Compression of View on Anonymous Networks — Folded View —

Seiichiro Tani
NTT Communication Science Laboratories, NTT Corporation, Atsugi, Japan
and
Quantum Computation and Information Project, SORST,
Japan Science and Technology Agency, Tokyo, Japan.
tani@theory.brl.ntt.co.jp

Abstract

View is a labeled directed graph containing all information about the network that a party can learn by exchanging messages with its neighbors. View can be used to solve distributed problems on an anonymous network (i.e., a network that does not guarantee that every party has a unique identifier). This paper presents an algorithm that constructs views in a compressed form on an anonymous n-party network of any topology in at most $2n$ rounds with $O(n^6 \log n)$ bit complexity, where the time complexity (i.e., the number of local computation steps per party) is $O(n^6 \log n)$. This is the first view-construction algorithm that runs in $O(n)$ rounds with polynomial bits complexity. The paper also gives an algorithm that counts the number of non-isomorphic views in the network in $O(n^6 \log n)$ time complexity if a view is given in the compressed form. These algorithms imply that some well-studied problems, including the leader election problem, can deterministically be solved in $O(n)$ rounds with polynomial bit and time complexity on an anonymous n-party network of any topology.

Keywords: Analysis of Algorithms and Problem Complexity, Distributed Networks.

1 Introduction

View is a labeled directed graph, which was introduced by Yamashita and Kameda [15, 16, 17]. The view $T(v)$ of a node v in an undirected graph G is a rooted tree obtained by maximally sharing the common prefix of every pair of labeled paths away from node v. Yamashita and Kameda showed that, for any anonymous network (i.e., a network that does not guarantee that every party has a unique identifier [3]) whose underlying graph is G, the view $T(v)$ represents all information about the network that the party corresponding to v can learn by exchanging messages with its neighbors. They gave the necessary and sufficient condition in terms of view under which a unique leader can deterministically be elected on an anonymous network. They further investigated the conditions on the solvability of other problems such as constructing a spanning tree and electing an edge (of the underlying graph) on an anonymous network. These conditions naturally give algorithms that output solutions to the problems. Subsequently, Kranakis, Krizanc and van den Berg [11] showed that Boolean functions with n input bits distributed over n parties (and more general problems) can be computed by using view.

In solving the above problems with views, the key steps are to construct a view for every party and to test if two views are isomorphic to each other. The amount of resources such as time and communication required for these key steps are finite for the following reason: Norris [13] proved that, for any two nodes u and v, view $T(u)$ is isomorphic to $T(v)$ if and only if $T^{n-1}(v)$ is isomorphic to $T^{n-1}(u)$ (improving the result of Yamashita and Kameda), where $T^{n-1}(u)$ [$T^{n-1}(v)$] is a subtree of depth $n-1$ obtained by truncating $T(u)$ [$T(v)$] at depth $n-1$. This means that it is sufficient for every party u to construct a view $T^{2(n-1)}(u)$ of depth $2(n-1)$ in order to know the views of all nodes, since $T^{2(n-1)}(u)$ contains $T^{n-1}(v)$ for every v as a subtree. However, the local computation time and the amount of communication taken to construct views or to test their isomorphism are still exponential in n, since the size of $T^{2(n-1)}(u)$ is exponential. Kranakis, Krizanc and van den Berg [11] introduced a polynomial-communication algorithm to construct views (in a recursive form), in which every party constructs view $T^{2(n-1)}(u)$ with $O(\Delta m n^3 \log n)$-bit communication in at most $2n^2$ rounds, if the number n of parties is given to every party,[*] where Δ is the maximum degree over all nodes and m is the number of edges in the underlying graph. This implies that the problems of electing a unique leader, electing a unique edge and computing a Boolean function can be solved (if they are solvable) on an anonymous network of any topology with the above communication cost. We are also interested in the maximum number of local computation steps per party of a distributed algorithm, where the maximum is over all parties, all possible inputs and all possible executions of the algorithm. Let us call the maximum number the *time complexity* or the *time* of the algorithm. As

[*]In another setting where the graph diameter D is given in addition to n, the algorithm in Ref. [11] runs in at most $2Dn$ rounds with $O(D\Delta m n^2 \log n)$ to construct $T^{2(n-1)}(u)$ for each u. For instance, if $D = O(1)$, it runs in $O(n)$ rounds with a better bit complexity than ours.

for the time complexity of local computation, It is easy to see that views can be (implicitly) constructed in polynomial time with the algorithm in Ref. [11], but it is unclear whether we can test the isomorphism of two views in polynomial time even if they are given in the recursive form.

Our results are the following two algorithms that work on an anonymous network of any unknown topology:

1. There is an algorithm in which every party u constructs view $T^{2(n-1)}(u)$ in a certain compressed form with $O(\Delta mn^3 \log \Delta)$-bit communication in at most $2n$ rounds, if the number n of parties is given to u. The time complexity of the algorithm is $O(\Delta^2 n^3 (\log n)^2)$. If the diameter of the underlying graph is $\Omega(n)$, the round complexity of our algorithm is optimal up to constant factor.

2. There is an algorithm in which every party locally computes the number of non-isomorphic views with time complexity $O(\Delta n^5 \log n)$ for a given view of depth $2(n-1)$ in the compressed form (actually, the algorithm can easily be modified so that it can output the quotient graph).

Specifically, a sequential run of these two algorithms deterministically solves each of the following problems (if it is solvable) in $2n$ rounds with $O(n^6 \log n)$-bit communication and $O(n^6 \log n)$ time for an anonymous network of any topology if the number n of parties is given to every party: the leader election problem, the edge election problem, computing any symmetric Boolean function with n input bits distributed over all parties (a symmetric Boolean function is a function whose value does not change for any permutation over input bits, e.g., AND and OR with n bits). These are the first linear-round algorithms with polynomial time and bit complexities that work for any network topology. Our algorithms can handle the case where each party has a label that is not necessarily unique, and/or the case where the underlying graph is directed (with some modifications).

The idea for constructing a view in the compressed form is based on the following simple observation: There are at most n non-isomorphic views, since there are exactly n parties on the network. Thus, sharing isomorphic subtrees rooted at the same level of the view results in a graph with at most n nodes at each level. We call this compressed form of a view a *folded view*. Our algorithm constructs a folded view level by level in a way similar to the straightforward algorithm for constructing a view. The crux is how to construct a folded view of depth k from folded views of depth $k-1$ *without unfolding* them, since otherwise the time complexity would be exponential in n. A useful fact is that views are not folded *too much* in the folded-view, i.e., only nodes at the same level may be shared. From this fact, we will prove that folded views can be constructed in polynomial time complexity. Our second algorithm computes a set of nodes in the given folded view such that no two nodes in the set were originally the roots of isomorphic views. For this, we want to test the isomorphism of the views associated with any two nodes in the folded view. An easy way would be to restore the views and then test their isomorphism. However, it would take exponential time. Our algorithm performs this test without unfolding the folded view.

Related Work For anonymous *directed* networks, Boldi et al. [4], and Boldi and Vigna [6] used certain kinds of graph homomorphism, i.e., fibration and covering, to define several notions that captures the information that each party can gather on the anonymous network: universal total graphs (coverings) and minimal bases (which are similar to view and quotient graphs [16]). They gave necessary and sufficient conditions for electing a unique leader in terms of these notions. They also presented an algorithm that partitions the set of parties into equivalence classes under the isomorphism of universal total graphs. Since the algorithm is based on the same idea as is used by Yamashita and Kameda [16] for constructing view, the bit complexity is exponential in n.

Computing on an anonymous network was first considered by Angluin [3]. Since her seminal work, there have been a lot of works on computing on anonymous networks. For the leader election problem on an anonymous network, interested readers should consult Refs. [2, 18] and the references in them. Recently, Das, Flocchini, Nayak and Nicola [7], and Chalopin and Métivier [8] developed leader election algorithms with polynomial bits of communication that do *not* use view (the former paper actually discusses the mobile agent election problem, but their algorithms can easily be transformed into ones for the leader election problem). However, their algorithms all require $\Omega(n^2)$ rounds in the worst case (with communication bits that are comparable to or fewer than ours). If we are allowed to use quantum computation and communication, the situation is quite different. It is proved in Refs. [14, 9] that there are quantum algorithms that exactly elect a unique leader with *polynomial* time and bit complexities on an anonymous quantum network of *any topology*. Many other problems, including solitude detection and function evaluation, have also been studied (e.g., Refs. [1, 11, 10, 5]).

We will often refer to Appendices A-G. They can be found in the supplemental file.

2 Preliminaries

2.1 The distributed network model

A *distributed system* (or *network*) is composed of multiple parties and bidirectional communication links connecting parties. When the parties and links are regarded as nodes and edges, respectively, the topology of the distributed system is expressed by an undirected connected graph with no self-loops and no multiple edges, denoted by $G = (V, E)$. In what follows, we may identify each party/link with its corresponding node/edge in the underlying graph for the system, if it is not confusing. Every party has *ports* corresponding one-to-one to communication links incident to the party. Every port of party l has a unique label i, $1 \le i \le d_l$, where d_l is the number of parties adjacent to l. More formally, G has a *port*

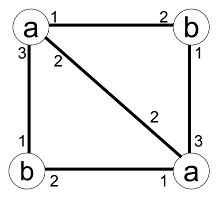

Figure 1: A graph $G = (V, E)$ with a port numbering σ and a node labeling $X\colon V \to \{a, b\}$.

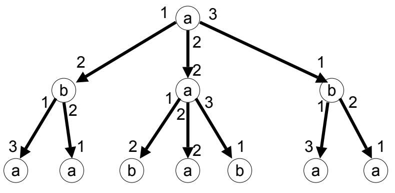

Figure 2: The view $T^2_{G,\sigma,X}(v)$ of depth 2, where v is the left-upper node of G in Fig. 1.

numbering [16], which is a set σ of functions $\{\sigma[v] \mid v \in V\}$ such that, for each node v of degree d_v, $\sigma[v]$ is a bijection from the set of edges incident to v to $\{1, \dots, d_v\}$. It is stressed that each function $\sigma[v]$ is defined independently of parties other than v. Just for ease of explanation, we assume that, for every party l, the port i of the party l corresponds to the link connected to the i^{th} adjacent party of the party l. In our model, each party knows the number of his ports and can appropriately choose one of his ports whenever he transmits or receives a message.

Initially, every party l has local information I_l, the information that only party l knows, such as his local state and the number of his adjacent parties, and global information I_G, the information shared by all parties (if it exists), such as the number of parties in the system (there may be some information shared by some but not all parties, but such a situation does not need to be considered to define anonymous networks). Every party l runs the same algorithm for the local and global information, I_l and I_G, given as its arguments. If all parties have the same local information except for the number of their ports, the system and the parties in the system are said to be *anonymous*. For instance, if the underlying graph of an anonymous network is regular, this is essentially equivalent to the situation in which every party has the same identifier (since we can regard the local information I_l of each party l as his identifier).

Distributed systems are either synchronous or asynchronous. In the synchronous case, message passing is performed synchronously. The unit interval of synchronization is called a *round* (for the definition, see, e.g., Ref. [12]). In the asynchronous setting, the above sequential steps of each party work asynchronously, and the time taken by each message passing is finite but unbounded.

