On The Smallest Grain of Salt to Get a Unique Identity*

Peva Blanchard and Rachid Guerraoui

EPFL, Lausanne, Switzerland (first.last@epfl.ch)

Abstract. We study the fundamental *enumeration* problem in asynchronous message-passing networks. Anonymous processes have to eventually decide on pairwise distinct identifiers, despite all starting in the same initial state. It is known since Angluin's seminal result [2] that some *grain of salt* is required for distributed algorithms to solve the problem, e.g., the system needs to have a non-symmetrical topology or unbiased independent random bits.

The starting point of this paper is the observation that these approaches demand too strong assumptions. In short, by adding *time* to the picture, we show that the enumeration problem can be solved with far less. The idea is to consider a schedule of events in a distributed system as a *space-time structure* that is *gradually learnt* by the processes. We introduce the notion of *divergence time* which essentially measures the time by which the causal order induced by the system schedule has differentiated all the processes.

We prove lower bounds on the running time of any algorithm solving enumeration in terms of divergence time. In particular, we show that any adversary scheduler against which the enumeration problem can be solved *necessarily* selects schedules with *finite divergence time*.

We prove that this last condition is *sufficient*: we present the Torche algorithm which solves enumeration for all schedules with finite divergence time. In this sense, having finite divergence time is the smallest grain of salt required to solve the enumeration problem.

1 Introduction

Process identifiers are crucial in distributed systems, and are implicitly assumed in most distributed algorithms. The problem of assigning distinct identifiers to processes, however, is not trivial. This problem, called *enumeration*, has received a lot of attention in the past decades [1,2,6,9, 11,16,21,22,23].

The problem consists for a set of n processes, starting identically (in the same initial state), to each decide a value (an integer in the range

* Acknowledgment: This work has been supported in part by the European ERC Grant 339539 - AOC

$\{1, \ldots, n\}$) that is different from all the values decided by the other processes. The core difficulty has been pinpointed in Angluin's seminal paper [2]. In short, the problem is impossible to solve deterministically in any network with *spatial symmetry*. Roughly speaking, two processes in the network are said to be related by a spatial symmetry if they have the exact same view of their surrounding. To get an intuition of Angluin's argument, consider a deterministic algorithm running on an even-sized oriented network of processes. Assume, for the sake of simplicity, that when a process is activated, the process atomically reads the states of all its neighbours in the network, and updates deterministically its own state accordingly. Initially, all the processes have the same state. The neighbourhood of any process p is thus similar to the neighborhood of the diametrically opposite process q. If p and q are concurrently activated, then their states are updated to the same value. By repeating this kind of activation, one can design schedules of events during which no process is distinguishable from the opposite process, thus preventing enumeration.

The only way to circumvent Angluin's argument is to assume that the distributed system contains some *grain of salt*, i.e., that there is some breaking of symmetry somewhere which could be exploited by distributed algorithms to enumerate. So far, two sorts of grain of salts have been considered in the literature. One approach *(I)*, adopted in [6,9,16], assumes that the *topology of the network has no non-trivial spatial symmetries*. Another approach *(II)* is the use of randomization [1,11], with processes having local access to *unbiased independent* random bits.

The starting point of this paper is the observation that these approaches are not necessary. The idea is to consider a schedule of events in a distributed system as a *space-time structure* that is *gradually learnt* by the processes.

Instead of considering that only the spatial part (i.e. the network topology) has no non-trivial symmetries, we consider the much weaker assumption that the space-time structure, taken as a whole, has no non-trivial symmetries. Our approach encompasses naturally the two previous approaches. First, if the spatial part has no non-trivial symmetries, then the space-time structure necessarily lacks non-trivial symmetries; second, the use of independent random bits ensures, with probability one, that the space-time structure lacks non-trivial symmetries.

We consider a general asynchronous distributed system model where processes communicate with their neighbours by sending and/or receiving messages through communication channels. Moreover, the processes may

have access to unreliable[1] sources of randomness. The different possibilities of scheduling events are chosen by some external entity, called the *adversary scheduler*.

Our main conceptual contributions are twofold. First, we introduce the notion of *divergence time* of a schedule S of events, which, roughly speaking, measures the time by which all the processes have differentiated[2]. We then exhibit lower bounds on the running time of any algorithm solving the enumeration problem, in terms of divergence time. In particular, we show that any adversary scheduler against which it is possible to solve the enumeration problem can only select schedules that have *finite divergence time*. In other words, the finite divergence time condition is a *necessary* condition for solving the enumeration problem.

Second, we prove that the finite divergence time condition is actually *sufficient*. We present an algorithm, we call TORCHE, that solves the enumeration problem over *all* the schedules with finite divergence time, assuming only the knowledge of the network size[3].

The main consequence of the existence of this algorithm is that the finite divergence time condition is indeed the *smallest grain of salt* required to solve the enumeration problem. By "smallest grain of salt", we mean that any other grain of salt, i.e., any other property of the schedules which allows to break symmetry and solve enumeration (e.g., topology without symmetries, or use of randomness) necessarily implies that the divergence time is finite.

Two new techniques are involved in the design of TORCHE: *folded causal past reconstruction* and *phylogenetic tree extraction*.

