Non Trivial Computations in Anonymous Dynamic Networks.

Roberto Baldoni and Giuseppe Antonio Di Luna
Dipartimento di Ingegneria Informatica, Automatica e Gestionale Antonio Ruberti
Università degli Studi di Roma La Sapienza
Via Ariosto, 25, I-00185 Roma, Italy
{baldoni, diluna}@dis.uniroma1.it

Abstract

In this paper we consider a static set of anonymous processes, i.e., they do not have distinguished IDs, that communicate with neighbors using a local broadcast primitive. The communication graph changes at each computational round with the restriction of being always connected, i.e., the network topology guarantees 1-interval connectivity. In such setting non trivial computations, i.e., answering to a predicate like *"there exists at least one process with initial input a?"*, are impossible. In a recent work, it has been conjectured that the impossibility holds even if a distinguished leader process is available within the computation. In this paper we prove that the conjecture is false. We show this result by implementing a deterministic leader-based terminating counting algorithm. In order to build our counting algorithm we first develop a counting technique that is time optimal on a family of dynamic graphs where each process has a fixed distance h from the leader and such distance does not change along rounds. Using this technique we build an algorithm that counts in anonymous 1-interval connected networks.

1 Introduction

Highly dynamic distributed systems are attracting a lot of interest from the relevant research community [6, 12]. These models are well suited to study the new challenges introduced by distributed systems where there is an immanent dynamicity given by the presence of mobile devices, unstable communication links and environmental constraints. A critical element in such future distributed systems is the *anonymity* of the devices; the uniqueness of a process ID is not guaranteed due to operational limit (e.g., in highly dynamic networks maintaining unique IDs may be infeasible due to mobility and failure among processes [21]) or to maintaining user's privacy (e.g., where users may not wish to disclose information about their behavior [10]).

In this paper we consider a static set of anonymous process $|V|$, this set of processes is connected by a dynamic communication graph that is governed by a fictional omniscient entity *the adversary* who has the power to change at each round the graph. The adversary is able to read the local memory of each process in order to deploy the worst possible communication graph to challenge the computation. The only restriction imposed to the adversary is that the graph has to be connected at each round. This corresponds to the 1-interval connectivity model proposed in [11].

We focus on the problem of counting the number of processes in the system which is one of the fundamental problems of distributed computing [9, 11, 16, 20]. The system model employed in this paper also assumes each process communicates with its neighbors using a local broadcast primitive. Under this model, it has been proved that the presence of a leader process is *necessary*

in order to compute non trivial tasks [18]. In the case of leader absence, the adversary could indeed perpetually generate an anonymous ring graph of unknown size, and it is well known that in such graph non-trivial computation are impossible [1,24]. Let us remark that a leader is actually present in many realistic settings, such as a base station in a mobile network, a gateway in a sensors network etc. Additionally, the computability in the model where all processes are anonymous but a leader has been widely investigated in the static network case and in population protocols [3, 7, 23, 25]. Furthermore, having a leader can be sometimes simpler than ensuring an unique ID for each process. From a formal point of view, it has been proved when processes communicate using broadcast, assuming the existence of a leader is strictly weaker than assuming unique IDs [18].

Anonymity and the adversarial dynamic graph make the system model very challenging for performing non-trivial computation. More specifically, it has been conjectured in [17, 18] that, in such model, the presence of a leader is not sufficient to compute, non trivial tasks such as counting. The main result of this paper is to show that the conjecture is false, and that a distinguished leader process is necessary and sufficient to do deterministic non trivial computations on anonymous 1-interval connected dynamic networks with broadcast. This is shown by introducing a deterministic terminating counting algorithm, namely EXT.

The paper introduces one by one the main sub-algorithms forming EXT, namely OPT, VCD and InstanceCount. As second result presented in this paper, we show that OPT is a time optimal counting algorithm for graphs in $\mathcal{G}(\mathrm{PD})_2$, a specific subset of interval connected dynamic graphs where each process has a fixed distance h from the leader with $h \leq 2$ and such distance is fixed across rounds. We showed in [13] that counting on $\mathcal{G}(\mathrm{PD})_2$ is function of the network size even if there is unlimited bandwidth and a constant dynamic diameter w.r.t. $|V|$. Thus OPT shows that the bound introduced in [13] for counting in $\mathcal{G}(\mathrm{PD})_2$ is tight.

Outline: Section 4 presents an optimal algorithm for graphs belonging to $\mathcal{G}(\mathrm{PD})_2$. Section 5 illustrates the basic structure of the counting algorithm, EXT, for 1-interval connected networks. EXT has two main components: VCD and InstanceCount which are introduced in Section 6 and Section 7 respectively. Finally, we prove that the conjecture presented in [17, 18] is false in Section 8. Due to lack of space, some details can be found in the Appendix.

2 Related Work

The question concerning what can be computed on top of static anonymous networks, has been pioneered by Angluin in [1] and it has been the further investigated in many papers [4,5,24,25]. In a static anonymous network with broadcast, the presence of a leader is enough to have a terminating counting algorithm as shown in [17].

Considering dynamic non anonymous networks, counting has been studied under several dynamicity models. In [2], dynamicity corresponds to processes churn where processes leave and join the system. In [16, 22] dynamicity is governed by a random adversary to model peer-to-peer networks. Finally considering the dynamicity model employed in this paper (worst-case adversary), in [11], a counting algorithm for 1-interval connectivity has been proposed. Other results related to counting can be found in [21] where a model similar to 1-interval connected is considered. In the context of possibly disconnected adversarial network, counting has been studied in [19]. The approaches followed by the latter works are not suitable in the model proposed by this paper, they use the asymmetry introduced by IDs.

Counting in anonymous dynamic networks: In [9], the authors propose a gossip-based protocol to compute aggregation function. The network graph considered by [9] is governed by a fair random adversary, moreover the proposed approach converges to the actual count without having a terminating condition. A similar model and strategy is also used by [8]. The first work investigating the problem of terminating counting in an anonymous network with worst-case adversary and a leader node is [17]. They show that when a process is able to send a different messages to each neighbors, the presence of a leader is enough to have a terminating naming algorithm. For the broadcast case, under the assumption of a fixed known upper bound on the maximum process degree, they provided an algorithm that computes an upper bound on the network size. Building on this result, [14] proposes an exact counting algorithm under the same assumption. Finally, [15] provides a counting algorithm for 1-interval connected networks considering each process is equipped with a local degree detector, i.e. an oracle able to predict the degree of the process before exchanging messages. Other works [11, 20] have investigated leader-less randomized approaches to obtain approximated counting algorithms. We are interested in study how anonymity impacts the computational power of 1-interval connected networks with broadcast, for this reason we assume that processes do not have access to a source of randomness, e.g. they cannot break symmetry by using coin tosses.

3 Model of the computation

We consider a synchronous distributed system composed by a finite static set of processes V. Processes in V are *anonymous*, they initially have no identifiers and execute a deterministic *round-based* computation. Processes communicate through a communication network which is *dynamic*. We assume at each round r the network is stable and represented by a graph $G_r = (V, E(r))$ where $E(r) \subseteq V \times V$ is the set of bidirectional links at round r connecting processes in V.

Definition 1. *A* dynamic graph $G = \{G_0, G_1, \ldots, G_r, \ldots\}$ *is an infinite sequence of graphs one at each round r of the computation.*

A dynamic graph is 1-interval connected, if, and only if, $G \in \mathcal{G}(\text{1-IC})$, if $\forall G_r \in G$ we have that G_r is connected. The neighborhood of a process v at round r is denoted by $N(v,r) = \{v' : (v', v) \in E(r)\}$. We say that v has *degree* d at round r iff $|N(v,r)| = d$. Given a round r we denote with $p_{v,v'}$ a path on G_r between v and v'. Moreover we denote as $P_r(v', v)$, the set of all paths between v, v' on graph G_r. The distance $d_r(v', v)$ is the minimum length among the lengths of the paths in $P_r(v', v)$, the length of the path is defined as the number of edges. We consider the computation proceed by exchanging messages through synchronous rounds.

Every round is divided in two phases: (i) *send* where processes send the messages for the current round, (ii) *receive* where processes elaborates received messages and prepare those that will be sent in the next round. Processes can communicate with its neighbors through an *anonymous broadcast* primitive. Such primitive ensures that a message m sent by process v_i at the beginning of a certain round r will be delivered to all its neighbors during round r. A process v floods message m by broadcasting it for each round. If process receives a flooded message m then it starts the flooding of m. The flood of m terminates when it has been received by all processes. We say that a network has dynamic diameter D if for any v and any round r the flood of a message that starts at round r from process v terminates by at most round $r + D$. Intuitively the dynamic diameter is the maximum time needed to disseminate messages to all processes in the network.

Leader-based computation and worst case adversary We assume the selection of a topology graph at round r is done by an omniscient adversary that may choose at each step the worst configuration to challenge a counting algorithm. Due to the impossibility result shown in [17], we assume any counting algorithm that works over a dynamic graph has a leader process v_l starting with a different unique state w.r.t. all the other processes.

Definition 2. *Given a dynamic network G with $|V|$ processes, a distributed algorithm $\mathcal{A}$ solves the counting on G if it exists a round r at which the leader outputs $|V|$ and terminates.*

Persistent distance dynamic graphs Let us characterize dynamic graphs according to the distances among a process v and the leader v_l.

Definition 3. *(Persistent Distance over G) Consider a dynamic graph G. The distance between v and v_l over G, denoted $D(v, v_l) = d$, is defined as follow: $D(v, v_l) = d$ iff $\forall r, d_r(v, v_l) = d$.*

Let us now introduce a set of dynamic graphs based on the distance between the leader and the processes of a graph.