2.2 View

We will review the *view* according to the definition by Yamashita and Kameda [16]. Let $G = (V, E)$ be the underlying undirected graph of a network, and let $n = |V|$. Suppose that each party corresponding to node $v \in V$, *or* simply party v, has a value $x_v \in U$ for a non-empty finite set U, and define a mapping $X\colon V \to U$ as $X(v) = x_v$. We use the value given by X to identify the label of node in G. For each v in G with port numbering σ and node labeling mapping X, *view* $T_{G,\sigma,X}(v)$ is a labeled, rooted directed tree with infinite depth defined recursively as follows: (1) $T_{G,\sigma,X}(v)$ has a unique root u, which is labeled with $\mathsf{label}(u) = X(v)$, where u is a node corresponding to v; (2) For each vertex v_j adjacent to v in G, view $T_{G,\sigma,X}(v)$ has vertex u_j labeled with $X(v_j)$, and a directed edge from root u to u_j with label, $\mathsf{label}((v, v_j)) = (\sigma[v](v, v_j), \sigma[v_j](v, v_j))$; (3) u_j is the root of $T_{G,\sigma,X}(v_j)$. It should be noted that, although any underlying graph G is undirected, view $T_{G,\sigma,X}(v)$ is directed. It should also be stressed that v, v_j are not identifiers of parties and are introduced just for definition. For simplicity, we often use $T(v)$ or $T_X(v)$ instead of $T_{G,\sigma,X}(v)$, if it is not confusing. The *view of depth h for v*, denoted by $T^h_X(v)$, is the subtree of depth h in $T_X(v)$ with the same root as $T_X(v)$. For any node u of a view, we use $\mathsf{depth}(u)$ to represent the depth of u, i.e., the length of the path from the root to u, in the view. For instance, it holds that $\mathsf{depth}(u) = h$ if u is a leaf of $T^h_X(v)$. For any node u in a view and its corresponding party l, if an outgoing edge e of u corresponds to the communication link incident to party l via port i, we call edge e *the* i^{th} *edge of* u, and denote it by $e_i(u)$ and the destination of $e_i(u)$ by $\mathsf{Adj}_i(u)$. Here and hereafter, we mean the nodes w and w' of a directed edge (w, w') by the *source* and *destination*, respectively, of the edge. For example, the tree shown in Fig. 2 is the view of depth 2 for the left-upper node of a labeled undirected graph G given in Fig. 1.

If two views $T_X(v)$ and $T_X(v')$ for $v, v' \in V$ are isomorphic (with respect to edge labels and node labels, but ignoring local names of vertices such as u_i), their relation is denoted by $T_X(v) \equiv T_X(v')$. With this relation, V is partitioned into equivalence classes; v and v' are in the same class if and only if $T_X(v) \equiv T_X(v')$. In Refs. [16, 18], it was proved that all classes have the same cardinality for fixed G, σ and X. Let us denote the cardinality by $c_{G,\sigma,X}$, or simply c_X.[†] We denote the set of non-isomorphic views by $\Gamma_{G,\sigma,X}$, i.e., $\Gamma_{G,\sigma,X} = \{T_{G,\sigma,X}(v)\colon v \in V\}$, and the set of non-isomorphic views of depth h by $\Gamma^h_{G,\sigma,X}$, i.e., $\Gamma^h_{G,\sigma,X} = \{T^h_{G,\sigma,X}(v)\colon v \in V\}$. For simplicity, we may use Γ_X and Γ^h_X instead of $\Gamma_{G,\sigma,X}$ and $\Gamma^h_{G,\sigma,X}$, respectively. We can see that $c_X = n/|\Gamma_X|$, since the number of views isomorphic to $T_X \in \Gamma_X$ is constant over all T_X. For any subset S of U, let $\Gamma_X(S)$ be the maximal subset of Γ_X such that any view $T_X \in \Gamma_X(S)$ has its root labeled with a value in S. Thus the number, $c_X(S)$, of parties having values in S is expressed as follows: $c_X(S) = c_X \cdot |\Gamma_X(S)| = n \cdot |\Gamma_X(S)|/|\Gamma_X|$.

[†]The maximum value of $c_{G,\sigma,X}$ over all port numbering σ is called *symmetricity* $\gamma(G, X)$ and used to give the necessary and sufficient condition for exactly solving LE_n in anonymous classical networks [16].

To compute $c_X(S)$, every party v constructs $T_X^{2(n-1)}(v)$, and then computes $|\Gamma_X|$ and $|\Gamma_X(S)|$ from $T_X^{2(n-1)}(v)$ as follows: To construct $T_X^h(v)$, every party v constructs $T_X^0(v)$, i.e., the root of $T_X^h(v)$, in the first round. If every party v_j adjacent to v has $T_X^{i-1}(v_j)$ in the ith round, v can construct $T_X^i(v)$ in the $(i+1)$st round by exchanging a copy of $T_X^{i-1}(v)$ for a copy of $T_X^{i-1}(v_j)$ for each j. By induction, in the $(h+1)$st round, each party v can construct $T_X^h(v)$. It is clear that, for each $v' \in V$, at least one node in $T_X^{n-1}(v)$ corresponds to v', since there is at least one path of length of at most $(n-1)$ between any pair of parties. To compute $|\Gamma_X|$ and $|\Gamma_X(S)|$, every party v needs to check the equivalence of every pair of views rooted in $T_X^{n-1}(v)$. This can be done with $T_X^{2(n-1)}(v)$ in finite steps, since $T_X(v) \equiv T_X(v')$ if and only if $T_X^{n-1}(v) \equiv T_X^{n-1}(v')$ for $v, v' \in V$ [13]. This implies that $|\Gamma_X|$ and $|\Gamma_X(S)|$ can be computed from $T_X^{2(n-1)}(v)$. Obviously, this simple construction takes $O(n)$ rounds but requires an exponential number of communication bits.

3 Folded view and its Properties

This section defines a folded view (*or* an f-view for shorthand) and presents its basic properties. They are essential for the algorithms described in the following sections.

3.1 Terminology

We will show that the folded view has all information represented by the corresponding view. To describe such information, we introduce a notion, *path set,* which is equivalent to a view in the sense that any view can be reconstructed from the corresponding path set, and vice versa.

A path set, $P_{G,\sigma,X}(v)$, is defined for every view $T_{G,\sigma,X}(v)$. Let $u_{\rm root}$ be the root of $T_{G,\sigma,X}(v)$. Suppose that every edge of a view is directed and its source is the end closer to $u_{\rm root}$. The path set $P_{G,\sigma,X}(v)$ is the set of directed labeled paths away from $u_{\rm root}$ with infinite length in $T_{G,\sigma,X}(v)$. More formally, let $p = (u_0, e(u_0), u_1, \cdots)$ be an infinite-length directed labeled path away from $u_0(= u_{\rm root})$, where u_i is a labeled node in $T_{G,\sigma,X}(v)$ and $e(u_i)$ is the directed labeled edge from u_i to u_{i+1}. We define $P_{G,\sigma,X}(v)$ as the set of all such paths p in $T_{G,\sigma,X}(v)$. For any view $T^h_{G,\sigma,X}(v)$ of depth h, we naturally define $P^h_{G,\sigma,X}(v)$: $P^h_{G,\sigma,X}(v)$ is the set of all directed labeled paths of length h away from $u_{\rm root}$ in $T^h_{G,\sigma,X}(v)$. In the following, we simply call an element in a path set, a *path*, and identify the common length of the paths in a path set with the *length* of the path set. We define the isomorphism between path sets in a standard way: For two path sets $P_{G,\sigma,X}(u)$ and $P_{G,\sigma,X}(v)$, we say that $P_{G,\sigma,X}(u)$ is *isomorphic* to $P_{G,\sigma,X}(v)$, if there exists a directed-edge-preserving bijection η from the set of nodes in $P_{G,\sigma,X}(u)$ to the set of nodes in $P_{G,\sigma,X}(v)$ that preserves the edge and node labels. Similarly, we define the isomorphism between two path sets of finite length, $P^h_{G,\sigma,X}(u)$ and $P^h_{G,\sigma,X}(v)$.

By the above definition, $P^h_{G,\sigma,X}(v)$ is easily obtained by traversing view $T^h_{G,\sigma,X}(v)$. On the other hand, given $P^h_{G,\sigma,X}(v)$, we can construct the view rooted at $u_{\rm root}$ by sharing the maximal common (i.e., isomorphic) prefix of any pair of paths in $P^h_{G,\sigma,X}(v)$. In this sense, $P^h_{G,\sigma,X}(v)$ has all information represented by view $T^h_{G,\sigma,X}(v)$. Let u^j be any node at depth j in $T^h_{G,\sigma,X}(v)$, and suppose that u^j corresponds to node v_{u^j} in G. Since a view is defined recursively, we can define the path set $P^{h'}_{G,\sigma,X}(v_{u^j})$ for the h'-depth subtree rooted at u^j, as the set of h'-length directed paths away from u^j for $h' \leq h - j$. To avoid complicated notations, we may use $P^{h'}_{G,\sigma,X}(u^j)$ instead of $P^{h'}_{G,\sigma,X}(v_{u^j})$. We call $P^{h'}_{G,\sigma,X}(u^j)$ *the path set of length h' at u^j*. In particular, when h' is the length of the path from u^j to a leaf, i.e., $h' = h - j$, we simply call $P^{h'}_{G,\sigma,X}(u^j)$ *the path set at u^j.*

3.2 Folded view

We now define a key operation, called the *merging operation*, which folds a view. Although the formal definition described later is lengthy, the idea of the merging operation is very simple: two isomorphic directed subtrees rooted at the same level in a view can be merged without any loss of information, and repeating similar procedures will result in a compact graph. For instance, there are seven nodes at the bottom level in the view shown in Fig. 2. Among them, however, there are only two distinct nodes (i.e., two non-isomorphic subgraphs rooted at the bottom level). Let us repeatedly merge a pair of nodes with the same label at the bottom level into one node until no two nodes with the same label exist at the level. Then the resulting graph has only two nodes as shown in Fig. 3. The resulting graph has fewer nodes than the original view, but it is easy to see that we can restore the view from the graph. Hence, the graph has all the information that the view has. Next look at each node u in the middle level of the graph in Fig. 3, and identify the subgraph rooted at u, i.e., the subgraph induced by u and all nodes that can be reached from u via a directed path. Among these subgraphs, there are two non-isomorphic subgraphs. Then we merge isomorphic subgraphs to obtain the graph shown in Fig. 4. Now we give a formal definition of the merging operation. We first give the definition of the base case, where the merging operation is applied to a *view*, and then extend it so that it can be applied repeatedly.

Definition 1 (Merging Operation (Base Case)) *Let u and u' be two nodes at the same depth in a view T (possibly of finite depth). Suppose that the two nodes u and u' satisfy the following condition (1):*

(1) *u and u' have the same label, i.e.,* $\mathsf{label}(u) = \mathsf{label}(u')$.

Moreover, if both u and u' have at least one outgoing edge (i.e., neither u nor u' is a leaf), suppose that they satisfy the following conditions (2) and (3) in addition to (1).

(2) *u and u' have the same number of outgoing edges.*

(3) *The following hold for every $i = 1, \ldots, d_u$, where d_u is the number of outgoing edges of u.*

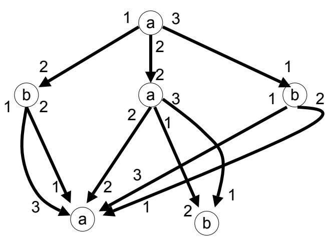

Figure 3: The folded view obtained from $T^2_{G,\sigma,X}(v)$ in Fig. 2 by applying the merging operation at the bottom level.