- First, the *folded causal past reconstruction* technique consists for each process p to maintain a compressed estimate of its causal past, by gluing together the estimates of its neighbours, and folding together the events that have isomorphic causal pasts. This technique may be seen as the spatio-temporal generalization of the *compressed view* technique from [20].
- Second, Each process p can extract from the folded estimate of its causal past a *phylogenetic tree* which, roughly speaking, represents the various differentiations that have occurred, as far as process p knows. The number of vertices that lies in the same level in this tree somehow

[1] The bits used by the processes may, to some extent, be correlated, across the network and through time.

[2] More precisely, they have pairwise non-isomorphic causal pasts. See details below.

[3] The knowledge of the network size, or any similar property, is a common assumption in the literature [21,22].

gives the *effective* number of distinct processes, i.e., the number of distinguishible (groups of) processes. Assuming that the schedule has finite divergence time, this tree eventually has n branches, where n is the network size. Process p can then detect the end of this divergence period, and, since p knows on which branch of the tree it lies, process p can also decide on a unique identifier.

Our TORCHE algoritm has several interesting properties. The running time is tight in the sense that processes decide right after the divergence time plus the time for the information at any process to reach the whole network (cover time). The space required for storing the process states and messages is polynomial in the network size, the divergence and cover time.

(Paper organization). We present our computational model, as well as the notions of divergence time in Section 2. We present our lower bounds on the running time of any algorithm solving the enumeration problem in Section 3. In Section 4, we present the TORCHE algorithm, and prove its main properties. In Section 5, we show how our notion of divergence time encompasses other notions used to circumvent Angluin's argument in previous work, namely *fiber-minimal* networks, and the *use of random bits*. Finally, we discuss the related work in Section 6.

2 Model and Definitions

We consider a general asynchronous model of computation where anonymous processes communicate by message passing.

2.1 Algorithms

(Graphs). We consider directed graphs with (possibly) multiple arrows between two vertices, and (possibly) self-loops. Formally, a *graph* G is given by a set $\mathcal{V}G$ of vertices, a set $\mathcal{A}G$ of arrows, and maps $s, t : \mathcal{A}G \to \mathcal{V}G$ specifying the source and target vertices of each arrow. A *vertex-labeling* (resp. *arrow-labeling*) is a map $\mathcal{V}G \to X$ (resp. $\mathcal{A}G \to \Lambda$). A *path* is a sequence $a_1 \dots a_l$ of arrows such that $t(a_i) = s(a_{i+1})$ for $1 \leq i < l$. This path is a *cycle* if moreover $t(a_l) = s(a_1)$. The graph G is *acyclic* if it does not contain any cycle.

A *morphism* $\phi : G \to H$ is given by a vertex function $\phi_V : \mathcal{V}G \to \mathcal{V}H$, and an arrow function $\phi_A : \mathcal{A}G \to \mathcal{A}H$ such that $s(\phi_A(a)) = \phi_V(s(a))$ and $t(\phi_A(a)) = \phi_V(t(a))$ for every arrow $a \in \mathcal{A}G$. If moreover the graphs G and H are equipped with a vertex-labeling and an arrow-labeling, it is

also required that ϕ_V and ϕ_A preserve the labels. The morphism ϕ is an *isomorphism* if both ϕ_V and ϕ_A are bijective. We denote by $G \simeq H$ the statement that G and H are isomorphic.

(Networks). A *network* is $\mathcal{N}$ simply modeled as a graph with at most one arrow from one vertex to another, and without self-loops. The vertices represent the processes. An arrow with source p and target q represents a communication channel transporting messages from p to q. For each process p, we denote by $\mathcal{N}_p$ the set of neighbour processes with arrow towards process p.

Algorithm 1: Normal Form - process p

```
1 variables:
2     state s_p of p initially set to some common value
3     set t_p of triples (ω, z) where ω ∈ Λ_p and z is a state value
4     variable b_p for random bits
5 for round r = 0, 1, ...
6     t_p ← scan()                                  /* scan neighbours */
7     b_p ← rand()                                  /* "random" bits */
8     s_p ← new-state(s_p, t_p, b_p)                /* update state */
```

(Processes). The processes are anonymous (no identifiers), they start in the same initial state, and all execute the same algorithm. We also assume that each process can read some bits from some local source of information. These bits can be thought as "random" bits, but we insist on the fact that, in the most general case, these bits may be correlated (both in time and across the network) in any possible way.

Each process p performs an infinite series of *asynchronous* rounds. Each round comprises two phases: a *scan* phase, and an *update* phase. The scan phase collects the (possibly not up to date) states of its neighbours and returns the multiset t_p of couples (ω, z) where z is the state of a neighbour q received along the incoming arrow a with label ω and source q. Then, during the update phase, process p reads some bit-string b_p from its local source of information, and updates its state by applying a deterministic transition function `new-state` to the tuple formed by its current state, the multiset t_p collected during the scan phase, and the bit-string b_p. Algorithm 1 sums up the normal form of an algorithm.

2.2 Schedules

We model a (finite or infinite) schedule S on a network $\mathcal{N}$ as a (finite or infinite) acyclic graph equipped with a labeling of the vertices with bit-strings, and a labeling of the arrows with the same set of arrow labels as the network $\mathcal{N}$. The vertices of the schedule are couples (p, r) where p is a process, $r \geq 0$ is an integer. We refer to (p, r) as an *event* at p in S. Each (p, r) with $r \geq 1$ is labeled with a bit-string b representing the "random" bit-string read by p during round $r - 1$. Intuitively, each event (p, r) represents the state of p at the beginning of round r at p. This state is the result of the previous update phase at round $r - 1$, and depends on the previous state of p, as well as the state values collected during the previous scan operation. These dependencies are modeled by arrows between events.

