Decentralized decision-making on robotic networks with hybrid performance metrics

Federico Rossi Marco Pavone

***Abstract*— The past decade has witnessed a rapidly growing interest in distributed algorithms for collective decision-making. For a large variety of settings algorithms are now known that *either* minimize time complexity (i.e., convergence time) *or* optimize communication complexity (i.e., number and size of exchanged messages). Yet, little attention has beed paid to the problem of studying the inherent trade off between time and communication complexity. Generally speaking, the optimization of the time complexity metric leads to fast and robust distributed algorithms; however, such algorithms often require a massive amount of messages to be exchanged and, consequently, might lead to unacceptable energy requirements for message transmission. On the other hand, the communication complexity metric leads to distributed algorithms that are very economical in terms of exchanged messages, but are extremely sensitive to link failures and converge too slowly in practical scenarios. In this paper we bridge this gap by designing and rigorously analyzing a *tunable* algorithm that solves a general version of the distributed consensus problem (that includes voting and mediation), on undirected network topologies and in the presence of infrequent link failures. The tuning parameter allows to gracefully transition from time-optimal to byte-optimal performance (hence, it allows to achieve hybrid performance metrics), and determines the algorithm's robustness, measured as either the number of single points of failure or the time required to recover from a failure. Our results leverage a novel connection between the consensus problem and the theory of gamma synchronizers. Simulation experiments that corroborate our findings are presented and discussed.**

I. Introduction

Distributed decision-making in robotic networks is an ubiquitous problem, with applications as diverse as state estimation [1], formation control [2], and cooperative task allocation [3]. In particular, the consensus problem, where the robots in the network have to agree on some common value, has seen a resurgence of interest in the last decade following the paper from Jadbabaie, Lin and Morse [4]. Most recent efforts in the control community have primarily focused on studying the properties and fundamental limitations of *average-based* consensus, a subclass of the consensus problem in which nodes average their status with their neighbors at each (continuous or discrete) time step [5]. In these works, the dominant performance metric is time complexity, i.e., the convergence time. The computer science community has, instead, focused the attention on the communication complexity, and "communication-optimal" algorithms for consensus problems are now known.

Despite the large interest on consensus problems in the last decade, little attention has been devoted to the problem of studying the inherent trade off between time and communication complexity. Generally speaking, the optimization of the time complexity metric leads to fast and robust distributed algorithms; however, such algorithms often require a massive amount of messages to be exchanged and, consequently, might lead to unacceptable energy requirements for message transmission. In this paper we bridge this gap by designing and rigorously analyzing a *tunable* algorithm that solves in *finite* time a general version of the distributed consensus problem, specifically the *convex* consensus problem, where the robots have to agree on a *convex* combination of their initial conditions. The convex consensus problem models decision-making problems as diverse as average consensus, leader election, collective data fusion, voting, and mediation. We consider settings where the underlying network topology is undirected and failures are *infrequent*. The tuning parameter allows to gracefully transition from time-optimal to byte-optimal performance (hence, it allows to achieve hybrid performance metrics), and determines the algorithm's robustness, measured as either the number of single points of failure or the time required to recover from a failure. Our results leverage a novel connection between the consensus problem and the theory of gamma synchronizers. The algorithm allows to achieve tradeoffs between time and communication (or byte) complexity, thus allowing to fulfill *hybrid* performance requirements.

This paper is structured as follows. After introducing some useful concepts in section II and presenting the problem in section III, in section IV we outline our algorithm's structure. Worst-case complexity is analyzed in section V and numerical simulations on random geometric graphs are presented in section VI. Potential applications and future research directions are discussed in section VII.

II. Preliminaries

In this section we first discuss the network model that we will consider in this paper and relevant performance metrics. Then, we will briefly overview the algorithms that are known to be either time or communication optimal.

A. Model for asynchronous robotic networks with infrequent failures

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

Federico Rossi and Marco Pavone are with the Department of Aeronautics and Astronautics, Stanford University, Stanford, CA, 94305, {frossi2, pavone}@stanford.edu

E. The neighborhood set of node $i \in V$, N_i, is the set of nodes $j \in V$ neighbors of node i.

Each node is internally modeled as a input/output (I/O) automaton, which is essentially a labeled state transition system commonly used to model reactive systems (we refer the reader to [6, ch. 8] for a formal definition). All nodes are identical except, possibly, a unique identifier (UID). The following key assumptions characterize the time evolution of each node in the graph G:

- Fairness [6, ch. 8]: the order in which transitions happen and messages are delivered is not fixed a priori. However, any enabled transition will *eventually* happen.
- Non-blocking [6, ch. 8]: every transition is activated within l time units of being enabled and every message is delivered within d time units of being dispatched.

Communication channels may experience *stopping failures*, modeled by deletion of the corresponding edge in G. Links may go offline but not come back: messages across the link at the time of failure are dropped. Agents on both sides of the link are notified of the failure immediately.

We assume that new links can not be added to the network during execution, hence, the network is *static*. Finally, we assume that each agent knows at least an upper bound $\bar{n}$, with $n + \underline{t} \geq \bar{n} \geq n$, on the number of nodes in the network. The role of the parameter $\underline{t}$ will be made clearer in section IV. This hypothesis is natural in many engineering applications, where the initial number of sensors or vehicles is known but a small number of nodes may fail during deployment.

B. Time, communication, and byte complexity

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

The following definitions naturally capture the notions of time complexity, communication complexity, and byte complexity and are widely used in the theory of distributed algorithms [6, ch. 8].