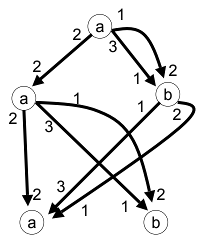

Figure 4: The smaller folded view obtained from the folded view in Fig. 3 by applying the merging operation at the middle level.

(3.1) *The* i^{th} *edge of* u *and the* i^{th} *edge of* u', $e_i(u)$ *and* $e_i(u')$, *have the same label, i.e.,* $\mathsf{label}(e_i(u)) = \mathsf{label}(e_i(u'))$.

(3.2) *The path sets at* $\mathsf{Adj}_i(u)$ *and* $\mathsf{Adj}_i(u')$ *are isomorphic.*

Then the merging operation is defined as follows for the nodes u *and* u' *satisfying the above conditions.*

(A) *Eliminate* u' *and all outgoing edges of* u' *from* T, *and then eliminate all nodes that cannot be reached from the root of* T *via a directed path.*[‡]

(B) *For every incoming edge* e' *of* u' *(say,* $e' = (v, u')$ *for some node* v *in* T*), eliminate the edge* $e' = (v, u')$ *from* T *and add a new edge* $e = (v, u)$ *labeled with* $\mathsf{label}(e')$ *(i.e., the label of the eliminated edge).*

After applying the merging operation to a view, the resulting graph $\hat{T}$ is not a view any more. However, we can naturally extend the definition of the merging operation so that it works against any graph obtained by applying the merging operation to a view. Before extending the definition, we need some observations. Obviously, the merging operation never eliminates the root of a view. Further, the merging operation does not change the length of the directed path from the root to each (remaining) node. Thus, we define the depth of each node u that remains after applying the merging operation as the length of the path from the root to u and denote it again by $\mathsf{depth}(u)$. In addition, the merging operation does not change the set of outgoing edges of every remaining node (hence we use $e_i(u)$ and $\mathsf{Adj}_i(u)$ in the same way as in the case of view). Finally, let us define the path set at a node u in a directed graph (in a way similar to the case of view) as the set of all directed labeled paths away from u. Based on these observations, we will extend the above definition of the merging operation by just replacing T with $\hat{T}$. It is easy to see that the above observations still hold after applying the merging operation twice. In this way, we inductively extend the base-case definition of the merging operation.

Definition 2 (Merging Operation (General Case)) *Let* $G^{(0)}$ *be a view and let* $G^{(i)}$ *be any rooted directed graph obtained by applying the merging operation to* $G^{(0)}$ i *times. Then the merging operation for* $G^{(i)}$ *is defined in the same way as Definition 1 except that view* T *is replaced with* $G^{(i)}$.

We call any directed graph obtained by applying the merging operation once or more to a view, a *folded view (f-view)*. Since views of finite depth are sufficient for our use, we only consider f-views that are obtained from views of finite depth hereafter. For any view $T^h_{G,\sigma,X}(v)$, its minimal f-view is uniquely determined up to isomorphism as will be proved later and is denoted by $\widetilde{T}^h_{G,\sigma,X}(v)$. For finite-depth f-views, we can define the path set similarly: the path set of length h at node u in an f-view is the set of all directed labeled paths of length h from u in the f-view. The following lemma states that an f-view contains all information that the original view has. The proof is given in Appendix A.

Lemma 3 *For any (f-)view, the merging operation does not change the path set at every node of the (f-)view up to isomorphism if the node exists after the operation. Thus, the path set of any f-view obtained from a view by applying the merging operation is isomorphic to the path set of the view.*

We can characterize f-views in terms of path sets. Informally, for every non-isomorphic path set P at a node at any depth j in a view, any f-view obtained from the view has at least one node u at depth j such that the path set at u is isomorphic to P. Before giving a formal characterization of f-views, we need to define some notations. We define $PF(j, T^h_{G,\sigma,X}(v))$ as the family of non-isomorphic path sets $P^{h-j}_{G,\sigma,X}(u^j)$ at all nodes u at depth j in $T^h_{G,\sigma,X}(v)$. Namely, $PF(j, T^h_{G,\sigma,X}(v))$ is defined as

$$\left\{ P^{h-j}_{G,\sigma,X}(u) \colon\ u \text{ is a node in } T^h_{G,\sigma,X}(v) \text{ with } \mathsf{depth}(u) = j \right\},$$

where no two isomorphic path sets are included. For any path set P, let $P|_x$ be the set obtained by cutting off the first node and edge from all those paths in P that have x as the first edge label. Namely, for every path in P of the form of $(u_0, e(u_0), u_1, e(u_1), \cdots)$ with $\mathsf{label}(e(u_0)) = x$, its suffix, $(u_1, e(u_1), \cdots)$, is an element of $P|_x$; conversely, every element of $P|_x$ is such a suffix of a path of the form $(u_0, e(u_0), u_1, e(u_1), \cdots)$ with $\mathsf{label}(e(u_0)) = x$. Finally, we

[‡]It is not trivial to eliminate all nodes that cannot be reached from the root via a directed path after eliminating u' and its outgoing edges. However, such a situation does not occur, i.e., all nodes still have incoming edges after eliminating u' and its outgoing edges, if we repeatedly apply the merging operation from the bottom level to upper levels as in our algorithm described later. Therefore, we virtually need to eliminate only u' and the outgoing edges of u' from T in our algorithm.

define a class $\mathcal{D}^h$ of labeled directed acyclic graphs $G^f = (V^f, E^f)$[§] satisfying the following conditions:

- V^f is the union of disjoint sets V_j^f ($j = 0, \ldots, h$) of nodes with $|V_0^f| = 1$,
- E^f is the union of disjoint sets E_j^f of labeled directed edges from nodes in V_j^f to nodes in V_{j+1}^f for $j = 0, \ldots, h-1$, and
- every node in V_j^f ($j = 1, \ldots, h$) can be reached from $u_r \in V_0^f$ via directed edges in E^f.

In addition, we define a subclass $\mathcal{D}^h(T^h_{G,\sigma,X}(v))$ of $\mathcal{D}^h$ as a set of graphs $G^f = (V^f, E^f) \in \mathcal{D}^h$ such that there is a mapping θ from V_j^f onto $PF(j, T^h_{G,\sigma,X}(v))$ for each $j = 0, \ldots, h$ satisfying the following two conditions:

C1 Each node $u \in V^f$ has the label that is identical to the common label of the first nodes of paths in $\theta(u)$

C2 For each $u \in V^f$, u has a unique outgoing edge (u, u') with label x if and only if there is a path in $\theta(u)$ whose first edge is labeled with x, and $\theta(u') = \theta(u)|_x$.

Now we give characterizations of f-views. The proofs are given in Appendix A.

Lemma 4 *A graph $\widehat{T}^h_{G,\sigma,X}(v)$ is an f-view of $T^h_{G,\sigma,X}(v)$ if and only if $\widehat{T}^h_{G,\sigma,X}(v)$ is in $\mathcal{D}^h(T^h_{G,\sigma,X}(v))$.*

Corollary 5 *Any minimal f-view $\widetilde{T}^h_{G,\sigma,X}(v)$ is unique up to isomorphism and has exactly $|PF(j, T^h_{G,\sigma,X}(v))|$ nodes at depth j ($0 \le j \le h$). The minimal f-view of depth h for any n-party distributed network has $O(hn)$ nodes and $O(h\Delta n)$ edges, where Δ is the maximum degree over all nodes of the underlying graph G.*

4 Folded-View Minimization

Lemma 4 and Corollary 5 imply that the minimal f-view can always be obtained by repeatedly apply the merging operation in any order to a view $T^h_{G,\sigma,X}(v)$ until it can no longer be applied (non-minimal f-view must have more than $PF(j, T^h_{G,\sigma,X}(v))$ nodes at depth j for some j, which means that there exists at least one pair of nodes at depth j such that the path sets at the nodes are isomorphic). Based on Lemma 4 and Corollary 5, this section presents an algorithm that outputs the minimal f-view for a given (f-)view. The algorithm will be used as a subroutine when constructing a minimal f-view from scratch.

Let $\widehat{T}^h$ be a (f-)view of depth h to be minimized. The minimization algorithm applies the merging operation to every node in $\widehat{T}^h$ in a bottom-up manner, i.e., in decreasing order of node depth. Clearly, this ensures that no application of the merging operation at any depth j creates a new node pair at any depth $j' > j$ to which the merging operation is applicable. Thus, no more merging operations can be applied after the algorithm traverses all nodes (in the bottom-up manner). It follows that the algorithm outputs the minimal f-view. In order to correctly apply the merging operation to nodes u and u' at depth j in $\widehat{T}^h$, we need to decide if the conditions (1), (2) and (3) of Definition 1 (and 2) hold. Specifically, condition (3.1) is equivalent to the fact that $\mathsf{Adj}_i(u)$ is identical to $\mathsf{Adj}_i(u')$, when the algorithm is processing nodes at depth j. This is because the algorithm works in the bottom-up manner and thus the path sets at no two nodes at depth $j+1$ are isomorphic. The following lemma states the time complexity of the minimization algorithm. The details of the algorithm and the proof of the lemma can be found in Appendix B.

Lemma 6 *Let U be a non-empty finite set. Let T be an f-view for a distributed network $G = (V, E)$ with n nodes labeled by $X\colon V \to U$, and let V^f be the set of nodes of T. Then, the minimization algorithm for input T runs with time complexity $O(|V^f|(\log|V^f|)(\log|U| + \Delta \log(n|V^f|)))$, where Δ is the maximum degree of the nodes in G.*

5 Constructing Minimal Folded-View

We now describe the entire algorithm that constructs a minimal f-view of depth h from scratch by using the f-view minimization algorithm as a subroutine. The algorithm for constructing a minimal f-view basically follows the straightforward algorithm for constructing an view (described in Section 2.2). The difference is that parties exchange *f-views* (instead of views) and perform the minimization subroutine on them: To construct an f-view $\widehat{T}^j_{G,\sigma,X}(v)$ of depth j, every party connects the root of each received minimal f-view $\widetilde{T}^{j-1}_{G,\sigma,X}(v_i)$ of depth $j-1$ with a newly created node (which will be the root of $\widehat{T}^j_{G,\sigma,X}(v)$) *without* unfolding them, and then applies the f-view minimization algorithm to $\widehat{T}^j_{G,\sigma,X}(v)$. It is easy to see that $\widehat{T}^j_{G,\sigma,X}(v)$ is an f-view (since every subgraph connected with the root is an f-view, which implies that $\widehat{T}^j_{G,\sigma,X}(v)$ can be obtained from view $T^j_{G,\sigma,X}(v)$ by applying the merging operation to every subtree rooted at depth 1). Thus, the minimization algorithm can be applied to $\widehat{T}^j_{G,\sigma,X}(v)$. More precisely, each party l with d_l adjacent parties and the label x_l performs the f-view construction algorithm described in Fig. 5 with h, d_l and x_l, in which we assume that v is the node corresponding to party l in the underlying graph. The proof of the following theorem can be found in Appendix C.

