

Computing on Anonymous Networks, Part I: Characterizing the Solvable Cases*

Masafumi Yamashita, *Member, IEEE*, and
Tsunehiko Kameda, *Affiliate Member, IEEE*

Abstract— In anonymous networks, the processors do not have identity numbers. We investigate the following representative problems on anonymous networks: (a) the leader election problem, (b) the edge election problem, (c) the spanning tree construction problem, and (d) the topology recognition problem. On a given network, the above problems may or may not be solvable, depending on the amount of information about the attributes of the network made available to the processors. Some possibilities are: (1) no network attribute information at all is available, (2) an upper bound on the number of processors in the network is available, (3) the exact number of processors in the network is available, and (4) the topology of the network is available.

In terms of a new graph property called “symmetricity,” in each of the four cases (1)–(4) above, we characterize the class of networks on which each of the four problems (a)–(d) is solvable. We then relate the symmetricity of a network to its 1- and 2-factors.

*To appear in *IEEE Trans. Parallel and Distributed Computing*, February, 1996. M. Yamashita is with the Department of Electrical Engineering, Hiroshima University, Higashi-Hiroshima, 724 Japan. T. Kameda is with the School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada V5A 1S6.

This work was supported in part by the Natural Sciences and Engineering Council of Canada and a Scientific Research Grant-in-Aid from the Ministry of Education, Science and Culture of Japan.

Index Terms— anonymous network, distributed computing, leader election, edge election, spanning tree construction, topology recognition, knowledge

1 Introduction

A *network* consists of a set of processors and a set of communication links connecting pairs of processors. In the past, dozens of papers have been written on the subject of efficient distributed algorithms for various problems about networks, including *leader election*, *spanning tree construction*, and *topology recognition* (i.e., the determination of network topology), under the assumption that each processor has a unique identity number (see, e.g., [10, 13, 21, 28]).

Suppose that the processors have unique identity numbers. Then for any network there is a distributed algorithm for electing a unique leader processor, which requires as input data no information about the network, such as the network topology or the number of processors in it. Also, if a unique initiator (leader) can be used, there are distributed algorithms for solving the problems listed above, which require no information about the network. Therefore, if the processors have unique identity numbers, those problems can be solved without input data containing information about network attributes.

We consider the above problems for the *anonymous* networks, in which the processors do not have identity numbers. One might argue that the leader election problem, for example, could be solved even for anonymous networks in general; a diffusing computation [11] or a probe/echo algorithm [9] could be used to construct a spanning tree, whose root we could choose as the leader. To explain informally the reason why such a solution does not work correctly, consider an anonymous 4-node ring network. Suppose that a node initiates a probe/echo algorithm A . Algorithm A would construct a spanning tree, if the other nodes never initiated A simultaneously. However, an arbitrary number of nodes can initiate A simultaneously,¹ and the correctness of A is no longer guaranteed, since, intuitively, when a node receives a message, it in general cannot tell who sent it, and therefore, in this particular example, different execution instances of A initiated by different nodes may get mixed up. In fact, Angluin [1] and Johnson and Schneider [17] have pointed out that there is a network (e.g., the 4-node ring network) for which the leader election problem is unsolvable, i.e., there exists no deterministic distributed leader election algorithm, even if it is allowed to construct an algorithm specific to the network (cf. the “universal” algorithms, which we will introduce later).² Hence, it is meaningful to investigate the problem of finding the class of networks for which the leader election problem, for example, is solvable. This paper discusses the following four problems. Suppose a network is represented by an undirected graph $G = (V, E)$, where V (E) is its node (edge) set. Each node represents a processor and each edge $(u, v) \in E$ represents a link between $u \in V$ and $v \in V$. In the rest of this paper, we use the terms, graph and network, interchangeably, although we tend to use the term graph to mean an abstract representation of a network.

Leader Election Problem (ELECT-LEADER): Elect a processor as the leader, in the sense that the elected processor knows that it has been elected and the other processors know that they have not.

Edge Election Problem (ELECT-EDGE): Select a link $e = (u, v)$ in the sense that

¹It is assumed that the same algorithm is installed on each node in an anonymous network (see Section 2).

²Randomized algorithms, which are outside the scope of this paper, can make use of the “coin-tossing” facility for generating random bits, and can solve the leader election problem (and hence the other problems listed above) with high probability, each node first generating a sufficiently large random identity number. See, e.g., [22, 24, 33].

processors u and v know which port corresponds to e and the other processors know that they are not incident with e .

Spanning Tree Construction Problem (SPANNING-TREE): Compute a spanning tree T of the network in the sense that each processor can tell which links incident to it are tree edges.

Topology Recognition Problem (FIND-TOPOLOGY): Compute on each processor a graph G isomorphic to the network it is running on.

We define the class \mathcal{P} as the class of the above four problems:

$$\mathcal{P} = \{\text{ELECT-LEADER, ELECT-EDGE, SPANNING-TREE, FIND-TOPOLOGY}\}.$$

It is clear that the class of networks for which a particular problem is solvable will, in general, become larger, as more information about the network is made available. Hence, finding the effect of available network attribute information on each solvable class, i.e., the class of networks for which a problem is solvable, is of interest. We investigate the following four conditions, depending on the amount of information available about the attributes of a given network.

No Information (*noinfo*): No network attribute information at all is available.

Upper Bound on Network Size (*upbound*): A constant upper bound on the number of processors in the network is available.

Network Size (*size*): The exact number of processors in the network is available.

Network Topology (*topology*): The topology of the network is available.

The last condition above, in which the network topology is available, doesn't appear to be practically very meaningful. Theoretically, however, the investigation of this case turns out to be of great importance. For any network, if there is an algorithm for solving a problem using the number of processors or an upper bound on it, then, clearly, there is an algorithm for solving the problem using the topology, since knowing the topology implies knowing the number of processors. Namely, the solvable classes of networks for the problem under the conditions *noinfo*, *upbound* and *size* are subsets of that under *topology*. We will prove that only for trees are all of the problems listed above solvable under *noinfo* and *upbound*, whereas any of the above problems is solvable for a class of non-tree networks under *size* and *topology*. For each of the above problems, the solvable classes under *noinfo* and *upbound* coincide. For each of ELECT-LEADER, ELECT-EDGE and SPANNING-TREE, the solvable classes under *size* and *topology* coincide. As will be shown, there is a network for which FIND-TOPOLOGY is unsolvable under *size*, while FIND-TOPOLOGY is trivially solvable for any network under *topology*. Moreover, for any problem $P \in \{\text{ELECT-LEADER, ELECT-EDGE, SPANNING-TREE}\}$ under *upbound*, *size* and *topology*, there is a "universal" algorithm, i.e., an algorithm that can solve P , if the network on which it is being executed belongs to the solvable class of P .

We will introduce a new property of a graph called *symmetry* (Section 3) and characterize all of the 16 solvable classes (i.e., all combinations of the four problems and four

conditions) in terms of this property. We will then characterize the class of networks having symmetricity k in terms of their 1- and 2-factors, where a p -factor is a spanning p -regular subgraph of a graph. As a result, we will be able to describe the solvable classes in graph theoretical terms.

As in [1], we make use of port numbering at each processor, and label each link by an ordered pair of port numbers, one at each end of the link. We call a particular way of port numbering by all the processors a “local edge labeling.” Then for each processor v , the set of all infinite walks (sequence of links) starting from v can be represented by a rooted tree with edge labels. We call this tree the “view” (of the network) from v (Section 3). The “similarity relation” [17] has been proposed to investigate concurrent systems. Informally, a set of processor labels is called a *similarity labeling*³ if any two processors having the same label behave similarly and produce the same output under certain communication timing.

Most results in this paper follow from the following facts, which we prove formally in subsequent sections.

- a) Each processor can compute its view, provided that an upper bound on the number of processors is known.
- b) The labeling which labels each processor by its view is a similarity labeling.
- c) The number of processors having the same view can be determined independently of the view – we will use this fact to define symmetricity.
- d) There is a strong relation between the equivalence classes induced by the similarity relation (two processors belong to the same equivalence class if and only if they have the same view) and the 1- and 2-factors of the network.
- e) Views contain (partial) information about the topology of the network.

In the next section, we formally define the network model on which our theory will be built. Section 3 introduces the concept of “view” that is used extensively in this paper and presents its basic properties. Section 4 is devoted to the discussions of the *topology* condition. It will form the basis of the investigation of the other conditions. In Section 5, we present a number of important results on symmetricity, and in Section 6, we treat the remaining three conditions, i.e., *noinfo*, *upbound* and *size*. Section 7 discusses related results. After Angluin [1], anonymous networks, especially anonymous rings, have been investigated extensively (see, e.g., [3, 4, 5, 12, 19, 20, 25, 35]). We will conclude the paper by briefly surveying related work in Section 8.

2 The Network Model

We model an (asynchronous) *anonymous network* by an undirected, connected, simple⁴ graph $G = (V, E)$, where the vertex set, $V = \{v_1, \dots, v_n\}$, represents the processors and the edge set E represents the bidirectional links among processors. An edge $e \in E$ is represented by (u, v) , if e connects $u \in V$ and $v \in V$. Let \mathcal{G} denote the set of all such graphs (networks).

³This should not be confused with port numbering.

⁴A graph is said to be simple, if it has neither self-loops nor parallel edges.

In what follows, we will use network and graph, processor and vertex, and link and edge interchangeably.

Each processor is assumed to have unlimited computational power; it has sufficiently large local memory and can access and change its memory content instantaneously.⁵ In executing a given sequential algorithm, in each step a processor, depending on the current memory content, either changes its memory content, sends a message via one of its ports, or receives a message via a port. The processors are *anonymous* in the sense that they do not have identity numbers, and the processors run the same deterministic algorithm.⁶ Although we label the processors in V by unique names v_1, \dots, v_n , these names are used only for description purposes, and the processors don't know their names. In other words, the algorithm that a processor executes does not use its identity number to make a decision or to compute a value. No assumption is made concerning relative execution speeds of processors besides fairness – unless a processor (i.e., an algorithm running on it) has terminated, it executes the next instruction in finite time.⁷

Communication is carried out by sending messages through links in E . A processor v is equipped with $deg(v)$ input/output ports, one for each link incident to it, named $1, \dots, deg(v)$, where $deg(v)$ denotes the degree of v . Let port j be processor u 's port for the link (u, v) . When processor u executes the instruction “send message M via port j ,” M is sent to the input queue of processor v for link e , in finite time, with no error, and in the FIFO order, i.e., messages sent through the link are placed in the input queue in the order they are sent. In order to receive a message placed in an input queue, the “receive” instruction is used. By the instruction “receive message M from port j ” executed by processor u , the first message in the input queue for link e is transferred to the variable M (stored in u 's local memory). If the input queue is empty, a special symbol is returned to M .

In our model, each processor v arbitrarily assigns names, $1, \dots, deg(v)$, to its local ports. In order to represent the correspondence between the port names and their associated links, we introduce the following definition. A *local edge labeling* (or *port numbering*) of G is a set of functions $\mathbf{f} = \{f_v \mid v \in V\}$ such that, for each $v \in V$, f_v is a bijection from the set of edges incident to v to the set of positive integers, $\{1, \dots, deg(v)\}$. Namely, $f_u(u, v) = i$ means that i is the name of the port of u corresponding to link (u, v) . Note that, in general, the same link may have different port numbers at its two ends, i.e., $f_u(u, v) \neq f_v(u, v)$, where $(u, v) \in E$. Finally, we assume that the network is reliable, i.e., the processors and the links never fail.

We assume that the local memory of each processor v initially contains algorithm A , $deg(v)$ and the information on G assumed to be known, and does not contain anything else. For example, under *topology*, each processor knows G , as well as A and $deg(v)$. We emphasize that this does not mean that a processor knows which vertex of G it is represented by, since processors having the same degree run the same algorithm with exactly the same initial information.

⁵This assumption is made, since we are interested in conditions for the four problems listed in Section 1 to become solvable and the message complexity of algorithms, but not in the local computation time.

⁶Our model assumes that each processor knows the number of ports belonging to it, and can make use of it. Therefore, processors with different number of ports can run different algorithms.

⁷Local clocks do not exist in our model. It is because local clocks cannot affect the processors' ability to solve the problems in the sense we will define later, provided that there is a possibility that they all indicate the same value at any time. Intuitively, the synchronized local clocks cannot be used to distinguish a processor from others.

From time to time, some processors (i.e., the initiators) spontaneously “wake up” and start the algorithm. Algorithm execution on the network will terminate when the algorithm terminates on every processor.

An algorithm A for problem P must work on any network G using the available attribute information about G , decide whether it can solve P for G , and solve P correctly if it can. If A determines that it cannot solve P for G , then it must report this fact. Under *upbound*, whether or not A can solve P for G may depend on the given upper bound \bar{n} on the size n of G , however. For example, if $\bar{n} - n \leq 1$ then knowing \bar{n} is just as good as knowing n . Furthermore, the behavior of A may also depend both on the timing of communications among the processors and on the naming (i.e., local edge labeling) of the ports. For a problem $P \in \mathcal{P}$ and an algorithm A for P , let $N(P, A)$ denote the set of networks for which A can solve P , no matter what the upper bound (under *upbound*), the communication timing and the port labeling are. Thus, by accident, A may solve P for G even if $G \notin N(P, A)$. Let $ALG_{topology}$ (resp. ALG_{size} , $ALG_{upbound}$, ALG_{noinfo}) be the set of all algorithms that use the topology of G (resp. the size n of G , an upper bound on n , no network attribute information). Define $D_{topology}(P)$, $D_{size}(P)$, $D_{upbound}(P)$ and $D_{noinfo}(P)$ as follows:

- $D_{topology}(P) = \{G \in \mathcal{G} \mid G \in N(P, A) \text{ for some } A \in ALG_{topology}\}$,
- $D_{size}(P) = \{G \in \mathcal{G} \mid G \in N(P, A) \text{ for some } A \in ALG_{size}\}$,
- $D_{upbound}(P) = \{G \in \mathcal{G} \mid G \in N(P, A) \text{ for some } A \in ALG_{upbound}\}$, and
- $D_{noinfo}(P) = \{G \in \mathcal{G} \mid G \in N(P, A) \text{ for some } A \in ALG_{noinfo}\}$.

Then by definition we have the following.

Proposition 1 For any problem $P \in \mathcal{P}$,

$$D_{noinfo}(P) \subseteq D_{upbound}(P) \subseteq D_{size}(P) \subseteq D_{topology}(P).$$

□

Proposition 2 $D_{topology}(\text{FIND-TOPOLOGY}) = G.$

□

In all sections an algorithm will mean a distributed algorithm in the sense defined above.

3 View and Its Properties

3.1 Definition and Basic Properties

Let $G = (V, E) \in \mathcal{G}$ and fix a local edge labeling $\mathbf{f} = \{f_v \mid v \in V\}$. Let v_1, \dots, v_d be the vertices adjacent to v , where $d = \text{deg}(v)$ is the degree of v .

