Optimal Computation in Leaderless and Multi-Leader Disconnected Anonymous Dynamic Networks

Giuseppe A. Di Luna[1]
DIAG, Sapienza University of Rome, Italy

Giovanni Viglietta[1]
Department of Computer Science and Engineering, University of Aizu, Japan

Abstract

We give a simple characterization of which functions can be computed deterministically by anonymous processes in dynamic networks, depending on the number of leaders in the network. In addition, we provide efficient distributed algorithms for computing all such functions assuming minimal or no knowledge about the network. Each of our algorithms comes in two versions: one that terminates with the correct output and a faster one that stabilizes on the correct output without explicit termination. Notably, these are the first deterministic algorithms whose running times scale *linearly* with both the number of processes and a parameter of the network which we call *dynamic disconnectivity* (meaning that our dynamic networks do not necessarily have to be connected at all times). We also provide matching lower bounds, showing that all our algorithms are asymptotically *optimal* for any fixed number of leaders.

While most of the existing literature on anonymous dynamic networks relies on classical mass-distribution techniques, our work makes use of a recently introduced combinatorial structure called *history tree*, also developing its theory in new directions. Among other contributions, our results make definitive progress on two popular fundamental problems for anonymous dynamic networks: leaderless *Average Consensus* (i.e., computing the mean value of input numbers distributed among the processes) and multi-leader *Counting* (i.e., determining the exact number of processes in the network). In fact, our approach unifies and improves upon several independent lines of research on anonymous networks, including Nedić et al., IEEE Trans. Automat. Contr. 2009; Olshevsky, SIAM J. Control Optim. 2017; Kowalski–Mosteiro, ICALP 2019, SPAA 2021; Di Luna–Viglietta, FOCS 2022.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Computing methodologies → Distributed algorithms

Keywords and phrases anonymous dynamic network, leaderless network, disconnected network, history tree

Digital Object Identifier 10.4230/LIPIcs.DISC.2023.18

Related Version *Full Version*: `https://arxiv.org/abs/2207.08061`

Funding Giovanni Viglietta was partially funded by JSPS KAKENHI Grant number 23K10985.

1 Introduction

Dynamic networks. An increasingly prominent area of distributed computing focuses on algorithmic aspects of *dynamic networks*, motivated by novel technologies such as wireless sensors networks, software-defined networks, networks of smart devices, and other networks with a continuously changing topology [9, 32, 34]. Typically, a network is modeled by a system of n *processes* that communicate in synchronous rounds; at each round, the network's topology changes unpredictably.

[1] Both authors contributed equally to this research.

Disconnected networks. In the dynamic setting, a common assumption is that the network is *1-interval-connected*, i.e., connected at all rounds [30, 37]. However, this is not a suitable model for many real systems, due to the very nature of dynamic entities (think of P2P networks of smart devices moving unpredictably) or due to transient communication failures, which may compromise the network's connectivity. A weaker assumption is that the union of all the network's links across any T consecutive rounds induces a connected graph on the processes [28, 39]. We say that such a network is *T-union-connected*, and we call $T \geq 1$ its *dynamic disconnectivity.*[2]

Anonymous processes. Several works have focused on processes with unique IDs, which allow for efficient algorithms for many different tasks [8, 29, 30, 31, 34, 37]. However, unique IDs may not be available due to operational limitations [37] or to protect user privacy: A famous example are COVID-19 tracking apps, where assigning temporary random IDs to users was not enough to eliminate privacy concerns [43]. Systems where processes are indistinguishable are called *anonymous*. The study of *static* anonymous networks has a long history, as well [6, 7, 10, 11, 12, 21, 42, 45].

Networks with leaders. It is known that several fundamental problems for anonymous networks (a notable example being the *Counting problem*, i.e., determining the total number of processes n) cannot be solved without additional "symmetry-breaking" assumptions. The most typical choice is the presence of a single distinguished process called *leader* [1, 2, 3, 4, 16, 21, 23, 25, 33, 41, 46] or, less commonly, a subset of several leaders (and knowledge of their number) [24, 26, 27, 28].

Apart from the theoretical importance of generalizing the usual single-leader scenario, studying networks with multiple leaders also has a practical impact in terms of privacy. Indeed, while the communications of a single leader can be traced, the addition of more leaders provides differential privacy for each of them.

Leaderless networks. In some networks, the presence of reliable leaders may not always be guaranteed: For example, in a mobile sensor network deployed by an aircraft, the leaders may be destroyed as a result of a bad landing; also, the leaders may malfunction during the system's lifetime. This justifies the extensive existing literature on networks with no leaders [14, 15, 35, 36, 38, 44, 47]. Notably, a large portion of works on leaderless networks have focused on the *Average Consensus problem*, where the goal is to compute the mean of a list of numbers distributed among the processes [5, 13, 14, 20, 39, 40].

1.1 Our Contributions

Summary. Focusing on anonymous dynamic networks, in this paper we completely elucidate the relationship between leaderless networks and networks with (multiple) leaders, as well as the impact of the dynamic disconnectivity T on the efficiency of distributed algorithms. We remark that only a minority of existing works consider networks that are not necessarily connected at all times.

The full version of this paper is found at [19].

[2] We use the term "disconnected" to refer to T-union-connected networks in the sense that they may not be connected at any round. It is worth noting that non-trivial (terminating) computation requires some conditions on temporal connectivity to be met, such as a finite dynamic disconnectivity and its knowledge by all processes (refer to Proposition 2).

Computability. We give an exact characterization of which functions can be computed in anonymous dynamic networks with and without leaders, respectively. Namely, with at least one leader, all the so-called *multi-aggregate functions* are computable; with no leaders, only the *frequency-based multi-aggregate functions* are computable (see Section 2 for definitions). Interestingly, computability is independent of the dynamic disconnectivity T. Our contribution considerably generalizes a recent result on the functions computable with exactly one leader and with $T = 1$ [17].

Complete problems. While computing the so-called *Generalized Counting function* F_{GC} was already known to be a complete problem for the class of multi-aggregate functions [17], in this work we expand the picture by identifying a complete problem for the class of frequency-based multi-aggregate functions, as well: the *Frequency function* F_R (both F_{GC} and F_R are defined in Section 2). By "complete problem" we mean that computing such a function allows the immediate computation of any other function in the class with no overhead in terms of communication rounds.

Algorithms. We give efficient deterministic algorithms for computing the Frequency function (Section 3) and the Generalized Counting function (Section 4). Since the two problems are complete, we automatically obtain efficient algorithms for computing *all* functions in the respective classes.

For each problem, we give two algorithms: a *terminating* version, where each process is required to commit on its output and never change it, and a *stabilizing* version, where processes are allowed to modify their outputs, provided that they eventually stabilize on the correct output.

The stabilizing algorithms for both problems run in $2Tn$ rounds regardless of the number of leaders, and do not require any knowledge of the dynamic disconnectivity T or the number of processes n. Our terminating algorithm for leaderless networks runs in $T(n + N)$ rounds with knowledge of T and an upper bound $N \geq n$; the terminating algorithm for $\ell \geq 1$ leaders runs in $(\ell^2 + \ell + 1)Tn$ rounds with no knowledge about n. The latter running time is reasonable (i.e., linear) in most applications, as ℓ is typically a constant or very small compared to n.

A comparison of our results with the state of the art on Average Consensus and Counting problems is illustrated in Table 1 and discussed in Appendix A.

Negative results. Some of our algorithms assume processes to have a-priori knowledge of some parameters of the network; in Section 5 we show that all of these assumptions are necessary. We also provide lower bounds that asymptotically match our algorithms' running times, assuming that the number of leaders ℓ is constant (which is a realistic assumption in most applications).

Multigraphs. All of our results hold more generally if networks are modeled as multigraphs, as opposed to the simple graphs traditionally encountered in nearly all of the literature. This is relevant in many applications: in radio communication, for instance, multiple links between processes naturally appear due to the multi-path propagation of radio waves.

Table 1 Comparing results for Average Consensus and Counting in anonymous dynamic networks. For algorithms that support disconnected networks, T indicates the dynamic disconnectivity.