Theorem 7 *Let U be a non-empty finite set. For a distributed network $G = (V, E)$ with n parties labeled by $X\colon V \to U$, there is an algorithm that constructs the minimal f-view of depth $h \in O(n)$ in $h+2$ rounds and $O(\Delta h^2 n(\log n)(\log|U|n^{\Delta}))$ time for each party with $O(mh^2 n \log(|U|\Delta^{\Delta}))$-bit communication over all parties, where m and Δ are the number of edges and the maximum degree, respectively, in G.*

[§]The superscript "f" is added to emphasize that G^f is not the underlying graph of the network and to implicitly mean that a subclass of $\mathcal{D}^h$ will be shown to be equivalent to f-views.

Folded-View Construction Algorithm

```
Input: integers $h$, $d$ and $x$.

Output:
    minimal f-view $\widetilde{T}^h_{G,\sigma,X}$, where $X$ is the underlying mapping naturally induced by the $x$ values of all parties.

Preprocess:
    For each $1 \le i \le d$, send out a message containing "$i$" via port $i$ to inform the corresponding neighbor of the port number $i$ of the sender.

1. Generate $\widetilde{T}^0_{G,\sigma,X}(v)$, which consists of only one node with label $x$.
2. For $j := 1$ to $h$, perform the following steps.
    2.1 Send a copy of the minimal f-view $\widetilde{T}^{j-1}_{G,\sigma,X}(v)$ to every adjacent party.
    2.2 Receive $\widetilde{T}^{j-1}_{G,\sigma,X}(v_i)$ via port $i$ for every $1 \le i \le d$, where $v_i$ is the node corresponding to the party connected via port $i$.
    2.3 Construct an f-view $\widehat{T}^j_{G,\sigma,X}(v)$ from the received $\widetilde{T}^{j-1}_{G,\sigma,X}(v_i)$'s as follows.
        2.3.1 Let root $u_{\mathrm{root}}$ of $\widehat{T}^j_{G,\sigma,X}(v)$ be $\widetilde{T}^0_{G,\sigma,X}(v)$.
        2.3.2 Connect $u_{\mathrm{root}}$ with the root of $\widetilde{T}^{j-1}_{G,\sigma,X}(v_i)$ via an edge labeled with $(i, i')$, where $i'$ is the port through which $v_i$ sent $\widetilde{T}^{j-1}_{G,\sigma,X}(v_i)$, i.e., $i' := \sigma[v_i](v, v_i)$.
    2.4 Perform the f-view minimization algorithm with $\widehat{T}^j_{G,\sigma,X}(v)$ to obtain $\widetilde{T}^j_{G,\sigma,X}(v)$.
3. Output $\widetilde{T}^h_{G,\sigma,X}(v)$.
```

Figure 5: Folded-view construction algorithm.

6 Counting the Number of Parties

In many cases, the purpose of constructing a view is to compute $\left|\Gamma^{(n-1)}_X(S)\right|$ for any set $S \subseteq U$ where $X: V \to U$ in order to compute the number of parties labeled with elements in S, i.e., $c_X(S) = n\left|\Gamma^{(n-1)}_X(S)\right| / \left|\Gamma^{(n-1)}_X(U)\right|$. We will describe an algorithm that computes $\left|\Gamma^{(n-1)}_X(S)\right|$ for given minimal f-view $\widetilde{T}^{2(n-1)}_X(v)$. It is obvious from the results in [16, 4, 6] that the quotient graph (or the minimum base) of the underlying graph directly gives $\left|\Gamma^{(n-1)}_X(S)\right|$. We should note that the minimal f-view is not the quotient graph of the underlying graph (see Appendix D for an example). For a given minimal f-view, the corresponding quotient graph can be obtained by picking up the subgraph induced by all nodes at depth up to $n-1$ and merging the nodes in the subgraph (and removing redundant edges) if the $(n-1)$-depth views rooted at them (when unfolding the f-view) are isomorphic. Our algorithm outputs the set of the nodes in a minimal f-view such that the views at them are not isomorphic (and it can easily be modified so that it will output the quotient graph). For this, the algorithm tests whether or not two sub-f-views are obtained from isomorphic views. Here "*a sub-f-view* " (of depth h) at a node u means the subgraph of an f-view induced by a node u in the f-view and all other nodes that can be reached from u via a directed path (of length at most h). The test would be trivial if we were allowed to restore the views corresponding to the sub-f-views. However, this would lead to exponential time complexity. We should also note that two sub-f-views are not necessarily isomorphic even if their corresponding views are isomorphic (see Appendix D for an example).

6.1 View Counting Algorithm

For a given minimal f-view $\widetilde{T}^{2(n-1)}_X(v)$, the algorithm computes a maximal set W of the nodes whose depth are at most $n-1$ in $\widetilde{T}^{2(n-1)}_X(v)$ such that no pair of the path sets of length $n-1$ at nodes in W are isomorphic. A more concrete description is given in Appendix E. We then compute $|\Gamma^{(n-1)}_X(S)|$ by counting the number of the nodes in W which are labeled with elements in S. Let $W_S \subseteq W$ be the set of nodes labeled with elements in S. Then, Lemma 3 implies that $c_X(S) = \frac{n}{|W|} \cdot |W_S|$.

We will describe how to implement a subroutine, Equivalence_Check, that tests whether or not a pair of sub-f-views in a minimal f-view (of depth $2(n-1)$) have isomorphic path sets of length $n-1$. There are at least three ways to achieve our purpose. They all have the same time complexity up to a constant factor. In particular, they have the time complexity $O(n^5 \Delta \log n)$, if the cardinality of U is bounded by a polynomial in n, where Δ is the maximum degree of the underlying graph. Since the three ways of implementations are interesting in their own right and they would help reader to deeply understand the properties of f-views, we shall show all of them. As stated above, even if such sub-f-views have isomorphic path sets of length $n-1$, they are not necessarily isomorphic to each other. The first implementation uses the following lemma, which says it is sufficient to test if there exists a kind of *homomorphism* between the pair of sub-f-views.

Lemma 8 *Suppose that* $\widehat{T}^{(n-1)}_{X,a}$ *and* $\widehat{T}^{(n-1)}_{X,b}$ *are any two sub-f-views of depth* $(n-1)$ *of a minimal f-view* $\widetilde{T}^{2(n-1)}_X(v)$*, such that, for roots* u_r *and* w_r *of* $\widehat{T}^{(n-1)}_{X,a}$ *and* $\widehat{T}^{(n-1)}_{X,b}$*, respectively,* $\mathsf{depth}(u_r) \le \mathsf{depth}(w_r) \le (n-1)$*. Let* V_a *and* V_b *be the node sets of* $\widehat{T}^{(n-1)}_{X,a}$ *and* $\widehat{T}^{(n-1)}_{X,b}$*, respectively, and let* E_a *and* E_b *be the edge sets of* $\widehat{T}^{(n-1)}_{X,a}$ *and* $\widehat{T}^{(n-1)}_{X,b}$*, respectively.*

Then, $\widehat{T}^{(n-1)}_{X,a}$ *and* $\widehat{T}^{(n-1)}_{X,b}$ *have isomorphic path sets of length* $(n-1)$ *if and only if the following conditions C1 and C2 hold.*

C1: *There is a unique surjective homomorphism* $\phi: V_a \to V_b$ *that preserves node labels:* $\mathsf{label}(u) = \mathsf{label}(\phi(u))$ *for each* $u \in V_a$.

C2: *Let* $E_a(u)$ *and* $E_b(v)$ *represent the set of outgoing edges of* $u \in V_a$ *and* $v \in V_b$*, respectively. Then, there is a family* ψ *of bijective mappings* $\psi_u: E_a(u) \to E_b(\phi(u))$ *for every* $u \in V_a$*, such that* ψ_u *preserves edge-labels,* *i.e.,* $\mathsf{label}(e) = \mathsf{label}(\psi_u(e))$ *for every* $e \in E_a(u)$*, and* ψ_u *maps any edge from* u *to* u' *to an edge from* $\phi(u)$ *to* $\phi(u')$ *for all possible* $u' \in V_a$.

Lemma 8 implies that, to check if sub-f-views $\widehat{T}^{(n-1)}_{X,a}$ and $\widehat{T}^{(n-1)}_{X,b}$ have isomorphic path sets of length $(n-1)$, we only need to test if we can construct ϕ and ψ that meet C1 and C2, or equivalently, test if we can construct ϕ for which ψ exists. To construct such ϕ, we simultaneously traverse $\widehat{T}^{(n-1)}_{X,a}$ and $\widehat{T}^{(n-1)}_{X,b}$ in a breadth-first manner. Every time we visit a node in $\widehat{T}^{(n-1)}_{X,a}$ and the corresponding node in $\widehat{T}^{(n-1)}_{X,b}$, we check if they have the same node label and isomorphic sets of labeled outgoing edges. During the traversal, if a node $\widehat{T}^{(n-1)}_{X,a}$ corresponds to two distinct nodes in $\widehat{T}^{(n-1)}_{X,b}$, we give up constructing ϕ (because this contradicts the fact that ϕ is homomorphism). This is the basic idea of subroutine Equivalence_Check. A more precise description of the subroutine and the proofs of the following lemma and theorem are given in Appendix E.

Lemma 9 *Given two sub-f-views* $\widehat{T}^{(n-1)}_{X,a}$ *and* $\widehat{T}^{(n-1)}_{X,b}$ *of depth* $(n-1)$ *of a minimal f-view* $\widehat{T}^{2(n-1)}_{X}(v)$*, there is an algorithm that outputs "*Yes*" if and only if* $\widehat{T}^{(n-1)}_{X,a}$ *and* $\widehat{T}^{(n-1)}_{X,b}$ *have isomorphic path sets of length* $(n-1)$*. The time complexity is* $O(n^2\Delta\log(n|U|))$*, where* Δ *is the maximum degree over all nodes of the underlying graph of the distributed network.*

Now we give the time complexity of the algorithm.

Theorem 10 *Let* U *be a non-empty finite set. For any distributed network* $G = (V, E)$ *of* n *parties labeled by mapping* $X\colon V \to U$*, there is an algorithm that computes* $|\Gamma^{(n-1)}_X(S)|$ *for any subset* S *of* U *in* $O(n^5\Delta\log(n|U|))$ *time, where* Δ *is the maximum degree over all nodes in* G*, if a minimal f-view* $\widehat{T}^{2(n-1)}_{X}(v)$ *is given to every party.*

Remark 1 *The above statement is somewhat weak, while this results in a simple description of the algorithm. With some elaboration, the time complexity in Lemma 9 can be improved to* $O(n^2\log(n^\Delta|U|))$*. Accordingly, the time complexity in Theorem 10 is improved to* $O(n^5\log(n^\Delta|U|))$*. However, this improvement is meaningful only if node labels are picked from a very large set compared with* n*, i.e.,* $|U| \in n^{\omega(\Delta)}$.

For the second implementation of Equivalence_Check, we need to perform the following preprocess before starting the view counting algorithm: For each node v at depth at most $n-1$ in the given minimal f-view, pick a copy of the sub-f-views of depth $n-1$ at v, and then apply the minimization algorithm to the copy. Then, Equivalence_Check has only to test if the minimized copies of the sub-f-views at the given pair of nodes are isomorphic. The above implementation requires the space to store the minimal sub-f-views at every node at depth at most $n-1$. To save the space, we can minimize the sub-f-views on demand, i.e., just before testing their isomorphism. This increases the complexity by a log factor.