Definition 1 The view of G from v under \mathbf{f} , $T_{\mathbf{f}}(v)$, is an infinite, labeled, rooted tree, defined recursively as follows. $T_{\mathbf{f}}(v)$ has the root x_0 corresponding to v . For each vertex v_i adjacent to v in G , $T_{\mathbf{f}}(v)$ has a node x_i and an edge from x_0 to x_i with labels $f_v(v, v_i)$ and $f_{v_i}(v, v_i)$ at its x_0 's and x_i 's ends, respectively. Node x_i is now the root of $T_{\mathbf{f}}(v_i)$ from v_i .

□

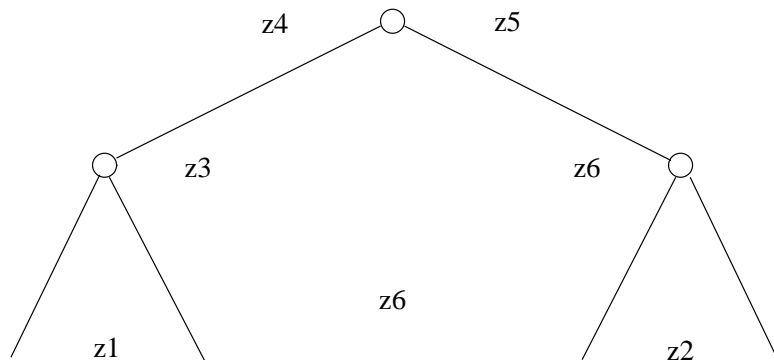


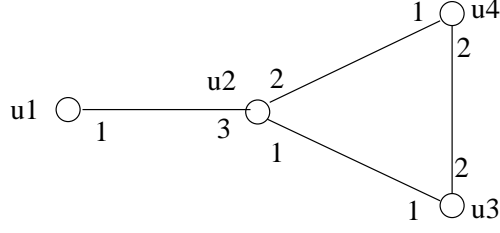
Figure 1: Definition of view $T_{\mathbf{f}}(v)$ from v under \mathbf{f} .

In the above definition, x_0 and x_i are used for definition purposes only and not part of the view. (See Figure 1.)

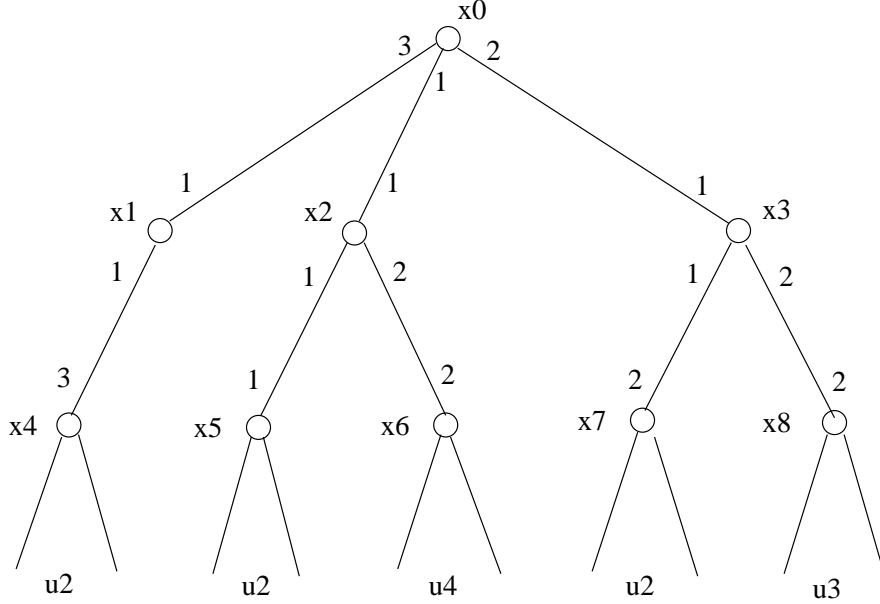
Note that the nodes of $T_{\mathbf{f}}(v)$ come from V , but they are not labeled as such. For example, x_i in the above definition is a local name known only to processor v and it is not labeled by v_i . But an algorithm (executed at a processor) can sometimes reveal the true identity, i.e., v_i , of a node of $T_{\mathbf{f}}(v)$, as will be discussed below. The view $T_{\mathbf{f}}(v)$ represents the set of all infinite walks in G starting at v , with port numbers appearing along each walk. To avoid confusion, in the rest of the paper, the vertices of a view will be referred to as *nodes*, to distinguish them from the vertices of networks under consideration. u, v and w (with or without subscripts) will denote vertices, and x, y and z (with or without subscripts) will denote nodes, unless otherwise stated. If x denotes a node of a view, then $\bar{x} \in V$ denotes the corresponding vertex of G .

Example 1 Consider the network G given in Figure 2(a). The integers attached to the edges in the figure present a local edge labeling \mathbf{f} for G . Figure 2(b) illustrates the view, $T_{\mathbf{f}}(u_2)$, from u_2 , which is constructed as follows. Starting with the root node x_0 , we draw an edge (x_0, x_1) with labels 3 and 1 corresponding to the walk from u_2 to u_1 in G . We thus have $\bar{x}_0 = u_2$ and $\bar{x}_1 = u_1$. Similarly, we draw (x_0, x_2) with labels 1 and 1 corresponding to the walk from u_2 to u_3 . The edge (x_1, x_4) with labels 1 and 3 corresponds to the walk from u_1 back to u_2 in G , and so forth. Note that labels 1 and 3 are attached in this order, since this time the edge (u_2, u_1) is traversed from u_1 to u_2 . Observe that $T_{\mathbf{f}}(u_2)$ is an infinite tree, and the three subtrees of its root are, from left to right, $T_{\mathbf{f}}(u_1)$, $T_{\mathbf{f}}(u_3)$ and $T_{\mathbf{f}}(u_4)$. $T_{\mathbf{f}}(u_2)$ and $T_{\mathbf{f}}(u_4)$ also appear as subtrees of $T_{\mathbf{f}}(u_3)$. \square

Clearly, the edge label sequences along different walks starting from the root of $T_{\mathbf{f}}(v)$ are distinct. Therefore, there is a one-to-one correspondence between the nodes of $T_{\mathbf{f}}(v)$ and the label sequences, from the root to those nodes. Let T be a view, and let $P(T, x)$ denote the shortest path connecting the root and node x in T , and let $L(T, x)$ denote the sequence of edge labels (two per edge) along $P(T, x)$. Recall that v_i 's are not shown in $T_{\mathbf{f}}(v)$ and that the correspondence between $P(T, x)$ and $L(T, x)$ is one-to-one.



(a)



(b)

Figure 2: An example of a view. (a) A graph G ; (b) $T_{\mathbf{f}}(u_2)$.

Two views T and T' are said to be *similar*, written $T \equiv T'$, if there is a graph isomorphism between them which preserves the root and edge labels. It is easy to see that \equiv is an equivalence relation. Note that there may be many vertices v in V having similar views. Let $\mathcal{T}_{\mathbf{f}}$ denote the set of all dissimilar views, i.e.,

$$\mathcal{T}_{\mathbf{f}} = \{T_{\mathbf{f}}(v) \mid v \in V\}$$

with one representative from each equivalence based on similarity. We may omit \mathbf{f} from $T_{\mathbf{f}}(v)$ and $\mathcal{T}_{\mathbf{f}}$, whenever it is obvious from the context.

Lemma 1 *Let \mathbf{f} be a local edge labeling of a network G , and let u_i and u_j be any two distinct vertices such that $T_{\mathbf{f}}(u_i) \equiv T_{\mathbf{f}}(u_j)$. Consider any two paths $P(T(u_i), z_i)$ and $P(T(u_j), z_j)$ in $T(u_i)$ and $T(u_j)$, respectively, such that the two label sequences $L(T(u_i), z_i)$ and $L(T(u_j), z_j)$ are identical. Then $T(\bar{z}_i) \equiv T(\bar{z}_j)$ and $\bar{z}_i \neq \bar{z}_j$ holds.*

Proof Consider two paths, π_i and π_j , in G corresponding to $P(T(u_i), z_i)$ and $P(T(u_j), z_j)$. They start at vertices u_i and u_j and end at vertices \bar{z}_i and \bar{z}_j , respectively. We assume

$T(\bar{z}_i) \not\equiv T(\bar{z}_j)$ and derive a contradiction. If $T(\bar{z}_i) \equiv T(\bar{z}_j)$, then without loss of generality there is a path π in G starting at \bar{z}_i such that there is no path starting at \bar{z}_j having the same label sequence as π . This means that $T(u_i)$ has a walk corresponding to $\pi_i\pi$, but $T(u_j)$ doesn't have a walk corresponding to $\pi_i\pi$, a contradiction.

Now, to prove the second assertion of the lemma, assume that $\bar{z}_i = \bar{z}_j$ for some $i \neq j$, and without loss of generality let $P(T(u_i), z_i)$ and $P(T(u_j), z_j)$ be the shortest paths such that $\bar{z}_i = \bar{z}_j$. Then, in G , the walk from u_i to \bar{z}_i and that from u_j to \bar{z}_j have the same label sequence and meet at $\bar{z}_i = \bar{z}_j$. This is impossible, since the ports of \bar{z}_i at which these walks end have different port numbers. \square

Corollary 1 *The cardinality of the set $\{v \mid T(v) \equiv T\}$ is the same for all $T \in \mathcal{T}_{\mathbf{f}}$.*

Proof Suppose that $T(u) \not\equiv T(v)$ for some u and v , and assume without loss of generality that the number of vertices having views similar to $T(u)$ is greater than that of those having views similar to $T(v)$. Let $u_1(=u), u_2, \dots, u_k$ be distinct vertices such that $T(u_1) \equiv \dots \equiv T(u_k)$. Since G is connected, there exists a path $P(T(u), z)$, where z is a node such that $\bar{z} = v$. For each $i = 1, 2, \dots, k$, define a node z_i of $T(u_i)$ by $L(T(u_i), z_i) = L(T(u), z)$. Note that z_i is uniquely determined. By Lemma 1, we have $T(\bar{z}_i) \equiv T(\bar{z}_j)$ and $\bar{z}_i \neq \bar{z}_j$ for any $1 \leq i, j \leq k$, a contradiction to the above assumption. \square

For any local edge labeling \mathbf{f} for a graph with n vertices, we define $s_{\mathbf{f}}$ by

$$s_{\mathbf{f}} = n/|\mathcal{T}_{\mathbf{f}}|,$$

where $|\mathcal{T}_{\mathbf{f}}|$ denotes the cardinality of $\mathcal{T}_{\mathbf{f}}$. Corollary 1 implies that $s_{\mathbf{f}}$ is an integer, and that $s_{\mathbf{f}}$ is the cardinality of the set $\{v \mid T(v) \equiv T\}$ for any $T \in \mathcal{T}_{\mathbf{f}}$. For the example of Figure 2(a), $s_{\mathbf{f}} = 1$, since no two views are similar, as easily checked by examining Figure 2(b).

For any positive integer d , let $\#_d(G)$ denote the number of vertices of G with degree d . Clearly, $\#_d(G) = 0$ for all $d \geq n$. Since two vertices with different degrees cannot be similar under \mathbf{f} , we have the following.

Proposition 3

1. *For any graph G and any local edge labeling \mathbf{f} for G , $s_{\mathbf{f}}$ divides the size n of G .*
2. *$s_{\mathbf{f}}$ also divides $\#_d(G)$ for each $d = 1, 2, \dots, n - 1$.* \square

Based on Corollary 1, we now introduce an important property, called *symmetry*, of a graph.

Definition 2 The *symmetry* of a graph G is defined by

$$\sigma(G) = \max\{s_{\mathbf{f}} \mid \mathbf{f} \text{ is a local edge labeling for } G\}.$$

\square

Intuitively, $T_{\mathbf{f}}(v)$ represents the maximum information that processor v can obtain in the worst case (i.e., under the most unfavorable communication timing) by exchanging messages with others. Therefore, the larger the number of processors which have similar views, the more difficult it is to identify their accurate positions in G . The symmetricity $\sigma(G)$ indicates, in the worst case (i.e., under the most unfavorable labeling), how many processors have similar views. Proposition 3 implies that $\sigma(G)$ divides the size n of G , since $\sigma(G) = s_{\mathbf{f}}$ for some \mathbf{f} .

Lemma 2 *For any two local edge labelings \mathbf{f} and \mathbf{g} , if $\mathcal{T}_{\mathbf{f}} \cap \mathcal{T}_{\mathbf{g}} \neq \emptyset$, then $\mathcal{T}_{\mathbf{f}} = \mathcal{T}_{\mathbf{g}}$.*

Proof Let $T \in (\mathcal{T}_{\mathbf{f}} \cap \mathcal{T}_{\mathbf{g}}) \neq \emptyset$ and let v and w be vertices in V such that $T_{\mathbf{f}}(v) \equiv T_{\mathbf{g}}(w) \equiv T$. We assume $\mathcal{T}_{\mathbf{f}} \neq \mathcal{T}_{\mathbf{g}}$ and derive a contradiction. Without loss of generality, we can assume that there exists a vertex u in V such that $\mathcal{T}_{\mathbf{g}}$ does not contain $T_{\mathbf{f}}(u)$. Since G is connected, there exists a path $P(T_{\mathbf{f}}(v), z)$ connecting the root of $T_{\mathbf{f}}(v)$ and a node z , satisfying $u = \bar{z}$. Since $T_{\mathbf{f}}(v) \equiv T_{\mathbf{g}}(w)$, there exists a node z' in $T_{\mathbf{g}}(w)$ such that $P(T_{\mathbf{f}}(v), z) = P(T_{\mathbf{g}}(w), z')$ and the subtree of $T_{\mathbf{g}}(w)$ rooted at z' is similar to the subtree of $T_{\mathbf{f}}(v)$ rooted at z , a contradiction. \square

Intuitively, the above lemma implies that any element of $\mathcal{T}_{\mathbf{f}}$ carries all the information about $\mathcal{T}_{\mathbf{f}}$. For any integer $d \geq 0$, let $T^d(v)$ denote $T(v)$ truncated to depth d , where depth is the distance (in the number of edges) from the root. We assume that each leaf of $T^d(v)$ contains the number of children the corresponding node in $T(v)$ has. The following lemma is by Norris[27].

Lemma 3 ⁸ *$T(u) \equiv T(v)$ if and only if $T^{n-1}(u) \equiv T^{n-1}(v)$.* \square

3.2 Pseudo-Synchronous Algorithms

The main objective of this subsection is to prove that, roughly speaking, the view from a vertex contains all the information that a distributed algorithm can possibly use (Lemma 5), and therefore, in the light of Lemma 3, the sole purpose of message exchanges is to construct a finite view at each vertex.