Problem	*Reference*	*Leaders*	*Disconn.*	*Term.*	*Notes*	*Running time*
Average Consensus	[35]	$\ell = 0$	✓		ϵ-convergence, T unknown, upper bound on processes' degrees known	$O(Tn^3 \log(1/\epsilon))$
	[14]	$\ell = 0$			ϵ-convergence	$O(n^4 \log(n/\epsilon))$
	[13]	$\ell = 0$			randomized Monte Carlo	$O(n)$
	[26]	$\ell \geq 1$		✓	ℓ known	$O(n^5 \log^3(n)/\ell)$
	this work	$\ell = 0$	✓		T unknown	$2Tn$
		$\ell = 0$	✓	✓	T and $N \geq n$ known	$T(n + N)$
(Generalized) Counting	[17]	$\ell = 1$				$2n - 2$
		$\ell = 1$		✓		$3n - 2$
	[28]	$\ell \geq 1$		✓	ℓ known	$O(n^4 \log^3(n)/\ell)$
	[27]	$\ell \geq 1$	✓	✓	ℓ and T known, $O(\log n)$-size messages	$\widetilde{O}(n^{2T+3}/\ell)$
	this work	$\ell \geq 1$	✓		ℓ known, T unknown	$2Tn$
		$\ell \geq 1$	✓	✓	ℓ and T known	$(\ell^2 + \ell + 1)Tn$

2 Definitions and Preliminaries

We will give preliminary definitions and results, and recall some properties of history trees from [17].

Processes and networks. A *dynamic network* is modeled by an infinite sequence $\mathcal{G} = (G_t)_{t \geq 1}$, where $G_t = (V, E_t)$ is an undirected multigraph whose vertex set $V = \{p_1, p_2, \ldots, p_n\}$ is a system of n *anonymous processes* and E_t is a multiset of edges representing *links* between processes.

Each process p_i starts with an *input* $\lambda(p_i)$, which is assigned to it at *round* 0. It also has an internal state, which is initially determined by $\lambda(p_i)$. At each *round* $t \geq 1$, every process composes a message (depending on its internal state) and broadcasts it to its neighbors in G_t through all its incident links. By the end of round t, each process reads all messages coming from its neighbors and updates its internal state according to a local algorithm $\mathcal{A}$. Note that $\mathcal{A}$ is deterministic and is the same for all processes. The input of each process also includes a *leader flag*. The processes whose leader flag is set are called *leaders* (or *supervisors*). We will denote the number of leaders as ℓ.

Each process also returns an *output* at the end of each round, which is determined by its current internal state. A system is said to *stabilize* if the outputs of all its processes remain constant from a certain round onward; note that a process' internal state may still change even when its output is constant. A process may also decide to explicitly *terminate* and no longer update its internal state. When all processes have terminated, the system is said to *terminate*, as well.

We say that $\mathcal{A}$ *computes* a function F if, whenever the processes are assigned inputs $\lambda(p_1)$, $\lambda(p_2)$, $\ldots$, $\lambda(p_n)$ and all processes execute the local algorithm $\mathcal{A}$ at every round, the system eventually stabilizes with each process p_i giving the desired output $F(p_i, \lambda)$. A stronger notion of computation requires the system to not only stabilize but also to explicitly terminate with the correct output. The (worst-case) *running time* of $\mathcal{A}$, as a function of n, is the maximum number of rounds it takes for the system to stabilize (and optionally terminate), taken across all possible dynamic networks of size n and all possible input assignments.

Classes of functions. Let $\mu_\lambda = \{(z_1, m_1), (z_2, m_2), \dots, (z_k, m_k)\}$ be the multiset of all processes' inputs. That is, for all $1 \le i \le k$, there are exactly m_i processes $p_{j_1}, p_{j_2}, \dots, p_{j_{m_i}}$ whose input is $z_i = \lambda(p_{j_1}) = \lambda(p_{j_2}) = \cdots = \lambda(p_{j_{m_i}})$; note that $n = \sum_{i=1}^{k} m_i$. A *multi-aggregate* function is defined as a function F of the form $F(p_i, \lambda) = \psi(\lambda(p_i), \mu_\lambda)$, i.e., such that the output of each process depends only on its own input and the multiset of all processes' inputs.

The special multi-aggregate functions $F_C(p_i, \lambda) = n$ and $F_{GC}(p_i, \lambda) = \mu_\lambda$ are called the *Counting* function and the *Generalized Counting* function, respectively. It is known that, if a system can compute the Generalized Counting function F_{GC}, then it can compute any multi-aggregate function in the same number of rounds: thus, F_{GC} is *complete* for the class of multi-aggregate functions [17].

For any $\alpha \in \mathbb{R}^+$, we define $\alpha \cdot \mu_\lambda$ as $\{(z_1, \alpha \cdot m_1), (z_2, \alpha \cdot m_2), \dots, (z_k, \alpha \cdot m_k)\}$. We say that a multi-aggregate function $F(p_i, \lambda) = \psi(\lambda(p_i), \mu_\lambda)$ is *frequency-based* if $\psi(z, \mu_\lambda) = \psi(z, \alpha \cdot \mu_\lambda)$ for every positive integer α and every input z (see [22]). That is, F depends only on the "frequency" of each input in the system, rather than on their actual multiplicities. Notable examples include statistical functions such as mean, variance, maximum, median, mode, etc. The problem of computing the mean of all input values is called *Average Consensus* [5, 13, 14, 15, 20, 26, 35, 36, 38, 39, 40, 44, 47].

The frequency-based multi-aggregate function $F_R(p_i, \lambda) = \frac{1}{n} \cdot \mu_\lambda$ is called *Frequency* function, and is complete for the class of frequency-based multi-aggregate functions, as stated below (the proof is simple, and is found in [19]).

▶ **Proposition 1.** *If F_R can be computed (with termination), then all frequency-based multi-aggregate functions can be computed (with termination) in the same number of rounds, as well.* ◀

History trees. *History trees* were introduced in [17] as a tool of investigation for anonymous dynamic networks; an example is found in Figure 1. A history tree is a representation of a dynamic network given some inputs to the processes. It is an infinite graph whose nodes are partitioned into *levels* L_t, with $t \ge -1$; each node in L_t represents a class of processes that are *indistinguishable* at the end of round t (with the exception of L_{-1}, which contains a single node r representing all processes). The definition of distinguishability is inductive: at the end of round 0, two processes are distinguishable if and only if they have different inputs. At the end of round $t \ge 1$, two processes are distinguishable if and only if they were already distinguishable at round $t-1$ or if they have received different multisets of messages at round t.

Each node in level L_0 has a label indicating the input of the processes it represents. There are also two types of edges connecting nodes in adjacent levels. The *black edges* induce an infinite tree rooted at node $r \in L_{-1}$ which spans all nodes. The presence of a black edge $\{v, v'\}$, with $v \in L_t$ and $v' \in L_{t+1}$, indicates that the *child node* v' represents a subset of the processes represented by the *parent node* v. The *red multi-edges* represent communications between processes. The presence of a red edge $\{v, v'\}$ with multiplicity m, with $v \in L_t$ and $v' \in L_{t+1}$, indicates that, at round $t+1$, each process represented by v' receives m (identical) messages from processes represented by v.