The third implementation is the simplest, but it requires as input a minimal f-view $\tilde{T}^{3(n-1)}_X$ of depth $3(n-1)$ (instead of $2(n-1)$), which needs $3(n-1)+2$ rounds to construct. For given nodes $\hat{u}$ and u at depth at most $n-1$, Equivalence_Check just tests whether the sub-f-view of depth $n-1$ at $\hat{u}$ and u are *isomorphic*. Here, we claim that the sub-f-views are isomorphic if and only if the path sets of length $n-1$ for the sub-f-views are isomorphic. The details of the second and third implementations are given in Appendix E.

6.2 Applications

To compute a symmetric function, it is sufficient to count the number of 1's among the inputs of all parties. Thus, every party constructs a folded view of depth $2(n-1)$ with $U = \{0, 1\}$, and then runs the view counting algorithm to compute $c_X(U)$ and $c_X(S)$, where S is the set $\{1\}$. The complexity follows from Theorems 7 and 10.

To solve the leader election problem, we define U as a singleton set, e.g., $\{0\}$. In the same way as the above, every party computes the set W of roots of non-isomorphic views. If $|W| < n$, then there are multiple parties whose corresponding views are isomorphic. This means that the set of all parties can be partitioned into equivalence classes under the isomorphism of view such that each equivalence class has the same cardinality. Thus, every party outputs "unsolvable". If $|W| = n$, then the view of each party is unique and the party with the first view under the lexicographical ordering will be elected as a unique leader. Notice that the time complexity of picking up the first view is negligible compared to the time complexity of the view counting algorithm. In a similar manner, the edge election problem can be solved by choosing a pair of the lexicographically-smallest isomorphic views that correspond to a pair of neighboring parties.

6.3 Directed Networks

Our algorithms, appeared in Sections 4, 5 and 6, can easily be modified for directed graphs. To handle directed graphs, we need to care about only Norris's result: $T_{G,\sigma,X}(v) \equiv T_{G,\sigma,X}(v')$ if and only if $T^{n-1}_{G,\sigma,X}(v) \equiv T^{n-1}_{G,\sigma,X}(v')$. However, a similar property holds if we appropriately define view for directed graphs, as shown by Boldi and Vigna [6]. More detailed discussions can be found in Appendix F.

Acknowledgments

I am grateful to anonymous referees and editors for suggesting simple alternative algorithms for view counting as well as a lot of comments regarding presentation and important references.

References

[1] Karl R. Abrahamson, Andrew Adler, Lisa Higham, and David G. Kirkpatrick. Tight lower bounds for probabilistic solitude verification on anonymous rings. *Journal of the ACM*, 41(2):277–310, 1994.

[2] Yehuda Afek and Yossi Matias. Elections in anonymous networks. *Information and Computation*, 113(2):312–330, 1994.

[3] Dana Angluin. Local and global properties in networks of processors (extended abstract). In *Proceedings of the Twentieth Annaul ACM Symposium on Theory of Computing*, pages 82–93, 1980.

[4] Paolo Boldi, Shella Shammah, Sebastiano Vigna, Bruno Codenotti, Peter Gemmell, and Janos Simon. Symmetry breaking in anonymous networks: Characterizations. In *Proceedings of the Fourth Israel Symposium on Theory of Computing and Systems*, pages 16–26. IEEE Computer Society, 1996.

[5] Paolo Boldi and Sebastiano Vigna. Computing anonymously with arbitrary knowledge. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 181–188, 1999.

[6] Paolo Boldi and Sebastiano Vigna. Fibrations of graphs. *Discrete Mathematics*, 243:21–66, 2002.

[7] Shantanu Das, Paola Flocchini, Amiya Nayak, and Nicola Santoro. Effective elections for anonymous mobile agents. In *Proceedings of the Seventeenth International Symposium on Algorithms and Computation*, volume 4288 of *Lecture Notes in Computer Science*, pages 732–743. Springer, 2006.

[8] Yves Métivier Jérémie Chalopin. An efficient message passing election algorithm based on Mazurkiewicz's algorithm. *Fundamenta Informaticae*, 80:221–246, 2007.

[9] Hirotada Kobayashi, Keiji Matsumoto, and Seiichiro Tani. Brief announcement: Exactly electing a unique leader is not harder than computing symmetric functions on anonymous quantum networks. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Principles of Distributed Computing*, pages 334–335, 2009.

[10] Evangelos Kranakis and Danny Krizanc. Distributed computing on anonymous hypercube networks. *Journal of Algorithms*, 23(1):32–50, 1997.

[11] Evangelos Kranakis, Danny Krizanc, and Jacov van den Berg. Computing boolean functions on anonymous networks. *Information and Computation*, 114(2):214–236, 1994.

[12] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufman Publishers, 1996.

[13] N. Norris. Universal covers of graphs: Isomoriphism to depth n-1 implies isomoriphism to all depths. *Discrete Applied Mathematics*, 56(1):61–74, 1995.

[14] Seiichiro Tani, Hirotada Kobayashi, and Keiji Matsumoto. Exact quantum algorithms for the leader election problem. In *Proceedings of the Twenty-Second Symposium on Theoretical Aspects of Computer Science*, volume 3404 of *Lecture Notes in Computer Science*, pages 581–592. Springer, 2005.

[15] Masafumi Yamashita and Tsunehiko Kameda. Computing on an anonymous network. In *Proceedings of the Seventh ACM Symposium on Principles of Distributed Computing*, pages 117–130, 1988.

[16] Masafumi Yamashita and Tsunehiko Kameda. Computing on anonymous networks: Part I – characterizing the solvable cases. *IEEE Transactions on Parallel Distributed Systems*, 7(1):69–89, 1996.

[17] Masafumi Yamashita and Tsunehiko Kameda. Computing on anonymous networks: Part II – decision and membership problems. *IEEE Transactions on Parallel Distributed Systems*, 7(1):90–96, 1996.

[18] Masafumi Yamashita and Tsunehiko 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, 1999.

Appendices

A Folded view and its Properties

This section gives the proofs of the lemmas and corollary appeared in Section 3.

Lemma 3 *For any (f-)view, the merging operation does not change the path set at every node of the (f-)view up to isomorphism if the node exists after the operation. Thus, the path set defined for any f-view obtained from a view by applying the merging operation is isomorphic to the path set defined for the view.*

Proof Let u' be the node that will be merged into u (i.e., u' will be eliminated). By the definition of the merging operation, the path set at u is isomorphic to the path set at u'. Thus, by eliminating u' and redirecting all incoming edges of u' to u, the path set at every remaining node does not change up to isomorphism. □

Lemma 4 *A graph $\widetilde{T}^h_{G,\sigma,X}(v)$ is an f-view of $T^h_{G,\sigma,X}(v)$ if and only if $\widetilde{T}^h_{G,\sigma,X}(v)$ is in $\mathcal{D}^h(T^h_{G,\sigma,X}(v))$.*

Proof (⇒) We will prove that, for any f-view obtained by applying the merging operation to the view $T^h_{G,\sigma,X}(v)$, there exists θ that satisfies C1 and C2. From Lemma 3, the merging operation does not change the path set at any node up to isomorphism (if it exists after the operation). It follows that the path set at any node at depth j in the f-view belongs to $PF(j, T^h_{G,\sigma,X}(v))$. Conversely, for every path set P in $PF(j, T^h_{G,\sigma,X}(v))$ there is at least one node at depth j in the f-view such that the path set at the node is isomorphic to P; this is because the merging operation just merges two nodes such that the path sets at the nodes are isomorphic. Let θ map every node u of the f-view to the path set at u. From the above argument, θ is a mapping from V^f_j onto $PF(j, T^h_{G,\sigma,X}(v))$ and satisfies C1. To show that θ satisfies C2, we use simple induction on the sequence of the merging operation. By the definition, view $T^h_{G,\sigma,X}(v)$ meets C2. Suppose that one application of the merging operation transformed an f-view to a smaller f-view, and that θ meets C2 for the f-view before the operation. If we define θ' for the smaller f-view by restricting the domain of θ to the node set of the smaller f-view, then θ' satisfies C2 by the definition of the merging operation.

(⇐) We will prove that any graph $G^f \in \mathcal{D}^h(T^h_{G,\sigma,X}(v))$ for which there is θ satisfying C1 and C2 can be obtained by applying the merging operation (possibly, more than once) to $T^h_{G,\sigma,X}(v)$. Let P be all maximal-length labeled directed paths away from $u_0 \in V^f_0$. We shall show that P is isomorphic to the path set $P^h_{G,\sigma,X}(v)$ defined for $T^h_{G,\sigma,X}(v)$. We first give an inversion of the merging operation that does not change P up to isomorphism when it is applied to G^f, and then show that the view which defines $P^h_{G,\sigma,X}(v)$ is obtained by repeatedly applying the inversion to G^f until the inversion cannot be applied any more. This view is isomorphic to $T^h_{G,\sigma,X}(v)$, since the view is uniquely determined for a fixed path set. It follows that G^f can be obtained from $T^h_{G,\sigma,X}(v)$ by reversing the sequence of the inversion, i.e., applying the merging operation repeatedly.

The inverse operation of the merging operation is defined as follows: if some node $u^j \in V^f_j$ $(1 \le j \le h)$ has multiple incoming edges, say, $e^1, \ldots, e^t \in E_{j-1}$, the inverse operation makes a copy u' of u^j together with its outgoing edges (i.e, creates a new node u' with the same label as u^j, and edge (u', w) with label x if and only if edge (u^j, w) has label x for every outgoing edge (u^j, w)) and redirects $e^2, \ldots, e^t$ to u' (e^1 is still directed to the original node u^j). Let $G'^f = (V'^f, E'^f)$ be the resulting graph. Define the mapping θ' as $\theta'(u') := \theta(u)$ and $\theta'(u) := \theta(u)$ for $u \neq u'$. Then θ' is a mapping from V'^f_j (i.e., the set of nodes at depth j in G'^f) onto $PF(j, T^h_{G,\sigma,X}(v))$ and meets C1 and C2. The sets of maximal-length paths from $u_0 \in V^f_0$ and $u'_0 \in V'^f_0$ are obviously isomorphic to each other. The inverse operation can be applied repeatedly until there are no nodes that have multiple incoming edges, which does not change the set of maximal-length paths up to isomorphism. It follows that G^f is transformed into a view that defines $P^h_{G,\sigma,X}(v)$, i.e., view $T^h_{G,\sigma,X}(v)$, by repeatedly applying the operation until it can no longer be applied. □

From this lemma, we obtain the next corollary.

Corollary 5 *Any minimal f-view $\widetilde{T}^h_{G,\sigma,X}(v)$ is unique up to isomorphism and has exactly $|PF(j, T^h_{G,\sigma,X}(v))|$ nodes at depth j $(0 \le j \le h)$. The minimal f-view of depth h for any n-party distributed network has $O(hn)$ nodes and $O(h\Delta n)$ edges, where Δ is the maximum degree over all nodes of the underlying graph G.*

Proof Lemma 4 implies that when the mapping θ is a *bijective* mapping from V^f_j to $PF(j, T^h_{G,\sigma,X}(v))$ for all j, the f-view is minimal. Thus, the f-view has $|PF(j, T^h_{G,\sigma,X}(v))|$ nodes at depth j $(0 \le j \le h)$.

