MinMax algorithms for stabilizing consensus

Bernadette Charron-Bost1 · Shlomo Moran2

Received: 11 March 2020 / Accepted: 19 March 2021 / Published online: 12 April 2021
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2021

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 and may join the system at any time (asynchronous start). 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 other agents. We stress that the bound on the time required for achieving this rootedness property is unknown to the agents. Such topologies are highly dynamic (in particular, roots may change arbitrarily over time) and may have very weak connectivity properties (an agent may be never a root). Our distributed MinMax algorithms thus require neither central control nor any global information and are also 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.

The stabilizing consensus problem [3,4,13] is a canonical problem in distributed control: each agent \( u \) starts with some initial value and repeatedly updates an output variable \( y_u \) which eventually stabilizes on the same input value. Until now, this problem has been studied in the context of Byzantine failures and a fixed topology. It is of basic interest also in the context of dynamic networks, as it lies between two fundamental problems in this model. One is the consensus problem (see e.g., [17]) in which decision is irrevocable, i.e., all the output variables \( y_u \) are write-once, and the other one is the asymptotic consensus problem [5,20], defined for the case of initial values that are real numbers, in which agents are only required to converge to the same outcome asymptotically. Moreover the limit value is only required to lie in the range of the initial values, while in stabilized consensus the limit value must be one of the initial values.

The original consensus problem, with irrevocable decisions, has been the subject of extensive study, specifically in the context of fault-tolerance and a fixed (complete) topology. There is also a large body of previous work on consensus in dynamic networks. In the latter works, agents are assumed to start synchronously, to share global information on the network, and to have distinct identifiers. Connectivity requirements that enable achieving consensus in such models were studied in [12,24]. Specifically, Coulouma and Godard [12] classified the oblivious message adversaries [1]—corresponding to the particular sets of dynamic graphs generated by a family of directed graphs—under which consensus is solvable. Thereafter Winkler et al. [24] extended such a characterization to any closed message adversaries (see [2,16]).

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 [23]. Averaging algorithms, where 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 (see for example [5,14,15,19]). Unlike the above mentioned consensus algorithms, averaging
algorithms remain valid in systems with unknown number of anonymous agents, and with asynchronous starts. 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. A close look at their proof shows that such an averaging algorithm highly takes benefit of the continuous domain of the output variables to force agreement despite the topology is not strongly connected (and thus despite some agents can never transmit their initial values to some others). Such algorithms may not achieve stabilizing consensus where the set of possible outcomes is finite. Moreover, it can be shown that, under the sole assumption of a continually rooted topology (possibly non-strongly connected), stabilizing consensus cannot be solved by just rounding the outputs of some averaging algorithm [18].

**Contribution.** The primary goal of this paper is the design of stabilizing consensus algorithms for synchronous, fault-free networked systems consisting of an unknown number of identical and anonymous agents connected by a time-varying topology and starting asynchronously. We assume neither strong connectivity, nor even the existence of permanent roots. It should be noted that while stabilizing consensus is trivially solved by a gossip algorithm when the time-varying topology is infinitely strongly connected over time, in the sense that for every pair of agents \(u\) and \(v\) there always exists a temporal path from \(u\) to \(v\), there is no obvious solution in the case where only a strict subset of agents, which is not given, can broadcast messages to all other agents. Observe moreover that in the absence of strong connectivity, 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 which are able, at any time, to send messages, possibly indirectly, to all other agents. A time-varying topology with a non-empty kernel is thus rooted with finite but a priori unbounded delays. We first observe that stabilizing consensus is not solvable in the case of an empty kernel. Then we show that under the rather weak assumption that these delays are bounded, and even if the bound is not given, the stabilizing consensus problem is solvable.

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 (anonymous network), and assume no global knowledge of the network structure or size. Moreover, they tolerate agents that 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 anonymous, asynchronous, and completely connected network with a minority of faulty agents that may crash or commit send omissions. This generalizes a similar result proved in [3] under the assumption that agents have unique identifiers that are mutually known. 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 finite network size \(n\).

We assume a round-based computational model in which 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 \(G(t) = (V, E_t)\), and is called the communication graph at 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 \(G = (G(t))_{t \in \mathbb{N}}\) is called a dynamic graph [8].

A network model is any non-empty set of dynamic graphs. Thus, a network model corresponds to the notion of communication predicates in the Heard-Of model [11] and to that of message adversaries in [1], which both define the rules by which messages can be blocked at each round.

In every run of an algorithm, each agent \(u\) is initially passive: it neither sends nor receives messages, and does 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 \( G(t) \) 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 \( G \), and the collection of the starting rounds. For each run, \( G^a(t) = (V, E^a_t) \) denotes the digraph where \( E^a_t \subseteq E_t \) is the set of edges that are either self-loops or connecting two agents that are active in round \( t \). The sets of \( u \)'s incoming neighbors (in-neighbors) in the digraphs \( G(t) \) and \( G^a(t) \) are denoted by \( \text{In}_u(t) \) and \( \text{In}^a_u(t) \), 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 \( G_1 \odot G_2 \), 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 \( G \) and any integers \( t' \geq t \geq 1 \), if \( G(t : t') = G(t) \odot \cdots \odot G(t') \). By convention, \( G(t : t) = G(t) \), and when \( 0 \leq t' < t \), \( G(t : t') \) is the digraph with only a self-loop at each node.

