following question: does there exist another class of stabilizing consensus algorithms that reach consensus in bounded time – which might depend on the network size – for this specific model of dynamic graphs?

## References

- [1] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC*, pages 290–299. ACM, 2004.
- [2] Dana Angluin, Michael J. Fischer, and Hong Jiang. Stabilizing consensus in mobile networks. In Phillip B. Gibbons, Tarek Abdelzaher, James Aspnes, and Ramesh Rao, editors, *Distributed Computing in Sensor Systems*, volume 4026 of *Lecture Notes in Computer Science*, pages 37–50. Springer Berlin Heidelberg, 2006.
- [3] L. Becchetti, A. Clementi, E. Natale, F. Pasquale, and L. Trevisan. Stabilizing consensus with many opinions. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium*

---

<sup>2</sup>A digraph is non-split if any two nodes have a common in-neighbor.

- on *Discrete Algorithms*, SODA '16, pages 620–635, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics.
- [4] Martin Biely, Peter Robinson, and Ulrich Schmid. Agreement in directed dynamic networks. In Guy Even and Magnús M. Halldorsson, editors, *Proceedings of the 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 7355 of *Lecture Notes in Computer Science*, pages 73–84. Springer, Heidelberg, 2012.
  - [5] Vincent D. Blondel, Julien M. Hendrickx, Alex Olshevsky, and John N. Tsitsiklis. Convergence in multiagent coordination, consensus, and flocking. In *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference (CDC-ECC)*, pages 2996–3000. IEEE, New York, NY, 2005.
  - [6] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 104–121, 2015.
  - [7] Ming Cao, A. Stephen Morse, and Brian D. O. Anderson. Reaching a consensus in a dynamically changing environment: a graphical approach. *SIAM Journal on Control and Optimization*, 47(2):575–600, 2008.
  - [8] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. In Hannes Frey, Xu Li, and Stefan Rührup, editors, *ADHOC-NOW*, volume 6811 of *Lecture Notes in Computer Science*, pages 346–359. Springer, 2011.
  - [9] Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. Approximate consensus in highly dynamic networks: The role of averaging algorithms. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming, ICALP15*, pages 528–539, 2015.
  - [10] Bernadette Charron-Bost and Shlomo Moran. The Firing Squad Problem Revisited. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, volume 96 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
  - [11] Bernadette Charron-Bost and André Schiper. The Heard-Of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.
  - [12] Étienne Coulouma and Emmanuel Godard. A characterization of dynamic networks where consensus is solvable. In *SIROCCO*, pages 24–35. Springer, 2013.
  - [13] 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, SPAA '11*, pages 149–158, New York, NY, USA, 2011. ACM.
  - [14] Shimon Even. *Graph Algorithms*. Cambridge University Press, New York, NY, USA, 2nd edition, 2011.

- [15] Fabian Kuhn, Yoram Moses, and Rotem Oshman. Coordinated consensus in dynamic networks. In *Proceedings of the 30th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–10. ACM, 2011.
- [16] Luc Moreau. Stability of multiagent systems with time-dependent communication links. *IEEE Transactions on Automatic Control*, 50(2):169–182, 2005.
- [17] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. <http://bitcoin.org/bitcoin.pdf>, 2008.
- [18] Alex Olshevsky and John N. Tsitsiklis. Convergence speed in distributed consensus and averaging. *SIAM Review*, 53(4):747–772, 2011.
- [19] 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):1226–1229, 1995.
- [20] Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, 2009.