Lemma 4 *Given any network G , there is an algorithm for each processor v on G to construct $T^d(v)$ for any given nonnegative integer d .*

Proof We give a sketch of a simple algorithm for each processor v to construct $T^d(v)$.

```

Let  $T_0(v)$  be the trivial tree consisting only of the root with information  $deg(v)$ ;
for  $i = 0$  to  $d - 1$  do {
    Send a message  $(T^i(v), j)$  via port  $j = 1, 2, \dots, deg(v)$ ;
    Wait until a message has been received from each port  $j = 1, 2, \dots, deg(v)$ ;
    Construct  $T^{i+1}(v)$  using the available information;
} od

```

⁸We originally proved the the following fact[36], which was recently improved by Norris:

$$T(u) \equiv T(v) \text{ if and only if } T^{n^2}(u) \equiv T^{n^2}(v).$$

It is easy to show that the above algorithm is correct. \square

Note that only the leaves of the received trees (and their incident edges) contain new information needed to extend $T^i(v)$ to $T^{i+1}(v)$. Alternatively, v can discard $T^i(v)$ and construct $T^{i+1}(v)$ by attaching the received trees to a single node (the root). Since processor v does not know that its identity number is v , neither does it know that the view it produces is $T^d(v)$.

We now define a *phase* of an algorithm as the following sequence of instructions.

- If the termination condition holds, send message `done` via each port j and terminate;
- Send messages via some ports (at most one per port);
- Send message `end-phase` via each port j ;
- Receive messages until exactly one `done` or `end-phase` message is received from each port j via which no `done` message has been received in an earlier phase;
- Process information in its local memory;

An algorithm is called *pseudo-synchronous* if it consists of phases. Number the phases of a distributed pseudo-synchronous algorithm, $1, 2, \dots$, from the initial phase onwards. Clearly, if the shortest distance (in terms of the number of links) between two processors is d , then it takes d phases for information to travel from one to the other.

Let A be an algorithm for solving a problem P . The execution of A on a processor consists of an interleaved sequence of processing activities and message exchanges. If, during a round of message exchanges, A does not send any message via some ports, modify A to send an `end-phase` message via each of those ports. Then, it can also expect to receive a message from each port. The resulting algorithm will be pseudo-synchronous. We thus have the following fact.

Proposition 4 *For any network $G \in \mathcal{G}$, there is an algorithm for solving a problem P on G , if and only if there is a pseudo-synchronous algorithm for solving P on G . \square*

The following lemma will form a basis of our subsequent discussions. Its implications are it is possible that any two processors having similar views behave in precisely the same way and, therefore, the labeling which labels each processor by its view is a similarity labeling (see [17]).

Lemma 5 *Let $P \in \mathcal{P}$ and $G \in \mathcal{G}$. There is an algorithm A for solving P on G , using some network attribute information I about G (e.g., size, topology, upbound), if and only if there is a pseudo-synchronous algorithm B for solving P on G using I , such that, in each phase $p + 1$ ($p \geq 0$), each processor v sends $T_{\mathbf{f}}^p(v)$, and nothing else, to all its neighbors.*

Proof The if part is trivial, so we concentrate on the only if part. By Proposition 4, we can assume without loss of generality that algorithm A is pseudo-synchronous. The essential assertion of this lemma is that all information that A , running on processor v , exchanges in its phase $p + 1$ is contained in $T_{\mathbf{f}}^p(v)$. Therefore, we need to show that algorithm B can simulate A using only the information contained in $T_{\mathbf{f}}^p(v)$.

We first describe how, for any phase p of A , algorithm B , running on processor v , can determine the state of A 's execution at v at the end of phase p , if B has a copy of A , information I , and $T_{\mathbf{f}}^p(v)$ at its disposal. Algorithm B simulates A 's phases 1 through p with the help of $T_{\mathbf{f}}^p(v)$ as follows. Note that the state of A 's execution at v at the end of phase p depends on the messages sent by the vertices at distances up to p from v . Therefore, B simulates phase p of A using p rounds. In the first round, for each node x in $T_{\mathbf{f}}^p(v)$, B simulates the phase 1 of A running on \bar{x} . In this simulation, x follows the steps of A , simulating the sending/receiving of messages via each port, and then doing some computation. The root of the tree, in particular, is now in the same state that A , running on v , would be in at the end of phase 1. Algorithm B now proceeds to round $i = 2, 3, \dots, p$ of simulation of A on \bar{x} for each node x at distances $\leq p-i+1$ from the root in $T_{\mathbf{f}}^p(v)$. After p rounds of simulation, the root of the tree will be in the same execution state that A running on v would be in at the end of phase p .

Algorithm B repeats the above simulation for each phase $p = 1, 2, \dots$, of A . Namely, B constructs $T_{\mathbf{f}}^p(v)$ using the messages $T_{\mathbf{f}}^{p-1}(u)$ sent from each neighbor u (except when $p = 1$, in which case $T_{\mathbf{f}}^0(v)$ is locally available), and simulates A for the first p phases. If A does not terminate in phase p , B sends $T_{\mathbf{f}}^p(v)$ to every neighbor in the next phase $p+1$. B eventually terminates since A does. \square

In the following several sections, we will investigate the classes of graphs for which the problems defined in Section 1 are solvable, applying the concept of graph symmetry introduced in this section.

4 The Known Topology Case

We start our case study with *topology*. Since the topology is assumed to be known, an algorithm A_G , specific to a particular network G on which it runs, will also be considered in this section. Recall, however, that each processor knows neither its identity number nor which vertex of G it is represented by. An algorithm must work correctly for any local edge labeling, since, by definition, it cannot expect any particular local edge labeling. The results of Subsections 4.1 and 4.2 apply equally well to *size*, where only the graph size n , not the topology, of the network is known.

In view of Lemma 3, in the rest of this section, we define $d = n - 1$, and let $\mathcal{T}_{\mathbf{f}}^d$ denote the set of all dissimilar (partial) views $T_{\mathbf{f}}^d(v)$, i.e.,

$$\mathcal{T}_{\mathbf{f}}^d = \{T_{\mathbf{f}}^d(v) \mid v \in V\}.$$

Note that Lemma 3 implies $|\mathcal{T}_{\mathbf{f}}^d| = |\mathcal{T}_{\mathbf{f}}|$.

4.1 Characterization of $D_{topology}$ (ELECT-LEADER)

Lemma 6 *A graph G is in $D_{topology}$ (ELECT-LEADER) if $\sigma(G) = 1$.*

Proof We give a sketch of an algorithm A_{EL} for solving ELECT-LEADER. Since the set of all views truncated to finite depths is enumerable, we can fix a total order $<$ among these finite views. A_{EL} contains a subalgorithm for sorting a given finite set of finite views in the order $<$. On each processor v , A_{EL} constructs $T_{\mathbf{f}}^k(v)$, where $k = 2(n - 1)$, based on the

messages exchanged with other processors. (See Lemma 4.) Note that A_{EL} running on v knows neither \mathbf{f} nor v . However, A_{EL} can construct $\mathcal{T}_{\mathbf{f}}^d$ from $T_{\mathbf{f}}^k(v)$ by Lemma 3, since every element in $\mathcal{T}_{\mathbf{f}}^d$ must appear in $T_{\mathbf{f}}^k(v)$ as a subtree whose root is within distance $n - 1$ from the root of $T_{\mathbf{f}}^k(v)$. Now all the processors have the same set $\mathcal{T}_{\mathbf{f}}^d$. Let T^* be the “smallest” element in $\mathcal{T}_{\mathbf{f}}^d$ with respect to the order $<$. Then A_{EL} elects v (the processor on which it is running) as the leader if and only if $T^* \equiv T_{\mathbf{f}}^d(v)$, i.e., the view of depth d it computes on v . The correctness of A_{EL} is guaranteed by the fact that the symmetricity of G is 1; if more than one processor constructed the same view T^* , the symmetricity of G would be greater than 1.⁹ \square

Lemma 7 $\sigma(G) = 1$ if G is in $D_{topology}(\text{ELECT-LEADER})$.

Proof We assume that there is an algorithm A for solving ELECT-LEADER for a graph G with symmetricity $\sigma(G) \geq 2$, and derive a contradiction. Since $\sigma(G) \geq 2$, there is a local edge labeling \mathbf{f} such that $s_{\mathbf{f}} \geq 2$ (for the definition of $s_{\mathbf{f}}$, see Section 3), namely, for any u in V , there exists v ($\neq u$) such that $T_{\mathbf{f}}(u) \equiv T_{\mathbf{f}}(v)$ (by Corollary 1). On the other hand, by Lemma 5, there exists a pseudo-synchronous algorithm $B \in ALG_{topology}$ for solving ELECT-LEADER such that, in each phase $p + 1$ ($p \geq 0$), each processor v sends $T^p(v)$, and nothing else. Consider the computation of B on each vertex on G . B on u and B on v proceed identically, since $T_{\mathbf{f}}(u) \equiv T_{\mathbf{f}}(v)$. Thus B on u decides that u is the leader if and only if B on v decides that v is the leader, a contradiction. \square

Theorem 1 $\sigma(G) = 1$ if and only if G is in $D_{topology}(\text{ELECT-LEADER})$.

Proof Follows from Lemmas 6 and 7. \square

4.2 Characterization of $D_{topology}(\text{ELECT-EDGE})$

Theorem 2 A graph G is in $D_{topology}(\text{ELECT-EDGE})$ if and only if either of the following two conditions holds:

1. $\sigma(G) = 1$, or
2. $\sigma(G) = 2$ and there exists an edge (u, v) such that $T_{\mathbf{f}}(u) \equiv T_{\mathbf{f}}(v)$, for any local edge labeling \mathbf{f} with $s_{\mathbf{f}} = 2$.

Proof If Part: We give a sketch of an algorithm A_{EE} for solving the ELECT-EDGE. A_{EE} works like A_{LE} in the proof of Lemma 6, so we adopt the notations used there. First, A_{EE} constructs the set $\mathcal{T}_{\mathbf{f}}^d$ and checks whether $|\mathcal{T}_{\mathbf{f}}^d| = n$, i.e., $s_{\mathbf{f}} = 1$. If so, it elects a vertex u having the view T^* as the leader. Then u elects the edge $e = (u, v)$ such that $f_u(e) = 1$. Otherwise, $\sigma(G) = 2$, i.e., $|\mathcal{T}_{\mathbf{f}}^d| = n/2$, and there exists an edge (u, v) such that $T_{\mathbf{f}}(u) \equiv T_{\mathbf{f}}(v)$. Let Δ be the set of all dissimilar views $T_{\mathbf{f}}^d(u)$ defined as follows:

$$\Delta = \{T_{\mathbf{f}}^d(u) \mid \text{there exists an edge } (u, v) \text{ such that } T_{\mathbf{f}}^d(u) \equiv T_{\mathbf{f}}^d(v)\}$$

⁹Algorithm A_{EL} does not use any information about G besides the size n . More precisely, for the purpose here, all A_{EL} needs to know is an upper bound on n . However, when the exact n is used, A_{EL} can elect a leader correctly if $|\mathcal{T}_{\mathbf{f}}^d| = n$, even if $\sigma(G) > 1$.

Δ is computable and is, by assumption, not empty. Let T^{**} be the “smallest” element in Δ with respect to the total order $<$. Then the edge (u', v') such that $T_{\mathbf{f}}^d(u') \equiv T_{\mathbf{f}}^d(v') \equiv T^{**}$ is elected. The correctness of A_{EE} is guaranteed by the fact that u' and v' are the only vertices having views similar to T^{**} .¹⁰

Only if part: Assume that $s_{\mathbf{f}} = k \geq 2$ for some local edge labeling \mathbf{f} , and that there exists an algorithm A for solving ELECT-EDGE on G . By Lemma 5, there is a pseudo-synchronous algorithm $B \in ALG_{topology}$ for solving the same problem such that, in each phase $p+1$ ($p \geq 0$), each processor v sends $T^p(v)$, and nothing else. Consider the computation of B on G . Since $s_{\mathbf{f}} = k$, let u_1, u_2, \dots, u_k be the vertices in V such that $T(u_1) \equiv T(u_2) \equiv \dots \equiv T(u_k)$. Assume without loss of generality that an edge $e = (u, v)$ is selected by u_1 , since by definition of ELECT-EDGE at least one processor selects an edge. Thus u_1 knows which edge (or edges) in $T(u_1)$ corresponds (or correspond) to e of G . Let $e_1 = (y_1, z_1)$ be an edge in $T(u_1)$ that corresponds to e . Now let $e_i = (y_i, z_i)$ ($2 \leq i \leq k$) be the edge in $T(u_i)$ which is at the same relative position as e_1 ; i.e., $L(T(u_1), y_1) = L(T(u_i), y_i)$ and $L(T(u_1), z_1) = L(T(u_i), z_i)$ hold. Then, by Lemma 1, we have $\bar{y}_i \neq \bar{y}_j$ and $\bar{z}_i \neq \bar{z}_j$ for any i, j ($i \neq j$). Since $T(u_1) \equiv T(u_i)$ for all i ($1 \leq i \leq k$), whenever u_1 selects e_1 , each u_i selects e_i . Therefore, we must have

$$\{\bar{y}_i, \bar{z}_i \mid 1 \leq i \leq k\} = \{u, v\}.$$

It thus follows that $k = 2$, $\bar{y}_1 = \bar{z}_2$ and $\bar{y}_2 = \bar{z}_1$. This implies that $T(\bar{y}_1) \equiv T(\bar{z}_1)$ since $T(\bar{y}_1) \equiv T(\bar{y}_2)$. This completes the proof. \square

To see that $\sigma(G) \leq 2$ alone is not sufficient for G to be in $D_{topology}(\text{ELECT-EDGE})$, consider the graph G and the local edge labeling \mathbf{f} for G shown in Figure 3. From the symmetry of G , it is easy to see that $T(u_i) \equiv T(v_i)$ for $i = 1, 2, 3$, and that $\sigma(G) = 2$. Intuitively, this means that u_i , for example, cannot know whether it is u_i or v_i . Therefore, u_i selects an edge e with labels p and q if and only if v_i selects the other edge having the same labels. It follows that $G \notin D_{topology}(\text{ELECT-EDGE})$. Note that G in Figure 3 has no pair (u, v) that satisfies the condition 2 of Theorem 2.

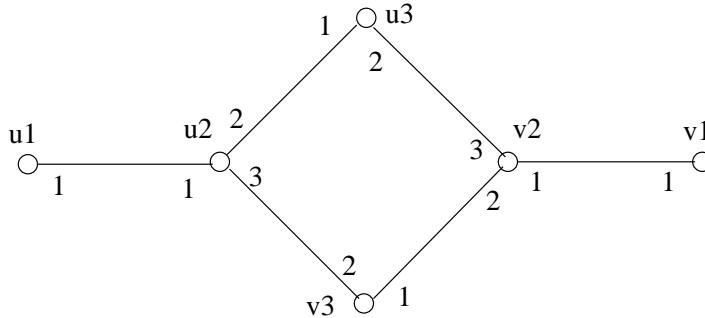


Figure 3: A graph $G \notin D_{topology}(\text{ELECT-EDGE})$ with $\sigma(G) = 2$.