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

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

where $T(a, k, \alpha, G)$ is the first time when all nodes have computed the correct value for problem P and have stopped. The order of the inf-sup operands in the above expression is naturally induced by our definition. By dropping the leading $\inf_{a \in \mathcal{A}}$, one recovers the complexity of a given algorithm a.

For synchronous algorithms, time complexity is typically expressed in rounds; in an asynchronous setting, we express time complexity in multiples of $l+d$, defined in section II-A. We will henceforth refer to $(l + d)$ as a *time unit*.

2) Byte complexity: In many instances, message size plays an important role in the energy needed for information transmission. The number of messages exchanged fails to capture this effect. To address this issue, one defines the byte as the infimum worst-case (over initial values and fair executions) *overall size* (in bytes) of all messages exchanged by an algorithm before its completion. Rigorously, the byte complexity for a given problem P with respect to the class of graphs $\mathcal{G}$ is

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

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

In this paper we are specifically interested in the asymptotic growth of TCand BC; accordingly, we briefly review some useful notation for asymptotic performance. For $f, g : \mathbb{N} \to \mathbb{R}$, $f \in O(g)$ (respectively, $f \in \Omega(g)$) if there exist $N_0 \in \mathbb{N}$ and $k \in \mathbb{R}_{>0}$ such that $|f(N)| \leq k|g(N)|$ for all $N \geq N_0$ (respectively, $|f(N)| \geq k|g(N)|$ for all $N \geq N_0$). If $f \in O(g)$ and $f \in \Omega(g)$, then the notation $f \in \Theta(g)$ is used.

C. Discussion of complexity measures for communication

Energy consumption is a limiting factor for a variety of cyber-physical systems, for example robotic swarms and wireless sensor networks, and wireless communication is often one of the main contributors to battery depletion. Yet, most of the research on distributed algorithms for robotic networks focuses on *time* complexity, with little attention to the *energy* required for their execution.

Our work strives to explore the "energy complexity" of the distributed consensus problem. Byte complexity is a reasonable proxy for the energy cost of a problem in settings where:

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

Linear dependence of the cost on payload size holds true for lightweight protocols whose handshakes, headers and acknowledgements are small with respect to the actual payload.

We remark that, if the energy cost of a message is *independent* of the message size, *message* complexity, i.e. the overall number of messages exchanged, is an appropriate proxy for energy cost. This may be the case whenever the cost of the message is dominated by the fixed cost to establish the connection, handshake, exchange connection parameters, and frame the payload.

Throughout this work, we will focus on byte complexity.

We remark that the exclusion of broadcast protocols is compatible with the presence of efficient narrow-band, high-gain mechanically or electronically steerable antennas on the agents participating in the process, or with use of a network protocol that does not implement broadcasts. The proper modeling of "energy complexity" for broadcast models is beyond the scope of this paper and is left for future research.

D. Time-optimal and byte-optimal algorithms for the consensus problem

If the consensus function depends on all nodes' initial values, $\mathrm{Diam}(G)$ represents a trivial lower bound on the number of time units required by *any* consensus algorithm. A simple flooding algorithm achieves this lower bound; in fact, one can easily show that the time complexity of flooding is $\mathrm{Diam}(G)$ time units. Hence, flooding is time-optimal. On the other hand, one can also show that the byte complexity is $O(|E|nb)$, where b is the size in bytes of one agent's initial value. Both of them are very large and limit the applicability of flooding algorithms (average-consensus algorithms belong to the family of flooding algorithms).

The GHS distributed minimum spanning tree algorithm, proposed by Gallager, Humblet and Spira [7], allows to build a *rooted* minimum spanning tree (MST) on synchronous and asynchronous static networks with $O(n\log(n))$ time and $O((n\log(n)+|E|)\log n)$ byte complexity. Once a rooted spanning tree is in place, the root can collect information from all nodes using the tree and compute any consensus function with $O(n)$ messages of size $O(\log n)$. Message size depends on the nature of the consensus function under consideration. Improved versions of the GHS algorithm such as [8] can yield a time complexity of $O(n)$ with no degradation in byte complexity. Under some mild assumptions, the improved GHS algorithm can be shown to be message-optimal [8]; if messages carry a sender or receiver ID, byte optimality follows. Its time-complexity, however, is significantly higher than that of the time-optimal flooding algorithms; furthermore, GHS-like algorithms are very fragile with respect to link failures.

III. Problem Formulation

In this paper we focus on finding distributed algorithms to solve *convex consensus problems* on *hierarchically computable* functions, which fulfill hybrid time/communication performance requirements. In this section, we first rigorously define the notion of convex consensus. Then we define the class of hierarchically computable functions. Finally we formally state the problem we wish to solve.

A. Convex consensus

The convex consensus problem is defined as follows:

Definition 3.1 (Convex consensus): Consider n nodes indexed by $\{1,\dots,n\}$ arranged in an undirected graph and capable of exchanging information according to the asynchronous network model presented in Section II-A. Each node is equipped with an initial value $k_i \in \mathbb{R}^n$ (representing, e.g., a local measurement of an environmental phenomenon). The goal is for all nodes to agree in a *finite* number of steps on a common value $\bar{k}$ lying within the convex hull of the initial values k_i, $i=1,\dots,n$; in other words, $\bar{k}$ can be *represented* as a convex combination of the initial values k_i's, i.e.:

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

where the weights c_i, $i=1,\dots,n$, are problem-dependent . In other words, the vector of weights $[c_i]_i$ *parameterizes* the convex consensus problem. The weights might be unknown to the agents.