More precisely, if during round r, process p received the state associated with the round s of the neighbour q, then there is an arrow $(q, s) \to (p, r)$ labeled with the same label as the arrow from q to p in the network. For every $r \geq 0$, there is also an arrow $(p, r) \to (p, r + 1)$ labeled with the distinguished symbol ϵ. Figure 1 gives an example of a schedule.

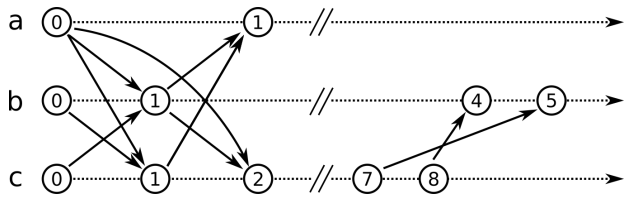

Fig. 1. Schedule - Process c in round 2 reads the initial state of process a, and the state at the beginning of round 1 of process b. Note (on the right) that a scan may return values older than the values returned by the previous scan. The labels of the arrows are omitted.

(Causality). For any two events e, e' in S, e *causally precedes* e' in S if there exists a path from e to e' in S. The *causal past* of an event e in S is the schedule which is the sub-graph of S spanned by e and all the events causally preceding e in S. Given a finite schedule S, for any process q participating in S, we refer to the causal past of the latest event at q in S as the *causal past of q in S*. A *past cone at p* is a finite schedule P which is the causal past of some event $e = (p, r) \in \mathcal{V}P$. The event e is necessarily unique, and is referred to as the *apex* of P. The *causal height*, or simply *height*, of an event e in S is the length of the longest directed

path in S reaching e. The height of a finite schedule S is the maximum height of its events, i.e., the length of the longest directed path contained in S. Note that the height of a past cone is the height of its apex.

(Cut). A *cut* C of a schedule S is a set of events in S such that each process of the network participates exactly once in C. For each process p, we denote by $C[p]$ the height of the event (p, r) in C. The *initial cut* is defined by $\{(p, 0) \mid p \in \mathcal{N}_V\}$. Cuts are partially ordered: $C \preccurlyeq C'$ if, for all p, $C[p] \leq C'[p]$. Equipped with this partial order, the set of cuts of a schedule form a lattice.

(Segment). Given two cuts $C \preccurlyeq C'$ in a schedule S, the *segment* $K = [C, C']$ is the sub-schedule of S comprising all the events (p, r) with height $C[p] \leq h \leq C'[p]$, where p runs over all the processes. The height of the segment K is the length of the longest causal chain in K. The *prefix* $[0, C]$ is the sub-schedule of S comprising all the events (p, r) with height at most $C[p]$. The *suffix* $[C, \infty)$ is the sub-schedule of S comprising all the events (p, r) with height at least $C[p]$.

(Fairness). The schedule S is *fair* if, for any two processes p and q, for all $t \geq 1$, there is a cut C such that $\forall p, C[p] \geq t$ and $[C, \infty)$ contains a path from some event at p to some event at q. Unless stated otherwise, all infinite schedules are assumed to be fair.

(Adversary scheduler). An *adversary scheduler* is modeled as a set of (fair)infinite schedules. Adversary $\mathcal{A}$ is *stronger* than adversary $\mathcal{B}$ if $\mathcal{B} \subseteq \mathcal{A}$.

(Divergence cut). The *divergence cut* of a schedule S is the minimum cut $C_{dv}(S)$ such that, for all cuts $C \succcurlyeq C_{dv}(S)$ in S, the causal pasts of any two distinct events in C are not isomorphic. The *divergence time* of a schedule S is defined as $\tau_{dv}(S) = \max_p C_{dv}[p]$. If the divergence cut is undefined, we write $C_{dv}(S) = \infty$ and $\tau_{dv}(S) = \infty$. We say that the schedule S has finite divergence time if the divergence cut exists. When it is clear from the context, we simply write C_{dv} and τ_{dv}, omitting the reference to the schedule S.

(Cover cut function). The *cover cut function* of a schedule S is defined as follows. For any cut C in S, $C_{cv}(C, S)$ is the minimum cut C' such that, for any two processes p, q, there is a path in $[C, C']$ from some event at p to some event at q. When it is clear from the context, we simply write $C_{cv}(C)$,omitting the reference to the schedule S.

(Decision event). Any process p is assumed to be able to trigger a *decide* action. Afterwards, its state remains unchanged. It is assumed that each process can trigger this action at most once during the execution.

The event at p at which this action is performed is the *decision event at* p.

(Decision cut). The *decision cut* is the cut formed by the decision events of all the processes. If some process never decides, the decision cut is undefined.

3 Lower Bounds on the Running Time

In this section, we present lower bounds on the (causal) height of the decision events of the processes, in terms of divergence and cover cuts. The following proposition states that any adversary scheduler against which enumeration is solvable *necessarily* provides schedules with finite divergence time. More precisely, it is impossible for all processes to decide strictly before the divergence cut.

Proposition 1. *Let $\mathcal{A}$ be a set of schedules over the network $\mathcal{N}$. Consider an algorithm solving enumeration over all the schedules in $\mathcal{A}$. Then all schedules in $\mathcal{A}$ have finite divergence time. And, more precisely, for any schedule in $\mathcal{A}$, the cut C defined by the decision events satisfies*

$$\exists p \in \mathcal{N},\ C[p] \geq C_{dv}[p]$$

Proof. Consider a schedule S in $\mathcal{A}$. Let C denote the decision cut. Assume first that the divergence cut of S is undefined. By definition, this implies that there exists a cut $D \succcurlyeq C$, and two distinct events in D, at two distinct processes p, q, with isomorphic causal pasts. In particular, p and q must have decided on the same value; whence a contradiction. Thus, S has finite divergence time.