¹⁰The only information needed by A_{EE} about G is its size n . It will become clear that an upper bound on n is the only information needed by A_{EE} about G when only “valid” graphs (i.e., graphs satisfying the condition (1) or (2)) are expected. (See Section 6.)

4.3 Inclusion Relationships among $\{D_{topology}(P) \mid P \in \mathcal{P}\}$

Theorem 3

$$\begin{aligned}
 D_{topology}(\text{ELECT-LEADER}) &\subset D_{topology}(\text{ELECT-LEADER}) \cup \mathcal{S} \\
 &\subset D_{topology}(\text{SPANNING-TREE}) \\
 &= D_{topology}(\text{ELECT-EDGE}) \\
 &\subset \mathcal{G},
 \end{aligned}$$

where \mathcal{G} and \mathcal{S} denote the sets of all networks (see Section 2) and all trees, respectively, and \subset denotes proper inclusion.

Proof (I) We show $D_{topology}(\text{SPANNING-TREE}) = D_{topology}(\text{ELECT-EDGE})$. Consider any $G \in D_{topology}(\text{SPANNING-TREE})$. We give a sketch of an algorithm for solving ELECT-EDGE on G . The algorithm first constructs a spanning tree T of G . Starting from each leaf of T , tokens are passed as follows:

- (a) Each leaf sends a token via a tree edge to its neighbor with respect to T .
- (b) Let $deg_T(v)$ be the degree of v with respect to T . In general, vertex v waits until it has received $deg_T(v) - 1$ tokens from neighbors, and sends a token via the tree edge through which no message has been received. (Thus, the leaves don't wait.)

Imagine that the edges of T are initially all white and a tree edge turns black when a token is transmitted over it. It is not difficult to see that at any instant, the subtree defined by the white edges, if any, is connected. Consider the last white edge to turn black, and let u and v be its two end vertices. They receive a token from each other as their last token to be received. Since the edge (u, v) is unique, u and v can select (u, v) . Hence $G \in D_{topology}(\text{ELECT-EDGE})$.

Next, assume $G \in D_{topology}(\text{ELECT-EDGE})$. Let $e = (u, v)$ be the edge elected by some algorithm. The following algorithm SE-ST (Single-source-Edge Spanning Tree) Algorithm, executed by each vertex, constructs a spanning tree of G , starting from the two vertices u and v . Its correctness proof is left for the reader.

[SE-ST Algorithm]

Let (u, v) be the source edge;

Initially, only u and v are *active* and they send **wake-up** to all their neighbors;

The other processors are *asleep*;

(Sleep state)

Wait until a **wake-up** message is received;

Change state to *active*;

(Active state)

Let L be the set of edges through which **wake-up** messages have been received;

Select an edge e in L and send back a message **tree-edge** through e ;

Send back a message **non-tree-edge** through each edge in L except for e ;

Send a message **wake-up** through each edge not in L ;

Wait until all acknowledgements (**tree-edge** or **non-tree-edge**)

have been received and terminate. [End of SE-ST]

(II) We show $D_{topology}(\text{ELECT-LEADER}) \subset D_{topology}(\text{ELECT-LEADER}) \cup \mathcal{S}$, in other words, there is a tree T such that $T \notin D_{topology}(\text{ELECT-LEADER})$. Consider a simple tree $T = (\{u, v\}, \{(u, v)\})$, i.e., two vertices with an edge connecting them. It is easy to show $\sigma(T) = 2$. By Theorem 1, the ELECT-LEADER is unsolvable on T .

(III) Next, we show $D_{topology}(\text{ELECT-LEADER}) \cup \mathcal{S} \subset D_{topology}(\text{SPANNING-TREE})$. \mathcal{S} is obviously contained in $D_{topology}(\text{SPANNING-TREE})$. By Theorems 1 and 2, and (I) above, it is sufficient to show that there is a non-tree graph G which is in $D_{topology}(\text{SPANNING-TREE})$ but not in $D_{topology}(\text{ELECT-LEADER})$. Consider the graph G in Figure 4 with a local edge labeling as shown. For this G , it is easy to see that $\sigma(G) = 2$. Therefore, G is not in $D_{topology}(\text{ELECT-LEADER})$. To show that G is in $D_{topology}(\text{SPANNING-TREE})$, we give a sketch of an algorithm for solving the SPANNING-TREE on G . Vertices u and v know that they are either u or v from their degrees. They decide that all edges incident to them are tree edges. Then each of them selects an edge incident to it; say, u selects (u, u') and v selects (v, v') . Vertices u and v can easily decide whether u' and v' are adjacent or not. If they are adjacent, u and v select edge (u', v') as a tree edge. Otherwise, the edge which is incident to neither u' nor v' is selected as a tree edge.

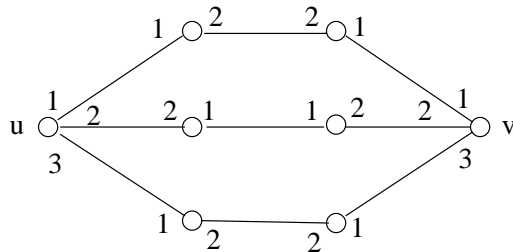


Figure 4: Illustration for the proof of Theorem 3.

(IV) Finally, it is clear that \mathcal{G} properly contains $D_{topology}(\text{SPANNING-TREE})$, since the SPANNING-TREE is unsolvable for the rings. \square

Corollary 2 *If G is a tree, then $\sigma(G) \leq 2$.*

Proof Follows from Theorems 2 and 3. \square

Besides the problems investigated above there are others worth investigating, which are apparently more difficult than ELECT-LEADER. The *location recognition problem*, i.e., the problem of computing on G an automorphism ψ from V to V in the sense that the processor v generates $\psi(v)$, and the *naming problem*, i.e., the problem of attaching a unique name to each processor v , are such problems. However, it turns out that $D_{topology}(P)$'s coincide with $D_{topology}(\text{ELECT-LEADER})$ for these problems P , since there are straightforward algorithms for solving these problems, provided that they are initiated by a unique initiator (leader).

5 Quotient Graphs

In this section, we first relate symmetricity $\sigma(G)$ with other familiar properties of a graph G . As a corollary, the class $D_{topology}(\text{ELECT-LEADER})$ is characterized in graph theoretical

terms. The characterization is achieved in terms of a new concept called the “quotient graph.” Informally, the quotient graph of G induced by a local edge labeling \mathbf{f} represents what the smallest possible G would look like to the processors in G under \mathbf{f} . We next characterize the trees in terms of their quotient graphs. This result will be used at the end of Section 5.1 to determine whether or not a given network is a tree. In the Section 5.2, we will discuss the problem of constructing the smallest G , given its quotient graph with respect to a particular local labeling \mathbf{f} . This is relevant to the discussion of FIND-TOPOLOGY in the next section. In Subsection 5.2, we prepare some theoretical tools for studying this problem.

In the rest of paper, we mostly investigate graphs with edge labels and label-preserving isomorphisms among them. However, we also sometimes consider graphs without edge labels. We use the notation $G \simeq G'$ to mean that two graphs G and G' are isomorphic (ignoring their edge labels, if any).

5.1 Relating Symmetricity to Factors

Example 2 Consider the graph G and a local edge labeling \mathbf{f} for G shown in Figure 5(a). A distributed algorithm running on one of the vertices represented by \circ in the network of Figure 5(a) will behave identically on the \circ vertex in the network of Figure 5(b). The same applies to the vertices represented by \diamond or \bullet . Another way of stating this fact is that, in G , the views from all vertices represented by the same symbol (\circ, \diamond or \bullet) are similar in Figures 5(a) and (b). \square

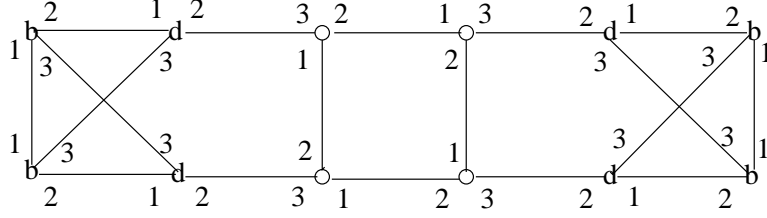
Let $G = (V, E)$ be a graph and \mathbf{f} be a local edge labeling for G . We partition V into g disjoint sets, each of which contains exactly $s_{\mathbf{f}}$ vertices having similar views, where $g = n/s_{\mathbf{f}}$. More formally, if $\mathcal{T}_{\mathbf{f}} = \{T_1, \dots, T_g\}$ is the set of all dissimilar views under \mathbf{f} , then $V_i = \{v \in V \mid T(v) \equiv T_i\}$ for $i = 1, 2, \dots, g$. Namely, V_1, \dots, V_g are the equivalence classes induced by the similarity relation, \equiv . We will refer to each V_i as a “similarity class.” In Figure 5(a), for example, V_1 (resp. V_2 and V_3) consists of the four vertices represented by \circ (resp. \diamond and \bullet). Intuitively, the “quotient graph” induced by \equiv is obtained from G by collapsing all vertices of V_i into one vertex for each i . (See Figure 5(b).)

Definition 3 The *quotient graph*, denoted G/\mathbf{f} , of G induced by \mathbf{f} is an undirected, connected (not necessary simple) graph $(\mathcal{T}_{\mathbf{f}}, E_{\mathbf{f}})$ with edge labels defined as follows. $E_{\mathbf{f}}$ contains an edge (T, T') with two labels, p at T 's end and q at T' 's end, denoted $(T, T' : p, q)$, if and only if there exists an edge $e = (u, v) \in E$ such that (1) $f_u(e) = p$, (2) $f_v(e) = q$, (3) $T(u) \equiv T$, and (4) $T(v) \equiv T'$. \square

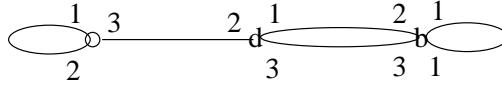
The graph shown in Figure 5(b) is the quotient graph for the example of Figure 5(a).

Lemma 8 If $G/\mathbf{f} = (\mathcal{T}_{\mathbf{f}}, E_{\mathbf{f}})$ has an edge $(T, T' : p, q)$, then for any vertex u of G such that $T(u) \equiv T$, there is a vertex u' in G such that (1) $T(u') \equiv T'$, (2) $e = (u, u') \in E$, (3) $f_u(e) = p$, and (4) $f_{u'}(e) = q$.

Proof Let u_1 and u_2 be any two vertices in V_i such that $T(u_1) \equiv T(u_2)$ holds. If an edge $e_1 = (u_1, w_1)$ is in E then, by Lemma 1, there exists an edge $e_2 = (u_2, w_2)$ in E such that $f_{u_1}(e_1) = f_{u_2}(e_2)$ and $f_{w_1}(e_1) = f_{w_2}(e_2)$, and that w_1 and w_2 are in the same similarity class V_j . \square



(a)



(b)

Figure 5: A graph with a local edge labeling and its quotient graph: (a) G with a local edge labeling \mathbf{f} ; (b) G/\mathbf{f} .

Based on the above lemma, we now introduce a label-preserving graph homomorphism $\tau_{\mathbf{f}}$ from $G = (V, E)$ with \mathbf{f} to $G/\mathbf{f} = (\mathcal{T}_{\mathbf{f}}, E_{\mathbf{f}})$ as follows.

1. $\tau_{\mathbf{f}}$ maps each vertex $u \in V$ to $T(u) \in \mathcal{T}_{\mathbf{f}}$.
2. $\tau_{\mathbf{f}}$ maps each edge $e = (u, v) \in E$ to $\tau_{\mathbf{f}}(e) = (T(u), T(v) : p, q)$, where $p = f_u(e)$ and $q = f_v(e)$.

Note that $\tau_{\mathbf{f}}$ is, in general, a many-to-one mapping. The inverse mapping of $\tau_{\mathbf{f}}$, denoted by $\tau_{\mathbf{f}}^{-1}$, is, therefore, one-to-many.

In the following theorem, $G_{V'} = (V', E')$ denotes the subgraph of G induced by $V' \subseteq V$; so, $E' = E \cap (V' \times V')$. From Lemma 8, we can derive the following theorem.

Theorem 4 *Let $G = (V, E)$ be a graph and \mathbf{f} be a local edge labeling for G . For any spanning tree S , including the edge labels, of G/\mathbf{f} , V can be partitioned into $s_{\mathbf{f}}$ subsets, $U_1, \dots, U_{s_{\mathbf{f}}}$, satisfying the following conditions:*

1. $|U_i| = n/s_{\mathbf{f}}$ for each $i = 1, 2, \dots, s_{\mathbf{f}}$, and
2. G_{U_i} has a spanning tree isomorphic to S including the edge labels for each $i = 1, 2, \dots, s_{\mathbf{f}}$.

Proof We only present a proof sketch. Construct vertex sets $U_1, \dots, U_{s_{\mathbf{f}}}$ that are “orthogonal” to the similarity classes under \equiv , V_1, \dots, V_g , where $g = n/s_{\mathbf{f}}$, i.e., for each i, j ($i = 1, \dots, s_{\mathbf{f}}$; $j = 1, \dots, g$) $|U_i \cap V_j| = 1$. Set U_1 is constructed as follows. Pick an arbitrary vertex $u \in V_1$ and put u in U_1 . Then embed S in G using $\tau_{\mathbf{f}}$, identifying $\tau_{\mathbf{f}}(u)$ with u , and let U_1 be the set of all vertices of G upon which S is embedded. Since S has

g vertices, each corresponding to a different similarity class, we have $|U_1| = g$. U_2 is constructed by starting with another vertex in $V_1 - \{u\}$, and so forth. It follows from Lemma 1 that $U_1, \dots, U_{s_{\mathbf{f}}}$ are disjoint. \square

As an example, consider the spanning tree S of G/\mathbf{f} in Figure 5(b) that contains the two horizontal edges with labels (3,2) and (1,2). Four instances of this tree can be embedded in G of Figure 5(a), one in each quadrant of Figure 5(a).

A graph $G = (V, E)$ is said to be *bipartite* if V can be partitioned into two subsets V_1 and V_2 such that every element of E joins a vertex of V_1 to a vertex of V_2 . To express V_1 and V_2 explicitly, we denote G as $(V_1//V_2, E)$. A graph $G = (V, E)$ is said to be *r -regular* if every vertex has degree r . For an integer p , a *p -factor* of G is a spanning p -regular subgraph of G . Note that a 2-factor is a set of cycles. G is said to be *p -factorable* if there exists a set of p -factors F_1, \dots, F_k where $F_i = (V, E_i)$, such that E_1, \dots, E_k constitute a partition of E . Such a set of p -factors is called a *p -factorization* of G . G is said to be *{1,2}-factorable* if there exists a set of 1- or 2-factors F_1, \dots, F_k ($F_i = (V, E_i)$) such that E_1, \dots, E_k constitute a partition of E . Such a set of 1- and 2-factors is called a *{1,2}-factorization* of G .

