Optimal Computation in Anonymous Dynamic Networks

Giuseppe A. Di Luna* Giovanni Viglietta†

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 classic mass-distribution techniques, our work makes use of a novel combinatorial structure called *history tree*, which is of independent interest. Among other contributions, our results make conclusive 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 Yamashita–Kameda, IEEE T. Parall. Distr. 1996; Boldi–Vigna, Discrete Math. 2002; Nedić et al., IEEE T. Automat. Contr. 2009; Kowalski–Mosteiro, J. ACM 2020.

Our contribution not only opens a promising line of research on applications of history trees, but also demonstrates that computation in anonymous dynamic networks is practically feasible and far less demanding than previously conjectured.

*DIAG, Sapienza University of Rome, Italy, g.a.diluna@gmail.com

†Department of Computer Science and Engineering, University of Aizu, Japan, viglietta@gmail.com

Contents

1 Introduction

1.1 Background

Dynamic networks. The study of theoretical and practical aspects of highly dynamic distributed systems has garnered significant attention in recent years [11, 40, 43]. These systems involve a constantly changing network of computational devices called *processes* (sometimes referred to as "processors" or "agents"). Pairs of processes may send each other messages only through *links* that appear or disappear unpredictably. This dynamicity is typical of modern real-world systems and is the result of technological innovations, such as the spread of mobile devices, software-defined networks, wirelesses sensor networks, wearable devices, smartphones, etc.

Connected networks. There are several models of dynamicity [11]; a popular choice is the *1-interval-connected* network model [38, 47]. Here, a fixed set of n processes communicate through links forming a time-varying graph, i.e., a graph whose edge set changes at discrete time units called *rounds* (thus, the system is synchronous); such a graph changes unpredictably, but is assumed to be connected at all times.

Disconnected networks. The 1-interval-connected network model may not be a suitable choice 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 more realistic assumption is that the union of all the network's links across any τ consecutive rounds induces a connected graph on the processes [36, 44, 49].[1] We say that such a network is *τ-union-connected*, and the smallest such parameter $\tau \geq 1$ is called *dynamic disconnectivity*.[2] Observe that 1-interval-connected networks can equivalently be characterized as 1-union-connected networks.

Networks with unique IDs. A large number of research papers have considered dynamic systems where each process has a distinct identity (*unique IDs*) [38]. In this setting, there are efficient algorithms for consensus [39], broadcast [10], counting [38, 47], and many other fundamental problems [37, 43].

It should be noted that networks with unique IDs allow for very simple algorithms for a wide variety of problems. Indeed, in a 1-interval-connected network of n processes, if each process broadcasts the set of its "known IDs" at every round (initially, a process only knows its own ID), then every process can learn the IDs of all other processes in n rounds. If input values are attached to these IDs, then the network can compute any function of such input values within n rounds.

Anonymous networks. Networks without IDs, also called *anonymous systems*, are a considerably more challenging scenario. In this model, all processes have identical initial states, and may only differ by their inputs. In the last thirty years, a large body of works have investigated the computational power of anonymous static networks, giving characterizations of what can be computed in various settings [8, 9, 13, 14, 15, 28, 52, 56].

Anonymous systems are not only important from a theoretical standpoint, but also have a remarkable practical relevance. In a highly dynamic system, IDs may not be guaranteed to be unique due to operational limitations [47], or may compromise user privacy. Indeed, users may not be willing to be tracked or to disclose information about their behavior; examples are COVID-19 tracking apps [53], where a threat to privacy was felt by a large share of the public even if these

[1] It is well known that non-trivial (terminating) computation requires some conditions on temporal connectivity to be met, as well as some a-priori knowledge by processes (refer to Proposition 2.3).

[2] Another widely used parameter for dynamic networks is the *dynamic diameter* d. The relationship between τ and d is discussed at the end of Section 2.3. In particular, it is worth noting that every occurrence of the parameter τ in the running times of our algorithms can be safely replaced with d.

apps were assigning a rotating random ID to each user. In fact, an adversary can easily track the continuous broadcast of a fixed random ID tracing the movements of a person [4]. Anonymity is also found in insect colonies and other biological systems [29].

Networks with leaders. In order to deterministically solve several non-trivial problems in anonymous systems, it is necessary to have some form of initial "asymmetry" [1, 9, 42, 56]. The most common assumption is the existence of a single distinguished process called *leader* [2, 3, 5, 6, 22, 28, 31, 33, 42, 51, 57] or, less commonly, a subset of several leaders (and knowledge of their number[3]) [32, 34, 35, 36]. The presence of leaders is a realistic assumption: examples include base stations in mobile networks, gateways in sensor networks, etc. For these reasons, the computational power of anonymous systems enriched with one or more leaders has been extensively studied in the traditional model of static networks [28, 51, 57], as well as in population protocols [2, 3, 5, 6, 22].

Apart from the theoretical importance of generalizing the usual single-leader scenario, studying networks with multiple leaders also has practical impacts 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 [16, 44, 45, 48, 55, 58]. 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 [7, 26, 49, 50].

Problem classes. Assume that each process is assigned an input at the beginning of the computation, i.e., at round 0. The *Input Frequency function* is the function that returns the percentage of processes that have each input value. Being able to compute the Input Frequency function allows a system to compute a wide class of functions called *frequency-based* with no loss in performance [30]. Thus, we say that the Input Frequency function is *complete* for this class of functions (refer to Section 2.2). The most prominent representative of this class of functions is given by the aforementioned Average Consensus problem, since the mean of a multiset of numeric input values is a frequency-based function (the percentage of processes that have each input can be used as weight to compute the mean of all inputs).

The *Input Multiset function* is the function that returns the number of processes that have each input value. This function is complete for a class of functions called *multiset-based*, which strictly includes the frequency-based ones (see Section 2.2 for definitions). A well-studied example of a multiset-based function that is not frequency-based is given by the *Counting problem*, which asks to determine the total number of processes in the system. The Counting problem is practically interesting in real-world scenarios such as large-scale ad-hoc sensor networks, where the individual processes may be unaware of the size of the system [47].

1.2 State of the Art

Average Consensus problem. In the Average Consensus problem, each process starts with an input value, and the goal is to compute the average of these initial values. The typical approach found in the literature is based on local averaging algorithms, where each process updates its local

[3]It is easy to see that a network with an unknown number of leaders is equivalent to a network with no leaders at all. Also, if the leaders are distinguishable from each other, then any one of them can be elected as a unique leader. Hence, the only genuinely interesting multi-leader case is the one with a known number of indistinguishable leaders.

value at each round based on a convex combination of the values of its neighbors. This and similar techniques for Average Consensus have been studied for decades by the distributed control and distributed computing communities [7, 16, 34, 44, 48, 49, 50, 55, 58]. Existing works can be divided into those that give *convergent* solutions, those that give *stabilizing* solutions, and those that give *terminating* solutions.

In convergent algorithms, the consensus is not reached in finite time, but each process's local value asymptotically converges to the average. A state-of-the-art ϵ-convergent algorithm based on Metropolis rules with a running time of $O\left(\tau n^3 \log(1/\epsilon)\right)$ communication rounds is given in [44]. However, this algorithm rests on the assumption that the degree of each process in the network has a known upper bound.[4] The running time can be improved to $O\left(n^2 \log(1/\epsilon)\right)$ communication rounds if the network is always connected (i.e., $\tau = 1$) and may only change every three rounds (as opposed to every round).

In stabilizing algorithms, the consensus is reached in a finite number of rounds, but no termination criteria are specified, and therefore all processes are assumed to run indefinitely. To the best of our knowledge, all stabilizing Average Consensus algorithms for leaderless networks are either probabilistic or assume the network to be static [26, 30, 48, 58]. However, some of these algorithms stabilize in a linear number of communication rounds [48, 58].

As for terminating Average Consensus algorithms, the one in [34] terminates in $O\left(n^{5+\epsilon} \log^3(n)/\ell\right)$ communication rounds (for $\epsilon > 0$) assuming the presence of a known number $\ell \geq 1$ of leaders and an always connected network.

A major research question left open in previous works is the following.

Research Question 1. In anonymous dynamic leaderless networks, are there deterministic Average Consensus algorithms that stabilize (or terminate) in linear time?

Counting problem. A long series of papers have focused the Counting problem in connected anonymous networks with a unique leader [12, 17, 18, 19, 20, 21, 31, 32, 36, 42].[5] These works have achieved better and better running times, with the first polynomial-time algorithm presented in J. ACM in 2020, having a running time of $O\left(n^5 \log^2(n)\right)$ rounds [33]. This was later improved and extended to networks with $\ell \geq 1$ leaders in [36], solving the Counting problem in $O\left(n^{4+\epsilon} \log^3(n)/\ell\right)$ communication rounds (for $\epsilon > 0$).

The only result for τ-union-connected networks is the recent preprint [35], which gives an algorithm that terminates in $\widetilde{O}\left(n^{3+2\tau(1+\epsilon)}/\ell\right)$ communication rounds, assuming the presence of $\ell \geq 1$ leaders. We remark that this algorithm is exponential in τ, and for $\tau = 1$ it has a running time of $\widetilde{O}\left(n^{5+\epsilon}/\ell\right)$.

Almost all of these works share the same basic approach of implementing a mass-distribution mechanism similar to the local averaging used to solve the Average Consensus problem.[6] In spite of the technical sophistication of this line of research, there is still a striking gap in terms of running time between state-of-the-art algorithms for anonymous networks and networks with unique IDs. The same gap exists with respect to static anonymous networks, where the Counting problem is known to be solvable in $O(n)$ rounds [42]. Given the current state of the art, solving non-trivial problems in large-scale dynamic networks is still impractical.

[4] More generally, averaging algorithms based on Metropolis rules cannot be applied to our network model, because they require all processes to know their outdegree at every round prior to sending their messages.

[5] It is folklore that the Counting problem cannot be solved in leaderless networks.

[6] We remark that the mass-distribution approach used for Counting and Average Consensus requires processes to communicate numbers whose representation size grows at least linearly with the number of rounds. Thus, all cited algorithms require processes to exchange messages of polynomial size.

Problem class	*Leaders*	*Terminating*	*Assumptions*	*Running time*	*Lower bound*
Frequency-based (e.g., Average Consensus)	$\ell = 0$	✗		$\tau(2n-2)+1$	$\tau(2n-6)$
		✓	τ and $N \geq n$ known	$\tau(n+N-2)+1$	$\tau(2n-4)$
			d known	$\tau(n-1)+d+1 \leq dn+1$	
Multiset-based (e.g., Counting)	$\ell \geq 1$	✗	ℓ known	$\tau(2n-3)+1$	$\tau(2n-\ell-5)$
		✓	ℓ and τ known	$\tau((\ell^2+\ell+1)(n-1)+1)$	$\tau(2n-\ell-3)$

Table 1: Summary of the results in this paper. The variable n indicates the number of processes in the system, ℓ is the number of leaders, N is an upper bound on n, τ is the dynamic disconnectivity of the network, and d is (an upper bound on) its dynamic diameter. If $\ell = 0$, only the frequency-based functions are computable; if $\ell \geq 1$, the multiset-based functions are also computable.

Research Question 2. In anonymous dynamic networks with a fixed number of leaders, are there deterministic Counting algorithms that stabilize (or terminate) in linear time?

1.3 Our Contributions

Summary. Focusing on anonymous dynamic networks, in this paper we completely elucidate the relationship between leaderless networks and networks with one or more leaders, as well as the impact of the dynamic disconnectivity τ on the efficiency of distributed algorithms. In particular, we characterize the solvable problems in each of these settings and we provide optimal linear-time algorithms for all solvable problems.

Technique. Our approach departs radically from the mass-distribution and averaging techniques traditionally adopted by most previous works on anonymous dynamic networks. Instead, all of our results are based on a novel combinatorial structure called *history tree*, which completely represents an anonymous dynamic network and naturally models the idea that processes can be distinguished if and only if they have different "histories" (see Section 3). Thanks to the simplicity of our technique, this paper is entirely self-contained, our proofs are transparent and easy to understand, and our algorithms are elegant and straightforward to implement.

An implementation can be found at `https://github.com/viglietta/Dynamic-Networks`. The repository includes a dynamic network simulator that can be used to test our algorithms and visualize the history trees of custom networks.

Computability. We give an exact characterization of the set of functions that can be computed deterministically in anonymous dynamic networks with and without leaders, respectively. Namely, in networks with at least one leader, a function is computable if and only if it is multiset-based;[7] in networks with no leaders, a function is computable if and only if it is frequency-based.[8] Interestingly, computability is independent of the dynamic disconnectivity τ.

Algorithms. Furthermore, we give efficient deterministic[9] algorithms for computing the Input Frequency function in leaderless networks (Sections 4.2 and 4.3) and the Input Multiset function

[7] Another way of stating this result is that it is sufficient to know the size of *any* subset of distinguished processes in order to compute all multiset-based functions.

[8] A similar result, limited to *static* leaderless networks and functions with additional constraints, was obtained in [30].

[9] An advantage of allowing randomization in our model would be the possibility of choosing unique IDs with high probability. Even though this would greatly simplify most problems, it would not afford an asymptotic improvement on the running times of our algorithms; see [38].

in networks with leaders (Sections 4.4 and 5). Since these functions are complete for the class of frequency-based and multiset-based functions, respectively, we automatically obtain efficient algorithms for computing *all* functions in these classes (Section 2.2).

For each of the aforementioned functions, we give two algorithms: a *terminating* version, where all processes are required to commit on their output and never change it, and a slightly more efficient *stabilizing* version, where processes are allowed to modify their outputs, provided that they eventually stabilize on the correct output. The running times of our algorithms are summarized in Table 1.

The stabilizing algorithms for both functions give the correct output within $2\tau n$ communication rounds regardless of the number of leaders, and do not require any knowledge of the dynamic disconnectivity τ or the number of processes n. Our terminating algorithm for leaderless networks runs in $\tau(n+N)$ communication rounds with knowledge of τ and an upper bound $N \geq n$; the terminating algorithm for $\ell \geq 1$ leaders runs in $\tau(\ell^2+\ell+1)n$ communication rounds[10] with no knowledge of n. The latter running time is reasonable (i.e., linear) in most applications, as ℓ is typically a constant or negligible compared to n.

We emphasize that the running times of our algorithms are specified as precise values, rather than being expressed in big-O notation. In fact, we made an effort to optimize the multiplicative constants, as well as the asymptotic complexity of our algorithms. To our knowledge, this feature is unique within the entire body of literature on anonymous dynamic networks.

We remark that the local computation time and the amount of processes' internal memory required by our terminating algorithms is only polynomial in the size of the network. Also, like in previous works on anonymous dynamic networks, processes need to send messages of polynomial size.

Negative results. Some of our algorithms assume processes to have a-priori knowledge of some parameters of the network; in Section 6 we show that none of these assumptions can be removed. 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). Table 1 summarizes our lower bounds, as well. We point out that our lower bounds of roughly $2\tau n$ rounds are the first non-trivial lower bounds for simple undirected anonymous dynamic networks (i.e., better than $n-1$).

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 existing 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. Furthermore, this turns out to be a remarkably powerful feature in light of Proposition 2.4, which establishes a relationship between multigraphs and τ-union-connected networks. This finding single-handedly allows us to generalize our algorithms to disconnected networks at the cost of a mere factor of τ in their running times, which is worst-case optimal.

Significance. Our general technique based on history trees enables us to approach all problems related to anonymous networks in a uniform and systematic manner. This technique also allows us to design straightforward algorithms that achieve previously unattainable running times. Indeed, our results make conclusive advancements on long-standing problems within anonymous dynamic networks, particularly with regards to the Average Consensus and Counting problems, in several aspects:

[10]Note that the case where all processes are leaders is not equivalent to the case with no leaders, because processes do not have the information that $\ell = n$, and have to "discover" that there are no non-leader processes in the network.

- *Running time.* Our algorithms are asymptotically worst-case optimal, i.e., linear (for networks with a fixed number of leaders ℓ and a fixed dynamic disconnectivity τ).
- *Assumptions on the network.* Unlike most previous solutions, our algorithms work in any dynamic network that has a finite dynamic disconnectivity.
- *Knowledge of the processes.* Our algorithms assume that the processes have a-priori knowledge about certain properties of the network only when the absence of such knowledge would render the Average Consensus or Counting problems unsolvable.
- *Quality of the solution.* Unlike several previous works, our algorithms are deterministic rather than probabilistic, and stabilize or terminate rather than converge.

Altogether, we settle open problems from several papers, including Nedić et al., IEEE Trans. Automat. Contr. 2009 [44]; Olshevsky, SIAM J. Control Optim. 2017 [48]; Kowalski–Mosteiro, J. ACM 2020 [33] and SPAA 2021 [34]. In particular, we settle the two research questions stated in Section 1.2:

Contribution 1. In anonymous dynamic leaderless networks, any problem that can be solved deterministically (including Average Consensus) has an algorithm that terminates (thus stabilizes) in linear time.

Contribution 2. In anonymous dynamic networks with a fixed number of leaders, any problem that can be solved deterministically (including Counting) has an algorithm that terminates (thus stabilizes) in linear time.

Our findings indicate that computing within anonymous dynamic networks entails an overhead when compared to networks equipped with unique IDs. However, this overhead is only linear, which is substantially lower than previously conjectured [33, 36]. In fact, our results demonstrate that general computations in anonymous and dynamic large-scale networks are both feasible and efficient in practice.

Unique-leader case. Our results concerning connected networks with a unique leader (i.e., $\tau = \ell = 1$) are especially noteworthy. We have an algorithm for the Counting problem that stabilizes in less than $2n$ rounds, which is worst-case optimal up to a small additive constant. We also have a terminating algorithm with a running time of less than $3n$ rounds, which is remarkably close to the lower bound of $2n$ rounds. We recall that, prior to our work, the state-of-the-art algorithm for this problem had a running time of $O\left(n^{4+\epsilon}\log^3(n)\right)$ rounds (with $\epsilon > 0$) [36].