Assume now that $C \prec C_{dv}$, i.e., for all processes p, $C[p] < C_{dv}[p]$. By definition of the divergence cut, this means that there exist two distinct processes p, q such that the causal pasts of their decision events are isomorphic. This implies that p and q decide on the same value; whence a contradiction. □

The following states that, in general, at least some process has to wait until it notices that all processes have differentiated.

Proposition 2. *Consider an algorithm solving enumeration. Then there exists a network $\mathcal{N}$ and a schedule on $\mathcal{N}$ such that the cut C defined by the decision events of the processes satisfies*

$$\exists p \in \mathcal{N},\ C[p] \geq C_{cv}(C_{dv})[p]$$

Proof. We consider a linear network $\mathcal{N}$ of 3 vertices, and the schedules S_1 and S_2 in Figure 2. The figure also depicts the divergence and cover cuts for schedule S_1. We claim that, in S_1, process c cannot decide before its second event. Indeed, assume that process c decides at its first event. However, at this point, S_1 and S_2 are indistinguishable for process c. In S_2, the first events of a and c have isomorphic causal pasts. Thus, in S_2, process a decides on the same value as c; whence a contradiction. In particular, the decision cut $C(S_1)$ in S_1 satisfisfies $C(S_1)[c] \geq C_{cv}(C_{dv}, S_1)[c]$. □

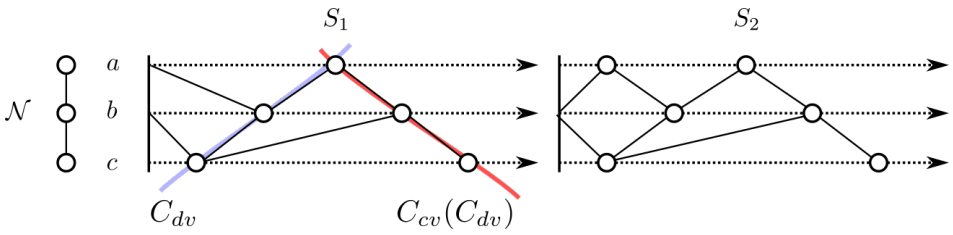

Fig. 2. Proof of Proposition 2 - Process c cannot decide before its second event.

4 The Torche Algorithm

Proposition 1 states that, in order to solve the enumeration problem, it is necessary that the schedules chosen by the adversary scheduler have a finite divergence time. In this section, we prove that having a finite divergence time is a sufficient condition for solving the enumeration problem. In this sense, having a finite divergence time is indeed the smallest grain of salt required to solve the enumeration problem.

The main idea underlying our approach consists in having processes trying to get an estimate, as accurate as possible, of their causal pasts. For the sake of simplicity, let us consider how a naive version of a full information algorithm would work. Process p starts in its initial state, represented as a single vertex. In the next round, process p collects the states of (some of) its neighbours, draws a random value b (if any), and updates its own state. In a naive full-information algorithm, this new state is encoded as a tree. The root corresponds to the new round that p has just performed, and is labeled with the value b that has been drawn. The root's children are the trees received from the neighbours, as well as the

tree corresponding to the previous state of p. If p has received the (tree) state of q along the incoming arrow $q \xrightarrow{\omega} p$, then the arrow connecting the tree of q to the root is labeled with ω as well. The arrow connecting the previous tree of p to the root is labeled with the distinguished symbol ϵ. The first row in Figure 4 illustrate how the full-information trees evolve along the schedule of Figure 3.

An *ϵ-path* is a path whose edges are all labeled with ϵ. A *maximal ϵ-path* is an ϵ-path which cannot be extended to a longer ϵ-path. A first observation is that, from the designer's point of view, the maximal ϵ-paths can be mapped to process identities. However, the tree structure has a lot of redundancy. For example, if p sees q' which has seen p before, then there are two maximal ϵ-paths that can be associated with p. For instance, in Figure 4, the tree T_p^3 contains two copies of T_p^1.

This example leads to a second observation: the tree of p that was observed by q is isomorphic to a sub-tree of the latest tree of p. A third observation is that there is one-to-one correspondance between such trees and the causal pasts that led to them. In particular, two processes have isomorphic causal pasts if and only if the corresponding trees are isomorphic.

From these observations, we adopt the following approach. We *fold* the tree by identifying vertices whose sub-trees are isomorphic. This amounts to identifying events in the schedule that have isomorphic causal pasts. Let W be the graph obtained this way. Since it is possible that two distinct processes have lived isomorphic causal pasts, this folding operation may identify them. See the second row in Figure 4. Moreover, the ϵ-paths in W form a tree we call the *phylogenetic tree* associated with W. This phylogenetic tree is interpreted as follows. Initially, all the processes are in the same initial state, so the *effective number* of distinct processes is 1, and is encoded by the fact that the ϵ-tree has a single root. As time flows, processes undergo specific events that differentiate them, and the tree branches. The number of leaves of this tree gives the effective number of distinct processes at the end of the schedule, or, mathematically speaking, the number of isomorphism classes of maximal causal pasts in the schedule. In particular, the number of leaves is at most n. By the assumption that the divergence time is finite, any two processes eventually have non isomorphic causal pasts, and thus, the number of leaves eventually reaches n.