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

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

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

\[
G(t : \infty) = (V, \cup_{t' \geq t} E(G(t : t'))),
\]
\[
G^a(t : \infty) = (V, \cup_{t' \geq t} E(G^a(t : t'))),
\]

1 To simplify notations, there are self-loops at all nodes of \( G^a(t) \), including those corresponding to the passive agents at round \( t \). and denote by \( \text{In}_u(t : \infty) \) and \( \text{In}^a_u(t : \infty) \) the sets of \( u \)'s in-neighbors in these two digraphs, i.e.,

\[
\text{In}_u(t : \infty) = \cup_{t' \geq t} \text{In}_u(t : t'), \quad \text{In}^a_u(t : \infty) = \cup_{t' \geq t} \text{In}^a_u(t : t').
\]

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

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

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

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

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

\[
\forall t \geq s, \quad \overline{G}(t) = \overline{G}(s).
\]

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

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

Let us recall that a digraph \( G \) is transitively closed if any edge of \( G \odot G \) is an edge of \( G \). In the case that \( G \) has a self-loop at each node, this is equivalent to \( G \odot G = G \).

The transitive closure of \( G \), denoted by \( G^+ \), is the minimal transitively closed digraph that contains all edges of \( G \).

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

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

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

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

\[ \square \]

### 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 \( G \). \( G \) 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 \( G \), denoted by \( \text{Ker}(G) \), is defined as

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

or equivalently,

\[ \text{Ker}(G) = \bigcap_{t \geq 1} \text{CRoots}(G(t)). \]

**Proposition 3** If \( G \) is a dynamic graph with a permanent self-loop at each node, then

\[ \text{Ker}(G) = \text{CRoots}(G(\infty)) = \text{Roots}(G(\infty)). \]

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

\[ \text{Ker}(G) = \bigcap_{t \geq 1} \text{CRoots}(G(t)) = \text{CRoots}(G(\infty)). \]

By Theorem 2, the digraph \( G(\infty) \) is the transitive closure of \( G(\infty) \), and so

\[ \text{Roots}(G(\infty)) = \text{CRoots}(G(\infty)) \]

which completes the proof. \( \square \)

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

### 2.4 Bounded delay rootedness

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

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

Informally, if \( G \) is rooted with bounded delay, then the fixed bound on it's delay is not given.

Let us consider a dynamic graph \( G \) that is rooted with delay \( T \), and let \( R(t) \) denote the non-empty set of roots of \( G(t : t + T - 1) \). Since the set of nodes is finite, there is a non-empty set of nodes \( R \) that appears infinitely often in the sequence \( (R(t))_{t \geq 1} \). Clearly, every node in \( R \) is in \( \text{Ker}(G) \). Hence, 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 \( G(t) \). Proposition 5 below shows that these nodes are actually central roots over bounded length intervals.

**Proposition 5** If \( G \) is a dynamic graph that is rooted with delay \( T \), then there exists a positive integer \( s \) such that

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

**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 \( G_T \) defined by \( G_T(t) = G((t - 1)T + 1 : tT) \) that is rooted with delay one, i.e., permanently rooted.

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

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

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

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

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

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 \( v \in \text{Ker}(G) \) in the digraph \( G(t : t + i + 1) \), 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 \( k = \lvert \text{Ker}(G) \rvert \), and hence \( U_{n-k} = V \). Thus for every node \( u \in V \), it holds that \( \text{In}_u(t : t + T(n - k)) \cap \text{Ker}(G) \neq \emptyset \), 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 \( \alpha_u \in \mathcal{V} \) and an output variable \( y_u \) initialized to \( \alpha_u \). The algorithm \( A \) achieves stabilizing consensus in an active run with the initial values \( (\alpha_u)_{u \in \mathcal{V}} \) if the following properties hold:

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

**Eventual agreement** There exists some round \( s \) such that

\[
\forall t \geq s, \quad \forall u, v \in \mathcal{V}, \quad y_u(t) = y_v(s).
\]

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 \( \mathcal{G} \).

By a simple partitioning argument, we obtain the following basic impossibility result:

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

**Proof** Let \( \mathcal{G} \) 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 \( \mathcal{G}(\infty) \), called the condensation graph of \( \mathcal{G}(\infty) \), 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. From Propositions 3 we derive that the condensation graph of \( \mathcal{G}(\infty) \) 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 \( \mathcal{G} \). 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. \( \square \)

4 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 Sect. 6.

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 \( x_u \) that \( u \) repeatedly broadcasts and sets to the minimum input value it has heard of.