Previous versions. This work is an extension of two preliminary conference papers that appeared at FOCS 2022 [23] and DISC 2023 [25], respectively. In preparing the current version, we have reworked several sections, included all missing proofs, and provided a more accurate analysis of the performance of our algorithms. In addition, the upper bound in terms of the dynamic diameter d for leaderless networks is a new contribution (see Table 1). We also added a comparisons between history trees and previous structures and concepts related to anonymous and dynamic networks (see Section 3.3).

2 Definitions and Preliminaries

In Section 2.1 we define the model of computation of anonymous dynamic networks. In Section 2.2 we define two classes of functions and their respective "complete" functions, which will play an important role in the rest of the paper. Finally, in Section 2.3 we discuss τ-union-connected networks and the impact of the dynamic disconnectivity τ on the running time of algorithms.

2.1 Model of Computation

Processes and networks. A *dynamic network* on a finite non-empty set $V = \{p_1, p_2, \dots, p_n\}$ is an infinite sequence $\mathcal{G} = (G_t)_{t \geq 1}$, where $G_t = (V, E_t)$ is an undirected multigraph, i.e., E_t is a multiset of unordered pairs of elements of V. In this context, the set V is called *system*, and its $n \geq 1$ elements are the *processes*. The elements of the multiset E_t are called *links*; note that we allow any (finite) number of "parallel links" between two processes, as well as "self-loops".[11]

Input and internal states. 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.[12] 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 $\mathcal{A}$. That is, $\mathcal{A}$ takes as input the current internal state of the process and the multiset of incoming messages, and returns the new internal state of the process. Note that $\mathcal{A}$ is deterministic and is the same for all processes.[13] 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 by ℓ.

Stabilization and termination. Each process 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.

Computation. We say that an algorithm $\mathcal{A}$ *computes* a function F if, whenever the processes in the system are assigned inputs $\lambda(p_1)$, $\lambda(p_2)$, ..., $\lambda(p_n)$ 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 $F(p_i, \lambda)$.[14] 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 n processes and all possible input assignments.

[11] In the dynamic networks literature, G_t is typically assumed to be a simple graph, for at most one link between the same two processes is allowed. However, our results hold more generally for multigraphs.

[12] In order to model wireless radio communication, it is natural to assume that each process in a dynamic network broadcasts its messages to all its neighbors (a message is received by anyone within communication range). The network's anonymity prevents processes from specifying single destinations.

[13] This network model bears some similarities with Population Protocols, although there are radical differences in the way symmetry is handled. The point-to-point communication model of Population Protocols automatically breaks the symmetry between communicating agents, greatly simplifying problems such as Leader Election, Average Consensus, etc.

[14] Formally, a function computed by a system of n processes maps n-tuples of input values to n-tuples of output values. Writing such a function as $F(p_i, \lambda)$ emphasizes that the output of a process may depend on all processes' inputs, as well as on the process itself. That is, different processes may give different outputs.

2.2 Classes of Functions

Multiset-based 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 $z_i = \lambda(p_{j_1}) = \lambda(p_{j_2}) = \dots = \lambda(p_{j_{m_i}})$; note that $n = \sum_{i=1}^{k} m_i$. A function F is *multiset-based* if it has the form $F(p_i, \lambda) = \psi(\lambda(p_i), \mu_\lambda)$. That is, the output of each process depends only on its own input and the multiset of all processes' inputs.[15]

The special multiset-based functions $F_C(p_i, \lambda) = n$ and $F_{IM}(p_i, \lambda) = \mu_\lambda$ are called the *Counting* function and the *Input Multiset* function, respectively. It is easy to see that, if a system can compute the Input Multiset function F_{IM}, then it can compute any multiset-based function in the same number of rounds: thus, F_{IM} is *complete* for the class of multiset-based functions.

Proposition 2.1. *If the Input Multiset function F_{IM} can be computed (with termination), then all multiset-based functions can be computed (with termination) in the same number of rounds, as well.*

Proof. Once a process p_i with input $\lambda(p_i)$ has determined the multiset μ_λ of all processes' inputs, it can immediately compute any desired function $\psi(\lambda(p_i), \mu_\lambda)$ within the same local computation phase. □

We remark that Proposition 2.1 does not require a unique leader or a connected network.

Frequency-based functions. 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 multiset-based 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 [30]). That is, F depends only on the "frequency" of each input in the system, rather than on their actual multiplicities. Notable examples of frequency-based functions include statistical functions of input values, such as mean, variance, maximum, median, mode, etc. Observe that the sum of all input values is a multiset-based function but not a frequency-based function.

The problem of computing the mean of all input values is called *Average Consensus* [7, 16, 26, 34, 44, 45, 48, 49, 50, 55, 58]. The frequency-based function $F_{IF}(p_i, \lambda) = \frac{1}{n} \cdot \mu_\lambda$ is called *Input Frequency* function, and is complete for the class of frequency-based functions.

Proposition 2.2. *If the Input Frequency function F_{IF} can be computed (with termination), then all frequency-based functions can be computed (with termination) in the same number of rounds, as well.*

Proof. Suppose that a process p_i has determined $\frac{1}{n} \cdot \mu_\lambda = \{(z_1, m_1/n), (z_2, m_2/n), \dots, (z_k, m_k/n)\}$. Then it can immediately find the smallest integer $d > 0$ such that $d \cdot (m_i/n)$ is an integer for all $1 \le i \le k$. Note that $\frac{d}{n} \cdot \mu_\lambda$ is a multiset. Hence, in the same round, p_i can compute any desired function $\psi(\lambda(p_i), \frac{d}{n} \cdot \mu_\lambda)$, and thus any frequency-based function, by definition. □

2.3 Disconnected Networks

Although the network G_t at each individual round may be disconnected, in this paper we assume dynamic networks to be *τ-union-connected*. That is, there is a (smallest) parameter $\tau \ge 1$, called *dynamic disconnectivity*, such that the sum of any τ consecutive G_t's is a connected multigraph.[16]

[15] Multiset-based functions are also know as *multi-aggregation* or *multi-aggregate* functions [23, 25].
[16] By definition, the sum of (multi)graphs is obtained by adding together their adjacency matrices.

Thus, for all $i \geq 1$, the multigraph $\left(V, \bigcup_{t=i}^{i+\tau-1} E_t\right)$ is connected (we remark that a union of multisets adds together the multiplicities of equal elements).[17]

We will prove that terminating computations are impossible without some a-priori knowledge about τ. We say that a function $F(p_i, \lambda)$ is *trivial* if and only if it is of the form $F(p_i, \lambda) = \psi(\lambda(p_i))$. That is, the output of any process p_i depends only on its own input $\lambda(p_i)$ and not on the inputs of other processes; therefore, F can be computed "locally" and does not require communications between processes.

Proposition 2.3. *Any non-trivial function is impossible to compute with termination unless the processes have some knowledge about* τ*.*

Proof. Assume for a contradiction that a non-trivial function $F(p_i, \lambda)$ is computed with termination by an algorithm $\mathcal{A}$ with no knowledge of τ. Since F is non-trivial, there is an input value z and two distinct output values y and y' with the following properties. (i) If p_1 is the only process in a network (i.e., $n = 1$), and p_1 is assigned input z and executes $\mathcal{A}$, it terminates in t rounds with output y. (ii) There exists a network size $\tilde{n} > 1$ and an input assignment λ with $\lambda(p_1) = z$ such that, whenever the processes in a network of size $\tilde{n}$ are assigned input λ and execute $\mathcal{A}$, the process p_1 eventually terminates with output $y' \neq y$.

Let us now consider a dynamic network of $\tilde{n}$ processes, where p_1 is kept disconnected from all other processes for the first t rounds (hence $\tau > t$). Assign input λ to the processes, and let them execute algorithm $\mathcal{A}$. Due to property (i), since p_1 is isolated for t rounds and has no knowledge of τ, it terminates in t rounds with output y. This contradicts property (ii), which states that p_1 should terminate with output $y' \neq y$. □

The following result will be used repeatedly to reduce computations on τ-union connected networks to computations on 1-union-connected networks.

Proposition 2.4. *A function F can be computed (with termination) within $f(n)$ rounds in any dynamic network with $\tau = 1$ if and only if F can be computed (with termination) within $\tau \cdot f(n)$ rounds in any dynamic network with $\tau \geq 1$, assuming that τ is known to all processes.*

Proof. Subdivide time into *blocks* of τ consecutive rounds, and consider the following algorithm. Each process collects and stores all messages it receives within a same block, and updates its state all at once at the end of the block. This reduces any τ-union-connected network $\mathcal{G} = ((V, E_t))_{t \geq 1}$ to a 1-union-connected network $\mathcal{G}' = ((V, E'_t))_{t \geq 1}$, where $E'_t = \bigcup_{i=(t-1)\tau+1}^{t\tau} E_i$. Thus, if F can be computed within $f(n)$ rounds in all 1-union-connected networks (which include $\mathcal{G}'$), then F can be computed within $\tau f(n)$ rounds in the original network $\mathcal{G}$.[18]

Conversely, consider a 1-union-connected network $\mathcal{G}$, and construct a τ-union-connected network $\mathcal{G}'$ by inserting $\tau - 1$ empty rounds (i.e., rounds with no links at all) between every two consecutive rounds of $\mathcal{G}$. Since no information circulates during the empty rounds, if F cannot be computed within $f(n)$ rounds in $\mathcal{G}$, then F cannot be computed within $\tau f(n)$ rounds in $\mathcal{G}'$ (recall that running times are measured in the worst case across all possible networks). □

[17] Our τ-union-connected networks should not be confused with the τ-*interval*-connected networks from [38]. In those networks, the *intersection* (as opposed to the union) of any τ consecutive E_t's induces a connected (multi)graph. In particular, a τ-interval-connected network is connected at every round, while a τ-union-connected network may not be, unless $\tau = 1$. Incidentally, a network is 1-interval-connected if and only if it is 1-union-connected.

[18] Note that this argument is correct because algorithms are required to work for all multigraphs, as opposed to simple graphs only. Indeed, since a process p_i may receive multiple messages from the same process p_j within a same block, the resulting network $\mathcal{G}'$ may have multiple links between p_i and p_j in a same round, even if $\mathcal{G}$ does not.

Relationship between dynamic disconnectivity and the dynamic diameter. A concept closely related to the dynamic disconnectivity τ 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 [11, 40]. It is a simple observation that $\tau \leq d \leq \tau(n-1)$.

We chose to predominantly use τ, as opposed to d, to measure the running times of our algorithms for several reasons. Firstly, τ is well defined (i.e., finite) if and only if d is; however, τ has a simpler definition, and is arguably easier to directly estimate or enforce in a real network. Secondly, Proposition 2.4, as well as all of our theorems, remain valid if we replace τ with d; nonetheless, stating the running times of our algorithms in terms of τ is preferable, because $\tau \leq d$.

3 History Trees

In this section, we introduce *history trees* as a natural tool of investigation for anonymous dynamic networks. In Section 3.1 we give definitions and we discuss basic structural properties of history trees. In Section 3.2 we give a local algorithm to incrementally construct history trees and we prove a *fundamental theorem* about this construction. Finally, in Section 3.3 we discuss related concepts found in previous literature.

3.1 Definition of History Tree

We will describe the structure of a history tree and its basic properties. For reference, an example showing part of a history tree is found in Figure 1.

Indistinguishable processes. In an anonymous network, processes can only be distinguished by their inputs or by the multisets of messages they have received thus far. This leads to an inductive definition of *indistinguishability*: Two processes are indistinguishable at the end of round 0 if and only if they have the same input. At the end of round $t \geq 1$, two processes p and q are indistinguishable if and only if they were indistinguishable at the end of round $t-1$ and, for every equivalence class A of processes that were indistinguishable at the end of round $t-1$, both p and q receive an equal number of (identical) messages from processes in A at round t.

Levels of a history tree. A *history tree* is a structure associated with a dynamic network. It is an infinite graph whose nodes are subdivided into *levels* L_{-1}, L_0, L_1, L_2, $\dots$, where each node in layer L_t, with $t \geq 0$, represents an equivalence class of processes that are indistinguishable at the end of round t. The level L_{-1} contains a unique node r, representing all processes in the system.

In addition, each node of L_0 has a *label* denoting the input of the processes it represents (by definition of indistinguishability at round 0, each node of L_0 represents all processes with a certain input). Nodes not in L_0 do not have any labels.

Black and red edges. A history tree has two types of undirected edges; each edge connects nodes in consecutive levels. The *black edges* induce an infinite tree rooted at r and spanning all nodes. 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 multiedges* represent messages. Each 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 a total of m (identical) messages from processes represented by v.

Anonymity of a node. The *anonymity* $a(v)$ of a node v of a history tree is defined as the number of processes represented by v. Since the nodes in a level represent a partition of all the processes, the sum of their anonymities must be equal to the total number of processes in the system, n. Moreover,

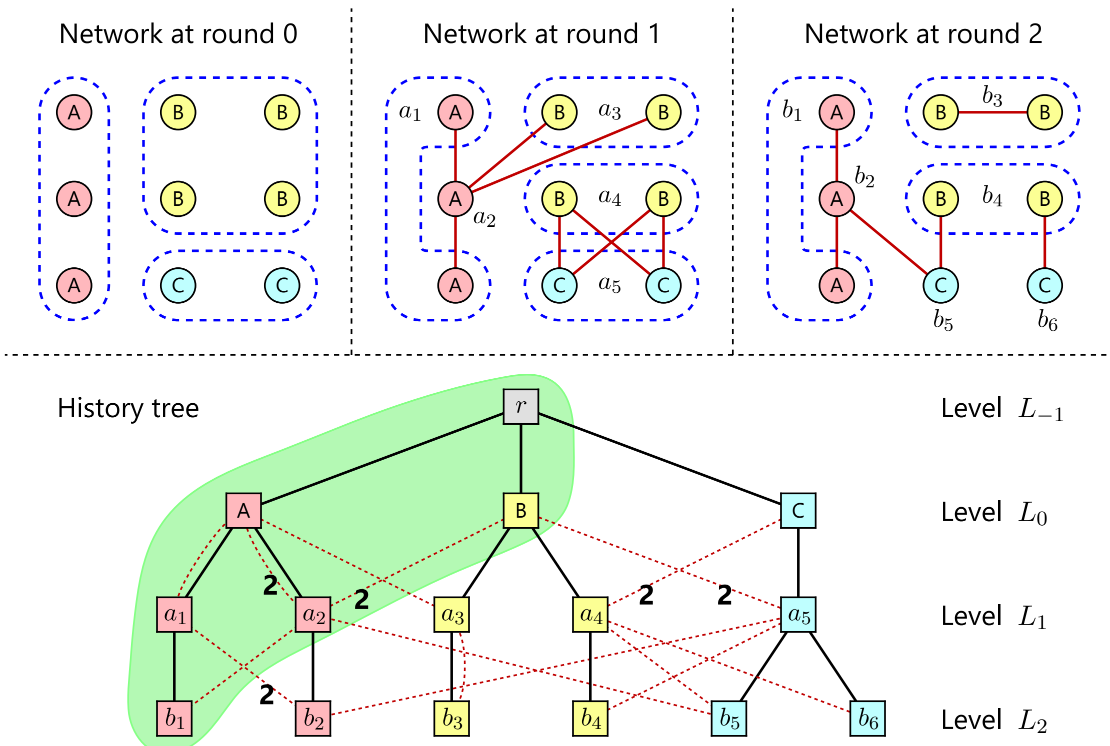

Figure 1: The first rounds of a dynamic network with $n = 9$ processes and the corresponding levels of the history tree. Level L_t consists of all nodes at distance $t + 1$ from the root r, which represent indistinguishable processes at the end of the tth communication round. The multiplicities of the 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 in the network). 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 induced by the vertices in the green blob is the view of the two processes in b_1. None of the levels of this view is complete, except L_{-1}.

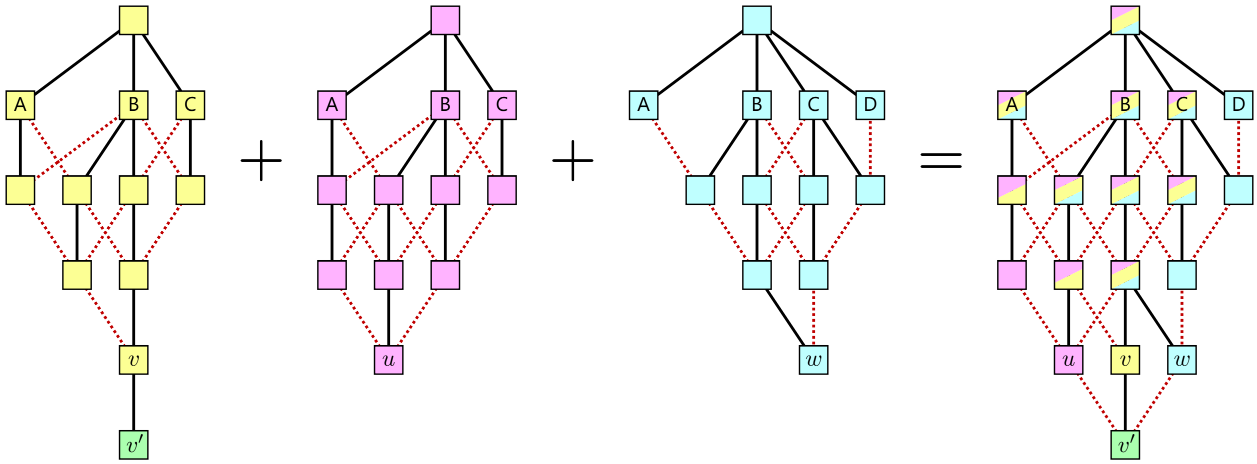

Figure 2: Updating the view of a process represented by node v after it receives messages from a process represented by u and a process represented by w. The node v' is created as a child of v, the views corresponding to u and w are merged with the old view, and red edges are created connecting v' with u and w.

by the definition of black edges, the anonymity of a node is equal to the sum of the anonymities of its children.