The convex consensus problem models a variety of decision problems of interest for robotic sensor networks. Some examples include:

- Computation of $\max_i k_i$ (equivalently, $\min_i k_i$), e.g., for leader election. This problem can be represented with the weight choice (assuming there exists a unique maximum or minimum): $c_i = 1$ if $k_i = \max_j(k_j)$ (equivalently, $k_i = \min_j(k_j)$) and $c_i = 0$ otherwise.
- Average consensus, which can be employed to solve problems as diverse as distributed sensing and filtering [1], formation control [4], rendezvous [9] and coverage control [10]. This problem can be represented with the weight choice: $c_i = 1/n$.
- Weighed average consensus, which can be employed for data fusion when information about the confidence of several measurements is available. This problem can be represented with the weight choice: $c_i = 1/\Big(\sigma_i \cdot \sum_j (1/\sigma_j)\Big)$, where σ_i is the uncertainty of each measurement.
- Any *logical* operation whose outcome lies within the convex hull of the nodes' initial "opinions" for the policy to follow. If policies are mutually exclusive, this problem can be represented with the weight choice (assuming only one agent proposes the selected policy) $c_i = 1$ if i is the selected policy, $c_i = 0$ otherwise. If the problem admits a notion of *mediation* between different policies, c_i can assume problem-dependent values between 0 and 1.

It is important to note three key differences with respect to "standard" average-based models for consensus problems: (i) convex consensus provides a generalization of average-based consensus, (ii) a solution should be provided in *finite* time (as opposed to asymptotically converging algorithms), and (iii) an algorithm is not restricted to evolve according to a (possibly discontinuous) differential equation, but can also perform logic operations, establish hierarchical relationship, route messages and, in general, make complex decisions, better exploiting the possibilities offered by the on-board processing capabilities.

B. Hierarchically computable functions

Hierarchically computable functions (related to the sensitively decomposable functions as defined in [8]) are defined as follows:

Definition 3.2 (Hierarchically computable function): A hierarchically computable function obeys the following property: given the value of a function on a number of

disjoint sets of nodes, it is possible to (i) compute the function on the *union* of these sets, and (ii) store the result in a string of the same order of magnitude as the size of the string needed to represent a single argument.

Average and weighed average are examples of hierarchically computable functions: given a subset of nodes, their contribution to the consensus value can be represented by their (weighed) average and the associated weight. Majority voting on a *limited* number of options is also hierarchically computable: it is sufficient to store the number of votes obtained by each option. Other examples of hierarchically computable functions include maximum and minimum. The name is inspired by the observation that hierarchically computable functions can be computed with messages of small size on a *hierarchical* structure such as a tree.

C. Problem formulation

In this paper, we wish to present an algorithm that solves the consensus problem and achieves tradeoffs between execution time and energy consumption (as measured by byte complexity according to our discussion in Section II-C). Accordingly, the problem statement reads as follows:

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

IV. A hybrid algorithm for convex consensus

In this section we present a hybrid, semi-hierarchical algorithm that solves the Problem defined in Section III-C. The algorithm is analyzed in Section V. Our algorithm is inspired by the *Gamma* synchronizers proposed by B. Awerbuch in [11].

A. High-level description

Our algorithm operates in four nominal phases compounded by two error recovery routines.

Phase 1 starts by building a forest of *minimum weight* trees (shown in Figure 1a) of height $O(n/m)$. The value of m is the algorithm's tuning parameter. All nodes run a modified version of the GHS algorithm [7]. The GHS algorithm builds a minimum spanning tree in stages: it grows a forest of minimum weight trees by incrementally merging clusters until they span the entire network. At each stage, nodes belonging to a cluster collectively identify the cluster's minimum weight outgoing edge; the cluster then absorbs the edge and merges with the tree across it. The algorithm terminates when the tree root is unable to identify a minimum weight outgoing edge because all nodes belong to the same cluster: it then informs all descendants, which stop.

We modify the GHS's stopping criterion. At each stage, the root keeps track of the number of nodes in its cluster: when the cluster size exceeds $\lfloor n/m \rfloor$, the root stops the tree-building phase and informs its descendants. At this point, other smaller groups may try and join the cluster: they are allowed to do so immediately, at which point they inherit the cluster's identity and they are notified that the tree-building phase is complete. When a node discovers that Phase 1 is over, it contacts all its neighbors, excluding its father and children, to inquire whether they are done. When all have replied, it switches to Phase 2.

In Phase 2, tree height is upper-bounded by splitting overgrown clusters while enforcing a lower bound on tree size. This phase of the algorithm starts at the leaves of each tree. Each node recursively counts the number of its descendants moving towards the root; agents with more than $\lfloor n/m \rfloor$ offspring create a new cluster, of which they become the root, and cut the connection with their fathers. The tree containing the original root may be left with too few nodes: the root can undo one cut to guarantee that all clusters contain a minimum number of nodes.

In Phase 3, each tree establishes a "certain" number of connections with neighbor clusters, as shown in Figure 1b.

When a node switches to Phase 3, it contacts all neighbors except for its father and children, inquiring about their cluster ID. Upon reception of an inquiry, a node replies as soon as it enters Phase 3 (and is therefore sure of its cluster ID). Information is then convergecast on the tree, starting from the leaves: each node informs the father about which clusters it is connected to (either directly or through its children) and how many connections per cluster are available.

Roots also exploit the tree structure to compute their cluster's consensus function after convergecasting information from their offspring; when they have received information from all offspring, they switch to Phase 4.