As time progresses and processes exchange messages, they are able to locally construct finite portions of the history tree. In [17], it is shown that there is a local algorithm $\mathcal{A}^*$ that allows each process to locally construct and update its own *view* of the history tree at every round. The view of a process p at round $t \ge 0$ is the subgraph of the history tree which is spanned by all the shortest paths (using black and red edges indifferently) from the root r to the node in L_t representing p (see Figure 1). As proved in [17, Theorem 3.1], the view of a process at round t contains all the information that the process may be able

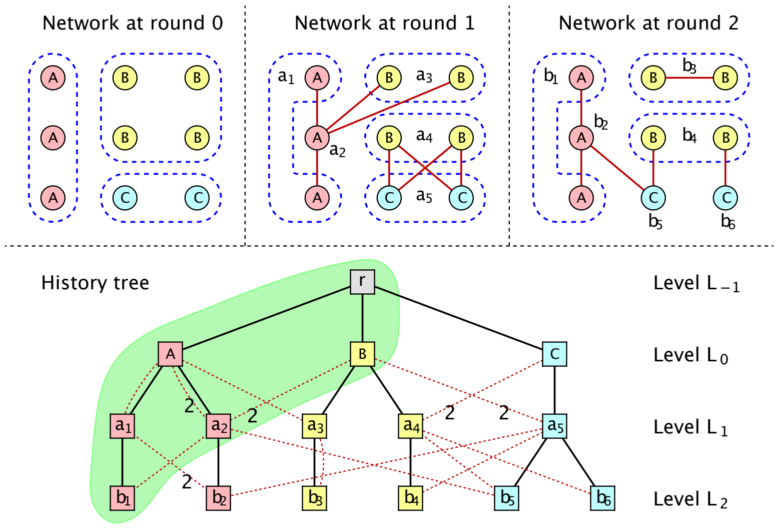

Figure 1 The first rounds of a dynamic network with $n = 9$ processes and the corresponding levels of the history tree. Level L_t consists of all nodes at distance $t + 1$ from the root r. The multiplicities of the red multi-edges of the history tree are explicitly indicated only when greater than 1. The letters A, B, C denote processes' inputs; all other labels have been added for the reader's convenience, and indicate classes of indistinguishable processes (non-trivial classes are also indicated by dashed blue lines). Note that the two processes in b_4 are still indistinguishable at the end of round 2, although they are linked to the distinguishable processes b_5 and b_6. This is because such processes were in the same class a_5 at round 1. The subgraph in the green blob is the *view* of the two processes in b_1.

to use at that round. This justifies the convention that all processes always execute $\mathcal{A}^*$, constructing their local view of the history tree and broadcasting (a representation of) it at every round, regardless of their task. Then, they simply compute their task-dependent outputs as a function of their respective views.

We define the *anonymity* of a node v of the history tree as the number of processes that v represents, and we denote it as $a(v)$. It follows that $\sum_{v \in L_t} a(v) = n$ for all $t \geq -1$, and that the anonymity of a node is equal to the sum of the anonymities of its children. Naturally, a process is not aware of the anonymities of the nodes in its view of the history tree, unless it can somehow infer them from the view's structure itself. In fact, computing the Generalized Counting function is equivalent to determining the anonymities of all the nodes in L_0. Similarly, computing the Frequency function corresponds to determining the value $a(v)/n$ for all $v \in L_0$.

Computation in disconnected networks. Although the network G_t at each individual round may be disconnected, we assume the dynamic network to be T*-union-connected*. That is, there is a *dynamic disconnectivity* parameter $T \geq 1$ such that the sum of any T consecutive G_t's is a connected multigraph. Thus, for all $i \geq 1$, the multigraph $\left(V, \bigcup_{t=i}^{i+T-1} E_t\right)$ is connected (we remark that a union of multisets adds together the multiplicities of equal elements).[3] The next two results are easy to prove (see [19]).

[3] Our T-union-connected networks should not be confused with the T-*interval*-connected networks from [30]. In those networks, the *intersection* (as opposed to the union) of any T consecutive E_t's induces a connected (multi)graph. In particular, a T-interval-connected network is connected at every round, while a T-union-connected network may not be, unless $T = 1$. Incidentally, a network is 1-interval-connected if and only if it is 1-union-connected.

▶ **Proposition 2.** *Any non-trivial function is impossible to compute with termination unless the processes have some knowledge about T. (A function is "trivial" if it can be computed locally.)* ⌟

▶ **Proposition 3.** *A function F can be computed (with termination) within $f(n)$ rounds in any dynamic network with $T = 1$ if and only if F can be computed (with termination) within $T \cdot f(n)$ rounds in any dynamic network with $T \geq 1$, assuming that T is known to all processes.* ⌟

Relationship with the dynamic diameter. A concept closely related to the dynamic disconnectivity T of a network is its *dynamic diameter* (or *temporal diameter*) D, which is defined as the maximum number of rounds it may take for information to travel from any process to any other process at any point in time [9, 32]. It is a simple observation that $T \leq D \leq T(n-1)$.

We chose to use T, as opposed to D, to measure the running times of our algorithms for several reasons. Firstly, T is well defined (i.e., finite) if and only if D is; however, T has a simpler definition, and is arguably easier to directly estimate or enforce in a real network. Secondly, Proposition 3, as well as all of our theorems, remain valid if we replace T with D; nonetheless, stating the running times of our algorithms in terms of T is better, because $T \leq D$.

3 Computation in Leaderless Networks

We will give a stabilizing and a terminating algorithm that efficiently compute the Frequency function F_R in all leaderless networks with finite dynamic disconnectivity T. As a consequence, *all* frequency-based multi-aggregate functions are efficiently computable as well, due to Proposition 1. Moreover, Proposition 16 states that no other functions are computable in leaderless networks, and Proposition 17 shows that our algorithms are asymptotically optimal. All missing proofs are found in [19].

3.1 Stabilizing Algorithm

We will use the procedure in Listing 1 as a subroutine in some of our algorithms. Its purpose is to construct a homogeneous system of $k-1$ independent linear equations involving the anonymities of all the k nodes in a level of a process' view. We will first give some definitions.

In (a view of) a history tree, if a node $v \in L_t$ has exactly one child (i.e., there is exactly one node $v' \in L_{t+1}$ such that $\{v, v'\}$ is a black edge), we say that v is *non-branching*. We say that two non-branching nodes $v_1, v_2 \in L_t$, whose respective children are $v'_1, v'_2 \in L_{t+1}$, are *exposed* with multiplicity (m_1, m_2) if the red edges $\{v'_1, v_2\}$ and $\{v'_2, v_1\}$ are present with multiplicities $m_1 \geq 1$ and $m_2 \geq 1$, respectively. A *strand* is a path $(w_1, w_2, \ldots, w_k)$ in (a view of) a history tree consisting of non-branching nodes such that, for all $1 \leq i < k$, the node w_i is the parent of w_{i+1}. We say that two strands P_1 and P_2 are *exposed* if there are two exposed nodes $v_1 \in P_1$ and $v_2 \in P_2$.

Intuitively, the procedure in Listing 1 searches for a long-enough sequence of levels in the given view $\mathcal{V}$, say from L_s to L_t, where all nodes are non-branching. That is, the nodes in $L_s \cup L_{s+1} \cup \cdots \cup L_t$ can be partitioned into $k = |L_s| = |L_t|$ strands. Then the procedure searches for pairs of exposed strands, each of which yields a linear equation involving the anonymities of some nodes of L_t, until it obtains $k-1$ linearly independent equations. Note

Listing 1 Constructing a system of equations in the anonymities of some nodes in a view.

```
# Input: a view $\mathcal{V}$ with levels $L_{-1}$, $L_0$, $L_1$, ..., $L_h$
# Output: $(t, S)$, where $t$ is an integer and $S$ is a system of linear equations

Assign $s := 0$
For $t := 0$ to $h$
   If $L_t$ contains a node with no children, return $(-1, \emptyset)$
   If $L_t$ contains a node with more than one child, assign $s := t + 1$
   Else
      Let $k = |L_s| = |L_t|$ and let $u_i$ be the $i$th node in $L_t$
      Let $P_i$ be the strand starting in $L_s$ and ending in $u_i \in L_t$
      Let $\mathcal{P} = \{P_1, P_2, \dots, P_k\}$
      Let $G$ be the graph on $\mathcal{P}$ whose edges are pairs of exposed strands
      If $G$ is connected
         Let $G' \subseteq G$ be any spanning tree of $G$
         Assign $S := \emptyset$
         For each edge $\{P_i, P_j\}$ of $G'$
            Find any two exposed nodes $v_1 \in P_i$ and $v_2 \in P_j$
            Let $(m_1, m_2)$ be the multiplicity of the exposed pair $(v_1, v_2)$
            Add to $S$ the equation $m_1 x_i = m_2 x_j$
         Return $(t, S)$
```

that the search may fail (in which case Listing 1 returns $t = -1$) or it may produce incorrect equations. The following lemma specifies sufficient conditions for Listing 1 to return a correct and non-trivial system of equations for some $t \geq 0$ (the proof is in [19]).