Observe that the problem of computing the Input Multiset function can be rephrased as the problem of determining the anonymities of all the nodes in L_0. Similarly, computing the Input Frequency function is the problem of determining the ratio $a(v)/n$ for each node v in L_0.

View of a process. A *monotonic path* in the history tree is a sequence of nodes in distinct levels, such that any two consecutive nodes are connected by a black or a red edge. For any process p, let $h(p,t)$ be the unique node representing p in the level L_t of the history tree. We define the *view* of p at round t as the finite subgraph of the history tree induced by all the nodes spanned by monotonic paths with endpoints $h(p,t)$ and the root r.

A node of the history tree is *missing* from a view if it is not among the nodes in the view. A level of a view is *complete* if no node is missing from that level. Observe that, if a level of a view is complete, then all previous levels of the view are also complete.

3.2 Construction of Views

Intuitively, the view of a process represents the portion of the history tree that the process "knows" at that time. In fact, there is an effective procedure that allows all processes in a network to locally construct a representation of their view at all rounds. We will present this procedure and discuss some related results, including the *fundamental theorem of history trees*, Theorem 3.1.

Merge operation. The basic operation that is used to construct and update views is called *merge*, and is illustrated in Figure 2. Merging two views $\mathcal{V}$ and $\mathcal{V}'$ is a very natural operation whose result is the minimum graph that contains both $\mathcal{V}$ and $\mathcal{V}'$ as induced subgraphs.

Procedurally, the nodes of $\mathcal{V}'$ are matched and merged with the nodes of $\mathcal{V}$ level by level starting at the root r. That is, the procedure attempts to match the next vertex in level L'_t of $\mathcal{V}'$ with an equivalent vertex already in level L_t of $\mathcal{V}$. If no such vertex exists, it is created and connected with the appropriate vertices, which are already in the previous level. The procedure continues until all vertices of $\mathcal{V}'$ have been matched and merged.

Local construction. We will now describe a local algorithm that allows processes to construct and update their views at every round. That is, assuming that each process has its view as its internal state at the beginning of round t and sends its view to all its neighbors at round t, there is a local algorithm $\mathcal{A}^*$ that allows the process to construct its new view at the end of round t.

Constructing the view at round 0 is simple, because the process just has to read its own input and create a root r with a single child bearing a label equal to its input.

After that, the view can be updated at every round as shown in Figure 2. Namely, the node v that currently represents the process gets a new child v', representing the same process in the next round. Then, if $m \geq 1$ copies of a view $\mathcal{V}$ are received as messages, then $\mathcal{V}$ is merged into the current view, and a red edge with multiplicity m is added to connect v' with the deepest node of $\mathcal{V}$. The correctness of this operation follows straightforwardly from the definition of view.

Size of the view. We remark that the size of the view of a process at round t is polynomial in t and n, and so is the amount of local computation needed to update the process' view at round t based on the messages it has received. More specifically, the view can be represented in $O(tn^2 \log M)$ bits, where M is the maximum number of messages that any process may receive at any round (if the network is simple, then $M < n$).

Indeed, each of the $O(t)$ levels in the view contains up to n nodes. Thus, there are $O(n^2)$ possible red edges between any two consecutive levels, each of which has a multiplicity that can be represented in $O(\log M)$ bits.

Fundamental theorem. We will now give a concrete meaning to the idea that the view of a process at round t contains all the information that the process can use at that round. This intuition is made precise by the following *fundamental theorem of history trees.*

Theorem 3.1. *If all processes in a system execute the same local algorithm $\mathcal{A}$ at every round, then the internal state of a process p at the end of round t is determined by a function $\mathcal{F}_\mathcal{A}$ of the view of p at round t. The mapping $\mathcal{F}_\mathcal{A}$ depends entirely on the algorithm $\mathcal{A}$ and is independent of p.*

Proof. Denote by $h(p, t)$ the node of the history tree that represents p at round t, by $\mathcal{V}(p, t)$ the view of p at round t, and by $\sigma(p, t)$ the internal state of p at round t. The construction of $\mathcal{F}_\mathcal{A}$ is done by induction on t in such a way that $\mathcal{F}_\mathcal{A}(\mathcal{V}(p, t)) = \sigma(p, t)$ for any process p and any round t.

If $t = 0$, then $\mathcal{V}(p, t)$ consists of a single node $h(p, t)$ in L_0 attached to the root r. Since each node of L_0 is labeled as the input of the processes it represents, $\mathcal{F}_\mathcal{A}$ can extract the input of p from $h(p, t)$ and use it to compute its internal state $\sigma(p, t)$, which at round 0 is just a function of the input.

Let $t > 0$ and assume that the inductive hypothesis $\mathcal{F}_\mathcal{A}(\mathcal{V}(p, t-1)) = \sigma(q, t-1)$ holds for any process q. Let p be any process, and let us prove that $\mathcal{F}_\mathcal{A}(\mathcal{V}(p, t)) = \sigma(p, t)$. Observe that, by definition, if a view contains a node in level L_i, it also contains the view of the processes represented by that node at round i.

It is easy to infer $h(p, t)$ from $\mathcal{V}(p, t)$, because it is the unique node of maximum depth. Thus, the node $h(p, t-1)$ is the parent of $h(p, t)$, which is in $\mathcal{V}(p, t)$ because it is connected to $h(p, t)$ by a black edge. It follows that $\mathcal{F}_\mathcal{A}$ can extract from $\mathcal{V}(p, t)$ the view of p at round $t-1$ and, by the inductive hypothesis, it can compute $\sigma(p, t-1)$.

Also, for each process q that sends messages to p at round t, the node $h(q, t-1)$ is in $\mathcal{V}(p, t)$, because it is connected to $h(p, t)$ by a red edge with a certain multiplicity $m \geq 1$. Hence, $\mathcal{F}_\mathcal{A}$ can extract m from $\mathcal{V}(p, t)$, as well as the view of q at round $t-1$, and compute $\sigma(q, t-1)$ by the inductive hypothesis. From the internal state $\sigma(q, t-1)$, the message received by p from q at round t can also be computed.

Thus, $\sigma(p,t)$ can be computed by running the algorithm $\mathcal{A}$ with input $\sigma(p,t-1)$ (i.e., the previous internal state of p) and the messages received by p at round t with the appropriate multiplicities. Since we have shown how $\mathcal{F}_\mathcal{A}$ can extract all this information from $\mathcal{V}(p,t)$, the theorem is proved. □

As a consequence, all the processes represented by the same node in level L_t of the history tree must have the same state (and give the same output) at the end of round t, regardless of the deterministic algorithm being executed. This is in agreement with the idea that the processes represented by the same node are indistinguishable.

Corollary 3.2. *At the end of any round t, all processes represented by the same node in L_t have the same internal state.*

Proof. If two processes are represented by the same node of L_t, they have the same view at round t. Thus, by Theorem 3.1, they have the same internal state at the end of round t. □

Significance. The significance of the fundamental theorem is that it allows us to shift our focus from dynamic networks to history trees. Recall that there is a local algorithm $\mathcal{A}^*$ that allows processes to construct and update their view at every round. Now, Theorem 3.1 guarantees that processes do not lose any information if they simply execute $\mathcal{A}^*$, regardless of their goal, and then compute their task-dependent outputs as a function of their respective views. Thus, in the following, we will assume without loss of generality that the internal state of every process at every round, as well as all the messages it sends to all its neighbors, always coincide with its view at that round.

3.3 Related Concepts

We will now delve into related literature to place our history trees within the broader context of a line of research dating back to the 1980s.

Graph coverings. Borrowing from topological graph theory, Angluin was the first to use the notion of *graph coverings* to prove that some problems cannot be solved in certain anonymous static networks [1]. However, the conditions she stated were necessary but not sufficient.

Views of Yamashita–Kameda. Precise graph-theoretic characterizations of when certain fundamental problems for anonymous static networks are solvable were later given by Yamashita and Kameda [57]. The same authors also introduced the concept of *view* of a process in a static network whose links are endowed with *port numbers* (i.e., the links incident to the same process have distinct IDs) [56]. For Yamashita and Kameda, the view of a process p in a static network G is an infinite tree rooted at p that encodes all the information about the network that can possibly be gathered by p. In the language of topological graph theory, the view of p is akin to the *universal cover* of G (i.e., the "largest possible" cover of G) rooted at p.

The notion of "view" in the sense of Yamashita–Kameda should not be confused with the views of history trees introduced in Section 3.1.[19] However, the two concepts are loosely related, as one may construct the view of a process in a history tree at round t by inspecting the Yamashita–Kameda view of this process truncated at depth t, and vice versa.

If processes with isomorphic views are considered identical, one obtains the *quotient graph* in the sense of Yamashita–Kameda. In the language of topological graph theory, this can also be characterized as the "smallest possible" graph that is covered by G.

Minimum bases of Boldi–Vigna. The above concepts were later extended by Boldi and Vigna to static networks with directed links and no port numbers [9]. In particular, they recognized that

[19]We emphasize that, outside of Section 3.3, we will use the word "view" only in reference to history trees.

the appropriate topological tool for these networks is not the graph covering but the more general *graph fibration*. In the language of Boldi–Vigna, the *minimum base* $\widehat{G}$ of a static network G is a structure analogous to the "quotient graph" of Yamashita–Kameda.

It is straightforward to construct the minimum base (equivalently, the quotient graph) of a static network G given its history tree $\mathcal{H}$. In fact, the nodes in the level L_t of $\mathcal{H}$ correspond to the isomorphism classes of the Yamashita–Kameda views truncated at round t. It is easy to see that, if G is a static network, the number of nodes in the levels of $\mathcal{H}$ strictly increases at every level until it becomes maximum, say, at level L_s. Now, the nodes of the minimum base $\widehat{G}$ are precisely the nodes of L_s, and their edges are given by the red edges between L_s and L_{s+1} (essentially, the endpoint in L_{s+1} of each red edge is redirected to its parent in L_s).

In summary, previous structures and theories related to anonymous static networks can effectively be reinterpreted within the framework of history trees. The advantage of using history trees is that they also incorporate information about the time at which specific classes of processes become distinguishable. This "temporal dimension" makes history trees an ideal tool for dynamic networks, where one cannot rely on topological regularity to infer temporal information.

Algorithms for static networks. From a computational standpoint, it is important to consider how a process in a network can algorithmically construct the aforementioned data structures and use them effectively to solve certain problems.

We already pointed out that the views of Yamashita–Kameda contain all the information that can be gathered locally by the processes; the same can be said of the minimum bases of Boldi–Vigna. Hence, constructing these structures allows a process to access all the information that it can possibly use for computations.

Yamashita and Kameda showed that, at round t, any process can algorithmically construct its view truncated at depth t. Thus, it can incrementally construct its view level by level. Moreover, they showed that, in a static network of n processes, if two processes have isomorphic views up to depth n^2, they have isomorphic views at any depth [56]. This leads to the conclusion that, if the number n is known to the processes, they can effectively solve general problems and terminate in $O(n^2)$ communication rounds.

For example, since processes with isomorphic views are always indistinguishable, it is possible to elect a unique leader in a static network in a finite number of rounds if and only if there is a unique process whose view truncated at depth n^2 is distinct from all others.

Linear bounds. The n^2 upper bound was later improved by Norris, who showed that truncating views at depth $n-1$ is sufficient and may be necessary in some networks. That is, if two processes have isomorphic views up to depth $n-1$, they have isomorphic views at any depth [46].

Although this bound is worst-case optimal, it is far from being optimal in most cases. Later, Fraigniaud and Pelc further improved Norris' upper bound to $\widehat{n}-1$, where $\widehat{n}$ is the number of nodes in the minimum base $\widehat{G}$ of the network, and of course $\widehat{n} \leq n$ [27].

Proving this statement is immediate if one reframes it in terms of history trees. We have already argued that, in the history tree of a static network, the number of nodes in a level increases by at least one unit per level until level L_s, where it becomes maximum. That is, $|L_s| = \widehat{n}$, and clearly $s < \widehat{n}$, because $|L_0| \geq 1$. Thus, if two processes have isomorphic views (in the history tree) at round $\widehat{n}-1$, they have isomorphic views at all rounds, which is equivalent to the statement of Fraigniaud and Pelc.

Size of a view. When designing efficient algorithms, one also wishes to optimize memory usage and message sizes, as well as running times. Unfortunately, in the worst case, a view of Yamashita and Kameda truncated at depth t grows exponentially in t, making their approach unsuitable for systems where memory limitations are an issue.

For this reason, Tani proposed a method to efficiently compress a truncated view by identifying isomorphic subtrees within the view itself. Using Tani's technique, a compressed view (of an undirected network with port numbers) truncated at depth t can be stored in $O(tnM \log M)$ bits, where M is the maximum degree of a process in the network [54].

This essentially matches the upper bound on the size of a view of a history tree of a τ-union-connected network at round t, which is $O(tn^2 \log \tau M)$ bits (the small discrepancy is due to the fact that our networks are modeled by disconnected dynamic multigraphs, as opposed to connected static simple graphs). Thus, for instance, our terminating Counting algorithm based on history trees requires $O(n^3 \log \tau M)$ bits of memory per process in the worst case.

Lamport causality. In distributed systems, certain events are often caused by the occurrence of previous events; the concept of *causal influence* was first introduced by Lamport in his seminal paper [41].

A similar notion of causal influence can be formulated in the framework of history trees by stating that a process p exerts a causal influence on a process q at round t if the view of q at round t includes a node v representing p. It is important to note that not all processes represented by v necessarily initiate a sequence of messages reaching q by round t. However, at least one process represented by v does so, and the specific identity of this process is irrelevant, since processes are anonymous.

Causality in dynamic networks. The concept of causal influence was further developed by Kuhn et al., who studied it in the context of dynamic networks with unique IDs [38]. However, their findings apply equally well to anonymous dynamic networks.

Kuhn et al. established that, in a 1-union-connected dynamic network, the set of processes that are causally influenced by a given process grows by at least one unit at every round, until all n processes have been influenced. We can rephrase this observation in the language of history trees as follows.

- The view of any process at any round t contains nodes representing at least $\min\{t+1, n\}$ processes.
- For any process p and any round t, there are at least $\min\{t+1, n\}$ processes whose view at round t contains a node representing p.

Broadcasting speed. As a consequence of the above, broadcasting a message, i.e., forwarding a message from a single process to the whole dynamic network, takes at most $n-1$ rounds. Therefore, the *dynamic diameter* d, defined as the maximum number of rounds it takes to forward a message from a process to any other process at any point in time, satisfies $d \leq n-1$ in any 1-union-connected dynamic network.

Since these facts will be used in the design of our algorithms, we give self-contained proofs below.

Lemma 3.3. *Let P be a set of processes in a 1-union-connected dynamic network of size n, such that $1 \leq |P| \leq n-1$, and let $t \geq 0$. Then, at every round $t' \geq t + |P|$, in the history of every process there is a node at level L_t representing at least one process not in P.*

Proof. Let Q be the complement of P (note that Q is not empty), and let Q_{t+i} be the set of processes represented by the nodes in L_{t+i} whose view contains a node in L_t representing at least one process in Q. We will prove by induction that $|Q_{t+i}| \geq |Q| + i$ for all $0 \leq i \leq |P|$. The base case holds because $Q_t = Q$. The induction step is implied by $Q_{t+i} \subsetneq Q_{t+i+1}$, which holds for all $0 \leq i < |P|$ as long as $|Q_{t+i}| < n$. Indeed, because G_{t+i+1} is connected, it must contain a link

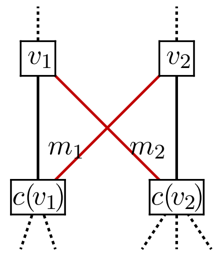

Figure 3: The nodes v_1 and v_2 are *exposed* with multiplicities m_1 and m_2, respectively.

between a process $p \in Q_{t+i}$ and a process $q \notin Q_{t+i}$. Thus, the history of q at round $t+i+1$ contains the history of p at round $t+i$, and so $Q_{t+i} \subsetneq Q_{t+i} \cup \{q\} \subseteq Q_{t+i+1}$.

Now, plugging $i := |P|$, we get $|Q_{t+|P|}| = n$. In other words, the history of each node in $L_{t+|P|}$ (and hence in subsequent levels) contains a node at level L_t representing a process in Q. □

Corollary 3.4. *In the history tree of a 1-union-connected dynamic network, every node at level L_t is in the view of every node at level $L_{t'}$, for all $t' \geq t+n-1$.*

Proof. Let $v \in L_t$, and let P be the set of processes not represented by v. If P is empty, then all nodes in $L_{t'}$ are descendants of v, and have v in their view. Otherwise, $1 \leq |P| \leq n-1$, and Lemma 3.3 implies that v is in the view of all nodes in $L_{t'}$. □

4 Leaderless and Stabilizing Algorithms

We will now present a general technique based on history trees, as well as three applications. Namely, we will give linear-time stabilizing and terminating algorithms for computing the Input Frequency function F_{IF} in leaderless networks, as well as a linear-time stabilizing algorithm for computing the Input Multiset function F_{IM} in networks with leaders. As we pointed out in Section 2.2, computing F_{IF} or F_{IM} is sufficient to compute the entire class of frequency-based functions or multiset-based functions, respectively, in the same number of rounds.

This basic technique, however, falls significantly short when it comes to developing terminating algorithms for computing F_{IM} in networks with leaders. This far more challenging problem will be discussed in Section 5.

Leveraging the theory developed in Section 3.2, we will assume, without any loss of generality, that all processes maintain their current view of the history tree as their internal state and broadcast this view across all available links at every round.

4.1 Basic Technique

Our basic technique makes use of the procedure in Listing 1, which will be a subroutine of all algorithms presented in this section. The purpose of this procedure is to construct a homogeneous system[20] 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.

Exposed nodes and strands. 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*. If v has multiple children, it is *branching* (thus, a leaf in a view is neither branching nor non-branching). We emphasize that, if v is a node of a view $\mathcal{V}$ of a history tree $\mathcal{H}$, then v may

[20]A linear system is *homogeneous* if all its constant terms are zero.

be branching in $\mathcal{H}$ but non-branching in $\mathcal{V}$. This happens if v has multiple children in $\mathcal{H}$, but only one of them appears in $\mathcal{V}$.