Definition 4. *(Persistent Distance set) A graph G belongs to Persistent Distance set, denoted $\mathcal{G}(PD)$, iff $\forall v \in G, \exists d \in \mathbb{N}^+ :: D(v, v_l) = d$*

Graphs in $\mathcal{G}(\mathbf{PD})_2$ Among the dynamic graphs belonging to $\mathcal{G}(\text{PD})$ we can further consider the set of graphs, denoted $\mathcal{G}(\text{PD})_h$, whose processes have maximum distance h from the leader with $1 < h \leq |V|$. Thus, given a graph in $\mathcal{G}(\text{PD})_h$ we can partition its processes in h sets, $\{V_0, V_1, \ldots, V_h\}$, according to their distance from the leader. In Figure 1 there is an example of $\mathcal{G}(\text{PD})_2$ graph. The depicted dynamic graph has dynamic diameter $D = 4$, if process v_0 starts a flood at round 0 this flood will reach process v_3 at round 3. The task of the leader process v_l is to count processes in V_2. Let us notice that if a process knows $|N(v, r) \cap V_1|$ before the receive phase of round r then counting in $\mathcal{G}(\text{PD})_2$ needs $O(1)$ rounds, the algorithm is trivial each process in V_2 sends a message $\frac{1}{|N(v,r) \cap V_1|}$ to processes in V_1. A process in V_1 collects these messages and send their sum to the leader. Also if IDs are present counting requires $O(1)$ rounds, in 2 rounds the leader collects the IDs of all processes. It is interesting to notice that if $|N(v, r)|$ is known only when a process receives messages from its neighbors then the time for counting become $\Omega(\log |V|)$ rounds, see Th.2 of [13].

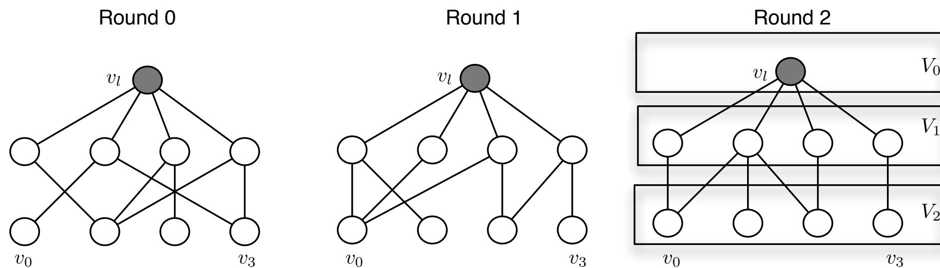

Figure 1: An example of a graph belonging to $\mathcal{G}(\text{PD})_2$ along three rounds

4 An asymptotically optimal algorithm for $\mathcal{G}(\text{PD})_2$

OPT initially starts a *get_distance* phase. At the end of this phase each process is aware of its distance from the leader. In $\mathcal{G}(\text{PD})_2$ this phase takes one round and it works as follow: Each process knows if it is the leader or not. This information is broadcast by each process (including the leader) to its neighbors at the beginning of round 0. Thus, at the end of round 0, each process knows if it belongs either to V_1 or to V_2.

Non-leader process behavior Starting from round 1, a process broadcasts its distance from the leader (i.e., 1 or 2) and each process v in V_2 builds its *degree history* $v.H(r)$ with $r \geq 0$ where $v.H(r)$ is an ordered list containing the number of neighbors of v belonging to V_1 at rounds $[0, \ldots, r-1]$. Thus $v.H(r) = [\bot, |N(v,1) \cap V_1|, \ldots, |N(v, r-1) \cap V_1|]$.

Starting from round $r > 0$, each $v \in V_2$ broadcasts $v.H(r)$. These histories are collected by each process $v' \in V_1$ and sent to the leader at the beginning of round $r + 1$.

Leader behavior Starting from the beginning of round $r \geq 2$ the leader receives degree histories from each process in V_1. The leader merges histories in a multiset denoted $v_l.M(r)$. Let us remark that $v_l.M(r)$ may contain the same history multiple times.

Data structure: The leader uses $v_l.M(r)$ to build a tree data structure T whose aim is to obtain $|V_2|$. For each distinct history $[A] \in v_l.M(r)$ the leader creates a node $t \in T$ with label $[A]$ and two variables $< m_{[A]}, n_{[A]} >$. $m_{[A]}$ denotes the number of histories $[A]$ in $v_l.M(r)$ and $n_{[A]}$ is the number of processes in V_2 that have sent $[A]$. Following the information flow, at round 2, $v_l.M(2)$ will be formed by a single history $[\bot]$ with multiplicity $m_{[\bot]}$. The leader creates the root of T with label $[\bot]$, value $m_{[\bot]}$, and $n_{[\bot]} =?$ (where ? means unknown value). It is important to remark that m values are directly computed from $v_l.M(r)$ while n values are set by the leader at a round $r' \geq r$ through a counting rule that will be explained later. The leader final target is to compute $n_{[\bot]}$ which corresponds to the number of processes in V_2.

At round $r + 2$ if the leader receives a history $h = [\bot, x_0, \ldots, x_{r-2}, x_{r-1}]$ and $n_{[\bot, x_0, \ldots, x_{r-2}]} =?$, then it creates a node in $t \in T$ with label h and value m_h, this node is a child of the node with label $[\bot, x_0, \ldots, x_{r-2}]$. Otherwise the leader ignores h. It is straightforward to see that the following equations hold:

$$\begin{cases} m_{[\bot,x_0,\ldots,x_{r-2},x_{r-1}]} = \sum_{i=1}^{|V_1|} i \cdot n_{[\bot,x_0,\ldots,x_{r-2},x_{r-1},i]} \\ n_{[\bot,x_0,\ldots,x_{r-2},x_{r-1}]} = \sum_{i=1}^{|V_1|} n_{[\bot,x_0,\ldots,x_{r-2},x_{r-1},i]} \end{cases} \tag{1}$$

Where $i \cdot n_{[\bot,x_0,\ldots,x_{r-2},x_{r-1},i]}$ means that the leader received i copies of history $[\bot, x_0, \ldots, x_{r-2}, x_{r-1}]$, one for each process in V_2 that at round $r + 1$ had history $[\bot, x_0, \ldots, x_{r-2}, x_{r-1}, i]$.

Counting Rule: When in T there is a non-leaf node with label $[\bot, x_0, \ldots, x_{r-2}, x_{r-1}, x_r]$ such that the leader knows the number of processes (i.e., $n_{[A]}$), for each of its children but one (i.e., $n_{[\bot,x_0,\ldots,x_{r-1},x_r,j]} =?$). Then the leader computes $n_{[\bot,x_0,\ldots,x_{r-1},x_r,j]}$ using $m_{[\bot,x_0,\ldots,x_{r-2},x_{r-1},x_r]} = \sum_{i=1}^{|V_1|} i \cdot n_{[\bot,x_0,\ldots,x_{r-2},x_{r-1},x_r,i]}$.

When the leader knows the values n for each of the children of a non leaf-node t, it sums the children values and sets the n_t (see the second equation of Eq. 1).

Due to the fact that the number of processes is finite, eventually there will be a non-leaf node in T with only one child (a leaf). Thanks to the counting rule, the n variables of the child and of

the father will be set. This will start a recursive procedure that will eventually set $n_{[\bot]}$ terminating the counting.

In Figure 2 is depicted an example run of the algorithm. In the Appendix the detailed pseudocode for T is provided.

Correctness proof

Lemma 1. *Let us consider the algorithm OPT. Eventually v_l sets a value for $n_{[\bot]}$ and this value is $|V_2|$.*

Proof. We first prove that eventually we reach a round in which the counting rule can be applied for any leaf of T. Let us consider the subtree of T rooted in the node with label $[A]$, if there is only one child then the counting rule can be applied and $n_{[A]}$ can be computed. Thus let us suppose that $[A]$ has at least two children with labels $[A, x], [A, x']$ with $x \neq x'$. We have that $n_{[A,x]} \geq 1$ and $n_{[A,x']} \geq 1$ because there must be at least one sending process for each degree-history. Considering that $n_{[A]} = \sum_{j=1}^{k} n_{[A,j]}$, it follows that $n_{[A,x]} \leq n_{[A]} - 1$. Iterating this reasoning we have that when the height of the subtree rooted in $[A]$ is greater than $n_{[A]} - 1$, then each leaf has no sibling: when there is a single process sending a certain degree history H, in the next round there will be only one degree history with H as suffix. As a consequence, after at most $n_{[A]}$ rounds, we may apply the counting rule for any leaf of the subtree rooted in $[A]$.

Now we prove by induction that: for each node $v \in T$ if $n_v \neq ?$, then n_v is equal to the number of processes in V_2 that had degree history equal to v at a given round.

Base case, leaf without siblings: Let $v_1 : [x_0, \ldots, x_{r+1}]$ be a leaf without siblings and $v_0 : [x_0, \ldots, x_r]$ its father. v_l sets, according to the counting rule, $n_{v_0} = n_{v_1} = \frac{m_{v_0}}{x_{r+1}}$. From Eq 1 we have $n_{v_0} = n_{v_1}$ which is equal to the number of processes in V_2 that had degree history $[x_0, \ldots, x_r]$.
Inductive case: Let us consider $v_0 : [x_0, \ldots, x_r]$ and the set of its children C_{v_0} with $|C_{v_0}| > 1$. Let introduce a set X_{v_0} formed by the children for which n is known and set, formally: $X_{v_0} : \{x \in C_{v_0} | n_x \neq ?\}$. If $\exists! v_1 : [x_0, \ldots, x_{r+1}] \in C_{v_0} \setminus X_{v_0}$, the leader sets (according to the counting rule) $n_{v_1} = \frac{m_{v_0} - \sum_{\forall [x_0,\ldots,x_k] \in X_{v_0}} (x_k \cdot n_{[x_0,\ldots,x_k]})}{x_{r+1}}$. By inductive hypothesis we have $\forall x \in X_{v_1}$, n_x is equal to the number of processes in V_2 with degree history equal to x. Due to Eq. 1, we have both n_{v_1} and n_{v_0} will be set to the correct value.