It is easy to see that on dynamic graphs that are infinitely connected, all the \( x_u \) 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 \( m_u^* \):

\[
m_u^* \overset{\text{def}}{=} \min_{v \in \mathcal{V}} (\alpha_v).
\]

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 \( \text{Ker}(\mathcal{G}) \), then \( m_u^* = m^* \), where

\[
m^* \overset{\text{def}}{=} \max_{v \in \mathcal{V}} (m_v^*).
\]

The above discussion suggests the following simple two phase scheme for solving 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 \( m_u^* \) value \( u \) has heard of.

There are two problems with implementing the min–max update rule described above: First, the borderline integer \( \theta \) is not given to the agents; second, round numbers \( t \) are not given either (in particular due to asynchronous start). The latter problem is handled in Sect. 6. To overcome the first problem, for each agent \( u \) and for each round \( t \), we assign an integer \( \theta_u(t) \in [1, t] \) such that \( \lim_{t \to \infty} \theta_u(t) = \infty \), and hence \( \theta_u(t) \geq \theta \) for all large enough \( t \). The value \( y_u(t) \) is then computed with the first phase consisting of the interval \([1, \theta_u(t)]\). This guarantees that for all large enough \( t \), each agent \( v \) has computed the value of \( m_v^* \) by round \( \theta_u(t) \).

We also need that, for all large enough \( t \), 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

\[
y_u(t) = \max_{v \in \text{Int}_u(\theta_u(t) + 1, t)} \left( \min_{w \in \text{Int}_u(1, \theta_u(t))} (y_w(0)) \right)
\]

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

**Proof** Let $w$ be an arbitrary agent in $\text{In}_u^1(1: \infty)$, and let $t_0 \in \mathbb{N}$ be such that $w \in \text{In}_w^1(1: t_0)$ and all agents are active at round $t_0$. Since $u$ is in Ker($G$), there is some round $t_2 > t_0$ such that $u \in \text{In}_u(t_0 + 1 : t_2) = \text{In}_u^1(t_0 + 1 : t_2)$. This implies that $(w, v)$ is an edge of $G(1 : t_2)$, and hence $w \in \text{In}_v^1(1: \infty)$.

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. \[\square\]

5 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 $m^*$ defined in Eq. (3).

5.1 Definition of safe MinMax algorithms

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

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

Using this notation, the update rule (4) can then be rewritten into

$$y_u(t) = \max_{v \in \text{In}_u^1(\theta_u(t) + 1: t)} (m_v(\theta_u(t))). \tag{5}$$

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 $t^* = \max\{t_u : v \in V\}$.

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

**Proof** By definition of $m_u^*$, there exist some agent $v$ and some round $t_u$ such that $v \in \text{In}_u^1(1 : t_u)$ and $m_u^* = \alpha_v$. Hence, we get $m_u(t) \leq m_v^*$ for all $t \geq t_u$. Since $\text{In}_u^1(1 : t) \subseteq \text{In}_u^1(1 : \infty)$ for every $t$, 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 $t$,

$$\forall v \in V, \quad m_v(\theta_u(t)) = m_v^*. \tag{6}$$

Because the sequence $(m_u(t))_{t \geq 1}$ is stabilizing and by Lemma 8, the condition (6) is satisfied for all large enough $t$ if $\lim_{t \to \infty} \theta_u(t) = \infty$.

Assuming that (6) holds for some $t \in \mathbb{N}$, we use Lemma 7 to show that if $\text{In}_u^1(\theta_u(t) + 1 : t)$ contains an agent from Ker($G$) then $y_u(t) = m^*$ as needed. In a large class of dynamic graphs with non-empty kernels, the latter condition is satisfied whenever $t - \theta(t) > t^*$ is larger than some constant (which may depend on the given dynamic graph). So our second restriction is that $\lim_{t \to \infty} t - \theta(t) = \infty$.

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

$$\forall u \in V, \lim_{t \to \infty} \theta_u(t) = \lim_{t \to \infty} (t - \theta(t)) = \infty. \tag{7}$$

A typical example of a safe MinMax algorithm that can be implemented locally at each agent $u$ is given by the cut-off function $\theta_u(t) = t - C_u(t)/2$, where $C_u(t)$ is the number of rounds since $u$ joined the algorithm.

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

$$\forall t \geq s, \forall u \in V, \quad \text{In}_u(\theta_u(t) + 1: t) \cap \text{Ker}(G) \neq \emptyset. \tag{8}$$

Moreover, the consensus value is $m^*$.

**Proof** Without loss of generality, assume that $s \geq t^*$ and $s \geq \max_{v \in V(s_v)}$, where $s_v$ is the round in which the agent $v$ starts running its local code. 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 $t \geq t_0$, Eq. (5) can be rewritten into

$$y_u(t) = \max_{v \in \text{In}_u(\theta_u(t) + 1: t)} (m_v^*). \tag{9}$$

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

We also obtain from (9) that for every agent $v \in \text{In}_u(\theta_u(t) + 1: t)$, it holds that $y_u(t) \geq m_v^*$. By (8), the set $\text{In}_u(\theta_u(t) + 1: t)$ contains at least one agent $v$ in Ker($G$). 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 $m^*$. \[\square\]
Equation (9) in the above proof allows for establishing the following useful property of safe MinMax algorithms.