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 (see Figure 3). Again, the same two nodes may be exposed in a view of a history tree, but not in the history tree itself.

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 $v_2 \in P_2$.

Thanks to the following lemma, if we know the anonymity of a node in an exposed pair, we can determine the anonymity of the other node. (Recall that we denote the anonymity of v by $a(v)$.)

Lemma 4.1. *In a history tree $\mathcal{H}$, if the nodes v_1 and v_2 are exposed with multiplicity (m_1, m_2), then $a(v_1) \cdot m_1 = a(v_2) \cdot m_2$.*

Proof. Let $v_1, v_2 \in L_t$, and let P_1 and P_2 be the sets of processes represented by v_1 and v_2, respectively. Since v_1 is non-branching in $\mathcal{H}$, we have $a(c(v_1)) = a(v_1)$, and therefore $c(v_1)$ represents P_1, as well. Hence, the number of links between P_1 and P_2 in G_{t+1} (counted with their multiplicities) is $a(c(v_1)) \cdot m_1 = a(v_1) \cdot m_1$. By a symmetric argument, this number is equal to $a(v_2) \cdot m_2$. □

Observe that Lemma 4.1 may not hold in a view $\mathcal{V}$ of $\mathcal{H}$, because v_1 (or v_2) may be non-branching in $\mathcal{V}$ but have multiple children in $\mathcal{H}$. Thus, it is not necessarily true that v_1 (or v_2) and its unique child in $\mathcal{V}$ have the same anonymity.

Main subroutine. 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.[21] Note 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 $t \geq 0$.

Lemma 4.2. *Let $\mathcal{V}$ be the view of a process in a τ-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 $t' \geq t + \tau(n-1) + 1$, or*
2. *$t' \geq \tau(2n - c - 1) + 1$, where c is the number of distinct inputs held by processes at round 0.*

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

Proof. If $\tau = 1$, it takes at most $n - 1$ rounds for information to travel from a process to any other process, due to Corollary 3.4. In general, if $\tau \geq 1$, it takes at most $\tau(n-1)$ rounds, by Proposition 2.4. Therefore, since $\mathcal{V}$ is a view taken at round t', all levels of $\mathcal{V}$ up to $L_{t'-\tau(n-1)}$ are complete (recall that a level of a view is complete if it has no missing nodes).

Assume Condition 2 first. Since $t' \geq \tau(2n - c - 1) + 1$, all levels of $\mathcal{V}$ up to $L_{\tau(n-c)+1}$ are complete. Thus, level L_0 is complete, and therefore it has exactly c nodes, because two processes are

[21] The reason why we have to consider strands spanning several levels of the history tree (as opposed to looking at a single level) is that the dynamic disconnectivity τ is not known, and thus Proposition 2.4 cannot be applied directly.

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

```
1  # Input: a view $\mathcal{V}$ with levels $L_{-1}$, $L_0$, $L_1$, ..., $L_h$
2  # Output: $(t, S)$, where $t$ is an integer and $S$ is a system of linear equations
3
4  Assign $s := 0$
5  For $t := 0$ to $h$
6     If $L_t$ contains a node with no children, return $(-1, \emptyset)$
7     If $L_t$ contains a node with more than one child, assign $s := t + 1$
8     Else
9        Let $k = |L_s| = |L_t|$ and let $u_i$ be the $i$th node in $L_t$
10       Let $P_i$ be the strand starting in $L_s$ and ending in $u_i \in L_t$
11       Let $\mathcal{P} = \{P_1, P_2, \dots, P_k\}$
12       Let $G$ be the graph on $\mathcal{P}$ whose edges are pairs of exposed strands
13       If $G$ is connected
14          Let $G' \subseteq G$ be any spanning tree of $G$
15          Assign $S := \emptyset$
16          For each edge $\{P_i, P_j\}$ of $G'$
17             Find any two exposed nodes $v_1 \in P_i$ and $v_2 \in P_j$
18             Let $(m_1, m_2)$ be the multiplicity of the exposed pair $(v_1, v_2)$
19             Add the equation $m_1 x_i = m_2 x_j$ to $S$
20          Return $(t, S)$
```

distinguishable at round 0 if and only if they have distinct inputs. Since the sum of the anonymities of these c nodes is n, there must be fewer than $n - c$ branching nodes in $\mathcal{V}$ (excluding the root). Hence, the levels up to $L_{\tau(n-c)}$ contain an interval of at least τ consecutive levels, say from L_r to $L_{r+\tau-1}$, where all nodes are non-branching and can be partitioned into $|L_r| = |L_{r+\tau-1}|$ strands P_1, P_2, ..., $P_{|L_r|}$.

Note that a link between two processes at any round r' in the interval $[r+1, r+\tau]$ determines a pair of exposed nodes in $L_{r'-1}$. Thus, by definition of τ-union-connected network, the graph of exposed strands between L_r and $L_{r+\tau-1}$ (constructed as G in Line 12) is connected. It follows that the execution of Listing 1 terminates at Line 20 (as opposed to Line 6) whenever $t \geq r + \tau - 1$. Thus, the procedure returns a pair (t, S) with $0 \leq t \leq r + \tau - 1 \leq \tau(n - c)$. In particular, all levels of $\mathcal{V}$ up to L_{t+1} are complete.

Now assume Condition 1. Since $t' \geq t + \tau(n-1) + 1$, all levels of $\mathcal{V}$ up to L_{t+1} are complete in this case, as well. Since $t \geq 0$ by assumption, the execution of Listing 1 terminates at Line 20. The termination condition is met when long-enough strands are found; as proved above, this event must occur while $t \leq \tau(n - c)$.

We have proved that, in both cases, the inequalities $0 \leq t \leq \tau(n - c)$ hold, and all levels of $\mathcal{V}$ up to L_{t+1} are complete. Let us now examine the linear system S. Observe that S is homogeneous because it consists of homogeneous linear equations (cf. Line 19). Also, since the spanning tree G' constructed at Line 14 has $k - 1$ edges, S contains $k - 1$ equations. We will prove that they are linearly independent by induction on k. If $k = 1$, there is nothing to prove. Otherwise, let P_i be a leaf of G', and let $\{P_i, P_j\}$ be its incident edge. Then, S contains an equation Q of the form $m_1 x_i = m_2 x_j$ with $m_1 m_2 \neq 0$. Let S' be the system obtained by removing Q from S; equivalently, S' corresponds to the tree obtained by removing the leaf P_i from G'. By the inductive hypothesis, no linear combination of equations in S' yields $0 = 0$. On the other hand, if Q is involved in a linear combination with a non-zero coefficient, then the variable x_i cannot vanish, because it only appears in Q. Therefore, the equations in S are independent.

It remains to prove that a solution to S is given by the anonymities of the nodes of L_t. Due to Lemma 4.1, if v_1 and v_2 are exposed in $\mathcal{V}$, as well as in the history tree containing $\mathcal{V}$, with multiplicity (m_1, m_2), then $m_1 a(v_1) = m_2 a(v_2)$. To conclude our proof, it is sufficient to note that, since the nodes of a strand P_i are non-branching in $\mathcal{V}$ as well as in the underlying history tree (recall that all levels of $\mathcal{V}$ up to L_{t+1} are complete), they all have the same anonymity, which is the anonymity of the ending node $w_i \in L_t$. □

Diameter bounds. We can also restate Lemma 4.2 with respect to the dynamic diameter d of the network.

Corollary 4.3. *The statement of Lemma 4.2 remains valid if the two conditions are replaced with:*

1. $t \geq 0$ *and* $t' \geq t + d + 1$, *or*
2. $t' \geq \tau(n - c) + d + 1$, *where* c *is the number of distinct inputs held by processes at round* 0,

where d *is the dynamic diameter of the network (or an upper bound thereof).*

Proof. Note that it takes at most d rounds for information to travel across the network. Hence, the proof of Lemma 4.2 can be repeated verbatim, substituting the term $\tau(n-1)$ with d throughout. □

4.2 Stabilizing Algorithm for Leaderless Networks

As a first application of Lemma 4.2, we will give stabilizing algorithm that efficiently computes the Input Frequency function F_{IF} in all leaderless networks with a finite dynamic disconnectivity τ. As a consequence, *all* frequency-based functions are efficiently computable as well, due to Proposition 2.2. Moreover, Proposition 6.3 states that no other functions are computable in leaderless networks, and Proposition 6.9 shows that our algorithm is asymptotically optimal.

Theorem 4.4. *There is an algorithm that computes* F_{IF} *in all* τ*-union-connected anonymous networks with no leader and stabilizes in at most* $\tau(2n - 2) + 1$ *rounds, assuming no knowledge of* τ *or* n.

Proof. Our local algorithm is as follows. Run Listing 1 on the process' view $\mathcal{V}$, obtaining a pair (t, S). If $t = -1$ or S is not a homogeneous system of $k - 1$ independent linear equations in k variables, output "Unknown". Otherwise, since the rank of the coefficient matrix of S is $k - 1$, the general solution to S has exactly one free parameter, due to the Rouché–Capelli theorem. Therefore, by Gaussian elimination, it is possible to express every variable x_i as a rational multiple of x_1, i.e., $x_i = \alpha_i x_1$ for some $\alpha_i \in \mathbb{Q}^+$ (recall that the coefficients of S are integers). Let $L_t = \{w_1, w_2, \dots, w_k\}$ and $L_0 = \{v_1, v_2, \dots, v_{k'}\}$. For every node $v_i \in L_0$, define $\beta_i \in \mathbb{Q}^+$ as $\beta_i = \sum_{w_j \in L_t \text{ descendant of } v_i} \alpha_j$, and let $\beta = \sum_i \beta_i$. Then, output

$$\{(\text{label}(v_1), \beta_1/\beta), (\text{label}(v_2), \beta_2/\beta), \dots, (\text{label}(v_{k'}), \beta_{k'}/\beta)\}.$$

The correctness and stabilization time of the above algorithm directly follow from Lemma 4.2. Specifically, at any round $t' \geq \tau(2n - 2) + 1$, Condition 2 of Lemma 4.2 is met (since $c \geq 1$), and the system S is satisfied by the anonymities of the nodes in L_t. Thus, $a(v_i) = \alpha_i a(v_1)$ for all $v_i \in L_0$, and therefore $\beta_i/\beta = a(v_i)/n$. We conclude that, for any input assignment λ, the algorithm stabilizes on the correct output $\frac{1}{n} \cdot \mu_\lambda$ within $2\tau n$ rounds. □

4.3 Terminating Algorithm for Leaderless Networks

We will now give a certificate of correctness that can be used to turn the stabilizing algorithm of Section 4.2 into a terminating algorithm. The certificate relies on a-priori knowledge of the dynamic disconnectivity τ and an upper bound N on the size of the network n; these assumptions are justified by Proposition 2.3 and Proposition 6.4, respectively. Again, Proposition 6.9 shows that our algorithm is asymptotically optimal.

Theorem 4.5. *There is an algorithm that computes F_{IF} in all τ-union-connected anonymous networks with no leader and terminates in at most $\tau(n + N - 2) + 1$ rounds, assuming that τ and an upper bound $N \geq n$ are known to all processes.*

Proof. Assume that τ and N are known. Our terminating local algorithm is as follows. Run Listing 1 on the process' view $\mathcal{V}$, obtaining a pair (t, S), and then do the same computations as in the algorithm for Theorem 4.4. If $t \geq 0$ and the current round t' satisfies $t' \geq t + \tau(N - 1) + 1$, then the output is correct, and the process terminates.

The correctness of this algorithm is a direct consequence of Lemma 4.2. Indeed, the algorithm only terminates when $t \geq 0$ and $t' \geq t + \tau(N - 1) + 1 \geq t + \tau(n - 1) + 1$, and hence it gives the correct output because Condition 1 of Lemma 4.2 is met. As for the running time, assume that $t' = \tau(n + N - 2) + 1 \geq \tau(2n - c - 1) + 1$. Since Condition 2 of Lemma 4.2 is met, we have $0 \leq t \leq \tau(n - c) \leq \tau(n - 1)$. Thus, $t' = \tau(n + N - 2) + 1 \geq t + \tau(N - 1) + 1$, and the algorithm terminates at round t'. □

We can also trade the knowledge of τ and N for the knowledge of the dynamic diameter d of the network.

Corollary 4.6. *There is an algorithm that computes F_{IF} in all τ-union-connected anonymous networks with no leader and terminates in at most $\tau(n - 1) + d + 1 \leq dn + 1$ rounds, assuming that the dynamic diameter (or an upper bound thereof) d is known to all processes, but assuming no knowledge of τ or n.*

Proof. We can repeat the proof of Theorem 4.5 verbatim, replacing the termination condition with $t' \geq t + d + 1$ and invoking Corollary 4.3 in lieu of Lemma 4.2. □

4.4 Stabilizing Algorithm for Networks with Leaders

To conclude this section, we will give a stabilizing algorithm that efficiently computes the Input Multiset function F_{IM} in all networks with $\ell \geq 1$ leaders and a finite dynamic disconnectivity τ. Therefore, *all* multiset-based functions are efficiently computable as well, due to Proposition 2.1. Moreover, Proposition 6.1 states that no other functions are computable in networks with leaders, and Proposition 6.8 shows that our algorithm is asymptotically optimal. 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 6.2.

Theorem 4.7. *There is an algorithm that computes F_{IM} in all τ-union-connected anonymous networks with $\ell \geq 1$ leaders and stabilizes in at most $\tau(2n - 3) + 1$ rounds, assuming that ℓ is known to all processes, but assuming no knowledge of τ or n.*

Proof. The algorithm proceeds as in Theorem 4.4, with two differences. First, instead of returning "Unknown", the new algorithm returns ℓ. Second, when the fractions β_1, β_2, ..., $\beta_{k'}$ have been computed, as well as their sum β, we perform the following additional steps. Let $L_0 = \{v_1, v_2, \ldots, v_{k'}\}$, and let $\{v_{j_1}, v_{j_2}, \ldots, v_{j_l}\} \subseteq L_0$ be the set of nodes in L_0 representing leader processes,[22] i.e., such

[22] In general we have $l \leq \ell$, because some nodes of L_0 may represent more than one leader.

that label(v_{j_i}) has the leader flag set for all $1 \le i \le l$. Compute $\beta' = \sum_{i=1}^{l} \beta_{j_i}$ and $\gamma_i = \ell\beta_i/\beta'$ for all $1 \le i \le k'$, and output

$$\{(\text{label}(v_1), \gamma_1), (\text{label}(v_2), \gamma_2), \ldots, (\text{label}(v_{k'}), \gamma_{k'})\}.$$

The correctness follows from the fact that, as shown in Theorem 4.4, at any round $t' \ge \tau(2n - c - 1) + 1$ we have $\beta_i/\beta = a(v_i)/n$ for all $1 \le i \le k'$. Adding up these equations for all $i \in \{j_1, j_2, \ldots, j_l\}$, we obtain $\beta'/\beta = \ell/n$, and therefore $n = \ell\beta/\beta'$. We conclude that

$$\gamma_i = \frac{\ell\beta_i}{\beta'} = \frac{\ell\beta\beta_i}{\beta'\beta} = \frac{n\beta_i}{\beta} = a(v_i).$$

Thus, within $\tau(2n - c - 1) + 1$ rounds, the algorithm stably outputs the anonymities of all nodes in L_0. As observed in Section 2, this is equivalent to computing the Input Multiset function F_{IM}.

If there are both leader and non-leader processes in the system, there are $c \ge 2$ distinct inputs, and therefore the stabilization time is at most $\tau(2n - 3) + 1$ rounds, as desired. In the special case where all processes are leaders, we have $\ell = n$, which is the algorithm's output at every round since round 0 (recall that, by design, the algorithm returns ℓ instead of "Unknown"). Thus, the stabilization time is $\tau(2n - 3) + 1$ rounds in any case. □

5 Terminating Algorithm for Networks with Leaders

We will now present the main result of this paper. As already remarked, giving an efficient certificate of correctness for the Input Multiset function with one or more leaders is a highly non-trivial task for which a radically new approach is required. In fact, there are two crucial difficulties to overcome.

Firstly, the strategies developed in Section 4 are too shallow and ineffective even for networks with a unique leader, which are the easiest to treat. In Section 5.1, we will give counterexamples to some naive termination strategies that one may devise in an attempt to generalize those in Section 4. This indicates that an entirely different technique is necessary.

Secondly, networks with more than one leader significantly add to the difficulty of the problem. For instance, once a terminating algorithm for networks with a unique leader has been designed, one may be tempted to simply adapt it to the multi-leader case by setting the anonymity of the leader node in the history tree to $\ell > 1$ instead of 1. However, this approach would be inconsistent. Indeed, while the history tree of a network with a unique leader contains a single leader branch, all of whose nodes have anonymity 1, this may not be the case for networks with multiple leaders. In such a network, as soon as some leaders get disambiguated, the leader node branches into several children nodes whose anonymities are unknown (we only know that their sum is ℓ). Moreover, some leader nodes may be missing from the view of other leaders. Thus, in the case of multi-leader networks, we must deal with the fact that even the view of a leader may have levels where the sum of the anonymities of any subset of nodes is unknown.

Section 5.3 contains the technical core of our algorithm. Here we develop a subroutine that, in $O(\tau\ell n)$ rounds, counts the number of processes in a network assuming that its history tree contains a leader node of known anonymity whose descendants are non-branching for sufficiently many rounds. In particular, in the case of networks with a unique leader, this subroutine yields a full-fledged linear-time terminating algorithm for the Counting problem (because in this case the leader nodes are necessarily non-branching).