From the previous arguments we have that after at most $|V_2|$ rounds all the leaves of $[\bot]$ have no siblings, thus the counting rule will be applied recursively until the value $n_{[\bot]}$ is set to $|V_2|$. □

Theorem 1. *Let G be a dynamic graph of size $|V|$ belonging to $\mathcal{G}(PD)_2$. A run of* OPT *on G terminates in at most $\lceil log_2 |V| \rceil + 3$ rounds.*

Proof. Let consider the algorithm OPT. The latter counts processes in V_2, since the number of processes in V_1 is immediately known by v_l at round 0, thus let us suppose that we are in the worst case i.e., $|V_2| = \mathcal{O}(|V|)$. Let us consider the tree T, given a node $[A]$ the maximum height of the subtree rooted in $[A]$ is a function $h_{max}(n_{[A]})$. We have that h_{max} is non decreasing, $h_{max}(n_{[A]} - 1) \leq h_{max}(n_{[A]})$: let us consider the worst scheduling that the adversary uses with $n_{[A]} - 1$ processes in order to obtain the maximum height. It easy to show that the same scheduling can be created with $n_{[A]}$ processes, the adversary will simply force two processes to follow the behavior of a single process in the old schedule. Let us restrict to the case when $[A]$ has only two children: $[A, x], [A, x']$, for the counting rule $h_{max}(n_{[A]}) = min(h_{max}(n_{[A,x]}), h_{max}(n_{[A,x']})) + 1$. Considering the second equation of

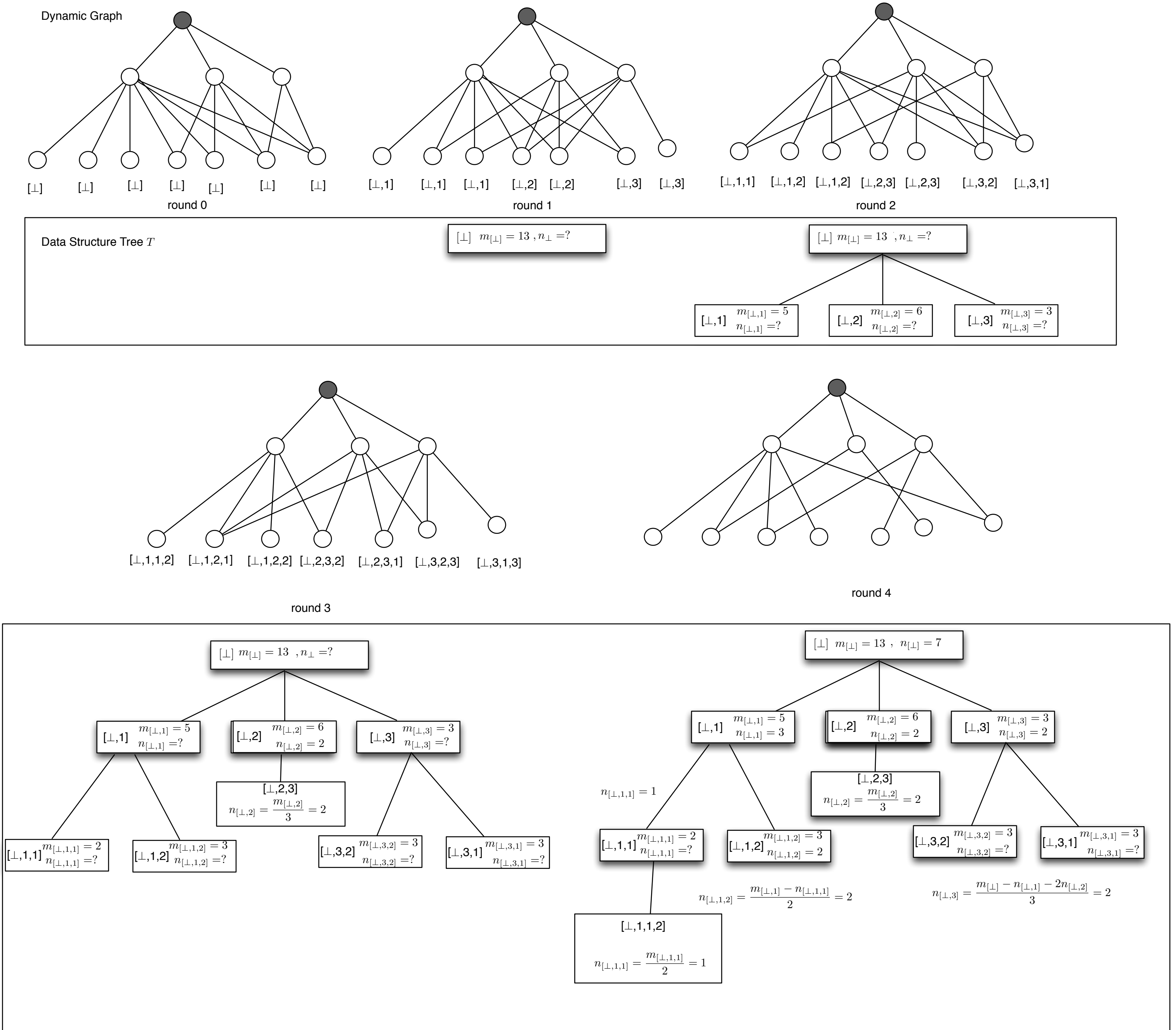

Figure 2: A run of OPT algorithm

Eq. 1, $h_{max}(n_{[A]})$ can be rewritten as follows: $h_{max}(n_{[A]}) = 1+min(h_{max}(\frac{n_{[A]}}{2}-\delta), h_{max}(\frac{n_{[A]}}{2}+\delta)) \leq 1+min(h_{max}(\frac{n_{[A]}}{2}), h_{max}(\frac{n_{[A]}}{2}))$ with $\delta \in [0, \frac{n_{[A]}}{2}]$. Thus, the optimal height can be reached by having $n_{[A,x]} = n_{[A,x']} = \frac{n_{[A]}}{2}$. Let us notice that when $[A]$ has more than two children, the maximum height of the subtree rooted in $[A]$ cannot be greater than the one obtained when $[A]$ has two children. Iterating this reasoning, in the worst case T is a balanced tree with degree at most 2 for each non leaf node and with exactly $|V|$ leaves. The height of this tree is $\lceil \log_2(|V|) \rceil$. Each level of T corresponds to one round of OPT, this completes the proof. □

A $\Omega(\log |V|)$ bound on $\mathcal{G}(\text{PD})_2$ has been shown in [13]. Therefore we have that OPT is asymptotically optimal.

5 High level view of $\mathcal{G}$(1-IC) counting algorithm

[18] and [17] conjectured: *It is impossible to compute (even with a leader) the predicate $Na \geq 1$, that is "exists an a in the input", in general anonymous unknown dynamic networks with broadcast.* In order to show that the conjecture is false we present a terminating counting algorithm, namely EXT, on $\mathcal{G}$(1-IC); this obviously implies the possibility to answer to any existence predicate confuting the conjecture.

Let us introduce the underlying structure we use to build EXT. The first conceptual step is to extend OTP to obtain a counting algorithm on $\mathcal{G}(\text{PD})_h$. We denote this extended algorithm OPT_h. As a second step, we consider networks in $\mathcal{G}$(1-IC). In such networks, at each round, processes can change their distance from the leader in $[1, V-1]$. When a process changes distance we say that the process *"moved"*.

5.1 Counting in $\mathcal{G}(\text{PD})_h$: OPT_h Algorithm

As OPT, OPT_h begins with a *get_distance* phase over $\mathcal{G}(\text{PD})_h$ where each process v obtains its distance from the leader, $v.distance$. This is done by using a simple flood and convergecast algorithm. After this phase the counting begin.

Each non leader process v keeps a degree history, where each element is the number of processes in $N(v,r)$ whose distance from v_l is $v.distance - 1$. Moreover v updates a multiset $v.M(r)$ that contains messages received by neighbors at distance $v.distance + 1$. The degree history and the multiset are broadcast at each round.

From an high level point of view the algorithm of v_l works as follow: the leader first computes the number of processes in V_1. Then by using messages sent by process in V_1, let MS_1 be this multiset[1], it executes OPT to count the processes in V_2. By counting processes in V_2 it also obtains the multiset MS_2 of messages sent by these processes. At this point, the leader simulates, using MS_2, an execution of OPT counting processes in V_3. Iterating this procedure till processes at distance h we obtain the final count.

Let us remark that OPT_h is an asymptotically time optimal algorithm for graphs in $\mathcal{G}(\text{PD})_h$. A more detailed explanation of OPT_h, with pseudo-code and formal proofs can be found in the Appendix.

[1] Due to anonymity multiple messages from different processes may be undistinguishable

5.2 Using $\mathcal{G}(\mathbf{PD})_h$ to count in $\mathcal{G}$(1-IC)

Let us introduce the notion of temporal subgraph G' of G:

Definition 5. *(Temporal Subgraph) Given a dynamic graph G, a dynamic graph G' is a temporal subgraph of G ($G' \subseteq G$) if and only if $G' : [G_{i_1}, G_{i_2}, \ldots]$ is an ordered subsequence of $G : [G_0, G_1, G_2, \ldots]$.*

We can show that in each $G \in \mathcal{G}$(1-IC) there exists a temporal subgraph G' that belongs to $\mathcal{G}(\mathrm{PD})_h$:

Lemma 2. *Let us consider a dynamic graph $G : [G_0, G_1, G_2, \ldots] \in \mathcal{G}$(1-IC). There exists $h \in \mathbb{N}^+$ and $\exists G' \subseteq G$ such that G' is infinite and $G' \in \mathcal{G}(PD)_h$.*

Now, let us define a counting algorithm InstanceCount. Such algorithm works on $G \in \mathcal{G}$(1-IC) and it has two properties: (P1) it terminates giving the correct count on instance $G' \in \mathcal{G}(\mathrm{PD})_h$; (P2) it does not give an incorrect count on $G' \notin \mathcal{G}(\mathrm{PD})_h$. Thus, if $G' \notin \mathcal{G}(\mathrm{PD})_h$ it can terminate giving either a correct count of the network or a special invalid value, i.e. INVCNT. The strategy of EXT is to run a different instance of InstanceCount on each temporal subgraph of G. Due to properties (P1) and (P2), EXT terminates correctly when an instance of InstanceCount outputs a valid count value. For the property (P1) and for Lemma 2, one instance of InstanceCount outputs a valid count value. Consequently, EXT is a correct terminating counting algorithm.