Definition 4 A graph $G = (V, E)$ with $|V| = n$ satisfies the *factorization condition* (or *F -condition*, for short) *with g* , if there is a partition $\{V_1, \dots, V_g\}$ of V , each of size n/g , satisfying both of the following conditions.

1. For any $i = 1, 2, \dots, g$, G_{V_i} is {1,2}-factorable.
2. For any $i, j = 1, 2, \dots, g$ ($i \neq j$), $B_{V_i, V_j} = (V_i//V_j, E \cap (V_i \times V_j))$ is a regular bipartite graph. \square

Then the symmetricity $\sigma(G)$ of a graph G is characterized by the following theorem.

Theorem 5 *For any graph $G = (V, E)$, g is the smallest integer such that G satisfies the F -condition with g , if and only if $\sigma(G) = n/g$. \square*

In order to prove this theorem, we first examine the relationship between G with \mathbf{f} and G/\mathbf{f} . In understanding the rest of this subsection, it is helpful to recall the embedding of several instances of a tree S in G that we described in the proof of Theorem 4. Note, for instance, that no edge of G connecting a pair of vertices in some V_i is used to embed S . In fact, such an edge connect two distinct embedded instances of S . Moreover, it is easy to see that this edge manifests itself as a self-loop in G/\mathbf{f} . If an edge e of G connects a vertex in V_i to another vertex in V_j ($i \neq j$), on the other hand, e may or may not belong to an embedded instance of S . We are now ready to prove the following lemma.

Lemma 9 *Given a graph G and a local edge labeling \mathbf{f} for G , let $G/\mathbf{f} = (T_{\mathbf{f}}, E_{\mathbf{f}})$. Consider any edge $e = (T_1, T_2 : p, q) \in E_{\mathbf{f}}$, and let $V_1 = \{u \mid T(u) \equiv T_1\}$ and $V_2 = \{u \mid T(u) \equiv T_2\}$.*

1. *If $T_1 = T_2$, i.e., e is a self-loop, and $p = q$, then $F_1 = (V_1, \tau_{\mathbf{f}}^{-1}(e))$ is a 1-factor of G_{V_1} .*
2. *If $T_1 \neq T_2$, i.e., e is a self-loop, and $p \neq q$, then $F_2 = (V_1, \tau_{\mathbf{f}}^{-1}(e))$ is a 2-factor of G_{V_1} .*

3. If $T_1 \neq T_2$, then $F_3 = (V_1 // V_2, \tau_{\mathbf{f}}^{-1}(e))$ is a 1-regular bipartite graph.

Proof (I) (In reading this part of proof, it will be helpful to use the self-loop at the \bullet vertex of Figure 5(b) as an example.) We show that F_1 is a 1-regular. By definition, this type of self-loop implies that there exist vertices u and u' in V_1 such that $(u, u') \in E$ and $f_u(u, u') = f_{u'}(u, u') = p$. Therefore, by Lemma 8, for any vertex $v \in V_1$, there exists a vertex $v' \in V_1$ such that $e' = (v, v') \in E$ and $f_v(e') = f_{v'}(e') = p$, which implies $e' \in \tau_{\mathbf{f}}^{-1}(e)$. Each vertex of F_1 has degree 1, since each vertex has just one port numbered p .

(II) (In reading this part of proof, it will be helpful to use the self-loop at the \circ vertex of Figure 5(b) as an example.) We show that F_2 is 2-regular. By definition, this type of self-loop implies that there exist vertices u and v in V_1 such that $(u, u'), (v, v') \in E$, $f_u(u, u') = p \neq q = f_{u'}(u, u')$, and $f_v(v, v') = p \neq q = f_{v'}(v, v')$. Therefore, by Lemma 8, there exists a vertex w in V_1 such that $e' = (u, w) \in E$, $f_u(e') = q$ and $f_w(e) = p$. Therefore, any vertex in F_2 has degree at least 2. No vertex in F_2 can have degree larger than 2, since its port numbers consist only of $\{p, q\}$.

(III) (In reading this part of proof, it will be helpful to use the edge between at the \diamond vertex and the \bullet vertex of Figure 5(b) as an example.) We show that F_3 is a 1-regular bipartite graph. Clearly, F_3 is bipartite. The fact that F_3 is 1-regular follows from the same argument as in (I) above. \square

Corollary 3 *Given a graph G and a local edge labeling \mathbf{f} for G , G satisfies the F -condition with $g = n/s_{\mathbf{f}}$.*

Proof Let $V_i = \{x \in V \mid T(x) \equiv T_i\}$ for each T_i in $\mathcal{T}_{\mathbf{f}}$. Repeated applications of Lemma 9 show that the F -condition is satisfied with the partition $\{V_1, \dots, V_g\}$. \square

Lemma 10 *If a graph G satisfies the F -condition with g , then there is a local edge labeling \mathbf{f} for G such that $s_{\mathbf{f}} \geq n/g$.*

Proof See Appendix. \square

[Proof of Theorem 5]

If part: Let G be any graph and \mathbf{f} be a local edge labeling for G such that $\sigma(G) = s_{\mathbf{f}}$ holds. Then by Corollary 3, G satisfies F -condition with $g = n/\sigma(G)$. Assume that for some $k > \sigma(G)$, G satisfies F -condition with $g = n/k$. Then there will be a labeling \mathbf{f} for G such that $s_{\mathbf{f}} \geq n/g > \sigma(G)$ by Lemma 10, a contradiction to the definition of symmetricity.

Only if part: Let g be the smallest integer such that G satisfies the F -condition with g . Then $\sigma(G) \geq n/g$ holds, since there is a local edge labeling \mathbf{f} for G such that $s_{\mathbf{f}} \geq n/g$ by Lemma 10. To show $\sigma(G) \leq n/g$, assume that $\sigma(G) > n/g$. Let \mathbf{f} be a local edge labeling for G such that $\sigma(G) = s_{\mathbf{f}} > n/g$ holds. Then by Corollary 3, G satisfies the F -condition with $n/s_{\mathbf{f}} < g$, a contradiction. \square

We now proceed to the problem of characterizing the set \mathcal{S} of all trees. To this end, we first characterize cycles in G in terms of the structure of G/\mathbf{f} .

Lemma 11 *A graph G has a cycle if, for any local edge labeling \mathbf{f} , G/\mathbf{f} has one of the following:*

1. a self-loop $(T_i, T_i : p, q)$, ($p \neq q$),
2. two self-loops, $(T_i, T_i : p, p)$ and $(T_j, T_j : q, q)$, where $p = q$ is not excluded if $i \neq j$, or
3. a cycle of length ≥ 2 .

Proof (I) Let $e = (T_i, T_i : p, q)$ be a self-loop in G/\mathbf{f} . By Lemma 9, $\tau_{\mathbf{f}}^{-1}(e)$ is a 2-factor in G_{V_i} . Therefore, G has a cycle.

(II) Let $e = (T_i, T_i : p, p)$ and $e' = (T_j, T_j : q, q)$ be two self-loops. If they are at the same vertex $T_i = T_j$ of G/\mathbf{f} , then clearly, each vertex in the subgraph of G , $F = (V_i, \tau_{\mathbf{f}}^{-1}(e) \cup \tau_{\mathbf{f}}^{-1}(e'))$, has degree 2, hence F contains a cycle. If e and e' are at two different vertices, then let $F = (V_i \cup V_j, \tau_{\mathbf{f}}^{-1}(e) \cup \tau_{\mathbf{f}}^{-1}(e'))$. Now, pick a spanning tree of G/\mathbf{f} and construct a partition $U_1, \dots, U_{s_{\mathbf{f}}}$, as in Theorem 4. In F , introduce $s_{\mathbf{f}}$ new edges between V_i and V_j such that $u \in V_i$ and $v \in V_j$ are joined by an edge if and only if $u, v \in U_h$ for some h . Such an edge indicates the fact that there is a path between u and v in the spanning tree in G_{U_h} . It is easy to see that this new graph, F' , has a cycle, since each vertex in it has degree 2. It follows that G has a cycle.

(III) Let C be a cycle in G/\mathbf{f} , and construct a spanning tree S of G/\mathbf{f} so that all but one edge $e = (T_i, T_j)$ of C belong to S . Let $F = (V_i \cup V_j, \tau_{\mathbf{f}}^{-1}(e))$ and add $s_{\mathbf{f}}$ new edges between V_i and V_j as in (II) above. The rest of the proof is analogous to (II). \square

The above lemma holds even if “if, for any local edge labeling \mathbf{f} for G ,” is replaced by “if there is a local edge labeling \mathbf{f} for G such that”. In which of the three forms a cycle in G manifests itself in G/\mathbf{f} depends on \mathbf{f} .

Theorem 6 *A graph G is a tree if and only if, for any¹¹ local edge labeling \mathbf{f} for G , the quotient graph $G/\mathbf{f} = (\mathcal{T}_{\mathbf{f}}, E_{\mathbf{f}})$ is a tree, except for at most one self-loop $e = (T_i, T_i : p, p)$ for some $T_i \in \mathcal{T}_{\mathbf{f}}$ and p .*

Proof The only if part follows directly from Lemma 11. To prove the if part, let $G/\mathbf{f} = (\mathcal{T}_{\mathbf{f}}, E_{\mathbf{f}})$, where $\mathcal{T}_{\mathbf{f}} = \{T_1, \dots, T_g\}$, be a tree, except possibly for one self-loop. We try to reconstruct $G = (V, E)$. V consists of g subsets V_1, \dots, V_g , corresponding to vertices T_1, \dots, T_g , of G/\mathbf{f} . There are two cases to consider.

- (a) G/\mathbf{f} has no self-loop. So G consists of $s_{\mathbf{f}}$ disjoint trees. Since, by definition, G is connected, $s_{\mathbf{f}} = 1$ must hold.
- (b) G/\mathbf{f} has a self-loop $e = (T_i, T_i : p, p)$. Let $S = (\mathcal{T}_{\mathbf{f}}, E_{\mathbf{f}} - \{e\})$ be the unique spanning tree of G/\mathbf{f} , and consider the graph $G' = (V, E')$, where $E' = \{\tau_{\mathbf{f}}^{-1}(e') \mid e' \in E_{\mathbf{f}} - \{e\}\}$. G' is clearly a collection of $s_{\mathbf{f}}$ ($= n/g$) disjoint trees, each isomorphic to S . The only difference between G and G' is that the edges in $\tau_{\mathbf{f}}^{-1}(e)$ are missing in G' . By Lemma 9, $F = (V_i, \tau_{\mathbf{f}}^{-1}(e))$ is a 1-factor of G_{V_i} , which implies that each edge in $\tau_{\mathbf{f}}^{-1}(e)$ joins a pair of trees in G' . Therefore, $s_{\mathbf{f}}$ is even and G consists of $s_{\mathbf{f}}/2$ disjoint trees. Since G is connected, we have $s_{\mathbf{f}} = 2$.

In either case, G is a tree. \square

¹¹This theorem is valid even if “if, for any local edge labeling \mathbf{f} for G ,” is replaced by “if there is a local edge labeling \mathbf{f} for G such that”.

5.2 Minimum Realization of a Quotient Graph

As we have seen, for any simple graph G' and a local edge labeling \mathbf{f}' for G' , the quotient graph G'/\mathbf{f}' is a connected (not necessary simple) graph with edge labels. We extend the definition of local edge labeling to non-simple graphs by allowing the same port numbers at both ends of a self-loop. Thus the edge labels on a quotient graph G'/\mathbf{f}' form a local edge labeling. The problem we address in this subsection is: given a quotient graph $Q = G'/\mathbf{f}'$, construct a simple, connected graph $G = (V, E)$ and a local edge labeling \mathbf{f} , called a *realization* of Q , such that G/\mathbf{f} is isomorphic to G'/\mathbf{f}' , including the edge labels. In general, there are many different realizations. For example, two different realizations of G/\mathbf{f} shown in Figure 6(c) are given in Figures 6(a) and (b). A realization $R = (G, \mathbf{f})$ of G'/\mathbf{f}' is said to be *minimum* if G has the minimum number of vertices among all the realizations of G'/\mathbf{f}' . We restrict Q to the quotient graphs, since otherwise G/\mathbf{f} may be smaller than Q for some “realizations” of Q .

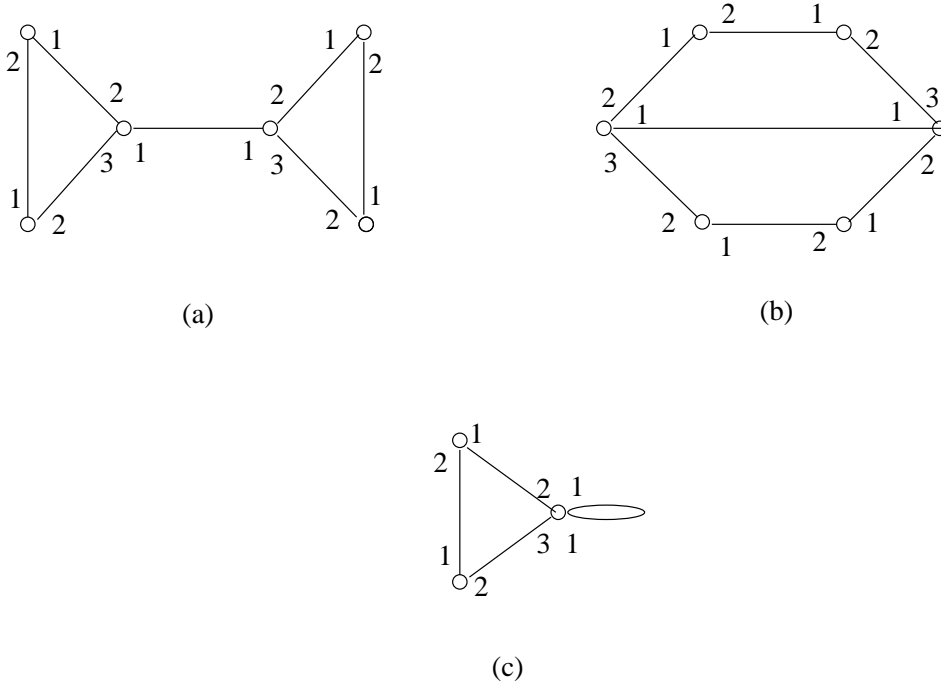


Figure 6: Two graphs and their quotient graph: (a) G with \mathbf{f} . (b) G' with \mathbf{f}' . (c) G/\mathbf{f} is isomorphic to G'/\mathbf{f}' including the edge labels.