In Section 5.2, we approach the general multi-leader problem by repeatedly guessing the anonymity of a leader node in the history tree and invoking the subroutine of Section 5.3 to confirm

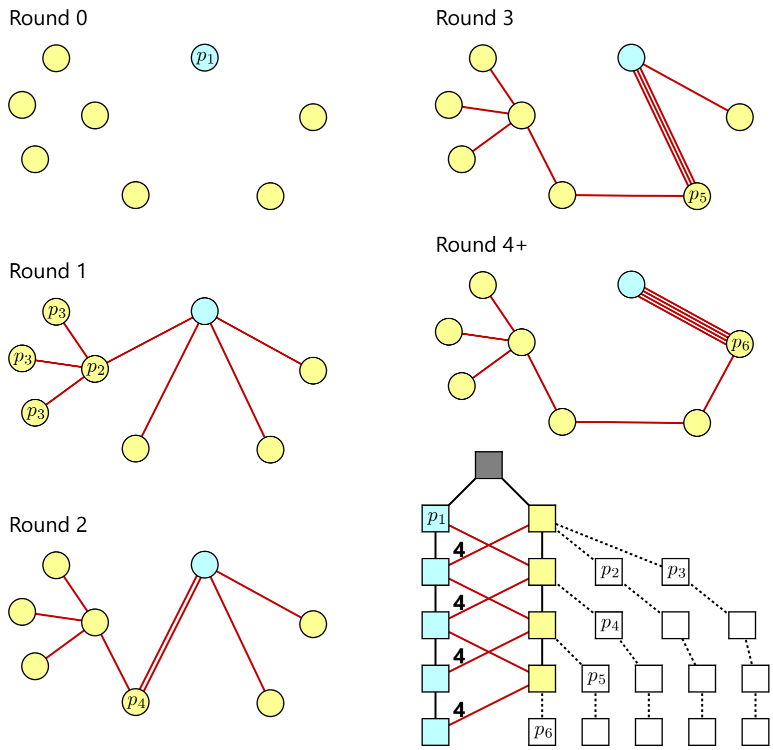

Figure 4: An example of a dynamic network where the naive technique of Section 4.4 fails to provide a correct termination condition. The white nodes in the history tree are not in the history of the leader at the last round; the red edges not in the view are not drawn. Same-colored processes have equal inputs. For the first four rounds, from the leader's perspective, this network is indistinguishable from the complete bipartite graph $K_{1,4}$. Thus, throughout this time, the leader is unaware of the processes labeled p_3, and therefore cannot compute the total number of processes.

our guesses. This procedure introduces a further bottleneck that causes our final algorithm to have a running time of $O(\tau\ell^2 n)$ rounds.

5.1 Naive Termination Strategies Are Incorrect

If we were to use a technique such as the one in Section 4.4 to devise a terminating algorithm for the Counting problem (or, more generally, for computing the Input Multiset function), our attempts would be bound to fail.

Basic naive strategy. As a first example, consider the dynamic network in Figure 4. Observe that the leader of this network always receives exactly four messages from indistinguishable non-leader processes. In turn, the processes that send messages to the leader in the first four rounds are still unaware of the processes labeled p_3. As a result, the view of the leader up to round 4 consists of only two strands whose nodes are exposed with multiplicity $(4, 1)$ at every level. Thus, according to the algorithm in Section 4.4, the leader assigns an anonymity of 4 to all the non-leader nodes in its view, concluding that there are only five processes in the network. Essentially, for the first four rounds, the leader cannot distinguish this network from a static star graph with the leader at the center.

It is straightforward to generalize this example to networks with $n = k_1 + k_2 + 1$ processes, $k_1 + 1$ of which are counted by the leader, while the other k_2 are not discovered by the leader until round $k_1 + 1$ (such as the processes labeled p_3 in Figure 4). In these networks, our naive algorithm consistently returns $k_1 + 1$ for several rounds, which can be made arbitrarily far from the true value $k_1 + k_2 + 1$ by increasing k_2 indefinitely.

Improved naive strategy. One may conjecture that a good termination certificate would be to compute the number of processes n' according to the algorithm in Section 4.4, and then wait for a number of rounds depending on n' to confirm that no relevant nodes were missing from the view. In fact, in the previous example, simply waiting for n' rounds would suffice.

Unfortunately, this strategy fails on the network in Figure 5. Here the leader's view at levels L_0 and L_1 causes the algorithm to count only $n' = 5$ processes (i.e., one leader, two processes represented by the purple node, and one process represented by the yellow node). Afterwards, the leader has to wait until round $k - 5$ for the appearance of the node that was missing from L_1. Since k is arbitrary, this type of strategy is bound to fail no matter what the waiting time is.

Challenges. Essentially, the recurring issue is that a process seems to have no way of knowing whether any level in its view of the history tree is complete; thus, it may end up terminating too soon with an incorrect output. To establish a correct termination condition, we will have to considerably develop the theory of history trees. This will be undertaken in Sections 5.2 and 5.3.

5.2 Main Algorithm

The subroutine `ApproxCount`. We first introduce the subroutine `ApproxCount`, whose formal description and proof of correctness are postponed to Section 5.3. 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.

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

Conditional anonymities. `ApproxCount` starts by assuming that the anonymity of ϑ 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 $a'(\vartheta) = x = \delta a(\vartheta)$.

Overview of `ApproxCount`. The procedure `ApproxCount` scans the levels of $\mathcal{V}$ starting from L_s, making "guesses" on the conditional anonymities of nodes based on already known conditional anonymities. Generalizing some lemmas from [23], we develop a criterion to determine when a guess is correct. This yields more nodes with known conditional anonymities, and therefore more guesses (the details are in Section 5.3). As soon as it has obtained enough information, the procedure stops and returns (n', t), where L_t is the level scanned thus far. If the information gathered satisfies certain criteria, then n' is an approximation of n. Otherwise, n' is an error code, as detailed below.

Error codes. If L_s contains no leader nodes, the procedure returns the error code $n' = -1$. If, before gathering enough information on n, the procedure encounters a descendant of ϑ with more than one child in $\mathcal{V}$, it returns the error code $n' = -2$. If it determines that the conditional anonymity of a node is not an integer, it returns the error code $n' = -3$. Finally, if it determines that the sum ℓ' of the conditional anonymities of the leader nodes is not ℓ, it returns $n' = -1$ if $\ell' < \ell$ and $n' = -3$ if $\ell' > \ell$.

Correctness of `ApproxCount`. The following lemma gives some conditions that guarantee that `ApproxCount` has the expected behavior; it also gives bounds on the number of rounds it takes for

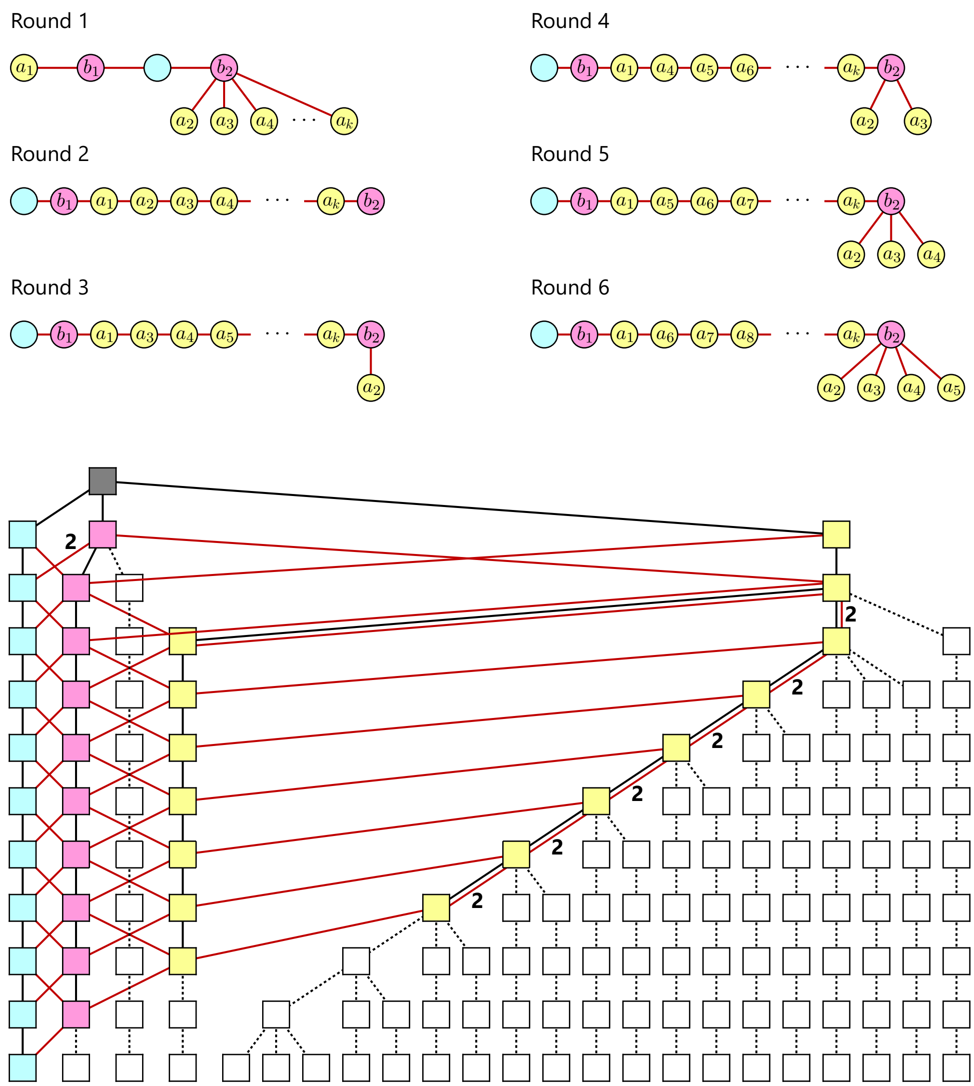

Figure 5: A dynamic network where, after level L_1, all levels in the leader's view are identical for an arbitrarily long sequence of rounds (depending on the parameter k). The color schemes and stylistic conventions are as in Figure 4.

Listing 2: Solving the Counting problem with $\ell \geq 1$ leaders.

```
1  # Input: a view $\mathcal{V}$ and a positive integer $\ell$
2  # Output: either a positive integer $n$ or "Unknown"
3
4  Assign $n^* := -1$ and $s := 0$ and $c := 0$
5  Let $b$ be the number of leader branches in $\mathcal{V}$
6  While $c \leq \ell - b$
7     Assign $t^* := -1$
8     For $x := \ell$ downto 1
9        Assign $(n', t) :=$ ApproxCount($\mathcal{V}, s, x, \ell$)      # see Listing 3 in Section 5.3
10       Assign $t^* := \max\{t^*, t\}$
11       If $n' = -1$, return "Unknown"
12       If $n' = -2$, break out of the for loop
13       If $n' > 0$
14          If $n^* = -1$, assign $n^* := n'$
15          Else if $n^* \neq n'$, return "Unknown"
16          Assign $c := c + 1$ and break out of the for loop
17    Assign $s := t^* + 1$
18 Let $L_{t'}$ be the last level of $\mathcal{V}$
19 If $t' \geq t^* + n^*$, return $n^*$
20 Else return "Unknown"
```

`ApproxCount` to produce an approximation n' of n, as well as a criterion to determine if $n' = n$. The lemma's proof is rather lengthy and technical, and is found in Section 5.3 (cf. Lemma 5.9).

Lemma 5.1. *Let* `ApproxCount`$(\mathcal{V}, s, x, \ell)$ *return* (n', t)*. Assume that* ϑ *exists and* $x \geq a(\vartheta)$*. Let* ϑ' *be the (unique) descendant of* ϑ *in* $\mathcal{V}$ *at level* L_t*, and let* $L_{t'}$ *be the last level of* $\mathcal{V}$*. Then:*

(i) If $x = a(\vartheta) = a(\vartheta')$*, then* $n' \neq -3$*.*

(ii) If $n' > 0$ *and* $t' \geq t + n'$ *and* $a(\vartheta) = a(\vartheta')$*, 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* $\mathcal{V}$*.* □

Our terminating algorithm assumes that all processes know the number of leaders $\ell \geq 1$ and the dynamic disconnectivity τ. Again, this is justified by Proposition 6.2 and Proposition 2.3.

Theorem 5.2. *There is an algorithm that computes* F_{IM} *in all* τ*-union-connected anonymous networks with* $\ell \geq 1$ *leaders and terminates in at most* $\tau((\ell^2 + \ell + 1)(n - 1) + 1)$ *rounds, assuming that* ℓ *and* τ *are known to all processes, but assuming no knowledge of* n*.*

Proof. Due to Proposition 2.4, since τ is known and appears as a factor in the claimed running time, we can assume that $\tau = 1$ without loss of generality. Also, note that determining n is enough to compute F_{IM}. Indeed, if a process determines n at round t', it can wait until round $\max\{t', \tau(2n-2)\}$ and run the algorithm in Theorem 4.7, which is guaranteed to give the correct output by that time.

In order to determine n assuming that $\tau = 1$, we let each process run the algorithm in Listing 2 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 $(\ell^2 + \ell + 1)(n - 1) + 1$ 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 $\vartheta \in L_s$ 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 ϑ 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 ϑ (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 $n^* = n$ is accepted as correct and returned; otherwise "Unknown" is returned (Lines 18–20).

Correctness and running time. We will prove that, if the output of Listing 2 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 5.1, this implies that $a(\vartheta) = a(\vartheta')$ whenever `ApproxCount` is called in $\mathcal{L}_j$. Also, since the option $x = \ell$ is tried first, the assumption $x \geq a(\vartheta)$ of Lemma 5.1 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 Lemma 5.1 (ii) and by the terminating condition (Line 19), if $n' > 0$ while $x \geq a(\vartheta)$, then $n^* = n' = n$. On the other hand, due to Lemma 5.1 (i), by the time $x = a(\vartheta)$ we necessarily have $n' \neq -3$ and therefore $n' > 0$.

It remains to prove that Listing 2 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 $t' = (\ell^2 + \ell + 1)(n - 1) + 1$. By Corollary 3.4, all nodes in the first $t' - n + 1 = \ell(\ell + 1)(n - 1) + 1$ levels of the history tree are contained in the view $\mathcal{V}$ at round t'. It is straightforward to prove by induction that the assumption $t' \geq s + (\ell + 2)(n - 1)$ of Lemma 5.1 (iii) holds every time `ApproxCount` is invoked. Indeed, as long as this condition is satisfied, Lemma 5.1 (iii) implies that $n' \neq -1$, and so "Unknown" is not returned at Line 11. Also, reasoning as in the previous paragraph, we infer that $n' \neq -3$ by the time $x = a(\vartheta)$. Thus, within $\Delta = (\ell + 1)(n - 1)$ levels, either a branching leader node is found ($n' = -2$) or a new approximation of n is computed ($n' > 0$). Every time either event occurs, s is increased by at most Δ at Line 17. Thus, after $\ell - 1$ (or fewer) updates of s, we have $s \leq (\ell - 1)\Delta = t' - (\ell + 2)(n - 1) - 1$. Hence the condition of Lemma 5.1 (iii) holds again, and the induction goes through for at least ℓ steps.

Observe that, since there can be at most $\ell - 1$ branching leader nodes in $\mathcal{V}$, at least one approximation $n' > 0$ of n is computed within the ℓth iteration of the loop at Lines 6–17. This occurs within $t^* \le \ell\Delta = t' - n$ levels. Because all nodes in these levels must appear in $\mathcal{V}$, the condition $a(\vartheta) = a(\vartheta')$ is satisfied in all intervals $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_{\ell-b+1}$. Due to Lemma 5.1 (ii), we conclude that all such intervals must yield the correct approximation $n' = n$. So, every time Line 15 is executed, we have $n^* = n'$, and the algorithm does not return "Unknown". Finally, when Line 19 is reached, we have $t^* \le t' - n = t' - n^*$, and therefore "Unknown" is not returned. Thus, an output other than "Unknown" is returned within the desired number of rounds. □

5.3 Subroutine `ApproxCount`

We will now define the subroutine `ApproxCount`$(\mathcal{V}, s, x, \ell)$ introduced in Section 5.2 and invoked in Listing 2. We will also give a proof to the technical Lemma 5.1 (restated as Lemma 5.9).

Overview. In `ApproxCount`, we are given a view $\mathcal{V}$ with a strand of leader nodes hanging from the first leader node ϑ in level L_s, where the anonymity $a(\vartheta)$ is an unknown number not greater than ℓ. The algorithm begins by assuming that $a(\vartheta)$ 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 ϑ, it reports failure if some descendants of ϑ (in the relevant levels of $\mathcal{V}$) have more than one child.

An 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, and all branches except one may be missing from $\mathcal{V}$.

We remark that `ApproxCount` assumes that the network is 1-union-connected, as this is sufficient for the main result of Section 5.2 to hold for any τ-union-connected network (see the proof of Theorem 5.2).

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

Conditional anonymity. `ApproxCount` starts by assuming that the anonymity of ϑ 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 $a'(\vartheta) = x = \delta a(\vartheta)$.

Guessing conditional anonymities. Let u be a node of a history tree, and assume that the conditional anonymities of all its children $u_1, u_2, \ldots, u_k$ have been computed: such a node u is called a *guesser*. If v is not among the children of u but is at their same level, and the red edge $\{v, u\}$ is present with multiplicity $m \ge 1$, we say that v is *guessable* by u (see Figure 6). 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 $1 \le i \le k$, 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 5.3. *If v is guessable, then $g(v) \ge a'(v)$. Moreover, if v has no siblings, $g(v) = a'(v)$.*

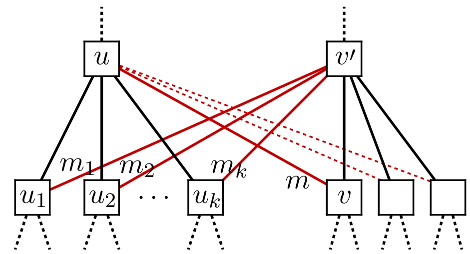

Figure 6: If the anonymities of u, u_1, u_2, …, u_k are known, then v is *guessable* by u.