InstanceCount counts as if the network is in $\mathcal{G}(\mathrm{PD})_h$. Therefore, the leader first counts processes in V_1, then processes in V_2 and so on. This is done until v_l counts processes of a set V_h such that no set V_{h+1} exists. The tricky part is to detect if the counting algorithm is operating on a network in $\mathcal{G}(\mathrm{PD})_h$. In the affirmative, the count done with such strategy will be correct. The procedure that counts each set V_j is a special algorithm, namely VCD. VCD allows to detect if the count obtained for V_j is correct, returning the count value, or if it is not possible to count V_j because some process moved during the counting, returning NOCOUNT. The VCD algorithm is explained in the next Section.

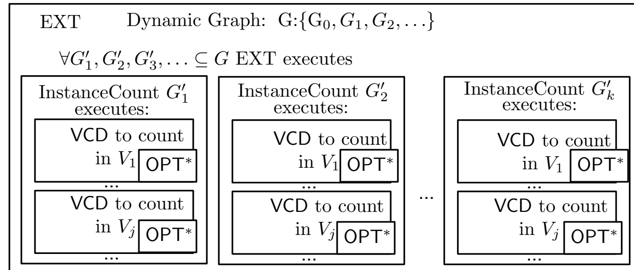

Figure 3: Counting algorithm EXT and the relationship among its subalgorithms. Algorithms VCD and OPT* are explained in Section 6; InstanceCount in Section 7. Finally, EXT is presented in Section 8

6 Valid Count Detection Algorithm (VCD)

Let us considering a network where processes in V_1 do not change distance from the leader along rounds. Remaining processes, including a proper subset V_2^M of processes in V_2 at round 0, may change distance along rounds. We wish to build an algorithm that solves this problem: *if no processes in V_2^M move during the counting, then the algorithm outputs the correct count of processes at distance 2 at round 0. Otherwise, the algorithm outputs either the correct count or a special invalid value.* Unfortunately, in case processes change their distance from the leader, OPT might fail outputting a wrong count.

One strategy to build such algorithm could be to first use OPT then, after OPT termination, to start a waiting phase in order to receive messages from processes that could have moved. Sadly, this simple OPT-based strategy does not work. If v_l outputs the final count at round r, the message from a process that moved could arrive at round $r+1$, invaliding the count. Thus, v_l should wait for some time before outputting the count but this time cannot be bounded as the size of the network is unknown. From this point of view, if a process changes distance across rounds in a network of unknown size it is like if this process halts and it does not send anymore messages. We denote this problem as Valid Count Detection.

Valid Count Detection Problem (VCDP) Let us consider a graph in $\mathcal{G}(\mathrm{PD})_2$ where processes in V_2 may halt at some point. We say that v_i *halts at round* r if it has send messages for any round $r'' < r$, and it does not send messages for any $r' \geq r$. We assume that processes halt from round $r \geq 1$; that is they send at least one message before their departure. Now we introduce the **VCDP** problem on $\mathcal{G}(\mathrm{PD})_2$:

Problem 1. VCDP *Given two run R, R_{NC} such that: in the run R no process halts; in the run R_{NC} there are processes that halt. An algorithm solves the Valid Count Detection Problem if at some round r it outputs a value and terminates. The output could be either a special value* NOCOUNT *or a number $C = |V_2|$. On run R the output value has to be C. On run R_{NC} it could be either C or* NOCOUNT.

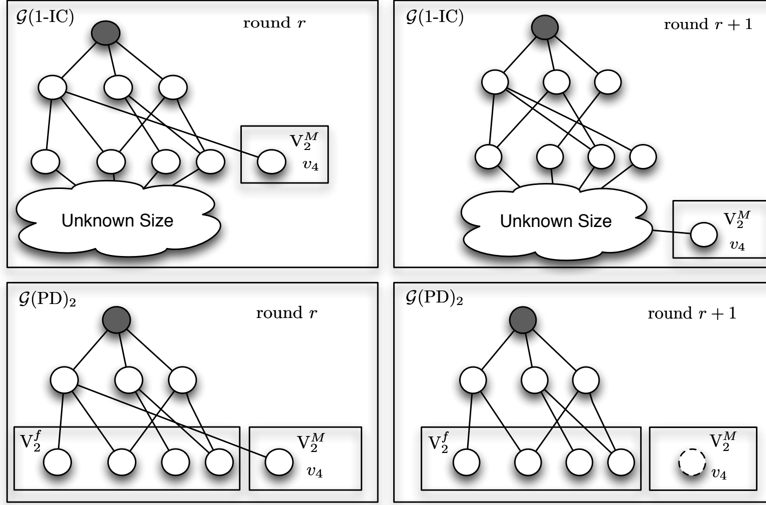

Figure 4: In general $\mathcal{G}$(1-IC) a subset V_2^M of processes in V_2 may move changing the distance from the leader and invalidating the correct count of processes in V_2. Network size is unknown therefore messages from process v_4 need an unknown number of rounds to reach the leader. We are interested in an algorithm that detects this using information from processes in $V_2 \setminus V_2^M$. This is equivalent to solve the problem on a networks in $\mathcal{G}(\mathrm{PD})_2$ where the subset V_2^M stops sending messages after a certain round. In the example process v_4 halts at round $r+1$.

The VCD Algorithm to solve VCDP

When identifiers are present a simple broadcast algorithm solves **VCDP** in $\mathcal{G}(\mathrm{PD})_2$. In our model we solve it by using an extension OPT, denoted as OPT*. When processes halt OPT* has a peculiar *"overestimation"* property (see Lemma 4).

Algorithm OPT* The algorithm OPT* differs from OPT in:

- Its output is considered not valid if we have one of the following: (i) the value n computed for some node of the tree is not in $\mathbb{N}^+$; (ii) if some of the Equations 1 are violated, i.e. $m_{[\perp,x_0,\ldots,x_{r-2},x_{r-1}]} \gtrless \sum_{i=1}^{|V_1|} i \cdot n_{[\perp,x_0,\ldots,x_{r-2},x_{r-1},i]}$; (iii) if at round $r+2$ there exists a node in T with label $[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r]$ and at round $r+3$ does not exists a node with label $[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r, *]$.
- Its counting rule is a restricted version of the OPT counting rule. Specifically: when in T there is a non-leaf node with label $[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r]$ such that *it has only one child* $[\perp, x_0, \ldots, x_{r-1}, x_r, j]$ the leader computes $n_{[\perp,x_0,\ldots,x_{r-1},x_r,j]}$ using:

$$m_{[\perp,x_0,\ldots,x_{r-2},x_{r-1},x_r]} = j \cdot n_{[\perp,x_0,\ldots,x_{r-2},x_{r-1},x_r,j]}$$

  When the leader knows the values n for each of the children of a non leaf-node t, it sums the children values and sets the n_t (see the second equation of Eq. 1).

Algorithm OPT^* has the following properties:

Lemma 3. *Let R be a run produced by OPT^*. R terminates in $\mathcal{O}(|V_2|)$ rounds.*

Lemma 4. *Let R be a run produced by OPT^* that starts at round 0 and $|V_2^f|$ be the number of non-halted processes in V_2 at the end of the execution of OPT^*. If at some round $r > 0$ processes in V_2 halt, then if the output C of OPT^* is valid we have $C > |V_2^f|$.*

Informally the previous Lemma says that, if there are halted processes, the output of OPT^* is always an overestimate on the number of non-halted processes. The following lemma states that if no process halts then the output is the number of processes.

Lemma 5. *Let R be a run produced by OPT^* that starts at round 0. If no process in V_2 halts during the run, then the output of OPT^* is valid and it is the correct count of processes in V_2.*

Algorithm VCD The algorithm executes sequentially k runs of OPT^* starting from round 0, for some $k > |V_2|$. The leader compares the output of these runs: if they are all equal and valid, then VCD outputs the count obtained by the first run of OPT^*. Otherwise VCD outputs NOCOUNT. The value k is computed by counting the edges connecting processes in V_1 with processes belonging to V_2 at round 0. This can be done trivially by v_l using messages from nodes in V_1. Each node in V_1 has to simply count neighbors in V_2, the sum of these partial count is equal to $k - 1$.

Theorem 2. *Algorithm* VCD *solves the* **VCDP** *problem.*

7 InstanceCount

This algorithm assumes that the communication graph belongs to $\mathcal{G}(\text{PD})_h$ then if InstanceCount notices that some process changed the distance from the leader along rounds, it invalids the count.

Non-leader process behavior (Figure 5) Each non leader process v has three variables: $v.distance$ indicating its distance from the leader and two lists $v.M$ and $v.H$. v assigns a value to $v.distance$ as follows: if, at round r, v has $v.distance = -1$ and it is neighbor of a process with $distance = r \neq -1$, v sets its distance to $r + 1$ (Line 9). Initially, the leader is the only process with $distance = 0$. As in OPT, v updates its degree history $v.H(r)$ by counting the number of processes in $N(v, r)$ whose distance is equal to $v.distance - 1$. Moreover v updates a multiset $v.M(r)$ that contains messages received by neighbors at distance $v.distance + 1$; if v has not received any of these messages, it adds $\perp$ to the multiset. In the sending phase, v broadcasts $< v.distance, v.M(r), v.H(r) >$ to its neighbors. This is done by using functions $count_distance_neighbors$ and $get_messages_from_distance$.