**Lemma 10** 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 an agent \( u \) such that \( m^*_u = m^* \). Then for all rounds \( t \) it holds that \( y_u(t) \geq m_u(t) \geq m^*_u = m^* \). Since \( y_u(t) = \tilde{y} \) for all large enough \( t \), we get that \( \tilde{y} \geq m^* \).

Conversely, let us consider an arbitrary agent \( u \). Since the algorithm is safe, (9) holds for all large enough \( t \), and hence \( y_u(t) \leq \max_{v \in V}(m_v) = m^* \) for all large enough \( t \). Since the value of \( y_u \) stabilizes to \( \tilde{y} \), we get that \( \tilde{y} \leq m^* \). \( \square \)

As observed, the Min (or the analogous Max) algorithm solves stabilizing consensus in any infinitely connected dynamic graph. The following example shows that this is not the case for some MinMax algorithms even when all agents wake up simultaneously: Let \( G \) be the dynamic graph over \( \{u, v\} \), defined by \( G(1) = \{(v, u)\} \) and \( G(t) = \{(u, v), (v, u)\} \) for \( t > 1 \). Assume that \( a_u = 1, a_v = 1 \). Then \( m_v(1) = 1, m_v(t) = 0 \) for \( t > 1 \), and \( m_u(t) = 0 \) for all \( t \).

Now, if \( \theta_u(t) = 1 \) for all \( t \) and \( \theta_v(t) > 1 \) for \( t > 1 \), we get by Eq. (5) that \( y_u(t) = 1 \) for all \( t \) and \( y_v(t) = 0 \) for all \( t \).

However, as a direct consequence of Theorem 9, we obtain the following result.

**Corollary 11** 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 12** 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 11 and 12 are the respective analogs for stabilizing consensus and MinMax algorithms of the fundamental solvability results by Moreau [19] and by Cao et al. [7] for asymptotic consensus and averaging algorithms. Observe, however, that Corollary 11 holds for all infinitely connected dynamic graphs while the Moreau’s theorem requires the communication graph to be bidirectional at every round.

### 5.2 A limitation of safe MinMax algorithms

A natural question raised by Theorem 9 and Corollary 12 is whether there exists a safe MinMax algorithm that solves stabilizing consensus for every dynamic graph with a non-empty kernel. The following theorem provides a negative answer to this question.

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

**Proof** Given any safe MinMax algorithm \( A \), we construct a dynamic graph \( G_\infty \) over a set of \( n > 1 \) nodes \( V = \{u, v_1, \ldots, v_{n-1}\} \), and a run of \( A \) on \( G_\infty \), such that \( \text{Ker}(G_\infty) \neq \emptyset \), but the output value of agent \( v_1 \) never stabilizes in that run. Clearly, this will prove the theorem.

For each \( t \in \mathbb{N} \), \( G_\infty(t) \) will be one of the graphs \( G \) and \( K \) below:

1. \( G \) is the directed path \( u, v_1, \ldots, v_{n-1}; \)
2. \( K \) is composed of the directed path \( v_1, \ldots, v_{n-1} \) and the isolated node \( u \).

Specifically, \( G_\infty \) begins with \( t_0 \) consecutive instances of \( G \), followed by \( t_1 \) consecutive instances of \( K \), then followed by \( t_2 \) consecutive instances of \( G \), etc. We set \( t_0 = 1 \) (i.e., \( G_\infty(1) = G \)), and then define inductively for each \( k \geq 0 \) two positive integers \( t_{2k+1} \) and \( t_{2k+2} \). \( G_\infty \) is then defined as follows. Let \( \sigma_1 = 0 + t_1 + \cdots + t_i \). Then for all \( k \geq 0 \), \( G_\infty(t) = K \) for \( \sigma_2k < t \leq \sigma_{2k+1} \) and \( G_\infty(t) = G \) for \( \sigma_{2k+1} < t \leq \sigma_{2k+2} \).

In our inductive construction we assume only runs of \( A \) in which all the agents start at round 1, the input values of \( v_1, \ldots, v_{n-1} \) are equal to 0, and the input value of \( u \) is 1.

So we assume that \( t_0, t_1, \ldots, t_{2k} \) are defined for some non-negative \( k \), and define \( t_{2k+1} \) and \( t_{2k+2} \).