Proof. Let $u, v' \in L_t$, and let P_1 and P_2 be the sets of processes represented by u and v', respectively. By counting the links between P_1 and P_2 in G_{t+1} in two ways, we have

$$\sum_i a(u_i)\, m_i = \sum_i a(v_i)\, m'_i \geq a(v)\, m,$$

where the two sums range over all children of u and v', respectively (note that $v = v_j$ for some j), and m'_i is the multiplicity of the red edge $\{v_i, u\}$ (so, $m = m'_j$). Therefore, we have the inequality

$$a(v) \leq \frac{\sum_i a(u_i)\, m_i}{m}$$

which becomes an equality if v has no siblings. Thus,

$$a'(v) = \delta a(v) \leq \frac{\sum_i \delta a(u_i)\, m_i}{m} = \frac{\sum_i a'(u_i)\, m_i}{m} = g(v)$$

and so $g(v) \geq a'(v)$, with equality if v has no siblings. □

Heavy nodes. The subroutine `ApproxCount` assigns guesses in a *well-spread* fashion, that is, in such a way that no two sibling nodes are assigned a guess. In other words, at most one of the children of each node 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 5.4. *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).*

Proof. Our proof is by well-founded induction on $w(v)$. Assume for a contradiction that no descendants of v are heavy. Let v_1, v_2, …, v_k be the "immediate" descendants of v that have been assigned guesses. That is, for all $1 \leq i \leq k$, no internal nodes of the black path with endpoints v and v_i have been assigned guesses. Observe that $k \geq 1$ because, by assumption, $w(v) > a'(v) = \delta a(v) \geq a(v) \geq 1$.

By the basic properties of history trees, $a(v) \geq \sum_i a(v_i)$, and therefore $a'(v) \geq \sum_i a'(v_i)$. Also, the induction hypothesis implies that $w(v_i) \leq a'(v_i)$ for all $1 \leq i \leq k$, or else one of the v_i's would have a heavy descendant. Therefore,

$$w(v) - 1 = \sum_i w(v_i) \leq \sum_i a'(v_i) \leq a'(v) < w(v).$$

Observe that all the terms in this chain of inequalities are between the two consecutive integers $w(v) - 1$ and $w(v)$. It follows that

$$w(v_i) \le a'(v_i) < w(v_i) + 1$$

for all $1 \le i \le k$. Also,

$$a'(v) - 1 < \sum_i a'(v_i) \le a'(v).$$

However, since every conditional anonymity is an integer multiple of the discrepancy $\delta \ge 1$, we conclude that $a'(v) = \sum_i a'(v_i)$. Hence, $a(v) = \sum_i a(v_i)$.

Let $1 \le d \le k$ be such that v_d has maximum depth. Since the assignment of guesses is well spread, no sibling of v_d has been assigned a guess. However, since $a(v) = \sum_i a(v_i)$, it follows that v_d has no siblings at all, for otherwise $a(v) > \sum_i a(v_i)$. Due to Lemma 5.3, we have $g(v_d) = a'(v_d)$. Thus,

$$w(v_d) \le a'(v_d) = g(v_d) < w(v_d) + 1,$$

which implies that $\lfloor g(v_d) \rfloor = w(v_d)$, and so v_d is heavy. □

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.

Lemma 5.5. *Assume that $\delta \ge 1$. 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.*

Proof. If $g(v)$ is not an integer, there is nothing to prove. Otherwise, because v is heavy, $g(v) = \lfloor g(v) \rfloor \le w(v)$. Since v has no heavy descendants, Lemma 5.4 implies $w(v) \le a'(v)$. Also, by Lemma 5.3, $a'(v) \le g(v)$. We conclude that

$$g(v) \le w(v) \le a'(v) \le g(v).$$

Therefore $g(v) = a'(v)$, and v has a correct guess. □

When the criterion in Lemma 5.5 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.

If s is an isle's root and F is its set of leaves, we have $a(s) \ge \sum_{v \in F} a(v)$, which is equivalent to $a'(s) \ge \sum_{v \in F} a'(v)$. Note that equality does not necessarily hold, because s may have some descendants in the history tree $\mathcal{H}$ that do not appear in the view $\mathcal{V}$. If equality holds, then the isle is said to be *complete.* In this case, given the conditional anonymities of s and of all nodes in F, 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.

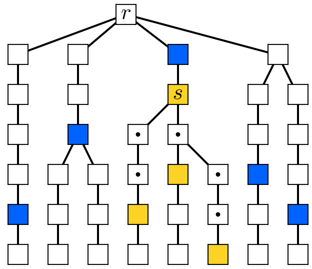

Figure 7: The first levels of (a view of) a history tree, where the colored nodes are counted. The blue nodes form a *counting cut*, and the orange ones define a non-trivial *isle* with root s, where the nodes with a dot are internal.

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

```
# 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 $\vartheta$ be the first leader node in $L_s$
Mark all nodes in $\mathcal{V}$ as not guessed, not counted, and not locked
Assign $u := \vartheta$; 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, non-counted, non-locked nodes in $\mathcal{V}$ and a counting cut has not been found
   Let $v$ be a guessable, non-counted, non-locked 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; lock $v$ and all its siblings
   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; unlock $v'$ and all its siblings
      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, not guessed, and not locked
If no counting cut has been found, return $(-2, t)$
Else
   Let $C$ be a counting cut (between $L_s$ and $L_t$)
   Assign $n' := \sum_{v \in C} a'(v)$
   Assign $\ell' := \sum_{v \text{ leader node in } C} a'(v)$
   If $\ell' < \ell$, return $(-1, t)$
   Else if $\ell' > \ell$, return $(-3, t)$
   Else return $(n', t)$
```

Overview of ApproxCount. Our subroutine `ApproxCount` is found in Listing 3. It repeatedly assigns guesses to nodes based on known conditional anonymities, starting from ϑ and its descendants. Eventually some nodes become heavy, and the criterion in Lemma 5.5 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", "counted", or "locked"; 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 node of $\mathcal{V}$ is guessed, all its siblings (including itself) become *locked*. While a node is locked, it cannot receive guesses. As defined earlier, a node v in level L_t of $\mathcal{V}$ is *guessable* if 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 $a'(\vartheta) = x$ to the first leader node $\vartheta \in L_s$. If no leader node exists in L_s, the error code -1 is immediately returned (Line 6). The algorithm also finds the longest strand P_ϑ hanging from ϑ, assigns the same conditional anonymity x to all of its nodes (including the unique child of the last node of P_ϑ) and marks them as counted (Lines 7–11). Then, as long as there are nodes that can receive a guess and no counting cut has been found, the algorithm keeps assigning guesses to nodes. A node can receive a guess if it is guessable, not counted, and not locked (Line 12).

When a guess is made on a node v, the node itself and all of its siblings become locked (Line 15). This is to ensure that guessed nodes will always be well spread. Moreover, as a result of v becoming guessed, some nodes in the path from v to its ancestor in L_s may become heavy; if this happens, let v' be the deepest heavy node (Line 18). 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_ϑ have children that are not in the view $\mathcal{V}$.) Otherwise, if $g(v')$ is an integer, the algorithm marks v' as counted and not guessed, in accordance with Lemma 5.5. Also, since v' is no longer guessed, its siblings become unlocked and are again eligible to receive guesses (Line 20). Furthermore, if the newly counted node v' is either 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 (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 ϑ 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 ϑ has a known conditional anonymity (by definition, $a'(\vartheta) = x$), the same is not necessarily true of the descendants of ϑ 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 ϑ has a (unique) descendant $\vartheta' \in L_{t'}$ and $a(\vartheta) = a(\vartheta')$.*

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 the job of our main Counting algorithm in Section 5.2 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 5.6. *Assume that $\delta \geq 1$. Then, as long as Condition 1 is satisfied, the following statements 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.

Proof. Statement (i) is true by design with no additional assumptions, because the algorithm only makes new guesses on unlocked nodes. In turn, a node is locked if and only if it is marked as guessed or has a sibling that is marked as guessed. Thus, no two nodes marked as guessed can be siblings, e.g., guesses are well spread.

All other statements can be proved collectively by induction. They certainly hold the first time Line 13 is ever reached. Indeed, the only nodes marked as counted up to this point are ϑ and some of its descendants, which are assigned the conditional anonymity x. Since $s = t$ and ϑ has conditional anonymity x by definition, statement (iv) is satisfied. Note that some descendants of ϑ that are marked as counted may not have been assigned their correct conditional anonymities, because some branches of the history tree may not appear in $\mathcal{V}$. However, no guesses have been made yet, and therefore no nodes are heavy; thus, statement (ii) is satisfied. Moreover, the only isles are formed by ϑ and its descendants, and are obviously all trivial; so, statement (iii) is satisfied.

Now assume that statements (ii), (iii), and (iv) are all satisfied up to some point in the execution of the algorithm. In particular, due to statement (iv), all nodes that have been identified as guessers by the algorithm up to this point were in fact guessers according to our definitions. For this reason, all guesses have been computed as expected, and all of our lemmas on guesses apply (because $\delta \geq 1$).

The next guess on a new node v is performed properly, as well. Indeed, Condition 1 states that ϑ has a descendant ϑ' at the same level as v such that $a(\vartheta') = a(\vartheta)$, and therefore $a'(\vartheta') = a'(\vartheta) = x$; so, ϑ' has the correct conditional anonymity. Thus, regardless of what the guesser of v is (either the parent of ϑ' or some other counted node), the guess at Line 15 is computed properly.

Hence, if a node is identified as heavy at Lines 17–18, it is indeed heavy according to our definitions. Because statement (ii) held before making the guess on v, it follows that any heavy node must have been created after the guess, and therefore should be on the path P_v, defined as in Line 16. If no heavy nodes are found on the path, then nothing is done and statements (ii), (iii), and (iv) keep being true.

Otherwise, by Lemma 5.5, the deepest heavy node v' on P_v has a correct guess and can be marked as counted, provided that the guess is an integer. Thus, statement (iv) is still true after Line 20. At this point, there are no heavy nodes left, because v' is no longer guessed and all of its ancestors along P_v end up having the same weight they had before the guess on v was made.

Now, because statement (iii) held before marking v' as counted, there can be at most one non-trivial complete isle, and v' must be its root or one of its leaves. Note that, due to statement (iv), any isle I identified as complete at Line 21 is indeed complete according to our definitions. Since I is complete, computing the conditional anonymities of its internal nodes as in Line 23 is correct, and therefore statement (iv) is still true after Line 24. Also, the unique non-trivial isle I is reduced to trivial isles, and statement (iii) holds again. Finally, since Lines 21–24 may only cause weights to decrease, statement (ii) keeps being true. □

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.

Recall that a node v of the history tree $\mathcal{H}$ is said to be 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 have no missing nodes, either.

Lemma 5.7. *Assume that $\delta \geq 1$. Then, as long as Condition 1 holds, whenever Line 13 is reached, at most $\delta(n-1)$ levels contain locked nodes.*

Proof. Note that the assumptions of Lemma 5.6 are satisfied, and therefore all the conditional anonymities and weights assigned to nodes up to this point are correct according to our definitions.

We will begin by proving that, if the subtree hanging from a node v of $\mathcal{V}$ contains more than $a'(v)$ nodes marked as guessed, then it contains a node v' marked as guessed such that $w(v') > a'(v')$. The proof is by well-founded induction based on the subtree relation in $\mathcal{V}$. If v is guessed, then we can take $v' = v$, for in this case $w(v) > a'(v)$. Otherwise, by the pigeonhole principle, v has at least one child u whose hanging subtree contains more than $a'(u)$ guessed nodes. Thus, v' is found in this subtree by the induction hypothesis.

Now, assume for a contradiction that more than $\delta(n-1)$ levels of $\mathcal{V}$ contain locked nodes; in particular, $\mathcal{V}$ contains more than $\delta(n-1)$ nodes marked as guessed. Consider the nodes in level L_s other than ϑ; the sum of their anonymities is at most $n - a(\vartheta)$ (note that some nodes may be missing from L_s), and so the sum of their conditional anonymities is at most $\delta(n - a(\vartheta)) \leq \delta(n-1)$. Thus, by the pigeonhole principle, there is a a node $v \neq \vartheta$ in L_s whose hanging subtree contains more than $a'(v)$ nodes marked as guessed.

Therefore, as proved above, the subtree hanging from v contains a guessed node v' such that $w(v') > a'(v')$. Since $\delta \geq 1$ and Lemma 5.6 (i) holds, we can apply Lemma 5.4 to v', which implies that there exist heavy nodes. In turn, this contradicts Lemma 5.6 (ii). We conclude that at most $\delta(n-1)$ levels contain locked nodes. □

Lemma 5.8. *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 where all guessable nodes are already counted.*

Proof. By definition of t, either $t = s$ or the algorithm has performed at least one guess on a node at level L_t with a guesser at level L_{t-1}. It is easy to prove by induction that the first guesser to perform a guess on this level must be the unique descendant $\vartheta' \in L_{t-1}$ of the selected leader node $\vartheta \in L_s$. Moreover, both ϑ' and its unique child in $\mathcal{V}$ have been assigned conditional anonymity x at Lines 9–11, and the same is true of all nodes in the black path P_ϑ from ϑ to ϑ', which is a strand in

$\mathcal{V}$. Since level L_t is not missing any nodes, then each of the nodes in P_ϑ has a unique child in the history tree, as well. It follows that all descendants of ϑ up to level L_t have the same anonymity as ϑ. Also, by definition of t and the way it is updated (Line 14), no guesses have been made on nodes at levels deeper than L_t, and hence Condition 1 is satisfied up to this point. Thus, Lemma 5.6 applies.

Observe that there are no counting cuts, or Line 13 would not be reachable. Due to Lines 9–11, all of the nodes in P_ϑ initially become guessers. Hence, all levels between L_s and L_{t-1} must have a non-empty set of guessers at all times. Consider any level L_i with $s < i \leq t$ such that all the guessable nodes in L_i are already counted. Let S be the set of guessers in L_{i-1}; note that not all nodes in L_{i-1} are guessers, or else they would give rise to a counting cut. Since the network is 1-union-connected, there is a red edge $\{u, v\}$ (with positive multiplicity) such that $u \in S$ and the parent of v is not in S. By definition, the node v is guessable; therefore, it is counted. Also, since the parent of v is not a guesser, v must have a non-counted parent or a non-counted sibling; note that such a non-counted node is in $\mathcal{V}$, because the levels up to L_t are not missing any nodes.

We have proved that every level between L_{s+1} and L_t where all guessable nodes are counted contains a counted node v having a parent or a sibling that is not counted: we call such a node v a *bad* node. To conclude the proof, it suffices to show that there are at most $n-2$ bad nodes between L_{s+1} and L_t. Observe that no nodes in P_ϑ can be bad.

We will prove that, if a subtree $\mathcal{W}$ of $\mathcal{V}$ contains the root r, the leader node ϑ, no counting cuts, and no non-trivial isles, then $\mathcal{W}$ contains at most $f-1$ bad nodes, where f is the number of leaves of $\mathcal{W}$ not in the subtree hanging from ϑ. We stress that, in the context of $\mathcal{W}$, a bad node is defined as a counted node in $\mathcal{W}$ (other than ϑ and its descendants) that has a non-counted parent or a non-counted sibling in $\mathcal{W}$.

The proof is by induction on f. The case $f = 0$ is impossible, because the single node ϑ yields a counting cut. Thus, the base case is $f = 1$, which holds because any bad node v in $\mathcal{W}$ and not in P_ϑ gives rise to the counting cut $\{\vartheta, v\}$ (recall that a bad node is counted by definition).

For the induction step, let v be a bad node of maximum depth in $\mathcal{W}$. Let $(v_1, v_2, \ldots, v_k)$ be the black path from $v_1 = v$ to the root $v_k = r$, and let $1 < i \leq k$ be the smallest index such that v_i has more than one child in $\mathcal{W}$ (i must exist, because this path eventually joins the black path from ϑ to r). Let $\mathcal{W}'$ be the tree obtained by deleting the black edge $\{v_{i-1}, v_i\}$ from $\mathcal{W}$, as well as the subtree hanging from it.

Notice that the induction hypothesis applies to $\mathcal{W}'$: since v_1 is counted, and each of the nodes $v_2, \ldots, v_{i-1}$ has a unique child in $\mathcal{W}$, the removal of $\{v_{i-1}, v_i\}$ does not create counting cuts or non-trivial isles. Also, v_2 is not counted (unless $v_2 = v_i$), because v_1 is bad. Furthermore, none of the nodes $v_3, \ldots, v_{i-1}$ is counted, or else v_2 would be an internal node of a (non-trivial) isle in $\mathcal{W}$. Therefore, none of the nodes $v_2, \ldots, v_{i-1}$ is counted. In particular, none of these nodes is bad in $\mathcal{W}$.

Moreover, a node of $\mathcal{W}'$ is bad if and only if it is bad in $\mathcal{W}$. This is trivial for all nodes, except for the siblings of v_{i-1}, which require a careful proof. Let $u \neq v_{i-1}$ be a sibling of v_{i-1} in $\mathcal{W}$. If u is not counted, then it is not a bad node in $\mathcal{W}$ nor in $\mathcal{W}'$. Hence, let us assume that u is counted. If v_i is not counted, then u is a bad node in $\mathcal{W}$ and in $\mathcal{W}'$. Thus, let us assume that v_i is counted. Note that $i = 2$ must hold, otherwise v and v_i would form a non-trivial isle in $\mathcal{W}$. In particular, v and u are siblings. Since v is a bad node and its parent v_2 is counted, it must have a non-counted sibling $u' \neq u$ in $\mathcal{W}$. However, u' is also a node of $\mathcal{W}'$, and therefore u is not a bad node in $\mathcal{W}$ nor in $\mathcal{W}'$.

It follows that $\mathcal{W}'$ has exactly one less bad node than $\mathcal{W}$ and at most $f-1$ leaves (because there is at least one leaf in the subtree of $\mathcal{W}$ hanging from v). Thus, the induction hypothesis implies that $\mathcal{W}'$ contains at most $f-2$ bad nodes, and therefore $\mathcal{W}$ contains at most $f-1$ bad nodes.