Given a quotient graph $Q = G'/\mathbf{f}' = (W, A)$, where $W = \{w_1, \dots, w_g\}$, for each $w_i \in W$, define three parameters, ℓ_i , m_i and \hat{m}_i , as follows:

- ℓ_i : the number of self-loops of the form $(w_i, w_i : p, p)$ for some w_i and p ,
- m_i : the number of self-loops of the form $(w_i, w_i : p, q)$ for some w_i and p, q ($p \neq q$).
- $\hat{m}_i = \max_{j(\neq i)} \{m_{ij}\}$, where m_{ij} is the number of parallel edges connecting w_i and w_j .

Define further

$$m = \max_{w_i \in W} \{1 + \ell_i + 2m_i, \hat{m}_i\}, \quad (1)$$

and

$$M = \begin{cases} m & \text{if } m \text{ is even or } \ell_i = 0 \text{ for all } w_i \in W, \\ m + 1 & \text{otherwise.} \end{cases}$$

We now want to determine the size of (i.e., the number of vertices in) the minimum realization. Note that, by Theorem 4, any realization (G, \mathbf{f}) of Q has $s_{\mathbf{f}}|W|$ vertices. However, $s_{\mathbf{f}}$ is a parameter of G , i.e., not G' .

Lemma 12 *Given a quotient graph $Q = G'/\mathbf{f}' = (W, A)$, every realization of Q has at least $M|W|$ vertices.*

Proof Consider an arbitrary realization (G, \mathbf{f}) with size $M'|W|$, where $G = (V, E)$. Let w_i satisfy $m = \max\{1 + \ell_i + 2m_i, \hat{m}_i\}$, and let ψ be a label preserving isomorphism from G/\mathbf{f} to Q . We have $|V_i| = M'$, where $V_i = \{v \in V \mid \psi(T(v)) = w_i\}$. Since G_{V_i} contains ℓ_i 1-factors, it must have at least $1 + \ell_i$ vertices. Further, since G_{V_i} contains m_i 2-factors, it must have at least $2m_i$ additional vertices. Therefore, we have $M' \geq 1 + \ell_i + 2m_i$. To see that $M' \geq \hat{m}_i$ also holds, pick a particular j satisfying $\hat{m}_i = m_{ij}$. Then, by Lemma 9, $B_{V_i, V_j} = (V_i//V_j, E \cap (V_i \times V_j))$ is a \hat{m}_i -regular bipartite graph, where $V_j = \{v \in V \mid \psi(T(v)) = w_j\}$. We thus have $M' \geq \hat{m}_i$. It follows that $M' \geq m$. Also, if $\ell_i > 0$ for some w_i , M' must be even in order for a 1-factor to exist. Hence $M' \geq M$. \square

The following theorem summarizes some known graph-decomposition results that we find useful in proving Lemma 13.

Theorem 7 [15]

1. Any complete graph K_n of even size n is 1-factorable.
2. Any complete graph K_n of even size n is the sum of a 1-factor and $(n-2)/2$ spanning cycles.
3. Any complete graph K_n of odd size n is the sum of $(n-1)/2$ spanning cycles.
4. Any regular bipartite graph is 1-factorable. \square

Lemma 13 *Given a quotient graph $Q = G'/\mathbf{f}' = (W, A)$, there is a realization of Q which has $M|W|$ vertices.*

Proof We construct a graph $G = (V, E)$ with size $M|W|$ and a local edge labeling \mathbf{f} for G such that G/\mathbf{f} is isomorphic to Q including the edge labels.

Intuitively, we make M copies of each vertex w_i of Q so that it is expanded into a set V_i of vertices of G . Theorem 7 states that the complete graph on M vertices has a sufficient number of vertices and edges to embed the $\{1, 2\}$ -factorable regular graph corresponding to the set of self-loops at any vertex w_i of Q (i.e., mutually edge-disjoint ℓ_i 1-factors and m_i 2-factors). We embed the $\{1, 2\}$ -factorable graph in the complete graph on V_i , and remove the remaining edges. Also, a complete bipartite graph on two sets of M vertices contains mutually edge disjoint \hat{m}_i 1-factors for any i . For each edge of Q between w_i and w_j , we introduce an edge set E' so that $(V_i//V_j, E')$ is m_{ij} -regular bipartite. We give below a more formal construction.

[MIN-LABEL]

Given a quotient graph $Q = G'/\mathbf{f}' = (W, A)$ with $|W| = g$, prepare $V = V_1 \cup V_2 \cup \dots \cup V_g$ as a vertex set of G , where V_i 's are pair-wise disjoint and $|V_i| = M$ for each $i = 1, \dots, g$. For each edge $(w_i, w_j : p, q)$ in Q , create a set of edges and define local edge labeling \mathbf{f} as follows.

Self-loops at w_i : Let $SL \subseteq A$ be the set of self-loops at $w_i \in W$.

(A) First consider the case $\ell_i = 0$, i.e., $|SL| = m_i$. Since $M \geq 1+2m_i$, by Theorem 7 the complete graph K_M on V_i contains at least m_i mutually edge disjoint spanning cycles. (If M is odd, apply Theorem 7 3., and if it is even, apply 2. to $M \geq 2+2m_i$.) For each edge $(w_i, w_i : p, q) \in SL$, we use a distinct spanning cycle $F = (V_i, E_F)$ selected from them, and include E_F in E . Local edge labeling \mathbf{f} for E_F is defined in such a way that labels p and q appear alternately along the cycle. More formally, let e_0, e_1, \dots, e_{M-1} ($e_h = (u_h, u_{h+1}), h = 0, 1, \dots, M-1$) be the spanning cycle, where $u_M = u_0$. Then define $f_{u_h}(u_h, u_{h+1}) = p$ and $f_{u_{h+1}}(u_h, u_{h+1}) = q$ for $h = 0, 1, \dots, M-1$.

(B) Suppose that $\ell_i > 0$. Then M is even by definition. By Theorem 7, in this case, the complete graph K_M on V_i contains a 1-facotor and $(M-2)/2$ spanning cycles, which are mutually edge disjoint. For each edge $(w_i, w_i : p, q) \in SL$ such that $p \neq q$, use a distinct spanning cycle $F = (V_i, E_F)$ selected from the $(M-2)/2$ spanning cycles, and include E_F in E . Then define local edge labeling \mathbf{f} for E_F as in Case (A).

Since $M \geq 1 + \ell_i + 2m_i$, the worst case is $M = 1 + \ell_i + 2m_i$, in which case ℓ_i is odd and $(M-2)/2 = (\ell_i - 1)/2 + m_i$. Since we have used m_i spanning cycles so far, there still remain $(\ell_i - 1)/2$ spanning cycles and one 1-factor, which translate to ℓ_i mutually edge disjoint 1-factors. For each edge $(w_i, w_i : p, p)$, we use a distinct 1-factor $F = (V_i, E_F)$ selected from these 1-factors. (If $\ell_i \geq 2$, then use at least one pair of 1-factors which belong to the same spanning cycle, to make G connected.) Then include E_F in E , and define local edge labeling \mathbf{f} for E_F as follows: For each $e = (u, v) \in E_F$, $f_u(e) = f_v(e) = p$.

Parallel edges (w_i, w_j) : Consider the complete regular bipartite graph K_{V_i, V_j} , and select an arbitrary spanning cycle in it. By Theorem 7 4., the remaining edges can be decomposed into $M - 2$ 1-factors. Decompose the spanning cycle into two 1-factors, F_1 and F_2 . Since $M \geq m_{ij}$, for each edge $(w_i, w_j : p, q)$, we can use a distinct 1-factor $F = (V_i // V_j, E_F)$ from the M 1-factors. Include E_F in E . (If $m_{ij} \geq 2$, use both F_1 and F_2 for some edges to make G connected.) Define a local edge labeling \mathbf{f} for E_F as follows: For each $e = (u, v) \in E_F$, $f_u(e) = p$ and $f_v(e) = q$. [End of MIN-LABEL]

It is easy to check that the graph G constructed by MIN-LABEL contains neither self-loops nor parallel edges. Also, if G is connected, it is easy to show that G/\mathbf{f} is isomorphic to Q including the edge labels. In the rest of the proof, we show that G is always connected.

Clearly, G is connected if $G_i = G_{V_i}$ is connected for some $i = 1, \dots, g$, since each copy of Q is connected. Therefore, suppose that none of $\{G_i \mid i = 1, \dots, g\}$ is not connected. By the construction MIN-LABEL, G_i is clearly connected if $\ell_i \geq 2$ or $m_i \geq 1$. Hence $\ell_i \leq 1$ and $m_i = 0$ for all i . Also, G is connected if $G_{ij} = (V_i // V_j, E \cap (V_i \times V_j))$ is connected for some pair of indices, $i, j = 1, \dots, g$, ($i \neq j$), since Q is connected. By the construction MIN-LABEL again, G_{ij} is connected if $m_{ij} \geq 2$. Hence, $m_{ij} \leq 1$ for all i and j , which implies that $\hat{m}_i = 1$ for every i .

If $\ell_i = 0$ for all i , then $M = 1$ and G is clearly connected. If $\ell_i = 1$ for some i , then $M = 2$ and two vertices in V_i are connected, because of the 1-factor created in G_i ; hence G is connected. \square

Theorem 8 *The minimum realization of $Q = G'/\mathbf{f}' = (W, A)$ has size $M|W|$.*

Proof Follows from Lemmas 12 and 13. \square

Theorem 9 *A graph G is a tree if and only if, for any local edge labeling \mathbf{f} , all realizations of the quotient graph G/\mathbf{f} are isomorphic (including the edge labels), i.e., for any two realizations (G_1, \mathbf{f}_1) and (G_2, \mathbf{f}_2) of G/\mathbf{f} , there is a label-preserving isomorphism.*

Proof *Only if part:* Let G be a tree and \mathbf{f} be a local edge labeling for G . We show that all realizations of the quotient graph G/\mathbf{f} are isomorphic including the edge labels. By Theorem 6, G/\mathbf{f} is a tree, except possibly for one self-loop. The proof of Theorem 6 shows that $s_{\mathbf{f}} \leq 2$ and that $s_{\mathbf{f}} = 2$ if and only if G/\mathbf{f} has a self-loop. First we examine the case $s_{\mathbf{f}} = 1$. In this case, G/\mathbf{f} is clearly isomorphic (including the edge labels) to (G, \mathbf{f}) . Suppose that there is a realization (G', \mathbf{f}') which is not isomorphic to (G, \mathbf{f}) , such that G'/\mathbf{f}' is isomorphic to G/\mathbf{f} . Then $c_{\mathbf{f}'} > 1$, since otherwise (G', \mathbf{f}') , which is isomorphic to G'/\mathbf{f}' , would be isomorphic to G/\mathbf{f} . Thus G'/\mathbf{f}' is not connected (see the proof of Theorem 6); therefore, $(G'/\mathbf{f}', \mathbf{f}')$ cannot be a realization.

If $s_{\mathbf{f}} = 2$, on the other hand, G/\mathbf{f} is a tree except for one self-loop labeled (p, p) for some p . Let (G', \mathbf{f}') be a realization of G/\mathbf{f} . Again, using the proof of Theorem 6, we have $c_{\mathbf{f}'} = 2$. From the structure of G/\mathbf{f} (i.e., a tree except for one self-loop corresponding to a 1-factor), (G', \mathbf{f}') is uniquely determined. Therefore, (G', \mathbf{f}') is isomorphic to G/\mathbf{f} .

If part: Suppose that G is not a tree. We want to show that there are two realizations of G/\mathbf{f} , (G_1, \mathbf{f}_1) and (G_2, \mathbf{f}_2) , that are of different sizes and hence not isomorphic to each other. Let (G_1, \mathbf{f}_1) be the minimum realization of size $M|W|$ known to exist by Theorem 8. Using any even number $M' (> M)$ instead of M in MIN-LABEL, we construct (G_2, \mathbf{f}_2) (which may not be connected) of size $M'|W|$, such that G_2/\mathbf{f}_2 is isomorphic to G/\mathbf{f} . It remains to show that this (G_2, \mathbf{f}_2) is indeed a realization of G/\mathbf{f} , i.e., it is connected.

As in the proof of Lemma 13, we can make G_2 connected when either $\ell_i \geq 2, m_i \geq 1$ or $\hat{m}_i \geq 2$ holds for some i , where ℓ_i, m_i , and \hat{m}_i are the parameters of G/\mathbf{f} . So, we consider the remaining cases, in which $\ell_i \leq 1, m_i = 0$ and $\hat{m}_i \leq 1$ hold. Since G is not a tree, there is a local edge labeling \mathbf{f} such that G/\mathbf{f} does not satisfy the condition of Theorem 6. Thus, G/\mathbf{f} must contain either (1) a cycle, or (2) two or more self-loops (but at most one at a given vertex since $\ell_i \leq 1$).

In case (2), we have $\ell_i = 1$ for some i , hence $M \geq 2$, which implies $M' \geq 4$. Let Q' be the graph obtained from $Q = G/\mathbf{f} = (W, A)$ by removing any two self-loops $e = (w_i, w_i : p, p)$ and $e' = (w_j, w_j : q, q)$, and let (G'_2, \mathbf{f}'_2) be the result of applying MIN-LABEL to Q' . Each vertex set V_k used in MIN-LABEL has size M' . We obtain (G_2, \mathbf{f}_2) from (G'_2, \mathbf{f}'_2) by adding a 1-factor of V_i representing e and a 1-factor of V_j representing e' . By Theorem 7 2., the complete graph on V_i has at least $(M' - 2)/2 \geq 1$ spanning cycles (call one of them C_i). Similarly, the complete graph on V_j has at least one spanning cycle (call it C_j). C_i (C_j) can be decomposed to two 1-factors, of which we use one to represent e (e'). To make the resulting G_2 connected, we need a certain relationship between C_i and C_j . Let U_1, U_2, \dots, U_g be the sets defined for G'_2/\mathbf{f}'_2 by Theorem 4. Without loss of generality, we assume

that each edge of C_i connects a vertex of U_k to a vertex of U_{k+1} , for some $k = 1, 2, \dots, g$, where $U_{g+1} = U_1$. Similarly for C_j . The 1-factor of C_i we use to represent e consists of the edges C_i connecting a vertex in U_k to a vertex in U_{k+1} , for $k = 1, 3, \dots$, while the 1-factor of C_j we use to represent e' consists of the edges of C_j connecting a vertex in U_k to a vertex in U_{k+1} , for $i = 2, 4, \dots$. It is easy to see that the resulting G_2 is connected.