As we assume that the processes know the size n of the network, process p can detect that all processes have differentiated (i.e., have pairwise non isomorphic causal pasts). Moreover, p is aware of the leaf representing

itself : it is the leaf with the highest height. Roughly speaking, process p finally sorts the leaves of W and decides on the rank of its leaf.

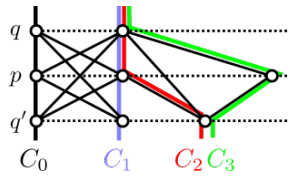

Fig. 3. Schedule with processes p, q and q'. Four cuts are represented. See Figure 4 for the states of the processes at these cuts.

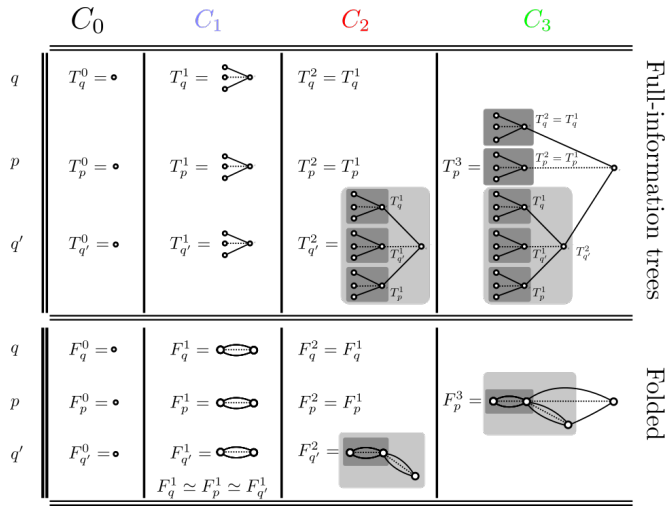

Fig. 4. Evolution of the full-information trees (first row) and their respective folding (second row) at the cuts from Figure 3. Due to space limitations, at cut C_3, we only represent the tree and folded tree of process p.

4.1 Fold Operation

Let P be a schedule. Recall that some arrows may be labeled with the distinguished symbol ϵ. There is a partial order on the vertices : $u \preccurlyeq v$ if there is a path from u to v. We define $\downarrow v$ as the subgraph of P induced by the vertices $u \preccurlyeq v$. We define an equivalence relation on the vertices : $u \sim v$ if the the graphs $\downarrow u$ and $\downarrow v$ are isomorphic. We denote by $[u]$

the equivalence class of u. We also define an equivalence relation on the arrows : $u \xrightarrow{\omega} v \sim u' \xrightarrow{\omega'} v'$ if there is some isomorphism between $\downarrow v$ and $\downarrow v'$ mapping the arrow $u \xrightarrow{\omega} v$ to $u' \xrightarrow{\omega'} v'$. This implies $v \sim v'$, $u \sim u'$ and $\omega = \omega'$. We denote by $[u \xrightarrow{\omega} v]$ the equivalence class of $u \xrightarrow{\omega} v$.

The graph $W = \texttt{fold}(P)$ is defined as follows. The vertices of W are the equivalence classes $[u]$ with $u \in \mathcal{V}P$. The arrows of W are the equivalence classes $[u \xrightarrow{\omega} v]$ with $u \xrightarrow{\omega} v \in \mathcal{A}P$. The source (resp. target) of $[u \xrightarrow{\omega} v]$ is $[u]$ (resp. $[v]$). The complexity of the fold operation is related to the complexity of graph isomorphism. Isomorphism of trees can be tested in linear time. However, the same problem for directed acyclic graphs is equivalent to the classic (undirected) graph isomorphism problem. By the recent work of [4], W can be computed from P in quasi-polynomial time.

Lemma 1. *The map $\Phi : P \to \texttt{fold}(P)$ defined on the vertices by $u \mapsto [u]$ and on the arrows by $u \xrightarrow{\omega} v \mapsto [u \xrightarrow{\omega} v]$ is a graph morphism preserving the labels of the vertices and arrows. Moreover, Φ also preserves the height of the vertices.*

Proof. The claim follows from the definition of $\texttt{fold}(P)$. □

Lemma 2. *Let P be a schedule. Let $W = \texttt{fold}(P)$. The number of maximal ϵ-paths in W equals the number of isomorphism classes of the maximal causal pasts of the processes in P. In particular, this number is at most the number n of processes in the network.*

Proof. Let E denote the set of maximal events in P, i.e., the events e which do not causally precede any other event. The set E can be partitioned into equivalence classes. Each maximal ϵ-path π in W can be identified with the last vertex l along this path (equivalently, the corresponding leaf in the tree formed by the maximal ϵ-paths). By definition of `fold`, the vertex l denotes an equivalence class of events in P. An event e in the class l is maximal, as otherwise (since the map Φ preserves the height) l would not be the last vertex along π. Therefore, there is one-to-one correspondance between the equivalence classes partitioning E and the final vertices of maximal ϵ-paths in W. □

4.2 Algorithm Details

Each process p maintains an estimate W_p of its folded causal past. At the beginning of a round, process p scans the set $\mathcal{N}_p$ of its neighbours, and pulls their estimates $(W_q)_{q \in \mathcal{N}_p}$. Process p then forms the disjoint union

of these graphs, including its previous estimate, adds a new vertex (r, b) labeled with the current round number r and a random value b (if any), and connects the apex of each estimate W_q with the new vertex (r, b); this arrow being labeled with the same label ω as the one connecting q to p. We denote this whole operation by $\left(\bigsqcup_{q \in \mathcal{N}_p \cup \{p\}} W_q\right) \oplus (r, b)$. The variable W_p is updated by folding the aforementioned graph.