By the induction hypothesis, \( G_\infty(t) \) is already defined for \( 1 \leq t \leq \sigma_{2k} \). Let \( G_{2k+1}(t) = G_\infty(t) \) for \( t \leq \sigma_{2k} \) and \( G_{2k+1}(t) = K \) for \( t > \sigma_{2k} \), i.e.,

\[
G_{2k+1} = (G_\infty(1), \ldots, G_\infty(\sigma_{2k}), K, K, \ldots)
\]

Let \( \theta_{v_1} \) be the cut-off function of \( v_1 \) in the specified run of \( A \) on \( G_{2k+1} \). Then \( t_{2k+1} \) is defined by

\[
t_{2k+1} = \min \{ t : \theta_{v_1}(\sigma_{2k} + t) > \sigma_{2k} \},
\]

that is, \( t_{2k+1} \) is the minimum integer \( t \) such that the value of the cut-off function of \( v_1 \) at round \( \sigma_{2k} + t \) is strictly greater than \( \sigma_{2k} \) (note that \( t_{2k+1} \) is finite since \( A \) is safe). By the definition of \( G_\infty \), \( G_\infty(t) = K \) for \( \sigma_{2k} < t \leq \sigma_{2k+1} \), hence \( \text{In}_{v_1}(t) = \{v_1\} \) for \( \sigma_{2k} < t \leq \sigma_{2k+1} \). Since \( \sigma_{2k} < \theta_{v_1}(\sigma_{2k+1}) \leq \sigma_{2k+1} \), we get that \( \text{In}_{v_1}(\theta(\sigma_{2k+1}) = \{v_1\} \). Since \( m_{v_1}(t) = 0 \) for all \( t \), Eq. (5) implies that \( y_{v_1}(\sigma_{2k+1}) = 0 \).

Similarly, \( t_{2k+2} \) is defined by considering the specified run of \( A \) on the dynamic graph

\[
G_{2k+2} = (G_\infty(1), \ldots, G_\infty(\sigma_{2k+1}), G, G, \ldots),
\]
and \( t_{2k+2} \) is defined by
\[
t_{2k+2} = \min \{ t : \theta_{v_1}(\sigma_{2k+1} + t) > \sigma_{2k+1} \}.
\]

By imitating the above arguments, we get that \( G_\infty(t) = G \) for \( \sigma_{2k+1} < t \leq \sigma_{2k+2} \), implying that \( \text{In}_{v_1}(t) = \{ u, v_1 \} \) for \( \theta(\sigma_{2k+2}) \leq t \leq \sigma_{2k+2} \), and hence that
\[
\text{In}_{v_1}(\theta(\sigma_{2k+2}) : \sigma_{2k+2}) = \{ u, v_1 \}.
\]

Thus, by Eq. (5) and the fact that \( m_u(t) = 1 \) for all \( t \), we get that \( y_{v_1}(\sigma_{2k+1}) = \max(0, 1) = 1 \).

Since \( G_\infty(t) = G \) in infinitely many rounds \( t \), Ker(\( G_\infty \)) = \{ \{ u \} \} ≠ \emptyset. Also, for each \( k \geq 0 \), \( y_{v_1}(\sigma_{2k}) = 1 \) and \( y_{v_1}(\sigma_{2k+1}) = 0 \)—i.e., the output value of \( v_1 \) never stabilizes, as claimed.

The counterpart of Theorem 13 for safe averaging algorithms and asymptotic consensus is proved in [6], thus strengthening the analogy between averaging algorithms and MinMax algorithms already pointed out above.

### 5.3 Convergence time of safe MinMax algorithms

We now examine the time complexity of safe MinMax algorithms. For that, we restrict ourselves to executions in which all agents wake up at round 1 and with dynamic graphs that are permanently rooted. By Proposition 5, these dynamic graphs have each a non-empty kernel.

First let us observe that by Theorem 9 and its proof, an execution of a safe MinMax algorithm on a dynamic graph \( G \) stabilizes no later than at round \( t_0 \), provided that \( t_0 \) satisfies the following: The stabilization value \( m^* \)—which is one of the initial values—has reached some agent \( v \in \text{Ker}(G) \) by round \( \theta_{v}(t_0) \), and the messages sent by agent \( v \) at round \( \theta(t_0) + 1 \) (possibly indirectly) reaches all other agents at time \( t_0 \). Thus the stabilizing time is of the order of the number of rounds required for \( m^* \) to reach an agent \( v \in \text{Ker}(G) \), plus the “dynamic” eccentricity of the node \( v \) in \( G \).

Unfortunately, the number of rounds required for \( m^* \) to reach an agent in the kernel can be arbitrarily large even when the (permanently rooted) dynamic graph \( G \) is almost constant: As an example, let \( V \) and \( G \) be as defined in the proof of Theorem 13, and let \( H \) be the directed path \( v_1, \ldots, v_{n-1}, u \); assume further that \( \alpha_{u} = 1 \) and \( \alpha_{v_i} = 0 \) for \( i = 1, \ldots, n-1 \) (as in that proof). Now let \( t_1 \) be some large integer, and let \( G \) be the dynamic graph defined by
\[
G(t) = \begin{cases} 
G & \text{if } 1 \leq t < t_1 \\
H & \text{if } t_1 \leq t.
\end{cases}
\]

The dynamic graph \( G \) is permanently rooted and \( \text{Ker}(G) = \{ v_1 \} \). The stabilization value is \( m^* = 0 \), and \( m^* \) reaches the agent \( u \) no earlier than at round \( t_1 \), which can be arbitrarily large.