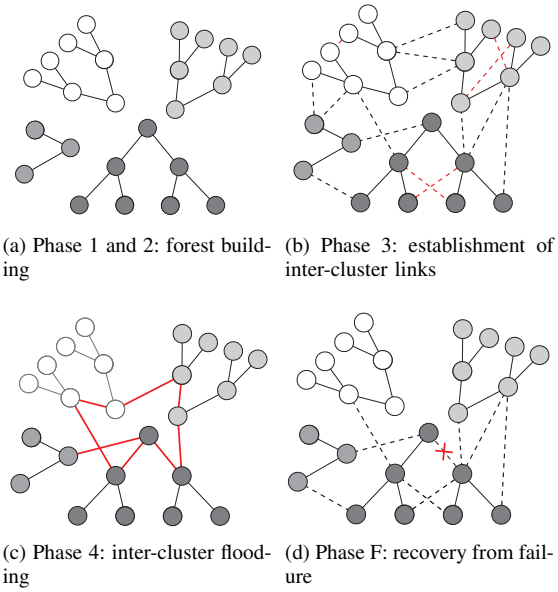
In Phase 4, cluster roots communicate with each other through the connections discovered in the previous stage. Conceptually, this phase of the algorithm is simply flooding across clusters. Each root sends a message containing its cluster's consensus function to each neighbor tree through the connections built in Phase 3. Each message is replicated a few times as a protection against link failures. When a root learns new information, it forwards it *once* to its neighbor clusters (sender excluded) via the same mechanism.

If a link failure breaks one of the trees (as in fig. 1d), the two halves evaluate their size. If either of the two halves is too small, it initiates a search for its minimum weight outgoing edge and rejoins the cluster across it; a splitting procedure guarantees that tree height stays bounded. After the failure, all nodes in the affected cluster contact their neighbors to update their routing tables.

When a link outside a tree fails, nodes on the two sides of the failure update their routing tables and notify their fathers, who do the same until the information reaches the root. Note that up to $k-1$ simultaneous, adversarial failures can occur while the algorithm updates its routing tables without disrupting cluster flooding

B. Detail description

As discussed in Section IV-A, the algorithm involves four phases, plus one phase (named Phase F), in case of intra-



(a) Phase 1 and 2: forest building

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

(c) Phase 4: inter-cluster flooding

(d) Phase F: recovery from failure

Fig. 1: Schematic representation of the algorithm behavior.

tree link failures and one phase (named Phase OF) to handle inter-tree failures. In the following, we discuss each phase in detail and show its correctness.

1) Phase 1: tree building: The proof of correctness relies on three preliminary lemmas.

Lemma 4.1 (Minimum weight tree structure): At the end of Phase 1, each cluster is a tree and contains only edges belonging to the graph's minimum spanning tree.

Proof: The claims follows from the correctness of the Awerbuch's algorithm [8], which eventually produces a minimum spanning tree, and from the observation that the algorithm never removes edges. Therefore, (i) at no point in time any of the clusters contains a cycle, and (ii) at any given time, only edges belonging to the graph's MST are present in any of the clusters. ■

Lemma 4.2 (Cluster size): At the end of Phase 1, all clusters contain at least $\lfloor n/m \rfloor$ nodes.

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

Lemma 4.3 (Participation): All nodes eventually join a cluster.

Proof: Once a node wakes up, it declares itself as root of an unary tree and executes the algorithm until either the size of the cluster is greater than or equal to $\lfloor n/m \rfloor$ or there are no outgoing edges left. Nodes wake up either spontaneously or when they receive a message from another node. At the end of Phase 1, each node contacts all of its neighbors (except its father and children, who are known to be awake). The network is connected: it follows that, as long as at one agent wakes up spontaneously, all agents are eventually contacted and therefore wake up. ■

We can now state the correctness claim for Phase 1.

Lemma 4.4 (Termination of Phase 1): All nodes are eventually informed of the end of Phase 1.

Proof: The algorithm terminates when the size of a cluster is greater than or equal to $\lfloor n/m \rfloor$. When this happens, the root of the cluster informs all of its offspring. Until then, each cluster at least doubles its size at each stage unless all nodes belong to the same cluster [8], [7]. Cluster size therefore monotonically increases until it exceeds $\lfloor n/m \rfloor$. ■

The trees obtained in Phase 1 are guaranteed to be strictly larger than $\lfloor n/m \rfloor$. Yet this is *not* sufficient: we wish to bound the number of clusters and the height of each tree. Despite this being a good heuristic, the stopping criterion does not offer worst-case guarantees: one can produce examples (in terms of network topologies and weight distributions) that give rise to a single tree spanning the whole network. This motivates the next phase of the algorithm.

2) Phase 2: tree splitting: Before executing Phase 2 of the algorithm, each node waits to be sure that all neighbors (excluding his father and children) are in Phase 2. Leaves (childless nodes) then send a message to their fathers. The algorithm proceeds recursively from here: once a node has heard from all of its children and made sure that its neighbors are in Phase 2, it can correctly compute the number of its offspring from received reports. It then sends this information to its father. If a node learns that it has more than $\lfloor n/m \rfloor$ offspring, it cuts the connection with its father after letting him know and tentatively declares itself as root (but waits before notifying its offspring). The (former) father makes a local note of this. Information about the cut which removed the least number of children (i.e., the new root UID, the father's UID, and the number of removed children) is relayed towards the root during the counting process.

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

If, on the other hand, the tree containing the original root is smaller than $\underline{t}$, the root asks its offspring to undo the cut that removed the least number of children (identified by the UIDs of father and children), then switches to Phase 3. The father of the cut to be undone asks the relevant child to do so and switches to Phase 3. The child notifies its offspring; all other nodes behave as in the previous case.

Correctness: The proof of correctness relies on two preliminary lemmas.