▶ **Lemma 4.** *Let $\mathcal{V}$ be the view of a process in a T-union-connected network of size n taken at round t', and let Listing 1 return (t, S) on input $\mathcal{V}$. Assume that one of the following conditions holds:*

1. *$t \geq 0$ and $t' \geq t + Tn$, or*

2. *$t' \geq 2Tn$.*

Then, $0 \leq t \leq Tn$, and S is a homogeneous system of $k - 1$ independent linear equations (with integer coefficients) in $k = |L_t|$ variables x_1, x_2, ..., x_k. Moreover, S is satisfied by assigning to x_i the anonymity of the ith node of L_t, for all $1 \leq i \leq k$. ⌟

▶ **Theorem 5.** *There is an algorithm that computes F_R in all T-union-connected anonymous networks with no leader and stabilizes in at most $2Tn$ rounds, assuming no knowledge of T or n.* ⌟

3.2 Terminating Algorithm

We will now give a certificate of correctness that can be used to turn the stabilizing algorithm of Theorem 5 into a terminating algorithm. The certificate relies on a-priori knowledge of the dynamic disconnectivity T and an upper bound N on the size of the network n; these assumptions are justified by Proposition 2 and Proposition 18, respectively.

▶ **Theorem 6.** *There is an algorithm that computes F_R in all T-union-connected anonymous networks with no leader and terminates in at most $T(n + N)$ rounds, assuming that T and an upper bound $N \geq n$ are known to all processes.*[4] ⌟

[4] If the dynamic diameter D of the network is known, the termination time improves to $Tn + D$ rounds.

4 Computation in Networks with Leaders

We will give a stabilizing and a terminating algorithm that efficiently compute the Generalized Counting function F_{GC} in all networks with $\ell \geq 1$ leaders and finite dynamic disconnectivity T. Therefore, *all* multi-aggregate functions are efficiently computable as well, due to [17, Theorem 2.1]. Moreover, Proposition 19 states that no other functions are computable in networks with leaders, and Proposition 21 shows that our algorithms are asymptotically optimal for any fixed $\ell \geq 1$.

4.1 Stabilizing Algorithm

We will once again make use of the subroutine in Listing 1, this time assuming that the number of leaders $\ell \geq 1$ is known to all processes. This assumption is justified by Proposition 20. The next theorem uses the same ideas as Theorems 5 and 6, and is proved in [19].

▶ **Theorem 7.** *There is an algorithm that computes F_{GC} in all T-union-connected anonymous networks with $\ell \geq 1$ leaders and stabilizes in at most $2Tn$ rounds, assuming that ℓ is known to all processes, but assuming no knowledge of T or n.* ◀

4.2 Terminating Algorithm

We will now present the main result of this paper. As already remarked, giving an efficient certificate of correctness for the (Generalized) Counting problem with multiple leaders is a highly non-trivial task for which a radically different approach is required. Note that it is not possible to simply adapt the single-leader algorithm in [17] by setting the anonymity of the leader node in the history tree to ℓ instead of 1. Indeed, as soon as some leaders get disambiguated, the leader node splits into several children nodes whose anonymities are unknown (we only know that their sum is ℓ). There is no way around this difficulty other than developing a new technique.

Our algorithm is rather involved, and the proofs of several technical lemmas are provided in [19], due to lack of space.

The subroutine `ApproxCount`. A large portion of this section is devoted to a subroutine called `ApproxCount`, which will be repeatedly invoked by our main algorithm. The purpose of `ApproxCount` is to compute an approximation n' of the total number of processes n (or report various types of failure). It takes as input a view $\mathcal{V}$ of a process, the number of leaders ℓ, and two integer parameters s and x, representing the index of a level of $\mathcal{V}$ and the anonymity of a leader node in L_s, respectively.

The subroutine roughly follows the general structure of the algorithm in [17, Section 4.2]: namely, anonymities are first "guessed" and then proven correct when some "certificates" are satisfied. However, the way these basic concepts are defined and the way the underlying principles are implemented is entirely new, due to the added difficulty that here we have a strand of leader nodes in the view $\mathcal{V}$ hanging from the first leader node τ in level L_s, where the anonymity $a(\tau)$ is an unknown number not greater than ℓ (as opposed to $a(\tau) = 1$, which is assumed in [17]).

`ApproxCount` begins by assuming that $a(\tau)$ is the given parameter x, and then it makes deductions on the anonymities of other nodes until it is able to make an estimate $n' > 0$ on the total number of processes, or report failure in the form of an *error code* $n' \in \{-1, -2, -3\}$. In particular, since the algorithm requires the existence of a long-enough strand hanging from τ, it reports failure if some descendants of τ (in the relevant levels of $\mathcal{V}$) have more than one child.

Another important difficulty that is unique to the multi-leader case is that, even if $\mathcal{V}$ contains a long-enough strand of leader nodes, some nodes in the strand may still be branching in the history tree (that is, the chain of leader nodes is branching, but only one branch appears in $\mathcal{V}$). We will have to keep this in mind when reasoning about $\mathcal{V}$.

In the following, we give some preliminary definitions and results in order to formally state our subroutine and prove its correctness and running time. We remark that `ApproxCount` assumes that the network is 1-union-connected, as this is sufficient for our main algorithm to work for any T-union-connected network (refer to the proof of Theorem 15).

Discrepancy δ. Suppose that `ApproxCount` is invoked with arguments $\mathcal{V}$, s, x, ℓ, where $1 \leq x \leq \ell$, and let τ be the first leader node in level L_s of $\mathcal{V}$ (if τ does not exist, the procedure immediately returns the error code $n' = -1$). We define the *discrepancy* δ as the ratio $x/a(\tau)$. Clearly, $1/\ell \leq \delta \leq \ell$. Note that, since $a(\tau)$ is not a-priori known by the process executing `ApproxCount`, then neither is δ.

Conditional anonymity. `ApproxCount` starts by assuming that the anonymity of τ is x, and makes deductions on other anonymities based on this assumption. Thus, we will distinguish between the actual anonymity of a node $a(v)$ and the *conditional anonymity* $a'(v) = \delta a(v)$ that `ApproxCount` may compute under the initial assumption that $a'(\tau) = x = \delta a(\tau)$.

Guessing conditional anonymities. Let u be a node of a history tree, and assume that the conditional anonymities of all its children u_1, u_2, ..., u_k have been computed: such a node u is called a *guesser*. If v is not among the children of u but it is at their same level, and the red edge $\{v, u\}$ is present with multiplicity $m \geq 1$, we say that v is *guessable* by u. In this case, we can make a *guess* $g(v)$ on the conditional anonymity $a'(v)$:

$$g(v) = \frac{a'(u_1) \cdot m_1 + a'(u_2) \cdot m_2 + \cdots + a'(u_k) \cdot m_k}{m}, \tag{1}$$

where m_i is the multiplicity of the red edge $\{u_i, v'\}$ for all $1 \leq i \leq k$, and v' is the parent of v (possibly, $m_i = 0$). Note that $g(v)$ may not be an integer. Although a guess may be inaccurate, it never underestimates the conditional anonymity:

▶ **Lemma 8.** *If v is guessable, then $g(v) \geq a'(v)$. Moreover, if v has no siblings, $g(v) = a'(v)$.* ⌟

Heavy nodes. The subroutine `ApproxCount` assigns guesses in a *well-spread* fashion, i.e., in such a way that at most one node per level is assigned a guess.

Suppose now that a node v has been assigned a guess. We define its *weight* $w(v)$ as the number of nodes in the subtree hanging from v that have been assigned a guess (this includes v itself). Recall that subtrees are determined by black edges only. We say that v is *heavy* if $w(v) \geq \lfloor g(v) \rfloor$.

▶ **Lemma 9.** *Assume that $\delta \geq 1$. In a well-spread assignment of guesses, if $w(v) > a'(v)$, then some descendants of v are heavy (the* descendants *of v are the nodes in the subtree hanging from v other than v itself).* ⌟

Correct guesses. We say that a node v has a *correct* guess if v has been assigned a guess and $g(v) = a'(v)$. The next lemma gives a criterion to determine if a guess is correct.