The above example shows that the convergence time of safe MinMax algorithms can be arbitrarily large even in highly restricted permanently rooted dynamic graphs. This can be interestingly compared with the convergence times of safe averaging algorithms which are at most exponential in the network size with any permanently rooted dynamic graph [7,9].

However, if we restrict our analysis to permanently rooted dynamic graphs \( G \) such that
\[
\forall t \geq 1, \quad \text{Roots}(G(t)) = \text{Ker}(G),
\]
we easily get that the stabilization value is actually equal to
\[
m^* = \min_{v \in \text{Ker}(G)} (\sigma_v),
\]
which implies that the bound on the convergence time of safe MinMax algorithms on such dynamic graphs is much smaller than the bound on the convergence time of safe averaging algorithms. Indeed let \( K = \text{Ker}(G) \) denote the assumed fixed root component. Then the above expression of \( m^* \) shows that within less than \( |K| \) rounds, each \( m_v \) with \( v \in K \) stabilizes to \( m^* \). Proposition 5 states that for each agent \( u \) 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 \( \lim_{t \to \infty} \theta_u(t) = \lim_{t \to \infty} t - \theta_u(t) = \infty \), there exists a positive integer \( k_u \) such that if \( t > k_u \), then it holds that
\[
\theta_u(t) \geq |K| - 1 \quad \text{and} \quad t - \theta_u(t) \geq n - |K| + 1.
\]
We conclude that the algorithm stabilizes by round \( \max_{u \in V}(k_u) \) rounds. In particular, setting \( \theta_u(t) = \lfloor t/2 \rfloor \) guarantees convergence within \( 2n \) rounds. For comparison, it is proved in [9,21] that the convergence time of any safe averaging algorithm is exponential in \( n \) on the fixed rooted topology of a Butterfly digraph.

### 6 Efficient distributed implementation of MinMax algorithms

In this section, we discuss distributed implementations of MinMax algorithms in our computing model. Figure 1 incorporates...
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 of 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 \( \theta_u \). Assume now that at round \( t \), the agent \( u \) receives \( k \) messages with the trees \( T_1, \ldots, T_k \). Then \( T_u(t) \) is the tree consisting of a root with \( k \) children on which the trees \( T_1, \ldots, T_k \) 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 the local variables \( y_u \) and \( \delta_u \), a variable \( x_u \) with values in \( V \). At each round \( t \), the agent \( u \) sets \( x_u \) to the minimal value it has heard of, i.e., \( x_u(t) = m_u(t) \).

We say that an input value \( \alpha \) is relevant for agent \( u \) at round \( t \) if there is an agent \( v \in \mathrm{In}_u^a(t - \delta_u(t) + 1 : t) \) such that \( x_v(t - \delta_u(t)) = \alpha \). 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 \( V \) 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 \( \mathrm{AGE}_u \) such that for each \( \alpha \in V \), \( \mathrm{AGE}_u[\alpha](t) \) is the 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 \( \alpha \). Thus \( \alpha \) is relevant for \( u \) at round \( t \) if and only if \( \mathrm{AGE}_u[\alpha](t) \leq \delta_u(t) \).

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.

**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 convention (cf. Section 2.1) that if \( t < s_u \) then \( x_u(t) = x_u(s_u - 1) \).

**Lemma 15** If the agent \( u \) is active at round \( t \), then for each integer \( k \in \{0, \ldots, t\} \),

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

**Proof** First, assume that there is an agent \( v \in \mathrm{In}_u^a(t - k + 1 : t) \) such that \( x_v(t - k) = \alpha \), and let \( t_0 = \max\{t - k, s_v - 1\} \). By the above-mentioned convention, for every integer \( t \) with \( 1 \leq t < s_v \), we have

\[ x_v(t) = x_v(s_v - 1) \]

Hence it always holds that \( x_v(t - k) = x_v(t_0) = \alpha \). Moreover, we easily check that \( v \) is in \( \mathrm{In}_u^a(t_0 + 1 : t) \). Hence there exists a \( v \sim u \) path in the interval \( [t_0 + 1, t] \) that we denote by \( v_0 = v, v_1, \ldots, v_{t_0 - t_0} = u \). Because of the update rule of the vectors \( \mathrm{AGE}_w \), we deduce by step by step that

\[ \mathrm{AGE}_{v_1}[\alpha](t_0 + 1) \leq 1, \quad \mathrm{AGE}_{v_2}[\alpha](t_0 + 2) \leq 2, \ldots, \quad \mathrm{AGE}_{v}[\alpha](t) \leq t - t_0. \]

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

We now show by induction on \( k \) the following implication:

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

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

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

(a) By inductive assumption, there is some agent \( w \in \mathrm{In}_u^a(t - k + 1 : t) \) such that \( x_w(t - k) = \alpha \). Then we consider the following two 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 \( \mathrm{In}_w^a(t - k) \) by executing line 7. Thus we have that \( x_w(t - k) = x_v(t - k - 1) = \alpha \).