Lemma 4.5 (Cluster height): At the end of Phase 2, trees all have height lower than $(\lfloor n/m \rfloor + \underline{t}) + 2 = \Theta(n/m)$.

Proof: It is easy to see that the procedure outlined in the previous paragraphs correctly counts the number of descendants of each node as long as cuts are not mended. When a node's offspring exceeds $\lfloor n/m \rfloor$, the node cuts the connection with its father: all children of a node therefore have fewer than $\lfloor n/m \rfloor$ offspring. The height of a tree is upper bounded by the number of nodes in the tree: the height

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

Lemma 4.6 (Number of clusters): At the end of Phase 2, there are at most $n/\underline{t} = O(m)$ clusters.

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

We can now state the correctness claim for Phase 2.

Lemma 4.7 (Termination of Ph. 2): All nodes are eventually notified of the end of Phase 2.

Proof: In Phase 2, each node sends a message to its father, either to inform it of the number of children or to cut the connection, as soon as it has received reports from all children and has confirmed that all neighbors are in Phase 2. All nodes eventually enter Phase 2 by Lemma 4.4: the algorithm behaves like a convergecast [6, par. 15.3].

Then each node, starting from the root, contacts its children to (i) announce the Cluster ID, (ii) confirm that a cut survives or (iii) ask to undo it. All three messages cause the child to switch to Phase 3. For termination purposes, the algorithm is a broadcast and terminates by [6, par 15.3]. ■

3) Phase 3: inter-cluster links: Each node waits until it has heard from all children (if any) and all neighbors before informing its father. Nodes maintain two local routing tables: one (the *neighbor* routing table) relates non-tree neighbors and their cluster, whereas the other (the *children* routing table) records which clusters each child is connected to (directly or indirectly) and how many connections are available per cluster. When informing its father, a node makes no distinction between direct and children-mediated connections.

Correctness: The proof of correctness is provided in the following lemma.

Lemma 4.8 (Termination of Phase 3): Each node is eventually informed of the correct number of neighbor clusters connected either to it or to its offspring.

Proof: Consider an execution α of the algorithm where (i) no node contacts its neighbors until all nodes are in Phase 3, and (ii) the convergecast of routing information does not start until *all* nodes have learned their direct neighbors' ClusterIDs. It is easy to see that any execution of Phase 3 of the algorithm is *similar* to execution α: two time units after the last node enters Phase 3, the *neighbor* routing table of each node is identical to the corresponding table in α for any execution; moreover, $r+2$ time units after the last node enters Phase 3, the *children* routing table of any node closer than r to the farthest leave among their offspring is identical to the corresponding table in α for any execution. The correctness of execution α is easy to verify: discovery of neighbors' Cluster IDs is trivial if all nodes are in Phase 3 and correctness of children routing tables follows from the correctness of the convergecast algorithm [6, par. 15.3]. ■

4) Phase 4: inter-cluster flooding: The root of each tree generates a message for each of its neighbor clusters with the value of its cluster's consensus function. Each message is replicated k times, where k is a user-defined parameter. The root then sends as many copies of each message as possible through its direct connections, stored in its *neighbor* routing table. Unsent copies of the message are distributed to children proportionally to the number of links available, stored in the *children* routing table. Children do the same: when required to forward a message to a cluster, they send as many copies as possible through their direct connections and divide the rest among their own children according to the number of connections available. When a node receives a message for its cluster, it checks whether it has already received this information, either from a non-cluster neighbor or from a child. If this is not the case, it forwards the message up the tree; otherwise, it discards it. The first time the root hears new information, it broadcasts it to neighbor clusters via the same mechanism as above, forwarding k copies of a new message with the new information and the origin cluster's ID. Roots include the number of their children in their cluster's information. When a root has heard from $n - \underline{t} + 1$ nodes, it terminates: after forwarding new information one last time, it computes the consensus value and informs its offspring.

Correctness: The proof of correctness relies on a preliminary lemma.

Lemma 4.9 (Diffusion of information): In absence of failures, all clusters eventually hear from each other.

Proof: Let us abstract Phase 4 by building a network G_c containing one node for each of the existing clusters in G. Nodes in G_c are connected if the corresponding clusters are "neighbors", as discovered in Phase 3. Now, thanks to the correctness of the routing tables (Lemma 4.8) Phase 4 of the algorithm reduces to flooding on G_c, whose correctness follows from [6, par. 4.1]. Certain messages may not be forwarded if (i) a node has already forwarded the same information to its root in the past, or (ii) a node has previously seen the information as a part of a message from its root to a neighbor. In both cases, the cluster root already holds the discarded information. ■

We can now state the correctness claim for Phase 4.

Lemma 4.10 (Termination of Phase 4): Phase 4 of the algorithm eventually terminates.

Proof: By Lemma 4.9, the roots of all cluster eventually hear from each other. By Lemma 4.6, no tree is smaller than $\underline{t}$. Messages from one cluster to another carry the number of nodes in the cluster at the time of dispatch: It follows that, once a tree has heard from $n - \underline{t} + 1$ nodes, it must have heard from all clusters. ■

We are now ready to prove correctness of our algorithm.

Theorem 4.11 (Correctness of the hybrid algorithm): Every node correctly computes the consensus function.