Listing 2 The subroutine `ApproxCount` invoked in Listing 3.

```
# Input: a view $\mathcal{V}$ and three integers $s$, $x$, $\ell$
# Output: a pair of integers $(n', t)$

Let $L_{-1}$, $L_0$, $L_1$, ... be the levels of $\mathcal{V}$
Assign $t := s$
If $L_s$ does not contain any leader nodes, return $(-1, t)$
Let $\tau$ be the first leader node in $L_s$
Mark all nodes in $\mathcal{V}$ as not guessed and not counted
Assign $u := \tau$; assign $a'(u) := x$; mark $u$ as counted
While $u$ has a unique child $u'$ in $\mathcal{V}$
   Assign $u := u'$; assign $a'(u) := x$; mark $u$ as counted
While there are guessable levels and a counting cut has not been found
   Let $v$ be a guessable non-counted node of smallest depth in $\mathcal{V}$
   Let $L_{t'}$ be the level of $v$; assign $t := \max\{t, t'\}$
   Assign a guess $g(v)$ to $v$ as in Equation (1); mark $v$ as guessed
   Let $P_v$ be the black path from $v$ to its ancestor in $L_s$
   If there is a heavy node in $P_v$
      Let $v'$ be the heavy node in $P_v$ of maximum depth
      If $g(v')$ is not an integer, return $(-3, t)$
      Assign $a'(v') := g(v')$; mark $v'$ as counted and not guessed
      If $v'$ is the root or a leaf of a non-trivial complete isle $I$
         For each internal node $w$ of $I$
            Assign $a'(w) := \sum_{w' \text{ leaf of } I \text{ and descendant of } w} a'(w')$
            Mark $w$ as counted and not guessed
If no counting cut has been found, return $(-2, t)$
Else
   Let $C$ be a counting cut between $L_s$ and $L_t$
   Let $n' = \sum_{v \in C} a'(v)$
   Let $\ell' = \sum_{v \text{ leader node in } C} a'(v)$
   If $\ell' < \ell$, return $(-1, t)$
   If $\ell' > \ell$, return $(-3, t)$
   Return $(n', t)$
```

▶ **Lemma 10.** *Assume that $\delta \geq 1$. In a well-spread assignment of guesses, if a node v is heavy and no descendant of v is heavy, then v has a correct guess or the guess on v is not an integer.*

When the criterion in Lemma 10 applies to a node v, we say that v has been *counted.* So, counted nodes are nodes that have been assigned a guess, which was then confirmed to be the correct conditional anonymity.

Cuts and isles. Fix a view $\mathcal{V}$ of a history tree $\mathcal{H}$. A set of nodes C in $\mathcal{V}$ is said to be a *cut* for a node $v \notin C$ of $\mathcal{V}$ if two conditions hold: (i) for every leaf v' of $\mathcal{V}$ that lies in the subtree hanging from v, the black path from v to v' contains a node of C, and (ii) no proper subset of C satisfies condition (i). A cut for the root r whose nodes are all counted is said to be a *counting cut.*

Let s be a counted node in $\mathcal{V}$, and let F be a cut for v whose nodes are all counted. Then, the set of nodes spanned by the black paths from s to the nodes of F is called *isle*; s is the *root* of the isle, while each node in F is a *leaf* of the isle. The nodes in an isle other than the root and the leaves are called *internal.* An isle is said to be *trivial* if it has no internal nodes.

Listing 3 Solving the Counting problem with $\ell \geq 1$ leaders.

```
# Input: a view $\mathcal{V}$ and a positive integer $\ell$
# Output: either a positive integer $n$ or "Unknown"

Assign $n^* := -1$ and $s := 0$ and $c := 0$
Let $b$ be the number of leader branches in $\mathcal{V}$
While $c \leq \ell - b$
   Assign $t^* := -1$
   For $x := \ell$ downto 1
      Assign $(n', t) :=$ ApproxCount$(\mathcal{V}, s, x, \ell)$      # see Listing 2
      Assign $t^* := \max\{t^*, t\}$
      If $n' = -1$, return "Unknown"
      If $n' = -2$, break out of the for loop
      If $n' > 0$
         If $n^* = -1$, assign $n^* := n'$
         Else if $n^* \neq n'$, return "Unknown"
         Assign $c := c + 1$ and break out of the for loop
   Assign $s := t^* + 1$
Let $L_{t'}$ be the last level of $\mathcal{V}$
If $t' \geq t^* + n^*$, return $n^*$
Else return "Unknown"
```

If s is an isle's root and F is its set of leaves, we have $a(s) \geq \sum_{v \in F} a(v)$, because s may have some descendants in the history tree $\mathcal{H}$ that do not appear in the view $\mathcal{V}$. This is equivalent to $a'(s) \geq \sum_{v \in F} a'(v)$. If equality holds, then the isle is said to be *complete*; in this case, we can easily compute the conditional anonymities of all the internal nodes by adding them up starting from the nodes in F and working our way up to s.

Overview of ApproxCount. Our subroutine ApproxCount is found in Listing 2. It repeatedly assigns guesses to nodes based on known conditional anonymities, starting from τ and its descendants. Eventually some nodes become heavy, and the criterion in Lemma 10 causes the deepest of them to become counted. In turn, counted nodes eventually form isles; the internal nodes of complete isles are marked as counted, which gives rise to more guessers, and so on. In the end, if a counting cut is created, the algorithm checks whether the conditional anonymities of the leader nodes in the cut add up to ℓ.

Algorithmic details of ApproxCount. The algorithm ApproxCount uses flags to mark nodes as "guessed" or "counted"; initially, no node is marked. Thanks to these flags, we can check if a node $u \in \mathcal{V}$ is a guesser: let $u_1, u_2, \ldots, u_k$ be the children of u that are also in $\mathcal{V}$ (recall that a view does not contain all nodes of a history tree); u is a *guesser* if and only if it is marked as counted, all the u_i's are marked as counted, and $a'(u) = \sum_i a'(u_i)$ (which implies $a(u) = \sum_i a(u_i)$, and thus no children of u are missing from $\mathcal{V}$).

ApproxCount will ensure that nodes marked as guessed are well-spread at all times; if a level of $\mathcal{V}$ contains a guessed node, it is said to be *locked*. A level L_t is *guessable* if it is not locked and has a non-counted node v that is guessable, i.e., there is a guesser u in L_{t-1} and the red edge $\{v, u\}$ is present in $\mathcal{V}$ with positive multiplicity.

The algorithm starts by assigning a conditional anonymity $a'(\tau) = x$ to the first leader node $\tau \in L_s$. (If no leader node exists in L_s, it immediately returns the error code -1, Line 6.) It also finds the longest strand P_τ hanging from τ, assigns the same conditional anonymity x to all of its nodes (including the unique child of the last node of P_τ) and marks them as counted (Lines 7–11). Then, as long as there are guessable levels and no counting cut has been found yet, the algorithm keeps assigning guesses to non-counted nodes (Line 12).

When a guess is made on a node v, some nodes in the path from v to its ancestor in L_s may become heavy; if so, let v' be the deepest heavy node. If $g(v')$ is not an integer, the algorithm returns the error code -3 (Line 19). (As we will prove later, this can only happen if $\delta \neq 1$ or some nodes in the strand P_τ have children that are not in the view $\mathcal{V}$.) Otherwise, if $g(v')$ is an integer, the algorithm marks v' as counted (Line 20), in accordance with Lemma 10. Furthermore, if the newly counted node v' is the root or a leaf of a complete isle I, then the conditional anonymities of all the internal nodes of I are determined, and such nodes are marked as counted; this also unlocks their levels if such nodes were marked as guessed (Lines 21–24).

In the end, the algorithm performs a "reality check" and possibly returns an estimate n' of n, as follows. If no counting cut was found, the algorithm returns the error code -2 (Line 25). Otherwise, a counting cut C has been found. The algorithm computes n' (respectively, ℓ') as the sum of the conditional anonymities of all nodes (respectively, all leader nodes) in C. If $\ell' = \ell$, then the algorithm returns n' (Line 32). Otherwise, it returns the error code -1 if $\ell' < \ell$ (Line 30) or the error code -3 if $\ell' > \ell$ (Line 31). In all cases, the algorithm also returns the maximum depth t of a guessed or counted node (excluding τ and its descendants), or s if no such node exists.

Consistency condition. In order for our algorithm to work properly, a condition has to be satisfied whenever a new guess is made. Indeed, note that all of our previous lemmas on guesses rest on the assumption that the conditional anonymities of a guesser and all of its children are known. However, while the node τ has a known conditional anonymity (by definition, $a'(\tau) = x$), the same is not necessarily true of the descendants of τ and all other nodes that are eventually marked as counted by the algorithm. This justifies the following definition.