In both cases, the proof of the claim in case (a) is completed by noting that since \( \mathcal{G}^a(t - k : t) = \mathcal{G}^a(t - k) \circ \mathcal{G}^a(t - k + 1 : t) \), we have that \( v \in \mathrm{In}_u^a(t - k - 1 : t) \).
By line 6, there is some agent $w$ in $\text{In}^a_u(t)$ such that $\text{AGE}_w[\alpha](t - 1) = k$. The inductive hypothesis implies that there exists some agent $\nu$ in $\text{In}^a_u(t - k : t - 1)$ such that

$$x_\nu(t - 1 - k) = \alpha.$$  

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

**Theorem 16** Any instance of the scheme in Fig. 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 $y_u(t) = \max \{\alpha \in \mathbb{V} : \text{AGE}_U[\alpha] \leq \delta_u(t)\}$ (cf. line 10). From Lemma 15, it follows that

$$y_u(t) = \max \{\alpha \in \mathbb{V} : \exists \nu \in \text{In}^a_u(\theta_u + 1 : t), x_\nu(\theta_u) = \alpha\} \quad (10)$$

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

$$x_\nu(\theta_u) = \min_{\nu \in \text{In}^a_u(\theta_u)} (\alpha_w). \quad (11)$$

By Eqs. (10) and (11), we get that, with $\theta_u = t - \delta_u(t)$,

$$y_u(t) = \max_{\nu \in \text{In}^a_u(\theta_u + 1 : t)} \left( \min_{\nu \in \text{In}^a_u(\theta_u)} \alpha_w \right).$$

6.1 Stabilizing consensus with failures

In the light of Corollary 12 and Theorem 16, we now revisit the problem of stabilizing consensus in the context of benign failures, in both synchronous and asynchronous settings. 

**Asynchronous systems and faulty senders** We first consider completely connected networks and the failure model of **faulty senders** in an asynchronous setting. We show that safe MinMax algorithms enable stabilizing consensus, provided that there are less than $n/2$ faulty agents.

Our strategy consists of first emulating synchronized rounds, and then 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, in an asynchronous and completely connected system of $n$ agents with at most $f$ faulty senders, the connectivity property of the communication graph $G$ that may be enforced is

$$\forall u \in \mathbb{V}, \forall t \geq 1, |\text{In}_u(t)| \geq n - f.$$  

Therefore if only a minority of agents may be faulty ($f < n/2$), then every pair of nodes in the communication digraph $G(t)$ at round $t$ has a common in-neighbor, i.e., at every round $t$, any two agents hear of a common agent. In other words, every digraph $G(t)$ is **non-split**. Since any non-split digraph is rooted, Corollary 12 shows that any safe MinMax algorithm on top of this emulation solves stabilizing consensus.

**Corollary 17** The stabilizing consensus problem is solvable in an asynchronous system with a complete topology, reliable links, and less than half of the agents commit send omissions.

**Synchronous systems and link failures** In synchronous systems, computations are directly organized into synchronized rounds (cf. [17]), and benign communication failures are then quantified by the number of message losses per round. As above, we consider a fixed completely connected network, and message losses then result in a time-varying...
communication graph. Charron-Bost et al. [9] showed that any digraph with \( n \) nodes and at least \( n^2 - 3n + 3 \) edges is rooted. Taking into account the \( n \) self-loops and since \( n^2 - 3n + 3 = (n^2 - n) - (2n - 3) \), we derive the following solvability result from Corollary 12.

**Corollary 18** The stabilizing consensus problem is solvable by any safe MinMax algorithm in a complete network with \( n \) agents if there are at most \( 2n - 3 \) message losses per round.

Compared with the impossibility result established by Santoro and Widmayer [22] for consensus with \( n - 1 \) faults per round, our result demonstrates that the number of link faults that can be tolerated increases by a factor 2 when solving stabilizing consensus instead of consensus.

### 7 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, captured by the condition of a non-empty kernel, namely rootedness over sufficiently long periods of time. First we observed 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 in the network model of dynamic graphs with non-empty kernels. Since by Theorem 13 no safe MinMax algorithm achieves this goal, such study may lead to the design of different techniques that solve stabilizing consensus with no strong connectivity.

A closely related question is to find the borderline between the connectivity requirements that enable solution to consensus, and those that do not enable such solutions but make stabilizing consensus solvable. A partial answer to this question is provided by our solvability result of stabilizing consensus in synchronous systems with link faults which demonstrates that the number of link faults that can be tolerated increases by a factor 2 when solving stabilizing consensus instead of consensus.