Proof: By Lemma 4.7, the network is partitioned in rooted trees: it is easy to see that a convergecast allows the root to correctly compute its *cluster*'s consensus function. Lemmas 4.9 and 4.10 show that every root is eventually

informed of all clusters' consensus function. It follows that, at the end of Phase 4, every root is able to correctly compute the consensus function on the initial values of all nodes. Every nonroot node is then informed of the result with a broadcast. ■

5) Phase F (recovery from in-tree failure): Upon being notified of a severed connection with a child, a node notifies its root. Conversely, a node losing a connection with its father declares itself a root. If either root has fewer than $\underline{t}$ offspring, it sends them a *unique* cut identifier and initiates a search for the cluster's minimum weight outgoing edge. If the number of offspring is high enough, the root just sends to the offspring the cut identifier, which includes the old cluster ID and the IDs of the two nodes immediately upstream and downstream of the cut.

The presence of a failure complicates the search for a minimum weight outgoing edge: nodes downstream of the cut, which receive a new Cluster ID, may mistakenly accept a connection with a node in the same cluster if the latter does not know about the cut yet. To avoid this, nodes disclose the unique cut identifier when looking for the minimum weight outgoing edge: if a node sports the old Cluster ID but does not hold the cut identifier, it delays the reply until it is informed of the cut. Once a small cluster finds its minimum weight outgoing edge, it rejoins the cluster on the other side. A splitting procedure, akin to the one outlined in Phase 2, is then initiated to maintain tree height bounded. The procedure is initiated by the node rejoining the cluster and proceeds up to the root: nodes outside this path see no change in the number of their offspring.

As nodes learn their final Cluster ID, they inform all non-cluster neighbors. Neighbors, in turn, update their routing tables and inform their fathers, as in Phase 3. When an unaffected node is contacted by a non-cluster neighbor, it does not immediately notify its father to avoid a multitude of expensive incremental updates: the node waits to hear from *all* offspring that were connected to the affected cluster (recorded in the children routing table) before updating its father. This way, routing tables are updated in a convergecast. When a root learns about a variation in the topology of neighbor clusters (either because a new cluster is formed or because the number of connections to an existing cluster decreases) it crafts a message with all information it holds and sends it to the modfied clusters as in Phase 4. The roots of clusters born or modified after the cut collect information from their children and craft messages for all their neighbors, too.

Correctness: The proof of correctness relies on three preliminary lemmas.

Lemma 4.12 (Cluster height): At the end of Phase F, trees all have height lower than $(\lfloor n/m \rfloor + \underline{t} + 2) = \Theta(n/m)$.

Proof: The proof is identical to that of Lemma 4.5 and follows from the correctness of the splitting procedure. ■

Lemma 4.13 (Number of clusters): At the end of Phase F, there are at most $n/\underline{t} = O(m)$ clusters.

Proof: The proof is identical to that of Lemma 4.6 and follows from the lower bound imposed by the splitting procedure on the size of each tree. ■

Lemma 4.14: All nodes in the tree affected by the cut eventually learn about the cut.

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

We can now state the correctness claim for Phase F.

Lemma 4.15 (Inter-cluster connections): Each node is eventually informed of the correct number of neighbor clusters connected to it either directly or through its offspring.

Proof: Before a failure occurs, routing tables are correct by Lemma 4.8. When a failure occurs, nodes in the affected cluster are informed by Lemma 4.14. Once informed, nodes in the affected cluster contact all their neighbors at once. These, in turn, update their routing tables with a convergecast. Nodes formerly belonging to the broken cluster rebuild their routing tables ex novo: the correctness of the procedure follows from Lemma 4.8. ■

6) Phase OF (recovery from out-of-tree failure): The proof of correctness relies on a preliminary lemma.

Lemma 4.16 (Termination of Phase OF): At the end of Phase OF, all nodes' routing tables are correct.

Proof: The proof is identical to that of Lemma 4.8. ■

We can now state the correctness claim for Phase OF.

Lemma 4.17 (Resilience to inter-cluster failures): Phase 4 of the algorithm correctly terminates even in presence of $k-1$ simultaneous adversarial failures.

Proof: Phase 4's routing strategy ensures that, if two clusters are connected by at least k edges, any message among the two clusters will be sent across k *distinct* edges. Up to $k-1$ failures of inter-cluster links can therefore occur without invalidating the similarity between Phase 4 and flooding outlined in Lemma 4.9. ■

V. Complexity analysis

The overall time and byte complexity of the proposed algorithm are reported in Table I. The complexity of byte-optimal GHS and time-optimal flooding are reported in Table II. We first give proofs of these results, and then we present a discussion.

TABLE I: Time, message and byte complexity of the proposed hybrid algorithm

	Time	Byte
Phase 1	$O(n \log(n/m))$	$O(\lvert E\rvert \log n)$
Phase 2	$O(2n)$	$O(2n \log n)$
Phase 3	$O(n/m)$	$O(2\lvert E\rvert \log n + nm \log n)$
Phase 4	$O(\text{Diam}(G_c)n/m)$	$O(m(n + k\lvert E_c\rvert) \log n)$
Phase F	$O(n/m)$	$O(\lvert E\rvert \log n + nm \log n)$
Phase OF	$O(n/m)$	$O(2n/m \log n)$

TABLE II: Time, message and byte complexity of flooding and GHS

	Time	Byte
Flooding	$O(\text{Diam}(G))$	$O(\lvert E\rvert nb)$
GHS	$O(n \log n)$	$O[(n \log n + \lvert E\rvert) \log n]$

A. Complexity analysis

Complexity of Phase 1:

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

Byte complexity: At each stage, communication within the clusters require $O(n)$ messages. Furthermore, $2n$ test-accept messages are sent at each stage: each node accepts exactly one connection. Each edge is also rejected at most once during the algorithm. The overall number of messages exchanged is therefore $O(n \log n/m + |E|)$. Messages carry one cluster ID (which can be represented with $\log n$ bytes) at most; hence, their size is upper bounded by $\log n$.