▶ **Condition 1.** *During the execution of* `ApproxCount`*, if a guess is made on a node v at level $L_{t'}$ of $\mathcal{V}$, then τ has a (unique) descendant $\tau' \in L_{t'}$ and $a(\tau) = a(\tau')$.*

As we will prove next, as long as Condition 1 is satisfied during the execution of `ApproxCount`, all of the nodes between levels L_s and L_t that are marked as counted do have correct guesses (i.e., their guesses coincide with their conditional anonymities). Note that in general there is no guarantee that Condition 1 will be satisfied at any point; it is up to the main counting algorithm that invokes `ApproxCount` to ensure that the condition is satisfied often enough for our computations to be successful.

Correctness. In order to prove the correctness of `ApproxCount`, it is convenient to show that it also maintains some *invariants*, i.e., properties that are always satisfied as long as some conditions are met.

▶ **Lemma 11.** *Assume that $\delta \geq 1$. Then, as long as Condition 1 is satisfied, the following hold.*

(i) *The nodes marked as guessed are always well spread.*
(ii) *Whenever Line 13 is reached, there are no heavy nodes.*
(iii) *Whenever Line 13 is reached, all complete isles are trivial.*
(iv) *The conditional anonymity of any node between L_s and L_t that is marked as counted has been correctly computed.* ◀

Running time. We will now study the running time of `ApproxCount`. We will prove two lemmas that allow us to give an upper bound on the number of rounds it takes for the algorithm to return an output, provided that some conditions are satisfied.

▶ **Lemma 12.** *Assume that $\delta \geq 1$. Then, as long as Condition 1 holds, whenever Line 13 is reached, at most δn levels are locked.* ◀

We say that a node v of the history tree $\mathcal{H}$ is *missing* from level L_i of the view $\mathcal{V}$ if v is at the level of $\mathcal{H}$ corresponding to L_i but does not appear in $\mathcal{V}$. Clearly, if a level of $\mathcal{V}$ has no missing nodes, all previous levels also have no missing nodes.

▶ **Lemma 13.** *Assume that $\delta \geq 1$. Then, as long as level L_t of $\mathcal{V}$ is not missing any nodes (where t is defined and updated as in `ApproxCount`), whenever Line 13 is reached, there are at most $n-2$ levels in the range from L_{s+1} to L_t that lack a guessable non-counted node.* ◀

Main lemma. The following lemma gives some conditions that guarantee that `ApproxCount` has the expected behavior; it also gives some bounds on the number of rounds it takes for `ApproxCount` to produce an approximation n' of n, as well as a criterion to determine if $n' = n$.

▶ **Lemma 14.** *Let `ApproxCount`$(\mathcal{V}, s, x, \ell)$ return (n', t). Assume that τ exists and $x \geq a(\tau)$. Let τ' be the (unique) descendant of τ in $\mathcal{V}$ at level L_t, and let $L_{t'}$ be the last level of $\mathcal{V}$. Then:*

(i) *If $x = a(\tau) = a(\tau')$, then $n' \neq -3$.*
(ii) *If $n' > 0$ and $t' \geq t + n'$ and $a(\tau) = a(\tau')$, then $n' = n$.*
(iii) *If $t' \geq s + (\ell+2)n - 1$, then $s \leq t \leq s + (\ell+1)n - 1$ and $n' \neq -1$. Moreover, if $n' = -2$, then L_t contains a leader node with at least two children in $\mathcal{V}$.* ◀

Terminating algorithm. We are finally able to state our main terminating algorithm. It assumes that all processes know the number of leaders $\ell \geq 1$ and the dynamic disconnectivity T. Again, this is justified by Proposition 20 and Proposition 2.

▶ **Theorem 15.** *There is an algorithm that computes F_{GC} in all T-union-connected anonymous networks with $\ell \geq 1$ leaders and terminates in at most $(\ell^2 + \ell + 1)Tn$ rounds, assuming that ℓ and T are known to all processes, but assuming no knowledge of n.*

Proof. Due to Proposition 3, since T is known and appears as a factor in the claimed running time, we can assume that $T = 1$ without loss of generality. Also, note that determining n is enough to compute F_{GC}. Indeed, if a process determines n at round t', it can wait until round $\max\{t', 2Tn\}$ and run the algorithm in Theorem 7, which is guaranteed to give the correct output by that time.

In order to determine n assuming that $T = 1$, we let each process run the algorithm in Listing 3 with input $(\mathcal{V}, \ell)$, where $\mathcal{V}$ is the view of the process at the current round t'. We will prove that this algorithm returns a positive integer (as opposed to "Unknown") within $(\ell^2 + \ell + 1)n$ rounds, and the returned number is indeed the correct size of the system n.

Algorithm description. Let b be the number of branches in $\mathcal{V}$ representing leader processes (Line 5). The initial goal of the algorithm is to compute $\ell - b + 1$ approximations of n using the information found in as many disjoint intervals $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_{\ell-b+1}$ of levels of $\mathcal{V}$ (Lines 6–17).

If there are not enough levels in $\mathcal{V}$ to compute the desired number of approximations, or if the approximations are not all equal, the algorithm returns "Unknown" (Lines 11 and 15).

In order to compute an approximation of n, say in an interval of levels $\mathcal{L}_i$ starting at L_s, the algorithm goes through at most ℓ phases (Lines 8–16). The first phase begins by calling `ApproxCount` with starting level L_s and $x = \ell$, i.e., the maximum possible value for the anonymity of a leader node (Line 9). Specifically, `ApproxCount` chooses a leader node in $\tau \in L_s$ and tries to estimate n using as few levels as possible.

Let (n', t) be the pair of values returned by `ApproxCount`. If $n' = -1$, this is evidence that $\mathcal{V}$ is still missing some relevant nodes, and therefore "Unknown" is immediately returned (Line 11). If $n' = -2$, then a descendant of τ with multiple children in $\mathcal{V}$ was found, say at level L_t, before an approximation of n could be determined. As this is an undesirable event, the algorithm moves $\mathcal{L}_i$ after L_t and tries again to estimate n (Line 12). If $n' = -3$, then x may not be the correct anonymity of the leader node τ (see the description of `ApproxCount`), and therefore the algorithm calls `ApproxCount` again, with the same starting level L_s, but now with $x = \ell - 1$. If $n' = -3$ is returned again, then $x = \ell - 2$ is tried, and so on. After all possible assignments down to $x = 1$ have failed, the algorithm just moves $\mathcal{L}_i$ forward and tries again from $x = \ell$.

As soon as $n' > 0$, this approximation of n is stored in the variable n^*. If it is different from the previous approximations, then "Unknown" is returned (Line 15). Otherwise, the algorithm proceeds with the next approximation in a new interval of levels $\mathcal{L}_{i+1}$, and so on.

Finally, when $\ell - b + 1$ approximations of n (all equal to n^*) have been found, a correctness check is performed: the algorithm takes the last level L_{t^*} visited thus far; if the current round t' satisfies $t' \geq t^* + n^*$, then n^* is accepted as correct; otherwise "Unknown" is returned (Lines 18–20).

Correctness and running time. We will prove that, if the output of Listing 3 is not "Unknown", then it is indeed the number of processes, i.e., $n^* = n$. Since the $\ell - b + 1$ approximations of n have been computed on disjoint intervals of levels, there is at least one such interval, say $\mathcal{L}_j$, where no leader node in the history tree has more than one child (because there can be at most ℓ leader branches). With the notation of Lemma 14, this implies that $a(\tau) = a(\tau')$ whenever `ApproxCount` is called in $\mathcal{L}_j$. Also, since the option $x = \ell$ is tried first, the assumption $x \geq a(\tau)$ of Lemma 14 is initially satisfied. Note that `ApproxCount` cannot return $n' = -1$ or $n' = -2$, or else $\mathcal{L}_j$ would not yield any approximation of n. Moreover, by statement (ii) and by the terminating condition (Line 19), if $n' > 0$ while $x \geq a(\tau)$, then $n^* = n' = n$. On the other hand, by statement (i), we necessarily have $n' > 0$ by the time $x = a(\tau)$.