In a more general standpoint, we propose to start with a recent result in [24] that characterizes closed network models in which consensus is solvable with distinct identities and synchronous starts, and to follow a similar approach by limiting our study to closed network models. An important step would then be to determine whether stabilizing consensus is solvable in any closed network model that contains only dynamic graphs with non-empty kernel.\(^3\)

Another open question concerns convergence time: As demonstrated in Sect. 5.3, 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 the network model of permanently rooted dynamic graphs?

**Acknowledgements** We would like to thank the anonymous referees for their helpful comments.

### Declaration

**Conflict of interest** The authors declare that they have no conflict of interest.

### References

1. Afek, Y., Gafni, E.: Asynchrony from synchrony. In: International Conference on Distributed Computing and Networking, pp. 225–239. Springer, Berlin (2013)
2. Alpern, B., Schneider, F.B.: Defining liveness. Inf. Process. Lett. 21(4), 181–185 (1985)
3. Angluin, D., Fischer, M.J., Jiang, H.: Stabilizing consensus in mobile networks. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R., eds.) Distributed Computing in Sensor Systems. Lecture Notes in Computer Science, vol. 4026, pp. 37–50. Springer, Berlin (2006)
4. Becchetti, L., Clementi, A., Natale, E., Pasquale, F., Trevisan, L.: Stabilizing consensus with many opinions. In: Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA’16, pp. 620–635, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics (2016)
5. Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and Distributed Computation: Numerical Methods. Athena Scientific, Belmont (1989)
6. Blondel, V.D., Hendrickx, J.M., Olshevsky, A., Tsitsiklis, J.N.: 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), pp. 2996–3000. IEEE, New York, NY (2005)
7. Cao, M., Stephen, M.A., Anderson, B.D.O.: Reaching a consensus in a dynamically changing environment: a graphical approach. SIAM J. Control Optim. 47(2), 575–600 (2008)
8. Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. Int. J. Parallel Emerg. Distrib. Syst. 27(5), 387–408 (2012)
9. Charron-Bost, B., Függer, M., Nowak, T.: Approximate consensus in highly dynamic networks: the role of averaging algorithms. In: Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming, ICALP’15, pp. 528–539 (2015)
10. Charron-Bost, B., Moran, S.: The firing squad problem revisited. In: Rolf, N., Brigitte, V. (eds.) Distributed Computing in Sensor Systems. Lecture Notes in Computer Science, vol. 4026, pp. 37–50. Springer, Berlin (2006)

\(^3\) Observe that the network model consisting of all dynamic graphs with non-empty kernel is not closed.
11. Charron-Bost, B., Schiper, A.: The Head-Of model: computing in distributed systems with benign faults. Distrib. Comput. 22(1), 49–71 (2009)
12. Coulouma, É., Godard, E., Peters, J.: A characterization of oblivious message adversaries for which consensus is solvable. Theor. Comput. Sci. 584, 80–90 (2015)
13. Doerr, B., Goldberg, L.A., Minder, L., Sauerwald, T., Scheideler, C.: 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, pp. 149–158, New York, NY, USA (2011)
14. Jadbabaie, A., Lin, J., Stephen, M.A.: Coordination of groups of mobile autonomous stability agents using nearest neighbor rules. IEEE Trans. Autom. Control 48(6), 988–1001 (2003)
15. Lorenz, D.A., Lorenz, J.: Convergence to consensus by general averaging. In: Bru, R., Romero-Vivó, S. (eds.) Positive Systems, Volume 389 of Lecture Notes in Control and Information Sciences, pp. 91–99, Springer, Berlin (2009)
16. Lubitch, R., Moran, S.: Closed schedulers: a novel technique for analyzing asynchronous protocols. Distrib. Comput. 8(4), 203–210 (1995)
17. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, San Francisco (1996)
18. Moran, S.: Averaging and rounding cannot achieve stabilizing consensus in rooted dynamic graphs. Unpublished note (2020). Available at https://www.dropbox.com/s/h08myw1huprxw/g/AC-note-SC2002.pdf?dl=0
19. Moreau, L.: Stability of multiagent systems with time-dependent communication links. IEEE Trans. Autom. Control 50(2), 169–182 (2005)
20. Olfati-Saber, R., Fax, J.A., Murray, R.M.: Consensus and cooperation in networked multi-agent systems. Proc. IEEE 95(1), 215–233 (2007)
21. Olshevsky, A., Tsitsiklis, J.N.: Convergence speed in distributed consensus and averaging. SIAM Rev. 53(4), 747–772 (2011)
22. Santoro, N., Widmayer, P.: Time is not a healer. In: Proceedings of the 6th Symposium on Theoretical Aspects of Computer Science, pp. 304–313, Paderborn, Germany (1989)
23. Vicsek, T., Czirok, A., Ben-Jacob, E., Cohen, I., Shochet, O.: Novel type of phase transition in a system of self-driven particles. Phys. Rev. Lett. 75(6), 1226–1229 (1995)
24. Winkler, K., Schmid, U., Moses, Y.: A characterization of consensus solvability for closed message adversaries. In: 23rd International Conference on Principles of Distributed Systems (OPODIS 2019). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.