Assume that $\widetilde{T}^h_{X,a}(v)$ and $\widetilde{T}^h_{X,b}(v)$ are any two minimal f-views of $T^h_{G,\sigma,X}(v)$, and let θ_a and θ_b be their corresponding bijective mappings θ, respectively. If we define $\zeta = \theta_b^{-1}\theta_a$ for the inverse mapping θ_b^{-1} of θ_b, then ζ is a bijective mapping from the node set of $\widetilde{T}^h_{X,a}(v)$ to that of $\widetilde{T}^h_{X,b}(v)$. Suppose that any node u_a at depth j of $\widetilde{T}^h_{X,a}(v)$ is mapped by θ_a to some path set P in $PF(j, T^h_{G,\sigma,X}(v))$, which is mapped to some node u_b at depth j by θ_b^{-1}. Obviously, u_a and u_b have the same degree and have the same label as that of the first node of paths in P. Let (u_a, u'_a) be an edge with a label x in $\widetilde{T}^h_{X,a}(v)$. Node u'_a is then mapped to $\theta_a(u)|_x$, which is mapped to node u'_b incident to the directed edge with label x emanating from u_b. For each u_a and x, thus, there is an edge $(\zeta(u_a), \zeta(u'_a))$ with label x in $\widetilde{T}^h_{X,b}(v)$ if the edge (u_a, u'_a) with label x exists in $\widetilde{T}^h_{X,a}(v)$. By a similar argument, there is an edge $(\zeta^{-1}(u_b), \zeta^{-1}(u'_b))$ with label x in $\widetilde{T}^h_{X,a}(v)$ if edge (u_b, u'_b) with label x exists in $\widetilde{T}^h_{X,b}(v)$ for each u_b and x. Thus, ζ is an isomorphism from $\widetilde{T}^h_{X,a}(v)$ to $\widetilde{T}^h_{X,b}(v)$.

As for the number of nodes in the minimal f-view, if there are n parties, it is obvious that $|PF(j, T^h_{G,\sigma,X}(v))| \leq n$ for any j $(0 \leq j \leq h)$. Since each node has at most Δ outgoing edges, the corollary follows. □

B Folded-View Minimization

This section gives a precise description of the folded-view minimization algorithm in Section 4 and the proof of its related lemma.

First, we define the following notion: Each node u is associated with the pair, called *key*,

$$(\mathsf{label}(u), \mathsf{ekey}(u)),$$

where $\mathsf{ekey}(u)$ is the lexicographically ordered list of all pairs $(\mathsf{label}(e_i(u)), \mathsf{Adj}_i(u))$ for $i = 1, \ldots, d_u$, where d_u is the number of the outgoing edges of u. We can see that two nodes can be merged if and only if they are associated with the same key. The reason is as follows. Suppose no more merging operation can be applied to the nodes at depth at least $j + 1$ (recall the algorithm proceeds in a bottom-up manner). If it holds that $(\mathsf{label}(u), \mathsf{ekey}(u))$ is equal to $(\mathsf{label}(v), \mathsf{ekey}(v))$ as bit strings for two distinct nodes u and v at depth j, then this implies that the conditions (1), (2) and (3) of Definition 1 (and 2) are satisfied. On the contrary, if u and v satisfy the conditions (1), (2) and (3) of the definition, the node $\mathsf{Adj}_i(u)$ must be identical to the node $\mathsf{Adj}_i(v)$ (since the path sets at no two distinct nodes at depth $j + 1$ are isomorphic). This implies that $(\mathsf{label}(u), \mathsf{ekey}(u))$ must be equal to $(\mathsf{label}(v), \mathsf{ekey}(v))$.

We first prepare $\mathsf{ekey}(u)$ for every node u by simply traversing $\widehat{T}^h$ in the breadth-first manner. We then perform the merging operations in the bottom-up manner. Let V^f_j be the set of all nodes at depth j (which can again be prepared by traversing $\widehat{T}^h$). For each depth j from h to 1, the following operations are performed (notice that the algorithm terminates at $j = 1$, since only the root lies at depth 0, which is never removed). First, we sort all nodes u's in V^f_j by regarding their keys, i.e., $(\mathsf{label}(u), \mathsf{ekey}(u))$, as a binary string. Notice that all nodes having the same key are adjacent to each other in the sorted order. Then, consider a maximal subsequence in the ordered sequence of elements in V^f_j such that the subsequence consists of nodes with the same key. For each of such subsequences, we eliminate all nodes but the first node in the subsequence and redirect all incoming edges of the eliminated nodes to the first node, which realizes the maximal applications of the merging operation to the nodes at depth j. For estimating the time complexity, we will describe a more precise description. We introduce variables **primary** and **primarykey** to store the first node u and its key $(\mathsf{label}(u), \mathsf{ekey}(u))$, respectively, of the subsequence currently being processed. The merging operations on V^f_j are realized by performing the following operations on every node u in V^f_j in the sorted order:

If $(\mathsf{label}(u), \mathsf{ekey}(u))$ is equal to **primarykey**,

- remove u from V^f_j,
- redirect all incoming edges of u to **primary**, and
- remove u and all its outgoing edges from $\widehat{T}^h$;

Otherwise, set **primary** to u and **primarykey** to $(\mathsf{label}(u), \mathsf{ekey}(u))$.

Lemma 6 *Let U be a non-empty finite set. Let T be an f-view for a distributed network $G = (V, E)$ with n nodes labeled by $X \colon V \to U$, and let V^f be the set of nodes of T. Then, the minimization algorithm for input T runs with time complexity $O(|V^f|(\log|V^f|)(\log|U| + \Delta\log(n|V^f|)))$, where Δ is the maximum degree of the nodes in G.*

Proof (Sketch). The ordered list $\mathsf{ekey}(u)$ of every node u and the set V^f_j for every j can be prepared by traversing the input f-view in a breadth-first manner. Since it takes $O(\log(n|V^f|))$ time per outgoing edge of each node, the time complexity is $O(\Delta|V^f|\log(n|V^f|))$, which is negligible compared to those of other operations as discussed below.

It takes $O(|V^f|\log|V^f|)$ comparisons in total to sorts all elements in V^f_j for all j. Each comparison takes $O(\log|U| + \Delta\log(\Delta|V^f|))$ time, since $\mathsf{label}(u) \in U$ and $\mathsf{ekey}(u)$ have $O(\log|U|)$ and $O(\Delta\log(n|V^f|))$ bits, respectively, for any u. Thus the time complexity of the sorting takes $O(|V^f|(\log|V^f|)(\log|U| + \Delta\log(n|V^f|))$ time.

Since no edge can be redirected and removed more than once, the time complexity taken to redirect and remove edges is at most the order of the number of edges in the input f-view, i.e., $O(\Delta|V^f|)$, with appropriate data structure (the details are given in Section G).

By summing up these elements, the total time complexity is given as $O(|V^f|(\log|V^f|)(\log|U| + \Delta\log(n|V^f|))$. □

C Constructing Minimal Folded-View

This section describes the proof of the theorem in Section 5.

Theorem 7 *Let U be a non-empty finite set. For a distributed network $G = (V, E)$ with n parties labeled by $X \colon V \to U$, there is an algorithm that constructs the minimal f-view of depth $h \in O(n)$ in $h+2$ rounds and $O(\Delta h^2 n(\log n)(\log|U|n^{\Delta}))$ time for each party with $O(mh^2 n\log(|U|\Delta^{\Delta}))$-bit communication over all parties, where m and Δ are the number of edges and the maximum degree, respectively, in G.*

Proof We will analyze the complexity of the algorithm in Fig. 5. Since the time and bit complexities of the preprocess are not dominant, we will consider the main part of the algorithm. Since there are n parties on the network there are at most n non-isomorphic path sets among those at all nodes at every depth j. Hence, Corollary 5 implies that the minimal f-view $\widetilde{T}^{j-1}_{G,\sigma,X}(v)$ has at most $j \cdot n$ nodes. By definition, every node has at most Δ outgoing edges, each of which is labeled with an $O(\log \Delta)$-bit value. Thus, $\widetilde{T}^{j-1}_{G,\sigma,X}(v)$ can be expressed by $O(jn \log |U| + j\Delta n \log \Delta) = O(jn \log(|U|\Delta^{\Delta}))$ bits. It follows that steps 2.1 and 2.2 take $O(j\Delta n \log(|U|\Delta^{\Delta}))$ time, since any party has at most Δ neighbors (strictly speaking, every party needs to encode the data structure representing an f-view as a bit-sequence before sending the f-view and decode it after receiving the f-view, but the time complexity of such encoding/decoding is negligible compared to other operations as discussed in Section G). Since $\widetilde{T}^{j}_{G,\sigma,X}(v)$ consists of a root and Δ minimal f-views of depth $j-1$, f-view $\widetilde{T}^{j}_{G,\sigma,X}(v)$ has at most $(j \cdot \Delta \cdot n + 1)$ nodes. From Lemma 6, step 2.4 in Fig. 5 takes

$$\begin{aligned} &O(j\Delta n \log(j\Delta n)(\log |U| + \Delta \log(n \cdot j\Delta n))) \\ &\qquad = O(j\Delta n(\log n)(\log |U| + \Delta \log n)) \end{aligned}$$

time for each j, since $j \in O(n)$. Thus the total time complexity is

$$\begin{aligned} &O\left(\sum_{j=1}^{h} j\Delta n(\log n)(\log |U| + \Delta \log n)\right) \\ &\qquad = O(\Delta h^2 n(\log n)(\log |U| n^{\Delta})). \end{aligned}$$

We now consider the bit-complexity. Since the minimal f-view of depth j can be expressed by $O(jn \log(|U|\Delta^{\Delta}))$ bits as described above, the total number of the bits exchanged by all parties is $O(jmn \log(|U|\Delta^{\Delta}))$ for each j. It follows that the total bit complexity to construct an f-view of depth h is

$$O\left(\sum_{j=1}^{h} (jmn \log(|U|\Delta^{\Delta}))\right) = O\left(mh^2 n \log(|U|\Delta^{\Delta})\right).$$

Since the number of message exchanges is $h+1$, the algorithm takes $h+2$ rounds. □

D Examples of Minimal F-Views

This section illustrates examples that exhibit the properties stated in the first paragraph in Section 6.

A minimal f-view is not the quotient graph of the underlying graph. For instance, Fig. 6 illustrates the minimal f-view of depth $2(n-1) = 6$ for the graph G in Fig. 1, while Fig. 7 shows the quotient graph of G.

Two sub-f-views are not necessarily isomorphic even if their corresponding views are isomorphic. For instance, Fig. 9 shows the minimal f-view with respect to the leftmost node of the graph G' given in Fig. 8. The two sub-f-views of depth 2 at the nodes indicated by bold circles are not isomorphic, while both nodes represent the leftmost node in the graph G'.

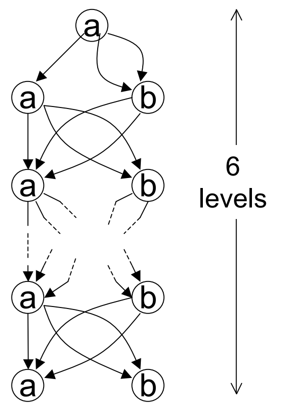

Figure 6: The minimal folded view $\widetilde{T}^{6}_{G,\sigma,X}(v)$ of depth $2(n-1) = 6$ for the graph G given in Fig. 1, where v is the left-upper node in G. Port numbers are omitted.