As will be shown below, W_p is isomorphic to the folding of the causal past of process p. Process p derives the phylogenetic tree associated with W_p and counts the number c of maximal ϵ-paths. If process p notices that $c = n$, i.e., that all processes have differentiated, then process p computes the shortest prefix K_p of W_p whose phylogenetic tree still has n leaves (maximal ϵ-paths). Process p sorts the maximal ϵ-paths of K_p (according to any predefined total order), and decides on the rank corresponding to its own ϵ-path (necessarily the longest ϵ-path in W_p).

Algorithm 2: TORCHE- process p

initial knowledge:
  the network size n;
variables:
  W_p : acyclic graph, reduced estimate of causal past, initially set to a single vertex $(0, \bot)$
for *round* $r = 1, 2, \ldots$
  $(W_q)_{q \in \mathcal{N}_p} \leftarrow$ `scan`()
  $b \leftarrow$ `rand`() `/* possibly poor quality random bits */`
  $W_p \leftarrow$ `fold` $\left(\left(\bigsqcup_{q \in \mathcal{N}_p \cup \{p\}} W_q\right) \oplus (r, b)\right)$
  let c be the number of maximal ϵ-paths in W_p
  if $c = n$ **then**
    let K_p be the shortest prefix of W_p with n maximal ϵ-paths
    sort the maximal ϵ-paths of K_p
    decide on the rank of the ϵ-path corresponding to self

Lemma 3. *Let P be a past cone at p. Then, at the end of P, there is an isomorphism $W_p \simeq$ `fold`(P).*

Proof. In this proof, we denote by W_p^r the value of the variable W_p at the beginning of round r. We prove the claim by induction on the height h of P. If $h = 0$, i.e., P is reduced to a single vertex $(p, 0)$ labeled with $\bot$, then, since W_p is initialized to a single vertex labeled with $\bot$, the folded graph $\texttt{fold}(W_p^0)$ equals W_p^0 and the claim holds. Assume the result holds for all causal pasts of height at most h. Let P be a causal past at p of

height $h+1$. Then, P can be written as $P = \left(\bigcup_{q \in \mathcal{N}_p \cup \{p\}} J_q\right) \oplus (p, r+1)$, where J_q is the maximal causal past of height at most h at q in P, and $r+1$ is the round number at p at the end of P. For every $q \in \mathcal{N}_p \cup \{p\}$, let r_q be the round number of process q at the end of J_q. By the induction hypothesis, we have $\forall q \in \mathcal{N}_p \cup \{p\},\ W_q^{r_q} \simeq \texttt{fold}(J_q)$. Moreover, according to Algorithm 2 (line 8), we have

$$\begin{aligned} W_p^{r+1} &= \texttt{fold}\left(\left(\bigsqcup_{q \in \mathcal{N}_p \cup \{p\}} W_q^{r_q}\right) \oplus (r+1, b)\right) \\ &\simeq \texttt{fold}\left(\left(\bigsqcup_{q \in \mathcal{N}_p \cup \{p\}} \texttt{fold}(J_q)\right) \oplus (r+1, b)\right) \\ &\simeq \texttt{fold}(P). \end{aligned}$$

□

Proposition 3. *Given that the processes know the network size n, the* Torche *algorithm solves enumeration. Moreover, for every fair schedule S with finite divergence time:* (i) *the height of the decision event at p is at most $C_{cv}(C_{dv})[p]$,* (ii) *state and message size is $O(n^2 \cdot T^2)$ where $T = \max_p\{C_{cv}(C_{dv})[p]\}$.*

Proof. (Termination). Let P be a past cone at p of height $C_{cv}(C_{dv})[p]$. By the definition of the cover time function, all processes participate in P. By definition, any two distinct events in C_{dv} have non isomorphic causal pasts. Therefore, by Lemma 2, the number of maximal ϵ-paths in $W = \texttt{fold}(P)$ is n. Thus p decides at most at the end of P.

(Uniqueness). Let p_1, p_2 be two distinct processes. Let P_1, P_2 the causal pasts of their decision events respectively. Let $Z = [0, C_{dv}]$ be the prefix corresponding to the divergence cut. Necessarily, Z is a prefix of P_1 and P_2. By Lemma 3, the values K_1 and K_2 computed at line 11 in Algorithm 2 both correspond to $\texttt{fold}(Z)$, thus $K_1 = K_2 \underset{def}{=} K$. In particular, processes p_1 and p_2 sort the maximal ϵ-paths of K the same way. And they decide on the ranks of two distinct ϵ-paths.

(State and message size). When a process decides, the variable W_p is (isomorphic to) the folding of a past cone of height at most T. Thus, it is possible to encode W_p as an adjacency matrix of dimension $n \cdot T$ at most, which requires $O(n^2T^2)$ bits. □

5 Encompassing Previous Approaches

In this section, we explain how previous approaches can be understood in terms of divergence time. More precisely, we explain how the underlying assumptions (lack of symmetry in the network topology, or use of independent random bits) imply that the divergence time is finite. Other conditions may be assumed. But were they sufficient to solve enumeration, they would necessarily imply that the divergence time is finite.