Complexity of Phase 2:

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

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

Messages contain numbers of offspring and/or cluster IDs. The message size in Phase 2 is therefore upper bounded by $O(\log n)$. The resulting byte complexity is $O(2n \log n)$.

Complexity of Phase 3:

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

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

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

Complexity of Phase 4:

Time complexity: Let us abstract Phase 4 by building an artificial network G_c composed of $O(m)$ nodes, each corresponding to one of the existing clusters in G and labeled accordingly. Nodes in G_c are connected if at least one edge exists among the corresponding clusters in G. Let us also define a *stage* time complexity as the time required for information to travel from the root of one cluster to the root of its neighbor. Phase 4 is simply flooding on G_c: the algorithm is therefore guaranteed to terminate in $\text{Diam}(G_c)$ stages. Note that $\text{Diam}(G_c) = O(\text{Diam}(G))$ and $\text{Diam}(G_c) = O(m)$. The time complexity of one stage is upper bounded by $2(\lfloor n/m \rfloor + \underline{t}) + 1$: in each stage, information travels away from the root across the cluster, then hops from a cluster to the next one and is finally convergecast to the root. The overall time complexity of Phase 4 is therefore $\text{Diam}(G_c)(2(\lfloor n/m \rfloor + \underline{t}) + 1) = O(\text{Diam}(G)(n/m))$.

Byte complexity: In absence of failures, each of the $O(n)$ edges belonging to a tree is crossed by information about one cluster at most twice: once when the cluster learns about the information, once when information is relayed to neighbors. The overall byte complexity of *intra-cluster* messages in Phase 4 is therefore upper bounded by $(2mn \log n)$: information about each of the m clusters is stored in $\log n$ bits. Each of the $k|E_c|$ inter-cluster connections is also crossed by information about each cluster once: clusters send new information once after they receive it. The associated byte complexity is $(k|E_c|m \log n)$. The overall byte complexity is therefore $O(m(n + k|E_c|) \log n)$.

Complexity of Phase F:

Time complexity: All nodes within a tree are informed of a link failure within $2(\lfloor n/m \rfloor + \underline{t})$ time units of the failure. The node downstream of the failure broadcasts the information to its offspring directly, whereas the upstream node informs the root which, in turn, broadcasts information to other nodes. If a tree is found to be too small, a search for the minimum weight outgoing edge is initiated. Any node can be rejected by $O(\underline{t})$ other nodes in the same group at most; furthermore, the first reply may be delayed by as much as $(\lfloor n/m \rfloor + \underline{t} - 1)$ time units as nodes are informed of the cut. The time complexity of splitting is upper bounded by twice the height of the tree, as in Phase 2. In Phase F, the maximum height of a tree is $(\lfloor n/m \rfloor + 2\underline{t})$: before the failure, no tree can be taller than $(\lfloor n/m \rfloor + \underline{t})$ and only trees smaller than $\underline{t}$ perform a minimum weight outer edge (MWOE) search. Once the tree has been reformed, it updates its neighbors about its cluster ID and rebuilds the internal routing tables. Neighbors update their own routing tables, too. As in Phase 3, the time complexity is upper bounded by $O(n/m)$.

Byte complexity: The number of messages required to inform all nodes in a broken cluster of a failure is $O(n)$: one message is charged to each node in the cluster, and cluster size (as opposed to cluster height) has a trivial upper bound. The corresponding byte complexity is $O(n \log n)$: messages carry a unique Cut ID containing two node IDs and one cluster ID. If a MWOE search is initiated, each of the $O(\underline{t})$ nodes is rejected by at most $\underline{t} - 2$ siblings and accepted by one neighbor: $O(\underline{t}^2)$ messages are exchanged. The subsequent convergecast requires $O(\underline{t})$ messages. The splitting procedure requires up to $2n$ messages, i.e., twice the size of a cluster. Finally, exploring connections with neighbor clusters can require up to $2|E|$ messages (which dominates the message complexity of Phase F) and updating routing tables requires up to n messages with a convergecast.

Messages informing nodes of a failure and exploring neighbor clusters carry a cluster ID and a unique cut identi-

fier. Nodes unaffected by the failure must update their routing tables by adding or removing information about *three* clusters at most: the original cluster may disappear and its two halves may join two existing trees. The size of all these messages is therefore $O(\log n)$.

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

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

Complexity of Phase OF:

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

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

B. Discussion

The theoretical analysis shows that (i) the worst-case time and byte performance of our algorithm is intermediate with respect to GHS and flooding and (ii) the algorithm has the same byte complexity as GHS for $m = 1$ and the same time complexity as flooding for $m = n$. The algorithm therefore solves the parametrized convex consensus problem.

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

- Phase 1, which only needs to be executed *once*, sports time complexity lower than GHS by $O(n \log m)$.
- The time complexity of Phase 4 is worse than flooding's by a factor of (n/m). It is also upper bounded by $O(n)$, since $\text{Diam}(G_c) = O(m)$.

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

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

Robustness: Our algorithm also exhibits robustness intermediate between GHS and flooding.

- Recovery from an intra-tree failure can be achieved in $O(n/m)$ time steps. The same failure recovery protocol requires $\Omega(h)$ time units on the spanning tree that GHS builds: all nodes must be informed before new edges are added to ensure that no cycles are created. Flooding does not require any reconfiguration after edge failures.
- Each of the $n-1$ edges belonging to the tree built by GHS is a single point of failure (SPF); in our hybrid algorithm, edge trees (and therefore SPFs) are $n - m$.