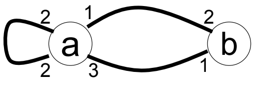

Figure 7: The quotient graph for the graph G given in Fig. 1.

E Counting the Number of Parties

This section presents a concrete description of the view counting algorithm and the proofs of related theorem and lemmas.

As stated in Section 6.1. the view counting algorithm computes a maximal set W of the nodes u at the depth of at most $n-1$ in $\widetilde{T}^{2(n-1)}_{X}(v)$ such that nodes $u \in W$ define non-isomorphic path sets of length $n-1$.

1. Set W to $\{u_{\mathrm{root}}\}$, where u_{root} is the root of $\widetilde{T}^{2(n-1)}_{X}(v)$.
2. Perform the following operations for each node u of $\widetilde{T}^{2(n-1)}_{X}(v)$ at the depth of at most $n-1$ in a breadth-first order:
   - 2.1 For each node $\hat{u}$ in W, run Subroutine Equivalence_Check (described later) to test if the sub-f-view at u has a path set of length $n-1$ that is isomorphic to that defined for the sub-f-view at $\hat{u}$.
   - 2.2 Set $W := W \cup \{u\}$ if the test is false (i.e., the sub-f-views do not have isomorphic path sets of length $n-1$).

Lemma 8 *Suppose that $\widetilde{T}^{(n-1)}_{X,a}$ and $\widetilde{T}^{(n-1)}_{X,b}$ are any two sub-f-views of depth $(n-1)$ of a minimal f-view $\widetilde{T}^{2(n-1)}_{X}(v)$, such that, for roots u_r and w_r of $\widetilde{T}^{(n-1)}_{X,a}$ and $\widetilde{T}^{(n-1)}_{X,b}$, respectively, $\mathsf{depth}(u_r) \leq \mathsf{depth}(w_r) \leq (n-1)$. Let V_a and V_b be the node sets of $\widetilde{T}^{(n-1)}_{X,a}$ and $\widetilde{T}^{(n-1)}_{X,b}$, respectively, and let E_a and E_b be the edge sets of $\widetilde{T}^{(n-1)}_{X,a}$ and $\widetilde{T}^{(n-1)}_{X,b}$, respectively.*

Then, $\widetilde{T}^{(n-1)}_{X,a}$ and $\widetilde{T}^{(n-1)}_{X,b}$ have isomorphic path sets of length $(n-1)$ if and only if the following conditions C1 and C2 hold.

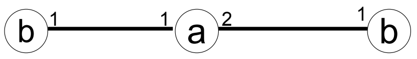

Figure 8: A graph $G' = (V', E')$ with a port numbering σ' and a node labeling $Y\colon V' \to \{a, b\}$.

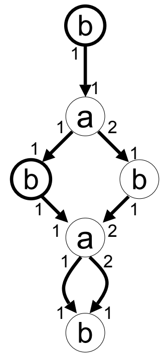

Figure 9: The minimal folded view of depth $2(n-1) = 4$ for the leftmost node in the graph G' given in Fig. 8. The two sub-f-views at the nodes indicated by bold circles are not isomorphic, while both nodes represent the leftmost node in the graph G'.

C1: *There is a unique surjective homomorphism*

$$\phi\colon V_a \to V_b$$

that preserves node labels: $\mathsf{label}(u) = \mathsf{label}(\phi(u))$ *for each* $u \in V_a$.

C2: *Let $E_a(u)$ and $E_b(v)$ represent the set of outgoing edges of $u \in V_a$ and $v \in V_b$, respectively. Then, there is a family ψ of bijective mappings*

$$\psi_u\colon E_a(u) \to E_b(\phi(u))$$

for every $u \in V_a$, such that ψ_u preserves edge-labels, i.e., $\mathsf{label}(e) = \mathsf{label}(\psi_u(e))$ *for every $e \in E_a(u)$, and ψ_u maps any edge from u to u' to an edge from $\phi(u)$ to $\phi(u')$ for all possible $u' \in V_a$.*

Proof ($\Rightarrow$) For each $j \in \{1, \ldots, 2(n-1) - \mathsf{depth}(u_r)\}$, let P_a^1 and P_a^2 be two distinct directed paths from u_r to a certain node at depth $j + \mathsf{depth}(u_r)$ (if they exists). Since $\widehat{T}_{X,a}^{(n-1)}$ and $\widehat{T}_{X,b}^{(n-1)}$ have isomorphic path sets, there are two directed paths P_b^1 and P_b^2 from w_r, isomorphic to P_a^1 and P_a^2. We claim that P_b^1 and P_b^2 cannot terminate at distinct nodes. This claim together with a simple induction implies that **C1** and **C2** hold. The proof of the claim is as follows. Let u be the node at which both P_a^1 and P_a^2 terminate, and let w_1 and w_2 be the nodes at which P_b^1 and P_b^2 terminate respectively. Consider the path sets $P_X^{2(n-1)-\mathsf{depth}(u)}(u)$ starting at u, $P_X^{2(n-1)-\mathsf{depth}(w_1)}(w_1)$ starting at w_1, and $P_X^{2(n-1)-\mathsf{depth}(w_2)}(w_2)$ starting at w_2. Notice that these path sets have the maximum length in $\widetilde{T}_X^{2(n-1)}(v)$. Since $\widehat{T}_{X,a}^{(n-1)}$ and $\widehat{T}_{X,b}^{(n-1)}$ have isomorphic path sets (and so do their extensions to infinite length), the path sets $P_X^{2(n-1)-\mathsf{depth}(w_1)}(w_1)$ and $P_X^{2(n-1)-\mathsf{depth}(w_2)}(w_2)$ can both be obtained by truncating every path in $P_X^{2(n-1)-\mathsf{depth}(u)}(u)$ at length $2(n-1) - \mathsf{depth}(w_1)$ $(= 2(n-1) - \mathsf{depth}(w_2))$. Therefore, nodes w_1 and w_2 must be merged in $\widetilde{T}_X^{2(n-1)}(v)$.

($\Leftarrow$) Suppose that ϕ and ψ satisfy conditions C1 and C2. Every directed edge (u, u') in $\widehat{T}_{X,a}^{(n-1)}$ is mapped by ψ_u to a directed edge $(\phi(u), \phi(u'))$ in $\widehat{T}_{X,b}^{(n-1)}$ with the same edge labels and the same end-node labels. It follows that any directed path in $\widehat{T}_{X,a}^{(n-1)}$ is mapped to an isomorphic directed path in $\widehat{T}_{X,b}^{(n-1)}$. Thus, any $(n-1)$-length directed path from u_r in $\widehat{T}_{X,a}^{(n-1)}$ has to be mapped to an isomorphic $(n-1)$-length directed path in $\widehat{T}_{X,b}^{(n-1)}$. Therefore, the path set of $\widehat{T}_{X,a}^{(n-1)}$ is a subset of that of $\widehat{T}_{X,b}^{(n-1)}$. Conversely, fix an $(n-1)$-length directed path $p = (e(w_1), \ldots, e(w_{n-1}))$ away from $w_r (= w_1)$, where $e(w_j) \in E_b$ is an outgoing edge of $w_j \in V_b$ for each $j = 1, \ldots, (n-1)$. Obviously, u_r is the unique preimage of w_r by ϕ. If u_j is a preimage of w_j (with respect to ϕ), condition C2 implies that there is exactly one preimage u_{j+1} of the destination node of $e(w_j)$ such that there is an edge (u_j, u_{j+1}) labeled with $\mathsf{label}(e(w_j))$. By induction, there is the unique path in $\widehat{T}_{X,a}^{(n-1)}$ which is isomorphic to p. Thus, the path set of $\widehat{T}_{X,b}^{(n-1)}$ is a subset of that of $\widehat{T}_{X,a}^{(n-1)}$. □

We will describe the first implementation of Subroutine Equivalence_Check. The subroutine first sets $\phi(u_r) := w_r$ if u_r and w_r have the same label, and then constructs ϕ defined in Lemma 8 by repeating the following operations for each j from 0 to $(n-1) - 1$: For every node $u \in V_a$ at depth $(j + \mathsf{depth}(u_r))$ and for every $i = 1, \ldots, d_u$, where d_u is the number of outgoing edges of u, set

$$\phi(\mathsf{Adj}_i(u)) := \mathsf{Adj}_i(\phi(u)) \in V_b,$$

if the following conditions hold:

1. The nodes u and $\phi(u)$ have the same label: $\mathsf{label}(u) = \mathsf{label}(\phi(u))$,
2. The nodes u and $\phi(u)$ have the same number of outgoing edges,
3. The i^{th} edge of u and the i^{th} edge of $\phi(u)$ have the same label: $\mathsf{label}(e_i(u))) = \mathsf{label}(e_i(\phi(u)))$,
4. When $\phi(\mathsf{Adj}_i(u))$ has already been defined, it holds that $\phi(\mathsf{Adj}_i(u))$ is identical to $\mathsf{Adj}_i(\phi(u))$.

We should note that conditions 2 and 3 correspond to the existence of ψ_u. Fig. 10 gives a more precise description of Subroutine Equivalence_Check. To construct ψ_u, one has only to add "set $\psi_u((u, \mathsf{Adj}_i(u))) := (\phi(u), \mathsf{Adj}_i(\phi(u)))$" in the case where the if-part fails in step 2.2.1. However, this is not essential for the subroutine and we omit it.

Lemma 9 *Given two sub-f-views $\widehat{T}_{X,a}^{(n-1)}$ and $\widehat{T}_{X,b}^{(n-1)}$ of depth $(n-1)$ of a minimal f-view $\widetilde{T}_X^{2(n-1)}(v)$, there is an algorithm that outputs "*Yes*" if and only if $\widehat{T}_{X,a}^{(n-1)}$ and $\widehat{T}_{X,b}^{(n-1)}$ have isomorphic path sets of length $(n-1)$. The time complexity is*

Subroutine Equivalence_Check

```
Input: Two sub-f-views, $\widetilde{T}^{(n-1)}_{X,a}$ at $u_r$ and $\widetilde{T}^{(n-1)}_{X,b}$ at $w_r$, of
    a minimal f-view $\widetilde{T}^{2(n-1)}_X(v)$ such that $\mathsf{depth}(u_r) \le \mathsf{depth}(w_r) \le n-1$.

Output: "Yes" or "No".

1. If $\mathsf{label}(u_r) = \mathsf{label}(w_r)$, then set $\phi(u_r) := w_r$; otherwise go to step 4.
2. Perform the following operations for each node $u$ in $\widetilde{T}^{(n-1)}_{X,a}$ at depth $j + \mathsf{depth}(u_r)$ for $j = 0, \dots, n-2$ in this order.
    2.1 If $u$ and $\phi(u)$ have the different number of outgoing edges, then go to step 4.
    2.2 Perform the following steps for $i := 1$ to $d_u$, where $d_u$ is the number of outgoing edges of $u$.
        2.2.1 If $\mathsf{label}(e_i(u)) \neq \mathsf{label}(e_i(\phi(u)))$, then go to step 4.
        2.2.2 If $\mathsf{label}(\mathsf{Adj}_i(u)) \neq \mathsf{label}(\mathsf{Adj}_i(\phi(u)))$, then go to step 4.
        2.2.3 If $\phi(\mathsf{Adj}_i(u))$ has already been defined and $\phi(\mathsf{Adj}_i(u)) \neq \mathsf{Adj}_i(\phi(u))$, then go to step 4.
        2.2.4 Set $\phi(\mathsf{Adj}_i(u)) := \mathsf{Adj}_i(\phi(u))$.
3. Output "Yes".
4. Output "No".
```

Figure 10: Subroutine Equivalence_Check.