It remains to prove that Listing 3 actually gives an output other than "Unknown" within the claimed number of rounds; it suffices to show that it does so if it is executed at round $t' = (\ell^2 + \ell + 1)n$. It is known that all nodes in the first $t' - n = \ell(\ell + 1)n$ levels of the history tree are contained in the view $\mathcal{V}$ at round t' (cf. [17, Corollary 4.3]). Also, it is straightforward to prove by induction that the assumption of statement (iii) of Lemma 14 holds every time `ApproxCount` is invoked. Indeed, in any interval of $(\ell + 1)n$ levels, either a branching leader node is found or a new approximation of n is computed. Since there can be at most ℓ leader branches, at least one approximation of n is computed within $\ell(\ell+1)n$ levels. Because all nodes in these levels must appear in $\mathcal{V}$, the condition $a(\tau) = a(\tau')$ of Lemma 14 is satisfied in all intervals $\mathcal{L}_1$, $\mathcal{L}_2$, ..., $\mathcal{L}_{\ell-b+1}$. Reasoning as in the previous paragraph, we conclude that all such intervals must yield the correct approximation of n. So, every time Line 15 is executed, we have $n^* = n'$, and the algorithm cannot return "Unknown". ◀

5 Negative Results

In this section we list several negative results and counterexamples, some of which are well known (in particular, Proposition 16 is implied by [22, Theorem III.1]). The purpose is to justify all of the assumptions made in Sections 3 and 4. All proofs are found in [19].

5.1 Leaderless Networks

▶ **Proposition 16.** *No function other than the frequency-based multi-aggregate functions can be computed with no leader, even when restricted to simple connected static networks.* ◀

▶ **Proposition 17.** *No algorithm can solve the Average Consensus problem in a T-union-connected leaderless network in less than $2Tn - O(T)$ rounds.* ◀

▶ **Proposition 18.** *No algorithm can solve the leaderless Average Consensus problem with explicit termination if nothing is known about the size of the network, even when restricted to simple connected static networks.* ◀

5.2 Networks with Leaders

▶ **Proposition 19.** *No function other than the multi-aggregate functions can be computed (with or without termination), even when restricted to simple connected static networks with a known number of leaders.* ◀

▶ **Proposition 20.** *No algorithm can compute the Counting function F_C (with or without termination) with no knowledge about ℓ, even when restricted to simple connected static networks with a known and arbitrarily small ratio ℓ/n.* ◀

▶ **Proposition 21.** *For any $\ell \geq 1$, no algorithm can compute the Counting function F_C (with or without termination) in all simple T-union-connected networks with ℓ leaders in less than $T(2n - \ell) - O(T)$ rounds.* ◀

6 Conclusions

We have shown that anonymous processes in disconnected dynamic networks can compute all the multi-aggregate functions and no other functions, provided that the network contains a known number of leaders $\ell \geq 1$. If there are no leaders or the number of leaders is unknown, the class of computable functions reduces to the frequency-based multi-aggregate functions. We have also identified the functions F_{GC} and F_R as the complete problems for each class. Notably, the network's dynamic disconnectivity T does not affect the computability of functions, but only makes computation slower.

We also gave efficient stabilizing and terminating algorithms for computing all the above functions. Some of our algorithms make assumptions on the processes' a-priori knowledge about the network; we proved that such assumptions are actually necessary. All our algorithms have optimal linear running times in terms of T and the size of the network n.

In one case, there is still a small gap in terms of the number of leaders ℓ. Namely, for terminating computation with $\ell \geq 1$ leaders, we have a lower bound of $T(2n - \ell) - O(T)$ rounds (Proposition 21) and an upper bound of $(\ell^2 + \ell + 1)Tn$ rounds (Theorem 15). Although these bounds asymptotically match if the number of leaders ℓ is constant (which is a realistic assumption in most applications), optimizing them with respect to ℓ is left as an open problem.

Observe that our stabilizing algorithms use an unbounded amount of memory, as processes keep adding nodes to their view at every round. This can be avoided if the dynamic disconnectivity T (as well as an upper bound on n, in case of a leaderless network) is known: In this case, processes can run the stabilizing and the terminating version of the relevant algorithm in parallel, and stop adding nodes to their views when the terminating algorithm halts. It is an open problem whether a stabilizing algorithm for F_{GC} or F_R can use a finite amount of memory with no knowledge of T.

Our algorithms require processes to send each other explicit representations of their history trees, which have cubic size in the worst case [17]. It would be interesting to develop algorithms that only send messages of logarithmic size, possibly with a trade-off in terms of running time. We are currently able to do so for leaderless networks and networks with a unique leader, but not for networks with more than one leader [18].

References

1 D. Angluin, J. Aspnes, and D. Eisenstat. Fast Computation by Population Protocols with a Leader. *Distributed Computing*, 21(3):61–75, 2008.

2 J. Aspnes, J. Beauquier, J. Burman, and D. Sohier. Time and Space Optimal Counting in Population Protocols. In *Proceedings of the 20th International Conference on Principles of Distributed Systems (OPODIS '16)*, pages 13:1–13:17, 2016.

3 J. Beauquier, J. Burman, S. Clavière, and D. Sohier. Space-Optimal Counting in Population Protocols. In *Proceedings of the 29th International Symposium on Distributed Computing (DISC '15)*, pages 631–646, 2015.

4 J. Beauquier, J. Burman, and S. Kutten. A Self-stabilizing Transformer for Population Protocols with Covering. *Theoretical Computer Science*, 412(33):4247–4259, 2011.

5 D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., USA, 1989.

6 P. Boldi and S. Vigna. An Effective Characterization of Computability in Anonymous Networks. In *Proceedings of the 15th International Conference on Distributed Computing (DISC '01)*, pages 33–47, 2001.

7 P. Boldi and S. Vigna. Fibrations of Graphs. *Discrete Mathematics*, 243:21–66, 2002.

8 A. Casteigts, F. Flocchini, B. Mans, and N. Santoro. Shortest, Fastest, and Foremost Broadcast in Dynamic Networks. *International Journal of Foundations of Computer Science*, 26(4):499–522, 2015.

9 A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-Varying Graphs and Dynamic Networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

10 J. Chalopin, S. Das, and N. Santoro. Groupings and Pairings in Anonymous Networks. In *Proceedings of the 20th International Conference on Distributed Computing (DISC '06)*, pages 105–119, 2006.

11 J. Chalopin, E. Godard, and Y. Métivier. Local Terminations and Distributed Computability in Anonymous Networks. In *Proceedings of the 22nd International Symposium on Distributed Computing (DISC '08)*, pages 47–62, 2008.

12 J. Chalopin, Y. Métivier, and T. Morsellino. Enumeration and Leader Election in Partially Anonymous and Multi-hop Broadcast Networks. *Fundamenta Informaticae*, 120(1):1–27, 2012.

13 B. Charron-Bost and P. Lambein-Monette. Randomization and Quantization for Average Consensus. In *Proceedings of the 57th IEEE Conference on Decision and Control (CDC '18)*, pages 3716–3721, 2018.

14 B. Charron-Bost and P. Lambein-Monette. Computing Outside the Box: Average Consensus over Dynamic Networks. In *Proceedings of the 1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND '22)*, pages 10:1–10:16, 2022.

15 B. Chazelle. The Total s-Energy of a Multiagent System. *SIAM Journal on Control and Optimization*, 49(4):1680–1706, 2011.

16 G. A. Di Luna, P. Flocchini, T. Izumi, T. Izumi, N. Santoro, and G. Viglietta. Population Protocols with Faulty Interactions: The Impact of a Leader. *Theoretical Computer Science*, 754:35–49, 2019.

17 G. A. Di Luna and G. Viglietta. Computing in Anonymous Dynamic Networks Is Linear. In *Proceedings of the 63rd IEEE Symposium on Foundations of Computer Science (FOCS '22)*, pages 1122–1133, 2022.

18 G. A. Di Luna and G. Viglietta. Brief Announcement: Efficient Computation in Congested Anonymous Dynamic Networks. In *Proceedings of the 42nd ACM Symposium on Principles of Distributed Computing (PODC '23)*, pages 176–179, 2023.

19 G. A. Di Luna and G. Viglietta. Optimal Computation in Leaderless and Multi-Leader Disconnected Anonymous Dynamic Networks. *arXiv:2207.08061 [cs.DC]*, pages 1–37, 2023.

20 L. Faramondi, R. Setola, and G. Oliva. Performance and Robustness of Discrete and Finite Time Average Consensus Algorithms. *International Journal of Systems Science*, 49(12):2704–2724, 2018.