A process that has $distance = r$ adds the messages from processes with $distance = -1$ to M list, let us recall that these processes with $distance = -1$ will set $distance = r + 1$ at round r. Finally at Line 16 a process adds an INVCNT message to M if it detects that at least one its neighbor changed its distance from the leader which implies that the communication graph is not in $\mathcal{G}(\text{PD})_h$ (see condition at Line 15). In the following when we refer to the set V_h, we consider processes setting their distance from the leader to h.

```
1: M(−1) = []
2: H(−1) = [⊥]
3: distance = −1
4:
5: procedure SENDING_PHASE
6:     send(Message :< distance, M(r), H(r) >)
7:
8: procedure RCV_PHASE(MultiSet MS)
9:     if distance == −1 ∧ ∃m ∈ MS | m.distance ≠ −1 ∧ m.distance == r then
10:        distance=m.distance+1
11:    if r == distance then
12:        for all m ∈ MS | m.distance == −1 do
13:            m.distance=distance+1
14:    if distance ≠ −1 then
15:        if r > distance ∧ ∃m ∈ MS | m.distance ∉ {distance − 1, distance, distance + 1} then
16:            M(r + 1) = M(r).append(INVCNT)
17:        H(r + 1) = H(r).append(count_distance_neighbors(MS, distance − 1))
18:        M(r + 1) = M(r).append(get_messages_from_distance(MS, distance + 1))
```

Figure 5: InstanceCount for $\mathcal{G}$(1−IC): pseudocode for Non-Leader process

Leader process behavior (Figure 6) The leader v_l first computes the number of processes in V_1, this is simply done by counting the messages received from these processes. After that, v_l executes VCD to count processes in V_2. This is done (i) by receiving the multi-set of messages MS from processes in V_1 (these processes are immediate neighbors of v_l) and (ii) by calling at Line 16 the function BUILDLASTSET. This function takes the multi set MS and starts an instance of VCD to construct the multi-set MS_{last} of messages sent by processes in V_2. We define as VCD(MS, r) the local leader side simulation of a run of VCD that starts at round r using the content of messages in MS. The function returns one out of three possible values: (i) $\perp$ if the messages in MS are not enough to terminate the execution of VCD; (ii) NOCOUNT if VCD detects an halt ; (iii) A multi-set MS_{last} of messages sent by processes belonging to V_2 at round r.

This multi-set leads to the actual count of processes in V_2 (see Line 21). This procedure is iterated: each time the leader obtains the multi-set MS sent by processes in V_{h-1}, v_l calls BUILDLASTSET to reconstruct the most recent multi-set sent by processes in V_h.

The leader returns INVCNT if either (i) there is a INVCNT message in some MS (see Lines 26) or (ii) if one of the instances of VCD terminates returning NOCOUNT. If an halt is detected then a process $v \in V_j$ at some round had a distance from v_l different than j. Additionally, at Line 8 the leader checks if processes in V_1, from which it receives messages, are stable; if this set changes the current instance is considered INVCNT.

The leader outputs the count when it counts a set V_h such that no process in V_h has a neighbor in V_{h+1}, see Line 13.

Correctness Proof

Lemma 6. *Let R be a run of* InstanceCount *on a dynamic graph $G \in \mathcal{G}(PD)_h$. We have that v_l will never output* INVCNT *in R.*

Lemma 7. *Let R be a run of* InstanceCount *on a dynamic graph $G \in \mathcal{G}$(1-IC). If $V_h \neq \emptyset$ in R, either (1) the leader obtains the count V_h or (2) the leader outputs* INVCNT.

```
1: distance_count[] = ⊥
2: distance = 0
3: procedure SENDING_PHASE
4:     send(< distance, ⊥, ⊥ >)
5:
6: procedure RCV_PHASE(MultiSet MS :< distance, M, H >)
7:     i = 1
8:     if (distance_count[i] ≠ ⊥ ∧ distance_count[i] ≠ |MS|) ∨ (∃m ∈ MS|m.distance > 1) then
9:         output(INVCNT)
10:    distance_count[i] = |MS|
11:    i++
12:    while true do
13:        if MS ≠ ∅ ∧ (∀m ∈ MS : m.M = [⊥, ..., ⊥] ∧ size(m.M) > 1) then
14:            count = Σ_{∀j|distance_count[j]≠⊥} distance_count[j]
15:            output(count)
16:        MS = BUILDLASTSET (MS)
17:        if ∃INVCNT ∈ MS then
18:            output(INVCNT)
19:        if MS = ⊥ then
20:            break
21:        distance_count[i] = |MS|
22:        i++
23:
24: function BUILDLASTSET(MS)
25:    MS_last = ⊥
26:    if MS.containsSymbol(INVCNT) then
27:        return {INVCNT}
28:    for r = MinRound(MS); r < MaxRound(MS); r++ do
29:        if VCD(MS, r) == NOCOUNT then
30:            return {INVCNT}
31:        if VCD(MS, r) ≠ ⊥ then
32:            if MS_last ≠ ⊥ ∧ |MS_last| ≠ | VCD(MS, r)| then
33:                return {INVCNT}
34:            MS_last = VCD(MS, r)
35:        else
36:            break
37:    return MS_last
```

Figure 6: InstanceCount for $\mathcal{G}$(1−IC): pseudocode for Leader process

Lemma 8. *Let R be a run of* InstanceCount *on a dynamic graph $G \in \mathcal{G}$(1-IC). If v_l outputs a value distinct from* INVCNT *in R, then that value is $|V|$.*

Lemma 9. *Let R be a run of* InstanceCount *on a dynamic graph $G \in \mathcal{G}(PD)_h$. We have that v_l terminate and it outputs $|V|$ in R.*

8 EXT Counting Algorithm

EXT executes an instance of InstanceCount for each temporal subgraph of G. Let us define as $\mathcal{P}_G$ as the set of such subgraphs of G. We want that processes execute for each $G' \in \mathcal{P}_G$ a different instance $I_{G'}$ of InstanceCount and that such instances do not interfere with each other. Let us remark that the system is synchronous and the current round number r is known by all processes. Therefore each $I_{G'}$ is uniquely identified by a binary string that has value 1 in position j if $G_{r_j} \in G'$ and 0 otherwise.

The uniqueness guarantees that instances can run in parallel. At each new round r the number of instances is doubled, half of the new instances will consider the messages exchanged within round r and the remaining ones will not consider these messages. As example at the end round 0 we have two instances I_1, I_0. In instance I_1 the counting is started and processes have received the message exchanged in G_0. In instance I_0 the counting has not been started, the messages exchanged in round 0 are ignored. At round 1 we have four instances $I_{11}, I_{10}, I_{01}, I_{00}$: I_{11} is an instance of counting in which messages exchanged in G_0, G_1 are considered; in I_{10} are considered only messages exchanged in G_1 and ignored messages exchanged in G_0; in I_{01} are considered only messages exchanged in G_0 and ignored messages exchanged in G_1; in I_{00} the counting has not been started. The pseudocode to implement the this procedure is trivial, thus it is omitted.

Theorem 3. *Let R be a run of* EXT *on a dynamic graph $G \in \mathcal{G}$(1-IC). Eventually, v_l terminates and it outputs the correct count in R.*

From the previous Theorem and from the impossibility of non trivial computation without a leader presented in [17, 18] we have:

Theorem 4. *Let us consider an anonymous unknown 1-interval connected networks with broadcast. A distinguished leader process is necessary and sufficient to do non trivial computations.*

Besides counting and existence predicates other non-trivial problems are solvable using simple variation of EXT. Let us assume that each process has an initial input value. If this initial input is attached in the messages of EXT the leader can compute the exact multiset of these values. Thanks to this multiset the leader may compute aggregation functions as average,min,max.

Complexity Discussion The EXT algorithm has an exponential complexity: Let us consider our $G : [G_0, G_1, \dots] \in \mathcal{G}(1\text{−IC})$ and $G' \subseteq G$ with $G' : [G'_{i_0}, G_{i_1}, \dots] \in \mathcal{G}(\text{PD})$. Let compute an upper bound on $max_j(|i_j - i_{j+1}|)$ with $G_{i_j}, G_{i_{j+1}} \in G'$. If we consider that distances of each node from v_l are in $[1, |V| - 1]$, then it is easy to see that the number of possible combinations of distances over the set of nodes is upper bounded by $|V|^{|V|}$, therefore by definition of $\mathcal{G}(\text{PD})$ we have $max_j(|i_j - i_{j+1}|) \leq |V|^{|V|}$. Now what we have to bound is the number of instances of G' needed by EXT to terminate, but this can be easily computed by considering when counting terminate with InstanceCount on a graph $\mathcal{G}(\text{PD})$. At each level we count in at most $\mathcal{O}(|V|^3)$ rounds, therefore it is easy to show by straightforward induction that the total cost is $\mathcal{O}(|V|^4)$. So EXT terminates in at most $\mathcal{O}(|V|^{|V|+4})$ rounds.

9 Conclusion

In this paper we have shown that, in anonymous interval connected network with broadcast, a leader node is enough to do non trivial computations. This answers negatively to the conjecture presented in [17, 18]. Moreover we have shown an optimal counting algorithm for $\mathcal{G}(\text{PD})_2$ networks, proving the tightness of the bound shown in [13]. However, our EXT algorithm has an exponential complexity, both in memory and in the number of rounds. In $\mathcal{G}$(1-IC) networks with IDs, when there is unlimited bandwidth, counting requires $O(|V|)$ rounds. It is unknown if handling anonymity in $\mathcal{G}$(1-IC) requires this exponential cost. A future line of work could be the investigation of this gap.

References

[1] D. Angluin. Local and global properties in networks of processors (extended abstract). In *STOC '80*, pages 82–93. ACM, 1980.

[2] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The price of validity in dynamic networks. *J. Comput. Syst. Sci.*, 73(3):245–264, May 2007.

[3] Joffroy Beauquier, Janna Burman, Simon Clavière, and Devan Sohier. Space-optimal counting in population protocols. In *(to appear) DISC '15*, 2015.