$O(n^2\Delta\log(n|U|))$, where Δ is the maximum degree over all nodes of the underlying graph of the distributed network.

Proof Subroutine Equivalence_Check attempts to construct ϕ defined in Lemma 8. It outputs "Yes" if and only if it visited all nodes in $\widetilde{T}^{(n-1)}_{X,a}$ and successfully defined $\phi(u)$ for all nodes $u \in V_a$. It is easy to see that, if the subroutine outputs "Yes", then ϕ meets C1 of Lemma 8 (due to steps 1, 2.2.2, and 2.2.3) and C2 (due to steps 2.1 and 2.2.1). If the subroutine outputs "No", then one of the steps 1, 2.1 and 2.2.1-2.2.3 fails, which implies that C1 or C2 fails. Thus, $\widetilde{T}^{(n-1)}_{X,a}$ and $\widetilde{T}^{(n-1)}_{X,b}$ have isomorphic path sets of length $(n-1)$ if and only if the subroutine outputs "Yes". This proves the correctness.

Step 2.1 in Fig. 10 is performed once with $O(\Delta)$ time at each node. Hence, it takes $O(|V_a|\Delta)$ time in total over all nodes in V_a. Step 2.2 takes $O(d_u(\log n + \log|U|))$ time for each u, since node and edge labels are represented by $O(\log|U|)$ bits and $O(\log n)$ bits, respectively. Hence, it takes $O(|E_a|\log(n|U|))$ time in total. The time complexity of step 2 is thus $O(|V_a|\Delta + |E_a|\log(n|U|))$, which is $O(n^2\Delta\log(n|U|))$ since $|V_a| = O(n^2)$ and $|E_a| = O(n^2\Delta)$. Obviously, step 2 is dominant over all steps. □

Now we gives the time complexity of the view counting algorithm.

Theorem 10 *Let U be a non-empty finite set. For any distributed network $G = (V, E)$ of n parties labeled by mapping $X\colon V \to U$, there is an algorithm that computes $|\Gamma^{(n-1)}_X(S)|$ for any subset S of U in $O(n^5\Delta\log(n|U|))$ time, where Δ is the maximum degree over all nodes in G, if a minimal f-view $\widetilde{T}^{2(n-1)}_X(v)$ is given to every party.*

Proof Consider the view counting algorithm shown in the first paragraph in this section. The correctness of the algorithm follows from the first paragraph in Section 6.1. As for complexity, we can see that (1) $|W|$ is at most n since there are n parties on the network, and that (2) there are $O(n^2)$ nodes whose depth is at most $n-1$ in $\widetilde{T}^{2(n-1)}_X(v)$ since there are at most n nodes at each depth. Hence, Equivalence_Check is performed for each of $O(n^3)$ pairs of sub-f-views. Since each run of Equivalence_Check takes $O(n^2\Delta\log(n|U|))$ time by Lemma 9, the total time complexity is $O(n^5\Delta\log(n|U|))$ time. □

Remark 1 *The above statement is somewhat weak, while this results in a simple description of the algorithm. The time complexity in Lemma 9 can be improved to $O(n^2\log(n^\Delta|U|))$ by performing step 2.2.2 in Fig. 10 only when $\phi(\mathsf{Adj}_i(u))$ has not been defined yet. Accordingly, the time complexity in Theorem 10 is improved to $O(n^5\log(n^\Delta|U|))$. This is the same order of the complexity of the second and third implementation as described later. However, this improvement is meaningful only if node labels are picked from a very large set compared with n, i.e., $|U| \in n^{\omega(\Delta)}$.*

For the second implementation of Equivalence_Check, we need to perform the following preprocess before starting the view counting algorithm: For each node v at depth at most $n-1$ in the given minimal f-view, pick a copy of the sub-f-views of depth $n-1$ at v, and then apply the minimization algorithm to the copy. Then, Equivalence_Check has only to test if the minimized copies of the sub-f-views at the given pair of nodes are isomorphic. Since there are at most $O(n^2)$ nodes at depth at most $n-1$, the time complexity of minimizing all the copies is $O(n^4(\log n)\log(|U|n^\Delta)$ by applying Theorem 6 with $|V^f| = O(n^2)$. For any pair of minimal f-views of depth $n-1$, it takes $O(n^2\Delta\log n + n^2\log|U|)$ to test whether they are isomorphic, since each edge label and each node label are represented by $O(\log n)$ bits and $O(\log|U|)$ bits, respectively. Since the test are performed at most n^3 times, the time complexity of testing isomorphism of n^3 pairs is $O(n^5\log(n^\Delta|U|))$, which is dominant in the time complexity of the entire view counting algorithm. The above implementation requires the space to store the minimal sub-f-views at all nodes at depth at most $n-1$. To save the space, we can minimize the sub-f-views on demand, i.e., just before testing their isomorphism. This increases the complexity by a log factor.

The third implementation is the simplest, but it requires as input a minimal f-view $\tilde{T}^{3(n-1)}_X$ of depth $3(n-1)$ (instead of $2(n-1)$), which needs $3(n-1)+2$ rounds to construct. For given nodes $\hat{u}$ and u at depth at most $n-1$, Equivalence_Check just tests whether the sub-f-view of depth $n-1$ at $\hat{u}$ and u are *isomorphic*. We claim that the sub-f-views are isomorphic if and only if the path sets of length $n-1$ for the sub-f-views are isomorphic. The only-if part is trivial. For the if-part, suppose that the path sets of length $n-1$ for the sub-f-views are isomorphic. Then, it is sufficient to show that for any path set P of length $n-1$, if there is a node u with $\mathsf{depth}(u) \le n-1$ such that the sub-f-view T of depth $n-1$ at u define P, then the sub-f-view T is unique up to isomorphism. The minimality of $\tilde{T}_X^{3(n-1)}$ implies that there are two distinct nodes u_1 and u_2 at the same depth in T, if and only if the path sets starting at u_1 and u_2 of maximal length, $l = 3(n-1) - \mathsf{depth}(u_1) = 3(n-1) - \mathsf{depth}(u_1)$, are not isomorphic. The latter part of the above sentence is equivalent to saying that the path sets starting at u_1 and u_2 of infinite length (which could be obtained if we constructed an infinite-depth view) are not isomorphic, since $l \ge n-1$ and, $T_X(v) \equiv T_X(v')$ if and only if $T_X^h(v) \equiv T_X^h(v')$ for every $h \ge n-1$ [13]. Therefore, if the path set of the sub-f-view T is isomorphic to P, then T is unique up to isomorphism. The time complexity is $O(n^5 \log(n^\Delta |U|))$, since we perform isomorphism test for each of n^3 pairs of sub-f-views and each test takes $O(n^2 \log(n^\Delta |U|))$.

F Directed network topologies

This section briefly discusses the reason why the folded view can work for directed networks. Suppose that there is an n-party distributed network whose underlying graph is strongly connected directed graph with a port-numbering σ and a node-labeling mapping X, where we assume that the ports for incoming edges and those for outgoing edges are independently numbered at each node. Let us define the view for a node v in the directed graph G as the labeled directed tree rooted at v obtained by recursively traversing incoming edges of every node (this view is called the *universal total graph* at v by Boldi and Vigna [6]). Let us denote the view again by $T_{G,\sigma,X}(v)$ and its finite-depth version by $T^h_{G,\sigma,X}(v)$.

Boldi and Vigna [6] showed that the view for a directed graph has a property similar to that proved in the undirected graph case. More concretely, if v and v' are any two nodes of the directed graph G, then it holds that $T_{G,\sigma,X}(v) \equiv T_{G,\sigma,X}(v')$ if and only if $T^{n-1}_{G,\sigma,X}(v) \equiv T^{n-1}_{G,\sigma,X}(v')$. Therefore, it is sufficient to look at views up to depth $n-1$ in order to count the number of non-isomorphic views of infinite depth (more generally, in order to gather all information that a party can obtain by exchanging messages). Then, a natural approach is that every party constructs an f-view of depth $2(n-1)$ and then counts the number of non-isomorphic views of depth $n-1$. This can work for the following reasons.

An f-view is obtained by just sharing isomorphic subgraphs of a view. In other words, the f-view construction algorithm in Fig. 5 does not care whether the view is derived from a directed network or an undirected network. Thus, the f-view construction algorithm can work for directed networks. It is obvious that the complexity for constructing a minimal f-view in the directed network case is the same order as in the undirected network case. The view-counting algorithm uses the fact that $T_{G,\sigma,X}(v) \equiv T_{G,\sigma,X}(v')$ if and only if $T^{n-1}_{G,\sigma,X}(v) \equiv T^{n-1}_{G,\sigma,X}(v')$. Since this also holds in the directed network case as stated in the above, the view counting algorithm can also work.

G Data Structure

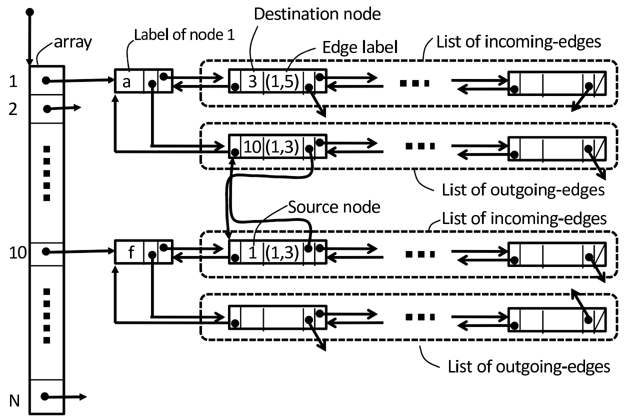

Figure 11: Data structure representing a folded view

Fig. 11 shows data structure that represents an f-view, where N is the number of nodes of the f-view. This is a slight modification of the standard representation of directed graphs. Each cell of the array stores the pointer to the record of a node. The record stores the label of the node and two pointers to the lists of incoming edges and outgoing edges, respectively, of the node. Each of the lists is a doubly-linked list of the records each of which stores the information of an edge such as the source or destination node (depending on whether it is an incoming edge or an outgoing edge) and the label of the edge. Thus, two records are stored for each edge, one in the list of outgoing edges of the source node, and the other in the list of incoming edges of the destination node. These two records are linked to each other, which makes it possible to perform the operation of minimizing f-view in low complexity. More concretely, consider the following operations that are performed to realize the merging operation. To redirect all incoming edges of a node u to **primary**, we move the incoming-edge list of u to that of **primary** in constant time and then, for every edge e in the incoming-edge list, we update the destination node field of the record in the outgoing-edges of the source node of e, which takes $O(d^I_u)$ steps for in-degree d^I_u of u. To remove a node u and all its outgoing edges, we remove the record for each of the edges from the incoming-edge list of its destination node, which takes $O(d_u)$ time in total for the out-degree d_u of u, and then put "`nil`" into the array cell for u. The total time complexity of the above two operations in the f-view minimization algorithm is

$$O\left(\sum_{u\in V^f} d_u^I + \sum_{u\in V^f} d_u\right) = O(\Delta|V^f|),$$

since no edge can be redirected or removed more than once.