21 P. Fraigniaud, A. Pelc, D. Peleg, and S. Pérennes. Assigning Labels in Unknown Anonymous Networks. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC '00)*, pages 101–111, 2000.

22 J. M. Hendrickx, A. Olshevsky, and J. N. Tsitsiklis. Distributed Anonymous Discrete Function Computation. *IEEE Transactions on Automatic Control*, 56(10):2276–2289, 2011.

23 D. R. Kowalski and M. A. Mosteiro. Polynomial Counting in Anonymous Dynamic Networks with Applications to Anonymous Dynamic Algebraic Computations. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP '18)*, pages 156:1–156:14, 2018.

24 D. R. Kowalski and M. A. Mosteiro. Polynomial Anonymous Dynamic Distributed Computing Without a Unique Leader. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP '19)*, pages 147:1–147:15, 2019.

25 D. R. Kowalski and M. A. Mosteiro. Polynomial Counting in Anonymous Dynamic Networks with Applications to Anonymous Dynamic Algebraic Computations. *Journal of the ACM*, 67(2):11:1–11:17, 2020.

26 D. R. Kowalski and M. A. Mosteiro. Supervised Average Consensus in Anonymous Dynamic Networks. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '21)*, pages 307–317, 2021.

27 D. R. Kowalski and M. A. Mosteiro. Efficient Distributed Computations in Anonymous Dynamic Congested Systems with Opportunistic Connectivity. *arXiv:2202.07167 [cs.DC]*, pages 1–28, 2022.

28 D. R. Kowalski and M. A. Mosteiro. Polynomial Anonymous Dynamic Distributed Computing Without a Unique Leader. *Journal of Computer and System Sciences*, 123:37–63, 2022.

29 F. Kuhn, T. Locher, and R. Oshman. Gradient Clock Synchronization in Dynamic Networks. *Theory of Computing Systems*, 49(4):781–816, 2011.

30 F. Kuhn, N. Lynch, and R. Oshman. Distributed Computation in Dynamic Networks. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC '10)*, pages 513–522, 2010.

31 F. Kuhn, Y. Moses, and R. Oshman. Coordinated Consensus in Dynamic Networks. In *Proceedings of the 30th ACM Symposium on Principles of Distributed Computing (PODC '11)*, pages 1–10, 2011.

32 F. Kuhn and R. Oshman. Dynamic Networks: Models and Algorithms. *SIGACT News*, 42(1):82–96, 2011.

33 O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Naming and Counting in Anonymous Unknown Dynamic Networks. In *Proceedings of the 15th International Symposium on Stabilizing, Safety, and Security of Distributed Systems (SSS '13)*, pages 281–295, 2013.

34 O. Michail and P. G. Spirakis. Elements of the Theory of Dynamic Networks. *Communications of the ACM*, 61(2):72, 2018.

35 A. Nedić, A. Olshevsky, A. E. Ozdaglar, and J. N. Tsitsiklis. On Distributed Averaging Algorithms and Quantization Effects. *IEEE Transactions on Automatic Control*, 54(11):2506–2517, 2009.

36 A. Nedić, A. Olshevsky, and M. G. Rabbat. Network Topology and Communication-Computation Tradeoffs in Decentralized Optimization. *Proceedings of the IEEE*, 106(5):953–976, 2018.

37 R. O'Dell and R. Wattenhofer. Information Dissemination in Highly Dynamic Graphs. In *Proceedings of the 5th Joint Workshop on Foundations of Mobile Computing (DIALM-POMC '05)*, pages 104–110, 2005.

38 A. Olshevsky. Linear Time Average Consensus and Distributed Optimization on Fixed Graphs. *SIAM Journal on Control and Optimization*, 55(6):3990–4014, 2017.

39 A. Olshevsky and J. N. Tsitsiklis. Convergence Speed in Distributed Consensus and Averaging. *SIAM Journal on Control and Optimization*, 48(1):33–55, 2009.

40 A. Olshevsky and J. N. Tsitsiklis. A Lower Bound for Distributed Averaging Algorithms on the Line Graph. *IEEE Transactions on Automatic Control*, 56(11):2694–2698, 2011.

41 N. Sakamoto. Comparison of Initial Conditions for Distributed Algorithms on Anonymous Networks. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing (PODC '99)*, pages 173–179, 1999.

42 J. Seidel, J. Uitto, and R. Wattenhofer. Randomness vs. Time in Anonymous Networks. In *Proceedings of the 29th International Symposium on Distributed Computing (DISC '15)*, pages 263–275, 2015.

43 T. Sharma and M. Bashir. Use of Apps in the COVID-19 Response and the Loss of Privacy Protection. *Nature Medicine*, 26(8):1165–1167, 2020.

44 J. N. Tsitsiklis. *Problems in Decentralized Decision Making and Computation.* PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1984.

45 M. Yamashita and T. Kameda. Computing on an Anonymous Network. In *Proceedings of the 7th ACM Symposium on Principles of Distributed Computing (PODC '88)*, pages 117–130, 1988.

46 M. Yamashita and T. Kameda. Computing on Anonymous Networks. I. Characterizing the Solvable Cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89, 1996.

47 Y. Yuan, G.-B. Stan, L. Shi, M. Barahona, and J. Goncalves. Decentralised Minimum-Time Consensus. *Automatica*, 49(5):1227–1235, 2013.

A Impact on Fundamental Problems and State of the Art

As a byproduct of the results mentioned in Section 1.1, we are able to optimally solve two popular fundamental problems: *Generalized Counting* for multi-leader networks (because it is a multi-aggregate function) and *Average Consensus* for leaderless networks (because the mean is a frequency-based multi-aggregate function). As summarized in Table 1 and as discussed below, our results improve upon the state of the art on both problems in terms of (i) running time, (ii) assumptions on the network and the processes' knowledge, and (iii) quality of the solution. Altogether, we settle open problems from ICALP 2019 [24], SPAA 2021 [26], and FOCS 2022 [17]. For a more thorough discussion and a comprehensive survey of related literature, refer to [19].

Average Consensus. This problem has been studied for decades by the distributed control and distributed computing communities [5, 13, 14, 15, 26, 35, 38, 40, 44, 47]. In the following, we argue that our results directly improve upon the current state of the art on this problem. A more detailed discussion can be found in the surveys [20, 36, 39] and in [19].

A convergent algorithm with a running time of $O\left(Tn^3 \log(1/\epsilon)\right)$ is given in [35]. The algorithm works in T-union-connected networks with no knowledge of T, but it rests on the assumption that the degree of each process in the network has a known upper bound. Assuming an always connected network, [14] gives an algorithm that converges in $O\left(n^4 \log(n/\epsilon)\right)$ rounds. We remark that both algorithms are only ϵ-convergent; therefore, not only does our stabilizing algorithm improve upon their running times, but it solves a more difficult problem under weaker assumptions.

The algorithm in [13] stabilizes to the actual average in a linear number of rounds, but it is a randomized Monte Carlo algorithm and requires the network to be connected at each round. In contrast, our linear-time stabilizing algorithm is deterministic and works in disconnected networks.

As for terminating algorithms, the one in [26] terminates in $O\left(n^5 \log^3(n)/\ell\right)$ rounds assuming the presence of a known number ℓ of leaders and an always connected network. Since the number of leaders is known, our terminating algorithm for Generalized Counting also solves Average Consensus with a running time that improves upon [26] and does not require the network to be connected. We remark that our algorithm terminates in linear time when ℓ is constant.

Generalized Counting. Our results on this problem are direct generalizations of [17] to the case of multiple leaders and disconnected networks. The best previous counting algorithm with multiple known leaders is the one in [28], which terminates in $O\left(n^4 \log^3(n)/\ell\right)$ rounds and assumes the network to be connected at each round. In the same setting, our stabilizing and terminating algorithms have running times of $2n$ rounds and $(\ell^2 + \ell + 1)n$ rounds, respectively.

The only other result for disconnected networks is the recent preprint [27], which gives an algorithm that terminates in $\widetilde{O}\left(n^{2T+3}/\ell\right)$ rounds using $O(\log n)$-sized messages. Our terminating algorithm has a linear dependence on both n and T, which is an exponential improvement upon the running time of [27], but it requires polynomial-sized messages.