Observe that the subtree $\mathcal{V}'$ of $\mathcal{V}$ formed by all levels up to L_t satisfies all of the above conditions, as it contains $\vartheta \in L_s$, the root r, and has no counting cuts, because a counting cut for $\mathcal{V}'$ would be a counting cut for $\mathcal{V}$, as well (recall that $\mathcal{V}$ has no counting cuts). Also, Lemma 5.6 (iii) ensures that $\mathcal{V}'$ contains no non-trivial complete isles. However, since no nodes are missing from the levels of $\mathcal{V}'$, all isles in $\mathcal{V}'$ are complete, and thus must be trivial. We conclude that, if $\mathcal{V}'$ has f leaves not in the subtree hanging from ϑ, it contains at most $f-1$ bad nodes. Since such leaves induce a partition of the at most $n-1$ processes not represented by ϑ, we have $f \leq n-1$, implying that the number of bad nodes up to L_t is at most $n-2$. □

Observe that the statement of Lemma 5.8 holds for $n = 1$ as well, because in this case the single node ϑ constitutes a counting cut, and Line 13 is never reached.

Main lemma. We are now ready to prove the salient properties of `ApproxCount` as summarized in Lemma 5.1, which we restate next.

Lemma 5.9. *Let* `ApproxCount`$(\mathcal{V}, s, x, \ell)$ *return* (n', t)*. Assume that* ϑ *exists and* $x \geq a(\vartheta)$*. Let* ϑ' *be the (unique) descendant of* ϑ *in* $\mathcal{V}$ *at level* L_t*, and let* $L_{t'}$ *be the last level of* $\mathcal{V}$*. Then:*

(i) If $x = a(\vartheta) = a(\vartheta')$*, then* $n' \neq -3$*.*

(ii) If $n' > 0$ *and* $t' \geq t + n'$ *and* $a(\vartheta) = a(\vartheta')$*, 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* $\mathcal{V}$*.*

Proof. Note that ϑ' is well defined, because the returned pair is (n', t), which means that either $t = s$, and thus $\vartheta = \vartheta'$, or $t > s$, and hence some guesses have been made on nodes in level L_t, the first of which must have had the parent of ϑ' as the guesser.

Let us prove statement (i). The assumption $x = a(\vartheta)$ implies $\delta = 1$. Moreover, since $a(\vartheta) = a(\vartheta')$, Condition 1 is satisfied whenever a guess is made (this is a straightforward induction). Therefore, by Lemma 5.6 (iv), all nodes marked as counted up to L_t indeed have the correct guesses. So, the conditional anonymity that is computed for any node is equal to its anonymity ($a'(v) = \delta a(v) = a(v)$), and hence is an integer. This implies that `ApproxCount` cannot return the error code -3 at Line 19. Also, either $\ell' = \ell$ if all leader processes have been counted, or $\ell' < \ell$ if some leader nodes are missing from the view. Either way, `ApproxCount` cannot return the error code -3 at Line 31. We conclude that $n' \neq -3$.

Let us prove statement (ii). Again, because $a(\vartheta) = a(\vartheta')$, Condition 1 is satisfied, and all nodes marked as counted have correct guesses. Also, $x \geq a(\vartheta)$ is equivalent to $\delta \geq 1$. By assumption, `ApproxCount` returns (n', t) with $n' > 0$ and $t' \geq t + n'$. Since $n' > 0$, a counting cut C was found whose nodes are within levels up to L_t, and n' is the sum of the conditional anonymities of all nodes in C. Let S_C be the set of processes represented by the nodes of C; note that $n' \geq |S_C|$, because $\delta \geq 1$. We will prove that S_C includes all processes in the system. Assume the contrary; Lemma 3.3 implies that, since $t' \geq t + n' \geq t + |S_C|$, there is a node $z \in L_t$ representing some process not in S_C. Thus, the black path from z to the root r does not contain any node of C, contradicting the fact that C is a counting cut with no nodes after L_t. Therefore, $|S_C| = n$, i.e., the nodes in C represent all processes in the system. Since `ApproxCount` returns $n' > 0$, the "reality check" $\ell' = \ell$ succeeds (Lines 30–32). However, ℓ' is the sum of the conditional anonymities of all leader nodes in C, and hence $\ell' = \delta\ell$, implying that $\delta = 1$. Thus, $n' = \delta n = n$, as claimed.

Let us prove statement (iii). Once again, $x \geq a(\vartheta)$ is equivalent to $\delta \geq 1$. By Corollary 3.4, if $L_{t'}$ is the last level of $\mathcal{V}$, then no nodes are missing from level $L_{t'-n+1}$. In fact, since $t' - n + 1 \geq s + (\ell + 1)(n - 1)$, no nodes are missing from any level up to $L_{s+(\ell+1)(n-1)}$. Let ϑ'' be the deepest descendant of ϑ that is marked as counted at Lines 9–11, and let L_p be the level of ϑ''. By construction, either all children of ϑ'' are missing from L_{p+1} or at least two children of ϑ'' are in L_{p+1}. Also note that ϑ' must be an ancestor of ϑ'', and so $t \leq p$.

Assume that $p < s + (\ell+1)(n-1)$. This implies that no nodes are missing from level L_{p+1}, and therefore ϑ'' must have at least two children in L_{p+1}. Since $t \leq p$, we have $t < s + (\ell+1)(n-1)$, as desired. Now assume that $n' = -2$, which implies that the algorithm was unable to find a counting cut. We claim that in this case $t = p$. So, assume for a contradiction that $t \leq p - 1$. It follows that $\vartheta' \in L_t$ is a guesser. Recall that L_t and L_{t+1} are not missing any nodes, because $t \leq p$. Since the network is connected at round $t+1$, there is at least one node $v \in L_{t+1}$ that is guessable by ϑ'. Also, since no guess has ever been made in level L_{t+1} (due to the way t is updated in Line 14), it follows that v is not counted and not locked. However, the algorithm cannot return $n' = -2$ as long as there are nodes such as v (Line 12). Thus, $t = p$, which means that $\vartheta' = \vartheta''$, and hence ϑ' has at least two children in $\mathcal{V}$, as desired.

Assume now that $p \geq s + (\ell+1)(n-1)$. If ϑ is the only node in L_s, it constitutes a counting cut. In this special case, Line 13 is never reached, $n' = -2$ is not returned at Line 25, and $t = s$. Hence, we may assume that there are nodes in L_s other than ϑ and Line 13 is reached (so, we have $n > 1$). Recall that no nodes are missing from any level up to $L_{s+(\ell+1)(n-1)}$. In particular, no nodes are missing from the levels in the non-empty interval $\mathcal{L}$ consisting of the $(\ell+1)(n-1)$ levels from L_{s+1} to $L_{s+(\ell+1)(n-1)}$. Thus, by definition of p, as long as no guesses are made outside of $\mathcal{L}$, Condition 1 holds, and therefore Lemmas 5.7 and 5.8 apply. Hence, as long as no guesses are made outside of $\mathcal{L}$, there are at most $\delta(n-1)$ levels of $\mathcal{L}$ containing locked nodes (Lemma 5.7) and there are at most $n-2$ levels of $\mathcal{L}$ where all guessable nodes are already counted (Lemma 5.8). So, there are at most

$$\delta(n-1) + n - 2 \leq \ell(n-1) + n - 2 = (\ell+1)(n-1) - 1$$

levels in $\mathcal{L}$ were the algorithm can make no new guesses. We conclude that $\mathcal{L}$ always contains at least one node where a guess can be made, and no guesses are ever made outside of $\mathcal{L}$ until either a counting cut is found or $n' = -3$ is returned at Line 19. In both cases, $n' = -2$ is not returned, and moreover $t \leq s + (\ell+1)(n-1)$, as desired.

It remains to prove that $n' \neq -1$. Since ϑ exists by assumption, the error code -1 cannot be returned at Line 6 and can only be returned at Line 30. In turn, this can only occur if a counting cut has been found and $\ell' < \ell$. However, we have already proved that no level up to L_t is missing any nodes, which implies that the counting cut contains nodes representing all processes, and in particular $\ell' = \delta\ell$. Since $\delta \geq 1$, we have $\ell' \geq \ell$, and the condition at Line 30 is not satisfied. ☐

Worst-case example. For $\ell = \tau = 1$, our Counting algorithm in Theorem 5.2 yields a running time of $3n-2$ rounds. The example in Figure 8, which can easily be generalized to networks of any size n, shows that our Counting algorithm can in fact terminate in $3n-3$ rounds. Indeed, the last node in a counting cut is at level L_{2n-3}, and then it takes an extra n rounds for the terminating condition $t \geq t^* + n^*$ of Listing 2 to be satisfied.

6 Negative Results

In this section we collect several negative results and counterexamples, which not only provide lower bounds asymptotically matching our algorithms' running times, but also justify all of the assumptions made in Sections 4 and 5 about the a-priori knowledge of processes. Although some of these facts were previously known, here we offer simple and self-contained proofs based on history trees.

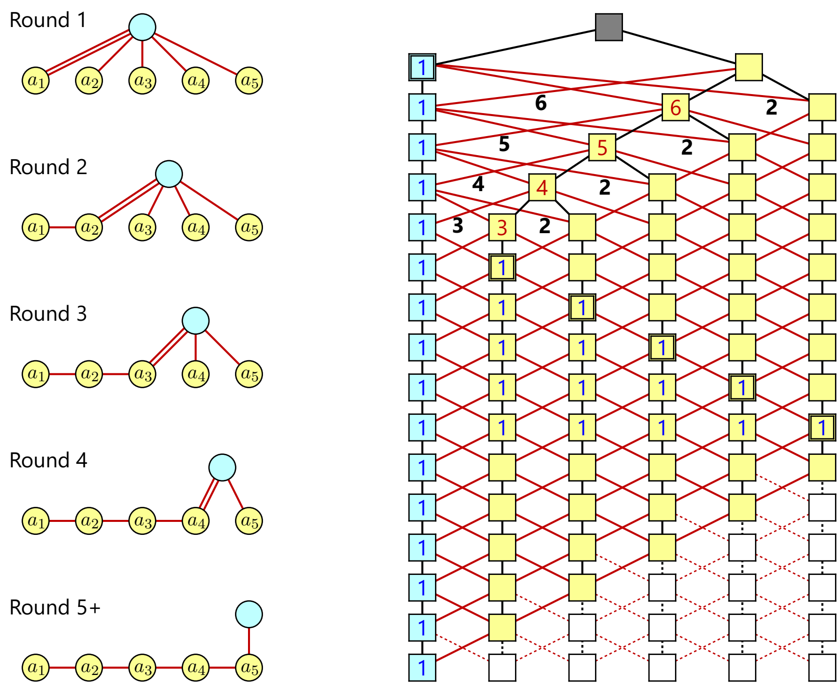

Figure 8: An example of a dynamic network with $\ell = \tau = 1$ and $n = 6$ where the algorithm of Theorem 5.2 terminates in $3n - 3$ rounds, almost matching the upper bound of $3n - 2$ rounds. The white nodes in the history tree are not in the view of the leader at the last round. The numbers inside nodes represent guesses; the blue ones correspond to nodes that are marked as counted. The six highlighted nodes constitute a counting cut.

6.1 Unsolvable Problems

Showing that certain problems are not solvable in certain network models allows us to argue that our algorithms are *universal*, i.e., they can be applied to all solvable problems. Remarkably, all of our algorithms exhibit the highest level of generality possible.

Networks with leaders. We will prove that the multiset-based functions introduced in Section 2 are the only functions that can be computed deterministically in anonymous networks with leaders. This result, together with Proposition 2.1, shows that the algorithms in Sections 4.4 and 5 can be generalized to all functions that can be computed in networks with leaders, i.e., the multiset-based functions.

Proposition 6.1. *No function other than the multiset-based functions can be computed (with or without termination), even when restricted to connected static simple networks with a known and arbitrary number of leaders.*

Proof. Let us consider the (static) network whose topology at round t is the complete graph $G_t = K_n$, i.e., each process receives messages from all other processes at every round. We can prove by induction that all nodes of the history tree other than the root have exactly one child. This is because any two processes with the same input always receive equal multisets of messages, and are therefore always indistinguishable. Thus, the history tree is completely determined by the multiset μ_λ of all processes' inputs; moreover, a process' view at any given round only depends on the process' own input and on μ_λ. By the fundamental theorem of history trees Theorem 3.1, this is enough to conclude that if a process' output stabilizes, that output must be a function of the process' own input and of μ_λ, which is the defining condition of a multiset-based function. $\square$

The following result justifies the assumption made in Sections 4.4 and 5 that processes have a-priori knowledge of the number of leaders ℓ in the system.

Proposition 6.2. *No algorithm can compute the Counting function F_C (with or without termination) with no knowledge about ℓ, even when restricted to connected static simple networks with a known and arbitrarily small ratio ℓ/n.*

Proof. Let us fix a positive integer k; we will construct an infinite class of networks whose ratio ℓ/n is $1/k$ as follows. For every $i \geq 3$, let $\mathcal{G}_i$ be the static network consisting of a cycle of $n_i = k \cdot i$ processes of which $\ell_i = i$ are leaders, such that the leaders are evenly spaced among the non-leaders. Assume that all processes get the same input (apart from their leader flags). Then, at any round, all the leaders in all of these networks have isomorphic views, which are independent of i. It follows that, if nothing is known about ℓ_i (other than the ratio ℓ_i/n_i, which is fixed), all the leaders in all the networks $\mathcal{G}_i$ always give equal outputs. Since the number of processes n_i depends on i, it follows that at most one of these networks can stabilize on the correct output n_i. $\square$

Leaderless networks. We will now prove that the frequency-based functions introduced in Section 2 are the only functions that can be computed deterministically in anonymous networks without leaders. This result, together with Proposition 2.2, shows that the algorithms in Sections 4.2 and 4.3 can be generalized to all functions that can be computed in leaderless networks, i.e., the frequency-based functions.

Although the next theorem is implied by [30, Theorem III.1], here we provide an alternative and simpler proof.

Proposition 6.3. *No function other than the frequency-based functions can be computed with no leader, even when restricted to connected static simple networks.*

Proof. Let $m_1, m_2, \dots, m_k$ be integers greater than 2 with $\gcd(m_1, m_2, \dots, m_k) = 1$, and let B be the complete k-partite graph with partite sets $V_1, V_2, \dots, V_k$ of sizes $m_1, m_2, \dots, m_k$, respectively. For any positive integer α, construct the static network G_α consisting of α disjoint copies of B, augmented with k cycles $C_1, C_2, \dots, C_k$ such that, for each $1 \leq i \leq k$, the cycle C_i spans all the αm_i processes in the α copies of V_i. Clearly, G_α is a connected static simple network.

Let the function λ_α assign input z_i to all processes in the α copies of V_i in G_α, and let $\lambda = \lambda_1$. As a result, $\mu_{\lambda_\alpha} = \{(z_1, \alpha m_1), (z_2, \alpha m_2), \dots, (z_k, \alpha m_k)\} = \alpha \cdot \mu_\lambda$ for all $\alpha \geq 1$. Moreover, all the networks G_α have isomorphic history trees. This is because, at every round, each process in any of the copies of V_i receives exactly two messages from other processes in copies of V_i and exactly m_j messages from processes in copies of V_j, for all $j \neq i$. Thus, it can be proved by induction that all processes in the copies of V_i have isomorphic views, regardless of α.

Due to the fundamental theorem of history trees Theorem 3.1, all the processes with input z_i must give the same output $\psi(z_i, \mu_\lambda) = \psi(z_i, \mu_{\lambda_\alpha}) = \psi(z_i, \alpha \cdot \mu_\lambda)$, regardless of α. Hence, by definition, only frequency-based functions can be computed in these networks. □

The following result justifies the assumptions made in Section 4.3 that processes have knowledge of an upper bound on n or on the dynamic diameter d of the network.

Proposition 6.4. *No algorithm can solve the leaderless Average Consensus problem with explicit termination if nothing is known about the size of the network or its diameter, even when restricted to connected static simple networks.*

Proof. Assume for a contradiction that there is such an algorithm $\mathcal{A}$. Let $\mathcal{G}$ be a static network consisting of three processes forming a cycle, and assign input 0 to all of them. If the processes execute $\mathcal{A}$, they eventually output the mean value 0 and terminate, say in t rounds.

Now construct a static network $\mathcal{G}'$ consisting of a cycle of $2t+2$ processes $p_1, p_2, \dots, p_{2t+2}$; assign input 1 to p_1 and input 0 to all other processes. It is easy to see that, from round 0 to round t, the view of the process p_{t+1} is isomorphic to the view of any process in $\mathcal{G}$. Therefore, if p_{t+1} executes $\mathcal{A}$, it terminates in t rounds with the incorrect output 0. Thus, $\mathcal{A}$ is incorrect. □

6.2 Lower Bounds

We will now give some lower bounds on the complexity of problems for anonymous dynamic networks. Since our algorithms have linear running times, our focus is on optimizing the multiplicative constants of the leading terms.

Preliminary results. We first prove some simple statements that will be used to derive lower bounds for stabilizing and terminating algorithms.

Lemma 6.5. *Let $\mathcal{G}$ and $\mathcal{G}'$ be two networks on n and n' processes respectively, where $n \neq n'$. Assume that there is a process p in $\mathcal{G}$ and a process p' in $\mathcal{G}'$ such that p and p' have isomorphic views at round t. Then,*

- *No algorithm computes the Counting function F_C and stabilizes within t rounds in both $\mathcal{G}$ and $\mathcal{G}'$.*
- *No algorithm computes the Counting function F_C in both $\mathcal{G}$ and $\mathcal{G}'$ and terminates within t rounds in $\mathcal{G}$ (or $\mathcal{G}'$).*

Proof. Since p and p' have isomorphic views at round t, they have isomorphic views at all rounds up to t. Thus, by Theorem 3.1, if p and p' execute the same algorithm, they give equal outputs up to round t. Since the Counting function F_C prescribes that p must output n and p' must output $n' \neq n$, it is impossible for both processes to simultaneously give the correct output within t rounds. In particular, no Counting algorithm can stabilize within t rounds.