Message complexity: We remark that our algorithm's message complexity can be higher than flooding's. Our algorithm is therefore unsuitable whenever message complexity, as opposed to byte complexity, is a good proxy for energy cost. Design of a tunable algorithm achieving time-optimal and *message*-optimal behavior will be the object a future paper.

VI. Numerical simulations on random geometric graphs

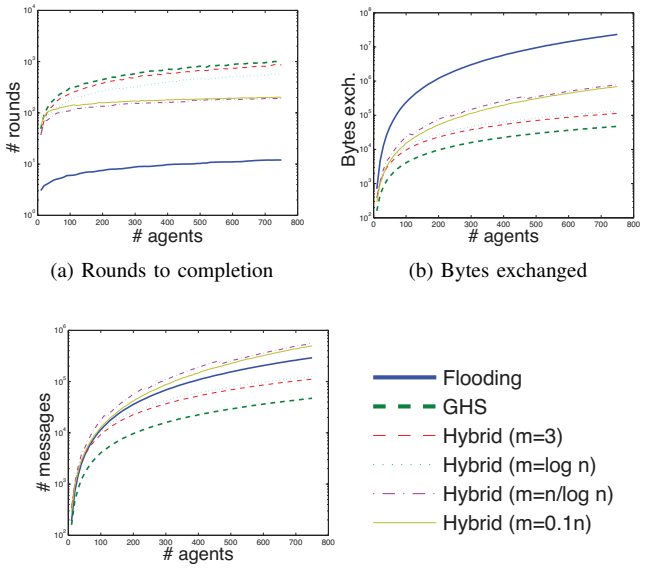
Performance of our algorithm on random geometric graphs in a *synchronous* setting was numerically evaluated and compared to GHS and flooding. Results are shown in figure 2. Simulations confirm our theoretical results: our algorithm achieves time and byte complexity intermediate between the time-optimal and the byte-optimal algorithm; performance varies smoothly as the tuning parameter is modified. Figure 3 shows the Pareto front formed by the executions of our algorithm for different tuning parameters.

Our hybrid algorithm, GHS and flooding were executed on random geometric graphs counting 10 to 750 nodes, in increments of 10. For each number of nodes, 100 executions on randomly generated networks were considered. The hybrid algorithm, GHS and flooding were executed on the same networks to ensure consistency of results. The hybrid algorithm was executed with four different values for m, decreasing from $m = n/10$ to $m = 3$ through $m = n/\log n$ and $m = \log n$.

VII. Conclusion

Our hybrid algorithm distributedly builds a semi-hierarchical structure to obtain intermediate performance between time optimality and byte optimality. A tuning parameter allows to achieve tradeoffs between execution time, energy consumption and robustness; time-optimal or byte-optimal behavior can also be recovered.

The algorithm allows to meet *hybrid* performance metrics on robotic networks of stationary or slow-moving agents. The consensus functions under consideration go beyond simple averaging and include voting and mediation.



(a) Rounds to completion

(b) Bytes exchanged

(c) Messages exchanged

Fig. 2: Time, byte and message complexity of our hybrid algorithm, GHS and flooding.

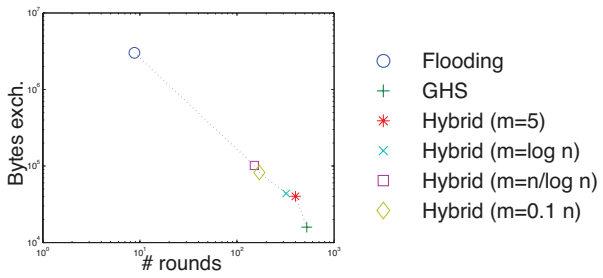


Fig. 3: Pareto front formed by executions of our algorithm for different values of m, GHS and flooding.

Future research will focus on extension of our results to fast-moving networks with frequent edge insertions and deletions, and on algorithms that meet hybrid performance parameters on *broadcast* robotic networks.

REFERENCES

[1] R. Olfati-Saber, "Distributed kalman filtering for sensor networks," in *Decision and Control, 2007 46th IEEE Conference on*, dec. 2007, pp. 5492 –5498.

[2] W. Ren, R. Beard, and E. Atkins, "Information consensus in multivehicle cooperative control," *Control Systems, IEEE*, vol. 27, no. 2, pp. 71 –82, april 2007.

[3] M. de Weerdt and B. Clement, "Introduction to planning in multiagent systems," *Multiagent and Grid Systems*, vol. 5, no. 4, pp. 345–355, 2009.

[4] A. Jadbabaie, J. Lin, and A. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *Automatic Control, IEEE Transactions on*, vol. 48, no. 6, pp. 988 – 1001, june 2003.

[5] A. Olshevsky, "Efficient information aggregation strategies for distributed control and signal processing," Ph.D. dissertation, MIT - Department of Electrical Engineering and Computer Science, September 2010.

[6] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[7] R. Gallager, P. Humblet, and P. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Transactions on Programming Languages and systems (TOPLAS)*, vol. 5, no. 1, pp. 66–77, 1983.

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

[9] J. Cortes, S. Martinez, and F. Bullo, "Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions," *Automatic Control, IEEE Transactions on*, vol. 51, no. 8, pp. 1289 –1298, aug. 2006.

[10] C. Gao, J. Cortés, and F. Bullo, "Notes on averaging over acyclic digraphs and discrete coverage control," *Automatica*, vol. 44, no. 8, pp. 2120 – 2127, 2008.

[11] B. Awerbuch, "Complexity of network synchronization," *Journal of the ACM (JACM)*, vol. 32, no. 4, pp. 804–823, 1985.