[4] P. Boldi and S. Vigna. Computing anonymously with arbitrary knowledge. In *PODC '99*, pages 181–188. ACM, 1999.

[5] P. Boldi and S. Vigna. Fibrations of graphs. *Discrete Mathematics*, 243(1-3):21–66, 2002.

[6] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *CoRR*, abs/1012.0009, 2010.

[7] Pierre Fraigniaud, Andrzej Pelc, David Peleg, and Stphane Prennes. Assigning labels in an unknown anonymous network with a leader. *Distributed Computing*, 14(3):163–183, 2001.

[8] M. Jelasity, A. Montresor, and Ö. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, 2005.

[9] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *FOCS '03*, pages 482–491. IEEE, 2003.

[10] J. Kong, X. Hong, and M. Gerla. An identity-free and on-demand routing scheme against anonymity threats in mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 6(8):888–902, 2007.

[11] F. Kuhn, N. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *STOC '10*, pages 513–522. ACM, 2010.

[12] F. Kuhn and R. Oshman. Dynamic networks: Models and algorithms. *SIGACT News*, 42(1):82–96, March 2011.

[13] G. Di Luna and R. Baldoni. Brief announcement: Investigating the cost of anonymity on dynamic networks. In *PODC '15*, pages 339–341. ACM, 2015.

[14] G. Di Luna, R. Baldoni, S. Bonomi, and I. Chatzigiannakis. Conscious and unconscious counting on anonymous dynamic networks. In *ICDCN '14*, pages 257–271. Springer, 2014.

[15] G. Di Luna, R. Baldoni, S. Bonomi, and I. Chatzigiannakis. Counting in anonymous dynamic networks under worst case adversary. In *ICDCS '14*, pages 338–347. IEEE, 2014.

[16] L. Massoulié, E. Le Merrer, A.-M. Kermarrec, and A. Ganesh. Peer counting and sampling in overlay networks: Random walk methods. In *PODC '06*, pages 123–132. ACM, 2006.

[17] O. Michail, I. Chatzigiannakis, and P. Spirakis. Brief announcement: Naming and counting in anonymous unknown dynamic networks. In *DISC '12*, pages 437–438. Springer, 2012.

[18] O. Michail, I. Chatzigiannakis, and P. Spirakis. Naming and counting in anonymous unknown dynamic networks. In *SSS '13*, pages 281–295. Springer, 2013.

[19] O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Causality, influence, and computation in possibly disconnected synchronous dynamic networks. In *OPODIS '12*, pages 269–283, 2012.

[20] D. Mosk-Aoyama and D. Shah. Computing separable functions via gossip. In *PODC' 06*, pages 113–122. ACM, 2006.

[21] R. O'Dell and R. Wattenhofer. Information dissemination in highly dynamic graphs. In *DIALM-POMC' 05*, pages 104–110, 2005.

[22] B. Ribeiro and D. Towsley. Estimating and sampling graphs with multidimensional random walks. In *IMC '10*, pages 390–403, New York, NY, USA, 2010. ACM.

[23] Naoshi Sakamoto. Comparison of initial conditions for distributed algorithms on anonymous networks. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, PODC '99, pages 173–179. ACM, 1999.

[24] M. Yamashita and T. Kameda. Computing on an anonymous network. In *PODC '88*, pages 117–130. ACM, 1988.

[25] M. Yamashita and T. Kameda. Computing on anonymous networks: Part 1-characterizing the solvable cases. *IEEE Trans. on Parallel and Distributed Systems*, 7(1):69–89, 1996.

Appendix

Proofs omitted in the main text

Lemma 2. *Let consider a dynamic graph $G : [G_0, G_1, G_2, \ldots] \in \mathcal{G}(1\text{-}IC)$. We have that exists $h \in \mathbb{N}^+$ and $\exists G' \subseteq G$ such that G' is infinite and $G' \in \mathcal{G}(PD)_h$.*

Proof. The proof is by contradiction. Given a $G : [G_0, G_1, \ldots] \in \mathcal{G}(\text{1-IC})$ let us assume that each distinct graph $G_j \in G$ appears a bounded number of times, let us say $m_{G_j} \in \mathbb{N}^+$. Now let us consider the set X of all possible graphs of $|V|$ processes, clearly we have that this set is finite. Now let us consider the round $x = \sum_{\forall G_j \in X} m_{G_j} + 1$ and the sub-sequence $S : [G_0, G_1 \ldots G_x]$ of G, let us consider the set of distinct graphs X_s of S, we have that $|S| \leq \sum_{\forall G_j \in X_s} m_{G_j} \leq \sum_{\forall G_j \in X} m_{G_j}$ but $|S| = x + 1$ that is a contradiction. Thus exists at least one graph $G_j \in G$ that appears in G an infinite number of times.

Let us consider the subsequence $G' = [G_{r_0}, G_{r_1}, G_{r_2}, \ldots]$ of G such that each $G_{r_i} = G_j$. It is clear that $G' \in \mathcal{G}(\text{PD})_h$ for some $h \leq \mathit{Diameter}(G_j)$ and that G' is infinite. □

Lemma 3. *Let R be a run produced by OPT^* that starts at round 0 we have that R terminates in $\mathcal{O}(|V_2|)$ rounds.*

Proof. The counting rule on leaf is applied only when the leaf has no siblings. Let x_0 be a node with at least two children x_1, x_2. We have by construction that the number of processes with history x is strictly greater that the number of processes with history x_1 and with history x_2. Since a node x may have more then one child if and only if the number of processes with history x is greater then one we have that after at most $\mathcal{O}(|V_2|)$ either each leaf has no siblings or the run R is not valid. In both cases OPT^* terminates. □

Lemma 4. *Let R be a run produced by OPT^* that starts at round 0 and $|V_2^f|$ be the number of non-halted processes in V_2 at the end of the execution of OPT^*. If at some round $r > 0$ processes in V_2 halt, then if the output C of OPT^* is valid we have $C > |V_2^f|$.*

Proof. For Lemma 3 OPT^* terminates we have to show that its output satisfies the lemma statement. Without loss of generality we consider only the case of a valid output of the algorithm. Let $l_{[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r]}$ be the number of processes with degree history $[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r]$ that halt at round $r + 1$. Let us first consider the application of the counting rule on a leaf $[\perp, x_0, \ldots, x_{r-1}, x_r, j]$ of T with father $[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r]$. For the counting rule of OPT^* we have that the leaf has no siblings. We have $m_{[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r]} = L + j(n_{[\perp, x_0, \ldots, x_{r-1}, x_r, j]} - l_{[\perp, x_0, \ldots, x_{r-1}, x_r, j]})$. Where $L = \sum_{\forall a : [\perp, x_0, \ldots, x_r, i] | l_a > 0} i \cdot l_{[\perp, x_0, \ldots, x_r, i]} \geq 0$ is the number of edges from halted processes that have been counted in $m_{[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r]}$. We have $(n_{[\perp, x_0, \ldots, x_{r-1}, x_r, j]} - l_{[\perp, x_0, \ldots, x_{r-1}, x_r, j]}) > 0$ otherwise the outputs is not valid. Thus the leader computes a value for the node with label $[\perp, x_0, \ldots, x_{r-1}, x_r, j]$ that is $n^*_{[\perp, x_0, \ldots, x_{r-1}, x_r, j]} = \frac{L}{j} + n_{[\perp, x_0, \ldots, x_{r-1}, x_r, j]} \geq n_{[\perp, x_0, \ldots, x_{r-1}, x_r, j]}$. If $L > 0$ the assigned value is clearly greater w.r.t to the actual number of non-halted processes with that degree history. The other case that we have to examine in our induction on T is when the counting rule sets the value of a non-leaf node $[\perp, x_0, \ldots, x_r]$. In this case we have that if $\exists\, l_{[\perp, x_0, \ldots, x_r, y]} > 0$ then $m_{[\perp, x_0, \ldots, x_r]} = L + \sum_1^{|V_1|} i(n_{[\perp, x_0, \ldots, x_r, i]} - l_{[\perp, x_0, \ldots, x_r, i]})$ thus if each $n_{[\perp, x_0, \ldots, x_r, i]}$ is not set to an overestimate of the number of processes with degree history $[\perp, x_0, \ldots, x_r, i]$ we would have $m_{[\perp, x_0, \ldots, x_r]} > \sum_1^{|V_1|} i(n_{[\perp, x_0, \ldots, x_r, i]} - l_{[\perp, x_0, \ldots, x_r, i]})$ leading to a non

valid output. From these observations it is easy to see that each $v \in V_2^f$ is counted at least once in the value associated to some leaf node. If $\exists l_{[X]} > 0$ we have, since the leader aggregates the value of nodes in T towards the root, that $n_{[\perp]} > |V_2^f|$.

□

Lemma 5. *Let R be a run produced by* OPT* *that starts at round* 0*. If no process in V_2 halts during the run, then the output of* OPT* *is valid and it is the correct count of processes in V_2.*

Proof. For Lemma 3 OPT* terminates we have to show that its output satisfies the lemma statement. When no process halts we have that the proof of correctness follows the same step of the correctness proof of OPT (Lemma 1): the base case is the same, the inductive case it is slightly different in the fact that the condition to set node value has to be $\nexists v_1 : [x_0, \ldots, x_{r+1}] \in C_{v_0} \setminus X_{v_0}$. It is straightforward that the modification on the counting rule only impacts on counting time, i.e. $\mathcal{O}(|V_2|)$ instead of $\mathcal{O}(\log |V_2|)$ (see proof of Th. 1 of OPT), and not on its correctness. □

Theorem 2. *Algorithm* VCD *solves the* **VCDP** *problem.*

Proof. Let $\{i_0, i_1, \ldots, i_k\}$ be k runs of VCD. Let us first consider the execution of VCD on R. In R no process halts, thus we have for Lemma 5 that all outputs $\{c_0, c_1, \ldots, c_k\}$ will be equal. Therefore VCD terminates outputting $c_0 = |V_2|$. Let us consider the execution of VCD on R_{NC}. Let us define as $\{|V_2^{f_0}|, |V_2^{f_1}|, \ldots, |V_2^{f_k}|\}$ the number of non-halted processes in the system at the end of instance $1, 2, \ldots$. Now two cases may arise:

- $|V_2| = |V_2^{f_0}|$, in this case for Lemma 5 we have $c_0 = |V_2|$. Therefore if all $\{c_0, c_1, \ldots, c_k\}$ are equal and valid then VCD outputs the correct count otherwise the algorithm outputs NOCOUNT. In any case the output is correct.
- $|V_2| > |V_2^{f_0}|$ in this case by using Lemma 4 we have $c_0 > |V_2^{f_0}|$. Processes in V_2 are less than k this implies that we have at least one instance i_j for which no process halts during its execution. For Lemma 5 the instance i_j outputs the value $c_j = |V_2^{f_{j-1}}| = |V_2^{f_j}| \neq c_0$. Thus the algorithm outputs NOCOUNT on R_{NC}. That is a correct output.