Moreover, if the execution of p terminated within t rounds, then the execution of p' would terminate at the same round, as well (again, due to Theorem 3.1). In that case, both processes would return the same output, which would be incorrect for at least one of them. In particular, no Counting algorithm can terminate within t rounds in $\mathcal{G}$. □

Corollary 6.6. *Let $(\mathcal{G}_n)_{n\geq 1}$ be a sequence where $\mathcal{G}_n$ is a network on n processes. Assume that there exists a constant $k > 0$ such that, for infinitely many values of $n \in \mathbb{N}$, there is a process in $\mathcal{G}_n$ and a process in $\mathcal{G}_{n+k}$ that have isomorphic views at round $an + b$, for integer constants $a > 0$ and b. Then, there is no algorithm that computes the Counting function F_C in every $\mathcal{G}_n$ in less than $an - ak + b + 1$ rounds. If termination is required, the bound improves to $an + b + 1$ rounds.*

Proof. Let $m \in \mathbb{N}$ be such that $\mathcal{G}_m$ and $\mathcal{G}_{m+k}$ contain processes with isomorphic views at round $t = am + b$. Assume for a contradiction that a Counting algorithm stabilized in $f(n) \leq an - ak + b$ rounds in every $\mathcal{G}_n$. Then, it would stabilize in $f(m) \leq am - ak + b \leq t$ rounds in $\mathcal{G}_m$ and in $f(m + k) \leq a(m + k) - ak + b = t$ rounds in $\mathcal{G}_{m+k}$. This contradicts Lemma 6.5, which states that no Counting algorithm stabilizes within t rounds in both networks.

Moreover, if a Counting algorithm terminated in $f(n) \leq an + b$ rounds in every $\mathcal{G}_n$, it would terminate in $f(m) \leq am + b = t$ rounds in $\mathcal{G}_n$, which again contradicts Lemma 6.5. □

Unique-leader networks. We will now prove a lower bound of roughly $2n$ rounds on the Counting problem for always connected networks with a unique leader.

We first introduce a family of 1-union-connected dynamic networks. For any $n \geq 1$, we consider the dynamic network $\mathcal{G}_n$ whose topology at round t is the graph $G_t^{(n)}$ defined on the system $\{p_1, p_2, \ldots, p_n\}$ as follows. If $t \geq n - 2$, then $G_t^{(n)}$ is the path graph P_n spanning all processes p_1, p_2, …, p_n in order. If $1 \leq t \leq n - 3$, then $G_t^{(n)}$ is P_n with the addition of the single edge $\{p_{t+1}, p_n\}$. We assume p_1 to be the leader and all other processes to have the same input.

Proposition 6.7. *In 1-union-connected dynamic simple networks with a unique leader, no algorithm can compute the Counting function F_C in less than $2n - 6$ rounds. If termination is required, the bound improves to $2n - 4$ rounds.*

Proof. Let us consider the network $\mathcal{G}_n$ as defined above. It is straightforward to prove by induction that, at every round $t \leq n - 3$, the process p_{t+1} gets disambiguated, while all processes p_{t+2}, p_{t+3}, …, p_n are still indistinguishable. So, the history tree of $\mathcal{G}_n$ has a very regular structure, which is illustrated in Figures 9 and 10. By comparing the history trees of $\mathcal{G}_n$ and $\mathcal{G}_{n+1}$, we see that the leaders of the two systems have identical views up to round $2n - 5$. Our claim now follows immediately from Corollary 6.6, with $k = 1$, $a = 2$, and $b = -5$ □

A slightly better lower bound can be obtained if we allow self-loops in the network. Consider the static network $\mathcal{G}_n$ consisting of a path graph spanning all n processes, where one endpoint of the path is the unique leader and the other endpoint has a self-loop. It is easy to see that the leaders in $\mathcal{G}_n$ and $\mathcal{G}_{n+1}$ have identical views up to round $2n - 2$. This implies a lower bound of $2n - 3$ rounds for stabilization and $2n - 1$ rounds for termination in static networks with self-loops and a unique leader.

Multi-leader networks. We can now generalize Proposition 6.7 to any τ and any $\ell \geq 1$. This lower bound shows that the algorithms in Sections 4.4 and 5 are asymptotically optimal for any constant number of leaders ℓ.

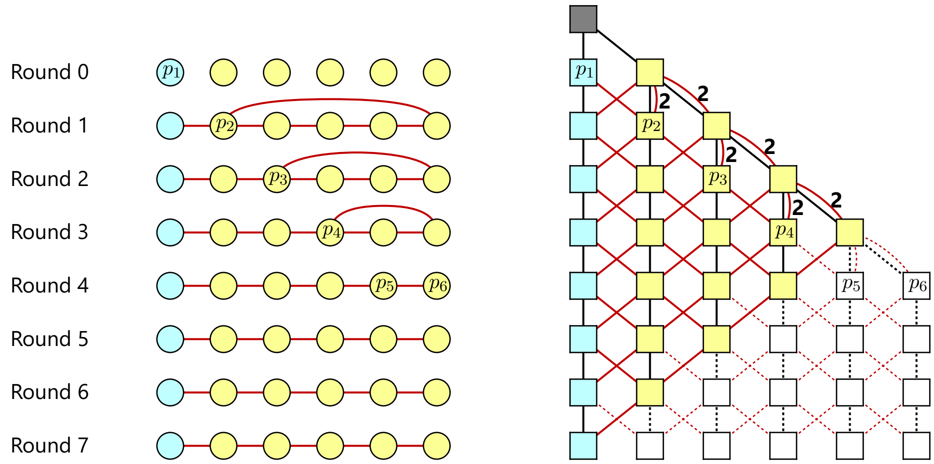

Figure 9: The first rounds of the dynamic network $\mathcal{G}_n$ used in Proposition 6.7 (left) and the corresponding levels of its history tree (right), where $n = 6$; the process in blue is the leader. The white nodes and the dashed edges in the history tree are not in the view of the leader at round 7. The labels $p_1, \ldots, p_6$ have been added for the reader's convenience, and mark the processes that get disambiguated, as well as their corresponding nodes of the history tree, which have anonymity 1.

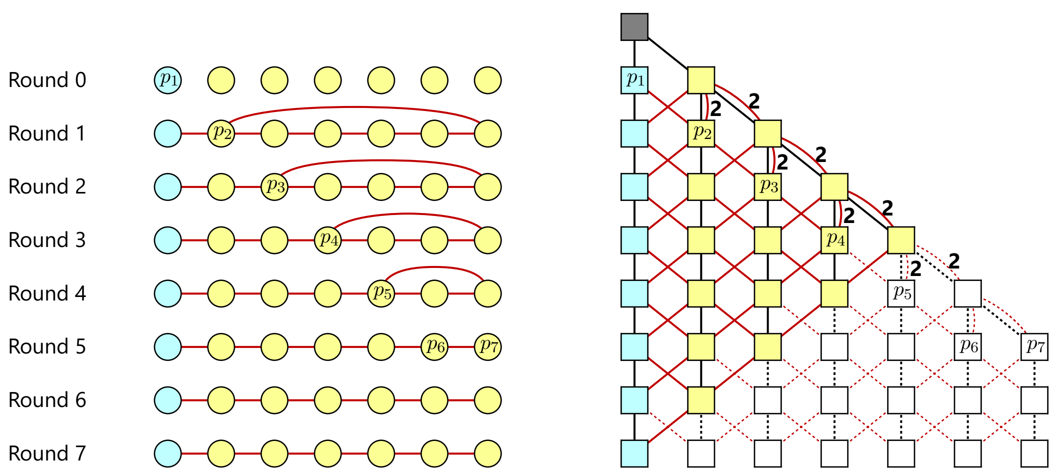

Figure 10: The first rounds of the dynamic network $\mathcal{G}_{n+1}$ with $n = 6$. Observe that the view of the leader at round 7 is identical to the view highlighted in Figure 9. The intuitive reason is that, from round 1 to round $n - 3$, both networks have a cycle whose processes are all indistinguishable (and are therefore represented by a single node in the history tree), except for the one process with degree 3. Thus, the history trees of $\mathcal{G}_n$ and $\mathcal{G}_{n+1}$ are identical up to level $n - 3$. After that, the two networks get disambiguated, but this information takes another $n - 3$ rounds to reach the leader. Therefore, if the leader of $\mathcal{G}_n$ and the leader of $\mathcal{G}_{n+1}$ execute the same algorithm, they must have the same internal state up to round $2n - 5$, due to Theorem 3.1. In particular, they cannot give different outputs up to that round, which leads to our lower bounds on stabilization and termination for the Counting problem.

Proposition 6.8. *For any $\ell \geq 1$, no algorithm can compute the Counting function F_C in all simple τ-union-connected networks with ℓ leaders in less than $\tau(2n - \ell - 5)$ rounds, which improves to $\tau(2n - \ell - 3)$ rounds if termination is required.*

Proof. As shown in Proposition 6.7, there is a family of simple 1-union-connected networks $\mathcal{G}_n$, with $n \geq 1$, with the following properties. $\mathcal{G}_n$ has $\ell = 1$ leader and n processes in total; moreover, up to round $2n - 5$, the leaders of $\mathcal{G}_n$ and $\mathcal{G}_{n+1}$ have isomorphic views.

Let us fix $\ell \geq 1$, and let us construct $\mathcal{G}'_n$, for $n \geq \ell$, by attaching a chain of $\ell - 1$ additional leaders $p_1, p_2, \ldots, p_{\ell-1}$ to the single leader p_ℓ of $\mathcal{G}_{n-\ell+1}$ at every round. Note that $\mathcal{G}'_n$ has n processes in total and a stable subpath $(p_1, p_2, \ldots, p_\ell)$ which is attached to the rest of the network via p_ℓ.

It is straightforward to see that the process p_ℓ in $\mathcal{G}'_n$ and the process p_ℓ in $\mathcal{G}'_{n+1}$, which correspond to the leaders of $\mathcal{G}_{n-\ell+1}$ and $\mathcal{G}_{n-\ell+2}$ respectively, have isomorphic views up to round $2(n - \ell) - 3$. Since the view of p_1 is completely determined by the view of p_ℓ, and it takes $\ell - 1$ rounds for any information to travel from p_ℓ to p_1, we conclude that the process p_1 in $\mathcal{G}'_n$ and the process p_1 in $\mathcal{G}'_{n+1}$ have isomorphic views up to round $2n - \ell - 4$.

It follows from Corollary 6.6 (with $k = 1$, $a = 2$, $b = -\ell - 4$) that the Counting function with $\ell \geq 1$ leaders and $\tau = 1$ cannot be computed in less than $2n - \ell - 5$ rounds in a stabilizing fashion or in less than $2n - \ell - 3$ if termination is required. These bounds generalize to an arbitrary τ by Proposition 2.4. □

Leaderless networks. Finally, we can use Proposition 6.7 to obtain a lower bound for the Average Consensus problem in leaderless networks. This lower bound shows that the leading term in the running time of the algorithm in Section 4.2, i.e., $2\tau n$, is optimal. The same holds for the algorithm in Section 4.3 assuming that $N = n$.

Proposition 6.9. *No algorithm can solve the Average Consensus problem (with or without termination) in all simple τ-union-connected leaderless networks in less than $\tau(2n - 6)$ rounds, which improves to $\tau(2n - 4)$ rounds if termination is required.*

Proof. According to Proposition 6.7, the number of processes n in a network with $\ell = 1$ and $\tau = 1$ cannot be determined in less than $2n - 6$ rounds ($2n - 4$ rounds if termination is required). We can reduce this problem to Average Consensus with $\ell = 0$ and $\tau = 1$ as follows. In any given network with $\ell = \tau = 1$, assign input 1 to the leader and clear its leader flag; assign input 0 to all other processes. If the processes can compute the mean input value, $1/n$, they can invert it to obtain n in the same number of rounds. It follows that Average Consensus with $\ell = 0$ and $\tau = 1$ cannot be solved in less than $2n - 6$ rounds ($2n - 4$ rounds if termination is required); this immediately generalizes to an arbitrary τ by Proposition 2.4. □

7 Conclusions

We introduced the novel concept of *history tree* and used it as our main investigation technique to study computation in anonymous dynamic networks. History trees are a powerful tool that completely and naturally captures the concept of symmetry and indistinguishability among processes. We have demonstrated the effectiveness of our methods by optimally solving a wide class of fundamental problems, and we believe that our techniques will find numerous applications in other settings, as well.

We have shown that anonymous processes in τ-union-connected dynamic networks can compute all the multiset-based 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 functions. We have also identified the Input Frequency function and the Input Multiset function as the complete problems for each class. Notably, the network's dynamic disconnectivity τ does not affect the computability of functions, but only makes computation proportionally slower.

Moreover, we gave efficient stabilizing and terminating algorithms for computing all the aforementioned 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 τ 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 computations with $\ell \geq 1$ leaders, we have a lower bound of roughly $\tau(2n-\ell)$ rounds (Proposition 6.8) and an upper bound of roughly $\tau(\ell^2 + \ell + 1)n$ rounds (Theorem 5.2). 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.

It is worth noting that for stabilizing computation (i.e., when explicit termination is not required) in networks with a constant number of leaders, our lower and upper bounds are essentially $2\tau n$ rounds; hence, in these case we were able to optimize the multiplicative constant, as well. As for terminating computation with a unique leader, we have a lower bound of $2\tau n$ rounds and an upper bound of $3\tau n$ rounds. Although we are still unable to completely close this gap, we emphasize that our findings demonstrate the practical feasibility of general computations in anonymous dynamic networks with a unique leader, which was a major open problem in this research area prior to our work.

Observe that our stabilizing algorithms use an unbounded amount of memory, as processes keep adding nodes to their views at every round. This can be avoided if the dynamic disconnectivity τ (as well as an upper bound on n, in the 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.

Our algorithms require processes to send each other explicit representations of their history trees, which have roughly cubic size in the worst case. 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 [24].

Finally, we wonder if our results hold more generally for networks where communications are not necessarily synchronous. We conjecture that our algorithms can be generalized to networks where messages may be delayed by a bounded number of rounds or processes may be inactive for some rounds (provided that a "global fairness" condition is met).

References

[1] D. Angluin. Local and Global Properties in Networks of Processors. In *Proceedings of the 12th ACM Symposium on Theory of Computing (STOC '80)*, pages 82–93, 1980.

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

[3] 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.

[4] L. Baumgärtner, A. Dmitrienko, B. Freisleben, A. Gruler, J. Höchst, J. Kühlberg, M. Mezini, R. Mitev, M. Miettinen, A. Muhamedagic, T. Nguyen, A. Penning, D. Pustelnik, F. Roos, A. Sadeghi, M. Schwarz, and C. Uhl. Mind the GAP: Security & Privacy Risks of Contact Tracing Apps. In *Proceedings of the 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom '20)*, pages 458–467, 2020.

[5] 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.

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

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

[8] 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.

[9] P. Boldi and S. Vigna. Fibrations of Graphs. *Discrete Mathematics*, 243(1):21–66, 2002.

[10] 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.

[11] 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.

[12] M. Chakraborty, A. Milani, and M. A. Mosteiro. Counting in Practical Anonymous Dynamic Networks Is Polynomial. In *Proceedings of the 4th International Conference on Networked Systems (NETYS '16)*, pages 131–136, 2016.

[13] 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.

[14] 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.

[15] 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.

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

[17] G. A. Di Luna and G. Baldoni. Brief Announcement: Investigating the Cost of Anonymity on Dynamic Networks. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC '15)*, pages 339–341, 2015.

[18] G. A. Di Luna and G. Baldoni. Non Trivial Computations in Anonymous Dynamic Networks. In *Proceedings of the 19th International Conference on Principles of Distributed Systems (OPODIS '15)*, pages 1–16, 2016.

[19] G. A. Di Luna, R. Baldoni, S. Bonomi, and I. Chatzigiannakis. Conscious and Unconscious Counting on Anonymous Dynamic Networks. In *Proceedings of the 15th International Conference on Distributed Computing and Networking (ICDCN '14)*, pages 257–271, 2014.

[20] G. A. Di Luna, R. Baldoni, S. Bonomi, and I. Chatzigiannakis. Counting in Anonymous Dynamic Networks Under Worst-Case Adversary. In *Proceedings of the 34th IEEE International Conference on Distributed Computing Systems (ICDCS '14)*, pages 338–347, 2014.

[21] G. A. Di Luna, S. Bonomi, I. Chatzigiannakis, and R. Baldoni. Counting in Anonymous Dynamic Networks: An Experimental Perspective. In *Proceedings of the 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS '13)*, pages 139–154, 2013.

[22] 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.

[23] 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.

[24] 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.

[25] G. A. Di Luna and G. Viglietta. Optimal computation in leaderless and multi-leader disconnected anonymous dynamic networks. In *Proceedings of the 37th International Symposium on Distributed Computing (DISC '23)*, pages 18:1–18:20, 2023.

[26] 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.

[27] P. Fraigniaud and A. Pelc. Decidability Classes for Mobile Agents Computing. *Journal of Parallel and Distributed Computing*, 109:117–128, 2017.

[28] 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.

[29] M. Ghaffari, C. Musco, T. Radeva, and N. Lynch. Distributed House-Hunting in Ant Colonies. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC '15)*, pages 57–66, 2015.

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

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

[38] 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.

[39] 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.

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

[41] L. B. Lamport. Time, Clocks and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 1978.

[42] 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.

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

[44] 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.

[45] 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.

[46] N. Norris. Universal Covers of Graphs: Isomorphism to Depth $n-1$ Implies Isomorphism to All Depths. *Discrete Applied Mathematics*, 56(1):61–74, 1995.

[47] 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.

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

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

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] S. Tani. Compression of View on Anonymous Networks—Folded View—. *IEEE Transactions on Parallel and Distributed Systems*, 23(2):255–262, 2012.

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

[56] 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.

[57] 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.

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