(Fiber minimal networks). As we pointed out, previous work [2,7, 8,9,21,22,23] mainly focused on the spatial aspect : the topology of the network. We briefly recall the definition of a fibration. A *fibration* $\Psi : \mathcal{N} \to \mathcal{G}$ is a surjective graph morphism (preserving labels if any) such that, for any vertex v in $\mathcal{G}$, for any vertex q in $\Psi^{-1}(v)$ (the *fiber* over v), Ψ induces a bijection (preserving labels if any) between the set of arrows into v and the set of arrows into q. The network $\mathcal{N}$ is *fiber-minimal* if any fibration $\Psi : \mathcal{N} \to \mathcal{G}$ is an isomorphism. Refer to [7] for further details. The notion of fibration captures the "spatial similarities" among the vertices of a network. The following proposition relates the concept of fibration with that of divergence time.

For the sake of simplicity, we consider, in this section, networks with *port-awareness*, i.e., the processes are able to distinguish their incoming arrows. Formally, given $q \xrightarrow{\omega} p$ and $q' \xrightarrow{\omega'} p$, then $q = q'$ iff $\omega = \omega'$.

Proposition 4. *A network $\mathcal{N}$ with port-awareness is fiber-minimal if and only if all fair schedules have a finite divergence time.*

Proof. If the network $\mathcal{N}$ is not fiber-minimal, then [6,7] have shown how to design a schedule in which at least two processes are always indistinguishable, i.e., for any height h, their respective causal pasts of height h are isomorphic. In particular, this schedule has an infinite divergence time. We proceed to prove the other direction : if there exists a schedule S with infinite divergence time then the network is not fiber-minimal.

Let S be a fair schedule with infinite divergence time on a network $\mathcal{N}$. Let $\Phi : S \to W = \texttt{fold}(S)$ be the surjective graph morphism as defined in Lemma 1. Let k denote the number of infinite ϵ-paths in W. By assumption, $k < n$. We define a network $\mathcal{G}$. The vertices of $\mathcal{G}$ are the infinite ϵ-paths in W. For any two vertices u, v in $\mathcal{G}$, there is an arrow $u \xrightarrow{\omega} v$ if and only if $u = u_1 \xrightarrow{\epsilon} u_2 \ldots$, $v = v_1 \xrightarrow{\epsilon} v_2 \ldots$, there exist $i, j \geq 1$ such that $u_i \xrightarrow{\omega} v_j$. Naturally, there is a surjective graph morphism $\Psi : \mathcal{N} \to \mathcal{G}$. This morphism maps a vertex p in $\mathcal{N}$ to the infinite ϵ-path

in W obtained as the image of the infinite ϵ-path associated with p in S under the folding operation. We claim that Ψ is a fibration. Indeed, let q be a vertex in $\mathcal{N}$. Let $b = b_1 \xrightarrow{\epsilon} b_2 \dots$ be the infinite ϵ-path in S corresponding to process q, and $v = \Phi(b) = \Psi(q) = v_1 \xrightarrow{\epsilon} v_2 \dots$ the corresponding infinite ϵ-path in W. Let $u \xrightarrow{\omega} v$ be a neighbour of v in $\mathcal{G}$. We have to prove that there exists a unique arrow $p \xrightarrow{\omega} q$ in $\mathcal{N}$ such that $\psi(p) = u$.

We write $u = u_1 \xrightarrow{\epsilon} u_2 \dots$. By definition, there exist $i, j \geq 1$ such that $u_i \xrightarrow{\omega} v_j$ in W. The existence of the arrow $u_i \xrightarrow{\omega} v_j$ implies that there exist vertices (events) a'_i, b'_j in S such that $\Phi(a'_i) = u_i$, $\Phi(b'_j) = v_j$ and $a'_i \xrightarrow{\omega} b'_j$. We also have $\Phi(b_j) = v_j$, that is, the events b_j and b'_j have isomorphic causal pasts. In particular, there exists an event a_i in S such that $\Phi(a_i) = u_i$ and $a_i \xrightarrow{\omega} b_j$. Let p be the process in $\mathcal{N}$ at which the event a_i occurs. We have $p \xrightarrow{\omega} q$, and, by the fact that incoming edges have pairwise distinct labels, p is the unique neighbour of q with such an arrow. It remains to show that $\Psi(p) = u$. Then $p \xrightarrow{\omega} q$. Moreover, the infinite ϵ-path $a = a_1 \xrightarrow{\epsilon} a_2 \dots$ in S corresponding to p is the one going through the event a_i. The infinite ϵ-path $\Phi(a)$ in W is necessarily the path u, because, by definition, u_i is such that a_i occurs in $[C_k, \infty)$, and thus there is a unique infinite ϵ-path in W going through u_i. In particular, $\Psi(p) = u$.

To conclude, we have shown that $\Psi : \mathcal{N} \to \mathcal{G}$ is a fibration. Since $\mathcal{G}$ has strictly less vertices ($k < n$), $\mathcal{N}$ is not fiber-minimal. □

(Randomness). We argue that the main purpose of using randomness is to have a finite divergence with high probability. For the sake of simplicity, we consider the case of synchronous bidirectional complete network (without arrow labels). At every round r, process p reads a random bit b_p^r.