□

Lemma 6. *Let R be a run of* InstanceCount *on a dynamic graph $G \in \mathcal{G}(PD)_h$. We have that v_l will never output* INVCNT *in R.*

Proof. The leader returns INVCNT at Line 18. The line is executed either (a) if an halt is detected by VCD, i.e. a process $v \in V_h$ at some round $r > h-1$ is not neighbor of processes in V_{h-1}, or (b) if some set MS contains a INVCNT element. The latter happens if a non leader-process v' executes Line 16-Figure 5, that is v' has a neighbor with distance value that is not in $\{v'.distance - 1, v'.distance, v'.distance + 1\}$. By definition of $\mathcal{G}(\text{PD})_h$ condition (a) cannot happen on G, see Th 2 , the same holds for condition (b). Both conditions would implies that a process v is at distance h in a graph $G_j \in G$ and at distance $h' \neq h$ in $G_i \in G$ with $i > j$. Therefore the claim follows. □

Lemma 7. *Let R be a run of* InstanceCount *on a dynamic graph $G \in \mathcal{G}(1\text{-}IC)$. If $V_h \neq \emptyset$ in R, either (1) the leader obtains the count V_h or (2) the leader outputs* INVCNT.

Proof. The processes in V_h set their distance at round $r = h - 1$, see Line 9 of Figure 5. At the same round an instance of VCD between processes in V_{h-1}, V_h is started, see Line 12 of Figure 5 where messages from Nodes in V_h are taken into account by Nodes in V_{h-1} starting from round $r = h - 1$.

Now we discuss how VCD could be used to reconstruct the exact multi set of memory of processes in V_2. This is equivalent to consider a case where each process v in V_2 have a certain initialisation input value i_v, and where we want to compute the multiset of these values when no process in V_2 halts. Essentially, the value i_v is attached to each algorithm message. The leader starts a different instance of VCD for each of these values. Each VCD instance outputs the multiplicity of a certain value, this is due to Th. 2.

Let us assume that processes in $V_0, \ldots, V_{h-1}$ do not change distance we have for Th. 2 that if no process in V_h moves the set of processes in V_{h-1} will eventually obtain the multiset MS_h . Now the multiset of messages MS_h sent at round $h - 1$ by process in V_h will be propagated from processes in V_h to v_l during rounds $[h - 1, \ldots, r_{count_h}]$. At each different frontier between distances this is done by using other instances of VCD: between processes in V_{h-1}, V_{h-2}, processes in V_{h-2}, V_{h-3} and so on.

Now let us consider condition (a) that is instances of VCD between processes in $V_0, \ldots, V_h$ that are propagating towards the leader MS_h never detect an invalid count and no processes in $V_0, \ldots, V_h$ has an INVCNT element in its multiset of messages MS. If condition (a) holds we have that the leader will obtains the correct count of processes in V_h by reconstructing the multiset of messages MS_h sent by process in V_h at round $h - 1$. This is ensured by Th. 2 of algorithm VCD and by a simple induction on the count for each set V_i.

Otherwise if condition (a) does not hold the leader will output INVCNT. This is done either because of Line 26 or because some instance of VCD terminated with NOCOUNT before the leader is able to reconstruct MS_h.

□

Lemma 8. *Let R be a run of* InstanceCount *on a dynamic graph $G \in \mathcal{G}(1\text{-}IC)$. If v_l outputs a value distinct from* INVCNT *in R, then that value is $|V|$.*

Proof. The Leader terminates at Line 13, that is the leader has reconstructed a multiset MS_h from processes in V_h such that for each $M \in MS_h$: M contains at least two elements and M contains only $\perp$ value. The condition on the size implies that MS_h has been sent at round $r' \geq h$. For Lemma 7 we have that if this happen the leader has correctly counted processes in $V_0, \ldots, V_h$, we have to show that when Line 13 is triggered we have $V \setminus V_0 \setminus \ldots \setminus V_h = \emptyset$.

Let us assume Line 13 is executed and that it exists $v \in V \setminus V_0 \setminus \ldots \setminus V_h$. We must have, for connectivity assumption, that such v at round r' is neighbor of some process in $V_0, \ldots, V_h$. If it is neighbor of a process $v_1 \in V_j$ then v_1 will put INVCNT in $v_1.M(r')$. In order to reconstruct MS_h the leader will also reconstruct the multiset of messages MS_j sent at round r'; two things may happen:

- (1) That the leader receives the INVCNT message thus the Line 26 will be triggered, then the leader outputs INVCNT and the Line 13 will not be executed;
- (2) That v_1, or some other process that is sending the INVCNT message to v_l, is moved away. This triggers the detection in VCD during the reconstruction of MS_h therefore Line 13 is not executed. If v is neighbor of processes in V_h and $r' > h$ we have the same behavior of the previous case.

The only possibility left is v neighbor of processes in V_h and $r' = h$. In this case at least one process v' in V_h will execute Line 12 setting $v'.M(h) = [\bot, \neg\bot]$ therefore Line 13 cannot be executed.

□

Lemma 9. *Let R be a run of* InstanceCount *on a dynamic graph $G \in \mathcal{G}(PD)_h$. We have that v_l terminate and it outputs $|V|$ in R.*

Proof. For Lemma 6 v_l will never outputs INVCNT. This means that, see Lemma 7, the leader eventually obtains the count for each set $V_1, \dots, V_h$. Since the set V_{h+1} is empty we have that v_l executes the terminating condition when it obtains MS_h, see Line 13. For Lemma 8 the leader output is correct. □

Theorem 3. *Let R be a run of* EXT *on a dynamic graph $G \in \mathcal{G}(1\text{-}IC)$. Eventually, v_l terminates and it outputs the correct count in R.*

Proof. For Lemma 2 there exists $G' \subseteq G$ with $G' \in \mathcal{G}(\mathrm{PD})_h$. Therefore for Lemma 9 the leader has to terminate correctly on $I_{G'}$. Moreover also for Lemma 8 if another instance $I_{G''}$ with $G'' \subseteq G$ outputs a value, then this value is also correct. From these considerations the claim follows.

□

OPT pseudocode

This appendix section reports in Figure 7 the pseudo code run by the leader to handle the data structure T.

```
1: T = ⊥
2:
3: function buildT(MultiSet M_r)
4:     if r == 2 then
5:         Assert(∃[⊥] ∈ M_1)
6:         T.setRoot([⊥] :< m_[⊥], ⊥ >)
7:     for all [x_0, ..., x_{r-2}, x_{r-1}] ∈ M_r do
8:         v_l creates a node [x_0, ..., x_{r-2}, x_r - 1] :< m_[x_0,...,x_{r-1}], n_|[x_0,...,x_{r-1}]| :? >
9:         t : T.findNode([x_0, ..., x_{r-2}])
10:        if n_t ≠?||t = null then
11:            continue
12:        t.addChild([x_0, ..., x_{r-2}, x_r - 1])
13:    compute(T)
14:    if T.root.n_[⊥] ≠? then
15:        output(T.root.n_[⊥])
16:
17: function compute(Tree T)
18:    for all t ∈ T starting from the level of the leaves until the root do
19:        C : T.findChildren(t)
20:        X ⊆ C such that [x_0, ..., x_k] ∈ X iff n_[x_0,...,x_k] ≠?
21:        if ∃!c : [y_0, ..., y_k] ∈ C \ X then
22:            n_c : (m_t - Σ_{∀[x_0,...,x_k]∈X} (x_k · (n_[x_0,...,x_k]))) / y_k
23:        if X = C then
24:            n_t : Σ_{∀[x_0,...,x_k]∈X} n_[x_0,...,x_k]
```

Figure 7: Handling of T by the Leader in OPT algorithm

OPT_h: Extending OPT to count in $\mathcal{G}(\mathbf{PD})_h$

As in OPT, OPT_h begins with a *get_distance* phase over $\mathcal{G}(\text{PD})_h$ where each process obtains its distance from the leader. Using a simple flooding and convergecast algorithm this phase takes at most $2h+1$ rounds. In the first h rounds (flooding step) each process computes its distance from v_l, in the $h+1$ successive rounds (convergecast step) the leader computes the maximum distance h.