In case (1), let $e(w_i, w_j)$ be an edge on a cycle C in $G/\mathbf{f} = (W, A)$. Apply MIN-LABEL to $(W, A - \{e\})$ to construct (G'_2, \mathbf{f}'_2) of size $M'|W|$. To convert (G'_2, \mathbf{f}'_2) into (G_2, \mathbf{f}_2) , we apply the step for the parallel edges of MIN-LABEL, using a 1-factor in K_{V_i, V_j} . This 1-factor can be so chosen that the resulting G_2 is connected. \square

Corollary 4 *A graph G is a tree if and only if the following condition holds for any labeling \mathbf{f} : For any two realizations (G_1, \mathbf{f}_1) and (G_2, \mathbf{f}_2) of G/\mathbf{f} , $G_1 \simeq G_2$.* \square

Proof The only if part follows from Theorem 9. The if part follows from the if part proof of Theorem 9. \square

It might appear that any non-tree graph G with $\sigma(G) = 1$ would also have the property of Theorem 9, since G/\mathbf{f} has neither self-loops nor parallel edges. However, it turns out that this is not the case. Consider, for example, the graph G and local edge labeling \mathbf{f} for G shown in Figure 2(a). Then $\sigma(G) = 1$, but the graph G' and the local edge labeling \mathbf{f}' for G' shown in Figure 7 form another realization of G/\mathbf{f} ($= G$) that is not isomorphic to G .

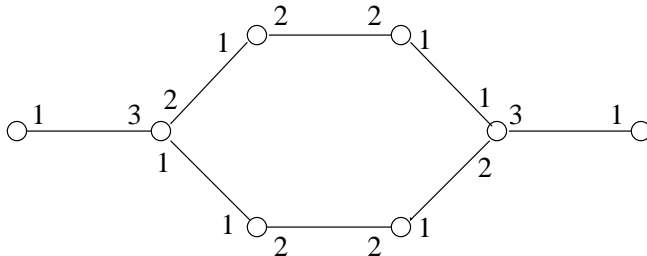


Figure 7: A realization of $G = G/\mathbf{f}$ of Figure 2(a).

6 Networks without Topology Information

In this section, we investigate networks G under the assumption that the processors running on G do not know the topology of G . In Subsection 6.1, we assume that the processors do know the size (i.e., the number of vertices) n of G , while in Subsection 6.2, we investigate model in which the processors only know an upper bound on n . Finally, in Subsection 6.3, computing under *noinfo* is investigated. In Subsections 6.1 and 6.2, the following theorem will play an important role.

Theorem 10 *For any graph G and local edge labeling \mathbf{f} for G , there is an algorithm $A \in \text{ALG}_{\text{upbound}}$ for constructing G/\mathbf{f} .*

Proof Let \bar{n} be an upper bound on the size n of G . A can construct $\mathcal{T}_{\mathbf{f}}^{\bar{n}-1}$ as in the proof of Lemma 6. By Lemma 3, A can thus construct the vertex set $\mathcal{T}_{\mathbf{f}}$ of G/\mathbf{f} . Its edge set can also be constructed from $\mathcal{T}_{\mathbf{f}}^{\bar{n}-1}$. \square

If an upper bound on the network size is known, then, by Theorem 10, we can assume without loss of generality that each processor knows the quotient graph G/\mathbf{f} for the given local edge labeling \mathbf{f} . Thus the difference between results in Section 4 and the corresponding results in subsections 6.1 and 6.2 are due to the difference in the information that the processors are given or can compute; in Section 4, the processors knew G , while in Subsections 6.1 and 6.2, they *can know* only G/\mathbf{f} .

For a problem P and an algorithm A , recall the definition of $N(P, A)$ given in Section 2. Let $I \in \{\textit{topology}, \textit{size}, \textit{upbound}, \textit{noinfo}\}$. In general, an algorithm A for solving problem P may actually solve P *regardless of the local edge labeling* only on a subset of networks in $D_I(P)$. For the other networks in $D_I(P)$, A may either solve P or report that it cannot solve P on the network it is running on with the given local edge labeling. A is said to be *universal* for $D_I(P)$, if $N(P, A) \supseteq D_I(P)$.¹²

6.1 Computing with a Known Number of Processors

6.1.1 ELECT-LEADER, ELECT-EDGE, and SPANNING-TREE

The algorithm A_{EL} (resp. A_{EE}) used in the proof of Lemma 6 (resp. Theorem 2) belongs to $ALG_{\textit{size}}$ ($\supset ALG_{\textit{topology}}$) and solves ELECT-LEADER (resp. ELECT-EDGE) on any graph in $D_{\textit{topology}}(\text{ELECT-LEADER})$ (resp. $D_{\textit{topology}}(\text{ELECT-EDGE})$).¹³ This implies that for $P = \text{ELECT-LEADER}$ and ELECT-EDGE , $D_{\textit{size}}(P) = D_{\textit{topology}}(P)$ and that A_{EL} (resp. A_{EE}) is universal for $D_{\textit{size}}(\text{ELECT-LEADER}) = D_{\textit{topology}}(\text{ELECT-LEADER})$ (resp. $D_{\textit{size}}(\text{ELECT-EDGE}) = D_{\textit{topology}}(\text{ELECT-EDGE})$). It turns out that SPANNING-TREE also shares the same property, as asserted by the next theorem.

Theorem 11 *For problem $P = \text{ELECT-LEADER}$, ELECT-EDGE , and SPANNING-TREE , there is an algorithm $A_P \in ALG_{\textit{size}}$ such that A_P is universal for $D_{\textit{topology}}(P)$.*

Proof Algorithms A_{EL} and A_{EE} provide a proof for $P = \text{ELECT-LEADER}$ and ELECT-EDGE , respectively. Consider the case $P = \text{SPANNING-TREE}$. We give a sketch of an algorithm A_{ST} satisfying the conditions of the theorem. A_{ST} first invokes A_{EE} on the given (G, \mathbf{f}) . When A_{EE} finds that it cannot elect an edge in (G, \mathbf{f}) , A_{ST} generates an output stating that it cannot solve SPANNING-TREE on (G, \mathbf{f}) . Otherwise, A_{EE} elects an edge. Let u and v be the two end vertices of the elected edge. Then on u and v , A_{ST} invokes the single-source-edge spanning tree algorithm SE-ST described in the proof of Theorem 3, and constructs a spanning tree. Note that SE-ST requires no network attribute information. It is clear that A_{ST} satisfies the conditions of the theorem, since $D_{\textit{topology}}(\text{SPANNING-TREE}) = D_{\textit{topology}}(\text{ELECT-EDGE})$ by Theorem 3. \square

Since no algorithm can solve problem P for all possible local labelings on any graph not belonging to $D_{\textit{topology}}(P)$, algorithms A_{EL} , A_{EE} and A_{ST} are the best possible.

¹²As stated in the footnote to Lemma 6, Algorithm A may be able to solve P for a network $G \notin N(P, A)$ for particular (but not for all) local edge labelings. Thus, A may not be able to decide whether the network on which it is running is in $D_I(P)$ or not.

¹³For the characterizations of $D_{\textit{topology}}(\text{ELECT-LEADER})$ and $D_{\textit{topology}}(\text{ELECT-EDGE})$, recall Theorems 1 and 2.

Corollary 5 For any $P \in \{\text{ELECT-LEADER}, \text{ELECT-EDGE}, \text{SPANNING-TREE}\}$, $D_{size}(P) = D_{topology}(P)$. \square

6.1.2 FIND-TOPOLOGY

The equality $D_{size}(P) = D_{topology}(P)$ in Corollary 5 implies that, when size n is known, knowing G or G/\mathbf{f} makes no difference for ELECT-LEADER, ELECT-EDGE, or SPANNING-TREE, as far as their solvable classes are concerned. However, as we see below, it does make an essential difference for FIND-TOPOLOGY.

Lemma 14 If $G \in D_{size}(\text{FIND-TOPOLOGY})$, then for any local edge labeling \mathbf{f} for G the following holds: For any realization (G', \mathbf{f}') of G/\mathbf{f} , if G' and G have the same size then we have $G' \simeq G$.¹⁴

Proof Assume that there is an algorithm $A \in ALG_{size}$ which solves FIND-TOPOLOGY on (G, \mathbf{f}) , where $G \in D_{size}(\text{FIND-TOPOLOGY})$ does not satisfy the condition of the lemma. We derive a contradiction out of this assumption. By Lemma 5, there is a pseudo synchronous algorithm $B \in ALG_{size}$ for solving FIND-TOPOLOGY on (G, \mathbf{f}) such that, in each phase $p+1$ ($p \geq 0$), each processor v sends $T_{\mathbf{f}}^p(v)$, and nothing else, to every one of its neighbors. Let (G', \mathbf{f}') be a realization of G/\mathbf{f} such that G and G' have the same size but $G' \not\simeq G$. Let B run on (G', \mathbf{f}') . Since G/\mathbf{f} is isomorphic to G'/\mathbf{f}' including edge labels, and G and G' have the same size, each processor, running B on (G', \mathbf{f}') , outputs (G, \mathbf{f}) . This is a contradiction, since each processor should either output (G', \mathbf{f}') or report that it cannot solve FIND-TOPOLOGY on the network it is running. \square

The following theorem is FIND-TOPOLOGY's counterpart to Theorem 11. Note that $D_{size}(\text{FIND-TOPOLOGY}) \neq D_{topology}(\text{FIND-TOPOLOGY})$.

Theorem 12 There is an algorithm $A_{TR} \in ALG_{size}$ which is universal for $D_{size}(\text{FIND-TOPOLOGY})$.¹⁵

Proof We give a sketch of an algorithm A_{TR} for solving FIND-TOPOLOGY. Running on a graph G with n vertices with a local edge labeling \mathbf{f} , A_{TR} first computes G/\mathbf{f} . Since A_{TR} knows n and since the number of graphs with n vertices is finite, it can enumerate all the realizations (G', \mathbf{f}') of G/\mathbf{f} such that G' has n vertices and check whether all G' 's are isomorphic. If they are isomorphic then A_{TR} outputs any one of them. Otherwise, based on Lemma 14, its output states that the graph on which it is running is not in $D_{size}(\text{FIND-TOPOLOGY})$. A 's universality, i.e., $N(\text{FIND-TOPOLOGY}, A_{TR}) = D_{size}(\text{FIND-TOPOLOGY})$, follows from Lemma 14. \square

Corollary 6 The condition of Lemma 14 is necessary and sufficient for G to be in $D_{size}(\text{FIND-TOPOLOGY})$. \square

¹⁴That two graphs G' and G without edge labels are isomorphic is denoted by $G' \simeq G$, as was defined in Section 5.

¹⁵For all $G \notin D_{size}(\text{FIND-TOPOLOGY})$, A_{TR} reports that G is not solvable, regardless of the given local edge labeling. This should be contrasted to Theorem 11, recalling the definition of the universality of an algorithm.

Proof Follows from Lemma 14 and Theorem 12. □

Based on the characterization of $D_{size}(\text{FIND-TOPOLOGY})$ given by Corollary 6, the following theorem clarifies the inclusion relationship between $D_{size}(\text{FIND-TOPOLOGY})$ and some other classes.

Theorem 13

1. $D_{topology}(\text{ELECT-LEADER}) \cup \mathcal{S} \subset D_{size}(\text{FIND-TOPOLOGY})$.
2. $D_{topology}(\text{ELECT-EDGE})$ and $D_{size}(\text{FIND-TOPOLOGY})$ are incomparable.

Proof (1.) By Theorem 9 and Corollary 6, we have $\mathcal{S} \subset D_{size}(\text{FIND-TOPOLOGY})$. The fact $D_{topology}(\text{ELECT-LEADER}) \subset D_{size}(\text{FIND-TOPOLOGY})$ follows from Theorem 1, Corollary 6, and from the fact that G is isomorphic to G/\mathbf{f} (including the edge labels) for any \mathbf{f} if $\sigma(G) = 1$.

(2.) Consider the ring R_3 on three vertices. Since $\sigma(R_3) = 3$, we have $R_3 \notin D_{topology}(\text{ELECT-EDGE})$ by Theorem 2. However, $R_3 \in D_{size}(\text{FIND-TOPOLOGY})$ since it satisfies the condition of Lemma 14 (and by Corollary 6).

Consider next the graph G shown in Figure 6(a). We have $G \in D_{topology}(\text{ELECT-EDGE})$ by Theorem 2. However, we can show that $G \notin D_{size}(\text{FIND-TOPOLOGY})$, using Corollary 6. To see this, consider the labeling \mathbf{f} for G shown in Figure 6(a). Figure 6(c) shows G/\mathbf{f} . There is another realization of G/\mathbf{f} , shown in Figure 6(b), which has the same size as G but is not isomorphic to G . □

Theorem 14 Algorithms A_{EL}, A_{EE}, A_{ST} and A_{TR} each have message complexity (i.e., the total number of messages exchanged in the worst case) $O(nm)$, where m is the number of edges.

Proof The majority of messages are needed to construct $T_{\mathbf{f}}^{2(n-1)}$. By Lemma 4, $T_{\mathbf{f}}^{2(n-1)}$ can be constructed with $O(nm)$ messages. □

6.2 Computing with an Upper Bound on the Number of Processors

Given a network G of size n , let \bar{n} be an upper bound on n . If $\bar{n} - n \leq 1$, then knowing \bar{n} is just as good as knowing n by Proposition 3. This is why we defined $N(P, A)$ in Section 2 as follows: For an algorithm $A \in \text{ALG}_{upbound}$, $N(P, A)$ is the set of networks G such that A can solve problem P for G , no matter how large an upper bound \bar{n} on the size n of G is given.

Theorem 15 For $P = \text{ELECT-LEADER}, \text{ELECT-EDGE},$ and SPANNING-TREE , there is an “algorithm” $A_P \in \text{ALG}_{upbound}$ which is “universal” for $D_{topology}(P)$.¹⁶

¹⁶However, A_P may fail to produce a valid output for a network not in $D_{topology}(P)$. Strictly speaking, therefore, an “algorithm” here is not an algorithm in the sense defined in Section 2. Therefore, the term “universal” used in this theorem should also be interpreted *mutatis mutandis*. Therefore, this theorem is not used anywhere else in this paper.

Proof By Corollary 5, it is sufficient to show that, in each case, the exact size n is computable.

Consider first the case $P = \text{ELECT-LEADER}$. Let $G \in D_{\text{topology}}(\text{ELECT-LEADER})$ and let \mathbf{f} be a local edge labeling for G . By Theorem 1, we have $\sigma(G) = 1$. By Theorem 10, each processor can compute G/\mathbf{f} , based on an upper bound \bar{n} on its size, and hence the exact size $n = |G/\mathbf{f}|$.¹⁷

Next, consider the case $P = \text{ELECT-EDGE}$ or SPANNING-TREE . Let $G \in D_{\text{topology}}(\text{ELECT-EDGE}) = D_{\text{topology}}(\text{SPANNING-TREE})$ (see Theorem 3) and let \mathbf{f} be a local edge labeling for G . By Theorem 10, each processor can compute G/\mathbf{f} , based on an upper bound \bar{n} on its size. Note that, by Theorem 2, we have $\sigma(G) \leq 2$, hence $s_{\mathbf{f}} \leq 2$. If $s_{\mathbf{f}} = 1$ then G has the same size as G/\mathbf{f} . If $s_{\mathbf{f}} = 2$, on the other hand, then G has exactly twice the size of G/\mathbf{f} .