Proposition 5. *Assuming that $(b_p^r)_{p \in \mathcal{VN}, r \in \mathbb{N}}$ are mutually independent uniform random bits, we have $\tau_{dv} = \max_p\{C_{dv}[p]\} = O(\log n)$ with high probability.*

Proof. Fix an arbitrary integer k. Consider, for each process p, the sequence $w_p = (b_p^1, \dots, b_p^k)$ of the k bits read during the first k rounds. The problem of computing the probability α that $w_p = w_q$ for at least two disinct processes p, q is an instance of the birthday paradox (n people with 2^k possible birthday dates). This yields $\alpha \simeq 1 - e^{-\frac{n^2}{2^{k+1}}}$. Setting $k = O(\log n)$, we obtain $\alpha \simeq 1 - e^{-O(n)}$. □

6 Related Work

Our computational model is the classical asynchronous message-passing model [14]. Our formulation of this model can be seen as the asynchronous generalization of the $\mathcal{LOCAL}$ model of Linial et al. [13]. Many approaches have addressed the issue of symmetry breaking in the $\mathcal{LOCAL}$ model. Leveraging the synchronous nature of $\mathcal{LOCAL}$, these approaches have mainly focused on computing the time complexity of problems like maximum independent set, maximal matching, coloring or network decomposition [3,5,10,13,17,18,19]. In most cases, these approaches assume processes with identifiers [3,17]. Our paper addresses the question of symmetry breaking in the more general settings of asynchronous computation in purely anonymous networks (no identifiers). In particular, we address the fundamental problem of enumeration on arbitrary networks. As explained in the introduction, the main obstacle to symmetry breaking problems in anonymous networks has been formulated by Angluin in [2], under the form of graph coverings, a concept borrowed from algebraic topology [15]. Later on, Yamashita et al. [21,22,23] extended Angluin's work, and inspired by the work of Johnson et al. [12], introduced the notion of *view* of a process, to encode all the information accessible to a process as the rooted tree of the finite labeled walks in the network from that process. In [20], the author presents a method to compress a view, thereby enhancing the space and time complexity of some symmetry breaking algorithms. In [6], Boldi et al. considered several synchronous and asynchronous models, and provided a characterization based on the notion of *graph fibration* [7], another concept borrowed from algebraic topology, which refines that of graph covering. In [8,9], Chalopin et al., inspired by the work of Mazurkiewicz [16], studied symmetry breaking in message-passing, and presented message-efficient algorithms in the context of networks without spatial symmetries.

References

1. Y. Afek and Y. Matias. Elections in anonymous networks. *Information and Computation*, 113(2):312–330, 1994.
2. D. Angluin. Local and global properties in networks of processors. In *12th Symposium on the Theory of Computing*, pages 82–93. ACM, 1980.
3. B. Awerbuch, A. V. Goldberg, M. Luby, and S. A. Plotkin. Network decomposition and locality in distributed computation. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 364–369, 1989.

4. L. Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697, 2016.
5. L. Barenboim, M. Elkin, and F. Kuhn. Distributed (delta+1)-coloring in linear (in delta) time. *SIAM Journal on Computing*, 43(1):72–95, 2014.
6. P. Boldi, S. Shammah, S. Vigna, B. Codenotti, P. Gemmell, and J. Simon. Symmetry breaking in anonymous networks: Characterizations. In *Israel Symposium on Theory of Computing and Systems*, pages 16–26, 1996.
7. P. Boldi and S. Vigna. Fibrations of graphs. *Discrete Mathematics*, 243(1-3):21–66, Jan. 2002.
8. J. Chalopin and Y. Métivier. An efficient message passing election algorithm based on Mazurkiewicz's algorithm. *Fundamenta Informaticae*, 80(1-3):221–246, Jan. 2007.
9. 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.
10. M. Hanckowiak, M. Karonski, and A. Panconesi. On the distributed complexity of computing maximal matchings. *SIAM Journal on Discrete Mathematics*, 15(1):41–57, 2001.
11. A. Itai and M. Rodeh. Symmetry breaking in distributed networks. In *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981*, pages 150–158, 1981.
12. R. E. Johnson and F. B. Schneider. Symmetry and similarity in distributed systems. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing*, PODC '85, pages 13–22, New York, NY, USA, 1985. ACM.
13. N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.
14. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., 1997.
15. W. Massey. *A basic course in algebraic topology*. Springer Verlag, 1991.
16. A. Mazurkiewicz. Distributed enumeration. *Information Processing Letters*, 61(5):233–239, Mar. 1997.
17. A. Panconesi and A. Srinivasan. On the complexity of distributed network decomposition. *Journal of Algorithms*, 20(2):356–374, 1996.
18. J. Schneider and R. Wattenhofer. An optimal maximal independent set algorithm for bounded-independence graphs. *Distributed Computing*, 22(5-6):349–361, 2010.
19. M. Szegedy and S. Vishwanathan. Locality based graph coloring. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 201–207, 1993.
20. S. Tani. Compression of view on anonymous networks - folded view -. *IEEE Trans. Parallel Distrib. Syst.*, 23(2):255–262, 2012.
21. M. Yamashita and T. Kameda. Computing on anonymous networks: Part I - Characterizing the solvable cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89, Jan. 1996.
22. M. Yamashita and T. Kameda. Computing on anonymous networks: Part II - Decision and membership problems. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):90–96, Jan. 1996.
23. M. Yamashita and T. Kameda. Leader election problem on networks in which processor identity numbers are not distinct. *IEEE Transactions on Parallel and Distributed Systems*, 10(9):878–887, Sept. 1999.