Non-Leader process behavior in OPT_h. The code of a non-leader process in OPT_h is reported in Figure 8, the function *count_distance_neighbors* returns the number of messages in MS generated by processes at distance $distance-1$, the function *get_messages_from_distance* returns only messages generated by processes at distance $distance+1$. If there is no such message the function returns $\perp$. As in OPT, a process v updates its degree history $v.H(r)$ by counting the number of processes in $N(v,r)$ whose distance is equal to $v.distance-1$. Moreover v updates a multiset $v.M(r)$ that contains messages received by neighbors at distance $v.distance+1$, if v has not received any of these messages, it adds $\perp$ to the multiset. In the sending phase, v broadcasts $< v.distance, v.M(r), v.H(r) >$ to its neighbors.

```
1: distance_count[]
2: procedure SENDING_PHASE
3:     send(< leader >)
4:
5: procedure RCV_PHASE(MultiSet MS :< distance, M, H >)
6:     i = 1
7:     distance_count[i] = |MS|
8:     i++
9:     while true do
10:        if  i > h then
11:            count = Σ_{∀j|distance_count[j]≠⊥} distance_count[j]
12:            output(count)
13:        MS =BUILDLASTSET (MS)
14:        if MS = ⊥ then
15:            break
16:        distance_count[i] = |MS|
17:        i++
18:
19: function BUILDLASTSET(MS)
20:     MS_last = ⊥
21:     if OPT(MS, 0) ≠ ⊥ then
22:         r_last = r'|OPT(MS, r') ≠ ⊥ ∧ ∄r'' > r'|OPT(MS, r'') ≠ ⊥
23:         MS_last = OPT(MS, r_last)
24:     return MS_last
```

Figure 9: OPT_h algorithm for $\mathcal{G}(\mathrm{PD})_h$: algorithm run by the leader

```
1: M(0) = [⊥]
2: H(0) = [⊥]
3: distance = −1
4:
5: procedure SENDING_PHASE
6:     send(Message :< distance, M(r), H(r) >)
7:
8: procedure RCV_PHASE(MultiSet MS)
9:     H(r + 1) = H(r).append(count_distance_neighbors(MS, distance − 1))
10:    M(r + 1) = M(r).append(get_messages_from_distance(MS, distance + 1))
```

Figure 8: OPT_h algorithm for $\mathcal{G}(\mathrm{PD})_h$: algorithm run by a non-leader process

Leader process behavior in OPT_h. From an high level point of view the algorithm works as follow: the leader first computes the number of processes in V_1, then it executes OPT to count the processes in V_2, this count will be completed by round $(2h+1)+(3+log(|V_2|))$ (see Theorem 7). At this point, the leader simulates an execution of OPT counting processes in V_3 exploiting the information obtained by processes in V_2, the leader uses OPT to obtain the exact multiset of messages received by processes in V_2. This counting will be completed by round $(2h+1)+6+log(|V_2|)+log(|V_3|)$. Iterating this procedure till processes at distance h we obtain the final count in $(2h+1)+3h+\sum_{i=2}^{h} log_2(|V_i|)$ rounds.

Operationally, the purpose of the leader is to reconstruct the multiset MS_j of messages $< distance, M(r), H(r) >$ sent by processes in V_j at some round r, from MS_j we have $|V_j| = |MS_j|$.

At each round the leader receives MS_1. Starting from MS_1 content, OPT_h iteratively recon-

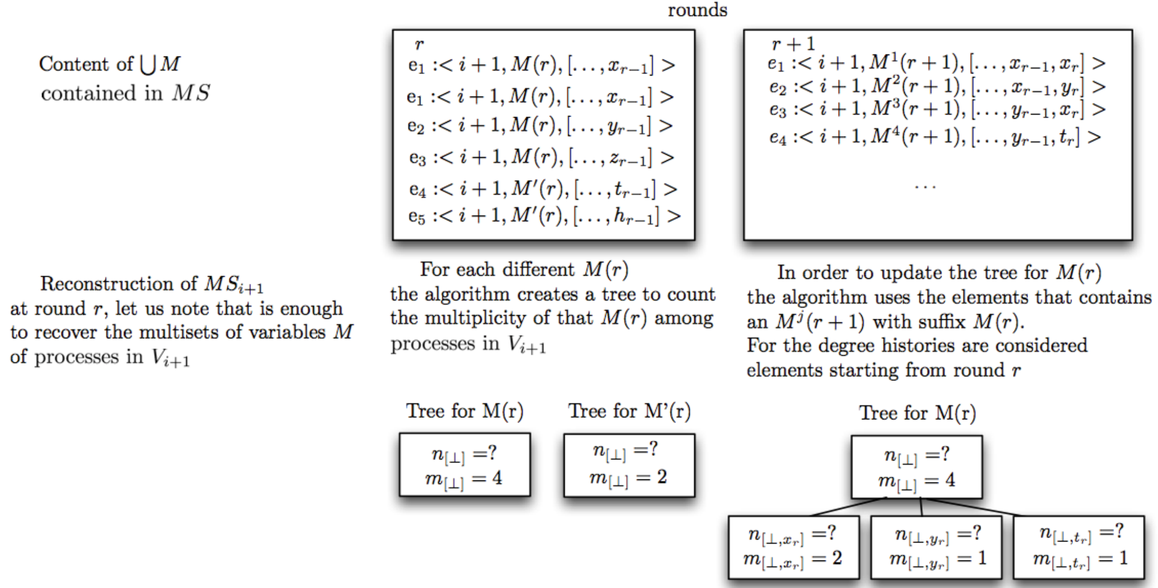

Figure 10: Reconstruction of M variables sent by the set of processes V_{i+1}

structs the sets MS_j for $j > 1$. This is done in the loop 9-17 of Figure 9. The leader uses the variable i to store the maximum distance at which processes of the network have been already counted by OPT_h (initially $i = 1$).

At the beginning of the loop, the leader checks if the count is over (i.e. it checks if $i > h$), in the affirmative the leader outputs the count. Otherwise v_l continues to execute the code in the loop (see Lines 10-12 of Figure 9). The leader now uses the information in the last reconstructed multiset, denoted MS_i, to obtain the multiset MS_{i+1}. Specifically v_l simulates an instance of the algorithm OPT for each element contained in MS_i, i.e., if MS_i contains only two different elements, namely $M(r)$ and $M'(r)$, then the leader uses two trees in order to count the exact number of processes that sent $M(r)$ and $M'(r)$. An example can be found in Figure 10.

The reconstruction is executed by the function BUILDLASTSET.

This function calls $\text{OPT}(MS_i, r')$. The function takes two parameters, and it works on the set of messages MS' where the elements in MS_i received before round r' are removed. Therefore the call $\text{OPT}(MS_i, r')$ returns either the multiset MS_{i+1} sent at round r' or $\bot$. The round r_{last} is the most recent round at which the multiset MS_{i+1} can be reconstructed (see Line 22). In the worst case $r_{last} \geq r - (log_2(|V_{i+1}|) + 3) + \sum_{j=2}^{i}(log_2(|V_j|) + 3)$. Thus the function BUILDLASTSET returns either MS_{i+1} sent at round r_{last} or $\bot$.

At line 14 of Figure 9 the leader obtains MS_{i+1} or $\bot$. If the value obtained is $\bot$, the leader exits from the loop and it waits for the next round; otherwise it computes the count of $|V_{i+1}|$, it updates the distance index and it starts the next iteration of the loop (lines 16-17, Figure 9).

Lemma 10. OPT_h *requires at most* $(2 \cdot h + 1) + 3 \cdot h + \sum_{1 \leq i \leq h} log_2(|V_i|)$ *rounds to output the*

count.

Proof. We consider a generic run after $2h+1$ rounds, so that each process has set $my_distance \neq -1$. For easy of explanation we consider that the algorithm starts at round 0 and that all processes know their distance. Let us consider the processes in V_h, at round $r = 0$. They start to send their degree history to processes in V_{h-1}, in the worst case at round $r_h = \log_2(|V_h|) + 3$ the union of variables of processes in V_{h-1} allows to compute the multiset MS_h sent at round $r = 0$, see Th. 1 for OPT algorithm, thus at round $r_{h-1} = 6 + \log_2(|V_h|) + \log_2(|V_{h-1}|)$ the multiset MS_{h-1} generated at round r_h can be reconstructed by the union of variables of processes in V_{h-2}, let us recall that MS_{h-1} at round r_{h-1} contains the information to reconstruct MS_h at round r_h. By induction is easy to show that at round $r_1 = \sum_{i=2}^{h}(\log_2(|V_i|) + 3)$ the multiset MS_1 contains all the information to reconstruct MS_2 at round $r_2 = \sum_{i=3}^{h}(\log_2(|V_i|) + 3)$ and so on. Thus the leader at round $r_1 + 1$ will execute the reconstruction loop, Lines 19-24 of Figure 9, on the multiset MS_1, and it will obtain the multiset MS_2 sent by processes in V_2 at round $r' \geq r_2$, thus using MS_2 it will reconstruct the multiset MS_3 sent at rounds $r' \geq r_3 = \sum_{i=4}^{h}(\log_2(|V_i|) + 3)$. The leader iterates the computation till it obtains the multiset MS_h, then leader terminates the reconstruction and the count. □

Complexity discussion From the bound shown in Th.2 of [13] we can easily obtain that a lower bound on counting time for $\mathcal{G}(\text{PD})_h$ is $h + max_{\forall V_i}(\log_3(2|V_i| + 1))$. This lower bound holds for a configuration where at each round processes at distance x could be connected to only two processes at distance $x - 1$. The complexity of OPT_h is upper bounded by $5h + 1 + \sum_{i=2}^{h}\log_2(|V_i|)$. Now let us discuss two cases:

- Case 1: If h is constant w.r.t $|V|$ we have $5h+1+\sum_{i=2}^{h}\log_2(|V_i|) \leq 5h+1+h \cdot max_{\forall V_i}(\log_3(2|V_i|+1))$ that is the same order of the lower bound.
- Case 2: If h is not constant w.r.t. $|V|$. We have that $\sum_{i=2}^{h}|V_i| \leq |V|$ and that $\Pi_{i=2}^{h}|V_i| \leq \frac{|V|}{x}^x$ since the product of numbers with a given sum is maximized when all numbers are equals [2]. The maximum of $\frac{|V|}{x}^x$ is obtained when $x = \frac{|V|}{e}$ and thus $\log_2(\Pi_{i=2}^{h}|V_i|) \leq \frac{|V|}{e}\log_2(e)$. Since also the lower bound is worst case $\mathcal{O}(|V|)$ we have our algorithm is asymptotically optimal.

[2] This is a well known result obtained by using the relationship between geometric and arithmetic mean.