The remaining problem is how to determine $s_{\mathbf{f}}$. We claim that $s_{\mathbf{f}} = 1$ if and only if G/\mathbf{f} has no self-loops. By definition, the only if part is clear. To prove the if part, assume that $s_{\mathbf{f}} = 2$. Then $\sigma(G) = 2$, and by Theorem 2, there are adjacent vertices u and v such that $T(u) \equiv T(v)$, which implies that there is a self-loop in G/\mathbf{f} . \square

Theorem 16 *Let \mathcal{S} denote the set of all trees. We have:*

1. $D_{\text{upbound}}(\text{FIND-TOPOLOGY}) = \mathcal{S}$,
2. $D_{\text{upbound}}(\text{ELECT-EDGE}) = D_{\text{upbound}}(\text{SPANNING-TREE}) = \mathcal{S}$, and
3. $D_{\text{upbound}}(\text{ELECT-LEADER}) = D_{\text{topology}}(\text{ELECT-LEADER}) \cap \mathcal{S}$.

Proof By Theorems 10 and 6, there is an algorithm in ALG_{upbound} that can decide if the network with a local edge labeling, (G, \mathbf{f}) , on which it is running is a tree and determine $s_{\mathbf{f}}$ (1 or 2 by Corollary 2) or the number n of vertices if it is a tree. Once it is discovered that G is a tree, it is easy to solve any of the four problems using G/\mathbf{f} . (As for ELECT-LEADER , if $s_{\mathbf{f}} = 2$, the algorithm reports that it cannot be solved for the tree.) It follows that \mathcal{S} is contained in $D_{\text{upbound}}(\text{FIND-TOPOLOGY})$, $D_{\text{upbound}}(\text{ELECT-EDGE})$ and $D_{\text{upbound}}(\text{SPANNING-TREE})$, and that $D_{\text{topology}}(\text{ELECT-LEADER}) \cap \mathcal{S}$ is contained in $D_{\text{upbound}}(\text{ELECT-LEADER})$, since $G \in D_{\text{topology}}(\text{ELECT-LEADER})$ if and only if $\sigma(G) = 1$ by Theorem 1.

To prove that these inclusion relationships are really equalities, consider any network $G \notin \mathcal{S}$. By Theorem 9 and MIN-LABEL in the proof of Lemma 13, there exists a local edge labeling \mathbf{f} for G such that, for any sufficiently large even number k , there is a realization (G', \mathbf{f}') with $c_{\mathbf{f}'} = k$. Note that G/\mathbf{f} is isomorphic to G'/\mathbf{f}' including edge labels, but $G' \not\cong G$ in general.

(I) To show that $G \notin D_{\text{upbound}}(\text{FIND-TOPOLOGY})$ for $\forall G \notin \mathcal{S}$, we assume that there is an algorithm $A \in ALG_{\text{upbound}}$ such that $G \in N(\text{FIND-TOPOLOGY}, A)$, and derive a contradiction. Let B be a pseudo synchronous algorithm for solving FIND-TOPOLOGY on G such that, in each phase $p+1$ ($p \geq 0$), each processor v sends $T_{\mathbf{f}}^p(v)$, and nothing else, to every one of its neighbors. Compare the computation of B on G with \mathbf{f} and G' with \mathbf{f}' , assuming that a sufficiently large upper bound \bar{n} is given as input data. Since $\mathcal{T}_{\mathbf{f}} = \mathcal{T}_{\mathbf{f}'}$,

¹⁷Here, the processor is working on the assumption that $\sigma(G) = 1$, which is not valid, in general. See the previous footnote.

any two vertices having similar views, u in G and v in G' , output the same answer, a contradiction. It follows that $D_{upbound}(\text{FIND-TOPOLOGY}) \subseteq \mathcal{S}$, which proves (1.).

(II) To show that $G \notin D_{upbound}(\text{ELECT-EDGE})$ for $\forall G \notin \mathcal{S}$, assume that there is an algorithm $A \in ALG_{upbound}$ such that $G \in N(\text{ELECT-EDGE}, A)$. By choosing $k \geq 3$, we can make $\sigma(G') \geq 3$, which means $G' \notin D_{topology}(\text{ELECT-EDGE})$ by Theorem 2. Let B be a pseudo synchronous version of A (see Lemma 5) such that, in each phase $p+1$ ($p \geq 0$), each processor v sends $T_{\mathbf{f}}^p(v)$, and nothing else, to every one of its neighbors. By the same argument as in (I) above, B can solve ELECT-EDGE for G' , a contradiction since $G' \notin D_{topology}(\text{ELECT-EDGE})$.

The fact that no non-tree network is in $D_{upbound}(\text{SPANNING-TREE})$ can be shown by a similar argument.

(III) The proof for (3.) is analogous to (2.), except that $D_{upbound}(\text{ELECT-LEADER}) \subseteq D_{topology}(\text{ELECT-LEADER})$ by Proposition 1. This additional condition is necessary because \mathcal{S} is not contained in $D_{topology}(\text{ELECT-LEADER})$. \square

Theorem 17 *For any problem $P \in \mathcal{P}$ there is an algorithm A with message complexity $O(\bar{n}m)$ such that $N(P, A) = D_{upbound}(P)$, where \bar{n} is the given upper bound on n .*

Proof Analogous to the proof of Theorem 14. (Replace n by \bar{n} .) \square

6.3 Computing without Network Attribute Information

Theorem 18 *For any problem $P \in \mathcal{P}$, we have*

$$D_{noinfo}(P) = D_{upbound}(P).$$

Proof For any graph $G \in D_{upbound}(P) (\subseteq \mathcal{S})$, we construct an algorithm $A_G \in ALG_{noinfo}$ such that each processor on G' solves P if and only if G' is isomorphic to G ; hence $G \in N(P, A_G)$. All A_G has to do is to recognize if network G' is isomorphic to G (without any attribute information on G'), since P is solvable on G if G is known (because $G \in D_{upbound}(P) \subseteq D_{topology}(P)$ for $\forall P \in \mathcal{P}$). Let G have n vertices.

To explain the main idea behind A_G , let us first reexamine view $T_{\mathbf{f}}(v)$, where v is a vertex of G' and \mathbf{f} is a local edge labeling for G' . It consists of all paths in G' originating at v . Each such path is infinitely long, and in general visits a vertex many times. Consider now the longest finite prefix of such a path, that is simple, i.e., does not visit any vertex more than once. (If G is a tree, in particular, then each such prefix ends at a leaf.) Moreover, the graph induced by the union of all such prefixes is exactly G' . Once G' is constructed, all that remains is to test if G' is isomorphic to G .

Running on G' , A_G first constructs $T_{\mathbf{f}}^n(v)$ on each processor v , where n is the size of G . Note that what is stated in the previous paragraph is still valid if $T_{\mathbf{f}}(v)$ is replaced by $T_{\mathbf{f}}^n(v)$, except that each path in $T_{\mathbf{f}}^n(v)$ has finite length n . We now describe an easy way to determine the longest simple prefix of each path in $T_{\mathbf{f}}^n(v)$. Let x and z be the parent and a child nodes of a node y in $T_{\mathbf{f}}^n(v)$, respectively. Observe that $\bar{x} = \bar{z}$ if and only if edges (x, y) and (y, z) are labeled by (p, q) and (q, p) , respectively, for some p and q . Therefore, to determine the longest simple prefix of a path in $T_{\mathbf{f}}^n(v)$, we truncate the path at the

first edge (y, z) on it satisfying the condition above for the parent node x of y . Call the resulting tree T^* . $T^* \simeq G'$ if T^* is “small” enough, or to be more precise, if T^* does not have a leaf vertex at depth n . If T^* has a leaf vertex at depth n , then A_G reports that it gives up solving P for the network. Otherwise, since $T^* \simeq G'$, it can invoke an appropriate algorithm (e.g., A_{EL}) in $ALG_{topology}$ described in Section 4. (A_G can simply output T^* if $P = \text{FIND-TOPOLOGY}$.) \square

The above theorem says that the upper bound information has no benefit, as far as the solvable classes are concerned. However, Theorem 19 below shows its effect on the existence of universal algorithms. To see the contrast more clearly, we first summarize what we already know about the cases where we do have some attribute information.

Corollary 7 *For any problem $P \in \mathcal{P}$ and any I in $\{\text{topology, size, upbound}\}$, there is a universal algorithm $A_P \in ALG_I$ for $D_I(P)$.* \square

Theorem 19 *For any problem $P \in \mathcal{P}$, there is no universal algorithm $A \in ALG_{noinfo}$ for $D_{noinfo}(P)$ ($= D_{upbound}(P)$).¹⁸*

Proof Let R_3 be the ring on three vertices and let \mathbf{f} a local edge labeling for R_3 such that each edge is labeled by $(1, 2)$. Consider any algorithm $A \in ALG_{noinfo}$ for solving P . By Proposition 4, we can assume without loss of generality that A is pseudo synchronous. Since R_3 is not a tree, by Theorem 16, no algorithm in $ALG_{upbound}$, *a fortiori*, not A , can solve P on R_3 . Therefore, the execution of A on R_3 must terminate in the k th phase, for some integer k , announcing its inability to solve P on R_3 . Lemma 5 implies that the execution of A on any processor v in any network G with any local edge labeling \mathbf{g} would produce the same output as that on processor u in R_3 , if $T_{\mathbf{f}}^k(u)$ on R_3 is similar to $T_{\mathbf{g}}^k(v)$ on G . Let L be a “chain” graph with $2k+3$ vertices, i.e., $L = (V, E)$, where $V = \{v_1, v_2, \dots, v_{2k+3}\}$ and $E = \{\{v_i, v_{i+1}\} \mid i = 1, \dots, 2k+2\}$. Consider the labeling \mathbf{g} for L such that all the edges are labeled by $(1, 2)$ except for edge $\{v_{2k+2}, v_{2k+3}\}$, which has label $(1, 1)$. Clearly, $T_{\mathbf{f}}^k(u)$ on R_3 is similar to $T_{\mathbf{g}}^k(v_k)$ on G . Thus $L \notin N(P, A)$, although $L \in D_{upbound}(P)$ by Theorems 1 and 16. It follows that A is not universal. \square

Figure 8 illustrates the inclusion relationships among $D_{topology}(P)$, $D_{size}(P)$, $D_{upbound}(P)$ and $D_{noinfo}(P)$, where $P \in \mathcal{P}$.

7 Other Related Results

7.1 The Link Information

In Section 6, we investigated networks under the assumption that each processor knows the size n or an upper bound on n . We may also consider the case where each processor knows the number of links m or an upper bound on m , although this assumption may be a little artificial.

¹⁸Therefore, for any algorithm $A \in ALG_{noinfo}$ for P , there is a network G such that P is not solvable on G by A , but is solvable by some other algorithm $A' \in ALG_{noinfo}$.

Lemma 15 *Given a graph G with a local edge labeling \mathbf{f} , let $G/\mathbf{f} = (W, F)$. If G/\mathbf{f} has s self-loops with the same label p at both ends, then*

$$m = s_{\mathbf{f}}(|F| - s/2).$$

Proof In Lemma 9, we considered three types of edges in F . For any edge e of type (1.), we showed there that $s_{\mathbf{f}} = |V_1| = 2|\tau_{\mathbf{f}}^{-1}(e)|$, while for any edge e of type (2.) or (3.), $s_{\mathbf{f}} = |\tau_{\mathbf{f}}^{-1}(e)|$. Therefore, an edge of type (1.) in G/\mathbf{f} corresponds to $s_{\mathbf{f}}/2$ edges in G , while an edge of type (2.) or (3.) in G/\mathbf{f} corresponds to $s_{\mathbf{f}}$ edges in G . \square

For each problem $P \in \mathcal{P}$ let $D_{Esize}(P)$ denote the set of networks for which P is solvable using m as an input. If each processor knows the number of vertices n , then it can construct G/\mathbf{f} (see Theorem 10) and compute m by Lemma 15. Conversely, if it knows m , then it can construct G/\mathbf{f} by Theorem 10, since $m+1$ is an upper bound on n if G is connected. It can then compute $s_{\mathbf{f}}$ by Lemma 15 and obtain $n = s_{\mathbf{f}}|W|$. We therefore have the following theorem.

Theorem 20 *For each problem $P \in \mathcal{P}$, we have*

$$D_{Esize}(P) = D_{size}(P).$$

\square

Next, consider the case where the processors know an upper bound on m . Let $D_{Eupbound}(P)$ denote the set of networks for which P is solvable using an upper bound on m as input. The proof of the following corollary is analogous to the of the above theorem.

Corollary 8 *For each problem $P \in \mathcal{P}$, we have*

$$D_{Eupbound}(P) = D_{upbound}(P).$$

\square

7.2 Automorphisms

From the definition of symmetricity, it appears that there might be a close relation between the number of different automorphisms and the symmetricity of a graph (e.g., the larger the number of automorphisms, the larger the symmetricity would be, or there might be an automorphism if the symmetricity is greater than 1, and so on). Unfortunately, however, due to the following two properties, it turns out that there is no direct relation between the two quantities. A complete bipartite graph on two sets V_1 and V_2 , where $|V_1| = 1$, is called a *star*. An edge in a graph is called a *bridge* if its removal disconnects the graph.

Proposition 5

1. *Any star with at least 3 vertices has symmetricity 1; the number of automorphisms, however, increases with the number of vertices.*

2. The symmetricity of any 3-regular bridge-less graph coincides with its size; however, there is a 3-regular bridge-less graph having no non-trivial automorphism.

Proof 1. is easy to show. 2. follows from Theorem 5, since any 3-regular bridge-less graph is $\{1,2\}$ -factorable [29]. \square

As for fixed-point-free automorphisms, we have the following result.

Theorem 21 *Let G be a graph with $\sigma(G) = 2$. Then, there is a fixed-point-free automorphism on G .*

Proof Let $G = (V, E)$ and consider two distinct vertices $u, v \in V$ such that $T_{\mathbf{f}}(u) \equiv T_{\mathbf{f}}(v)$ for some local edge labeling \mathbf{f} . Define a function $\psi : V \rightarrow V$ as follows. Let $w \in V$ and x be a node of $T_{\mathbf{f}}(u)$ such that $w = \bar{x}$. Then

$$\psi(w) = \bar{y},$$

where node y satisfies $L(T(u), x) = L(T(v), y)$.¹⁹ Since G is connected, ψ is defined for every w in V . First, we show that ψ is a fixed-point-free function. Let x_1 and x_2 be any two nodes in $T(u)$ such that $w = \bar{x}_1 = \bar{x}_2$. Define y_1 and y_2 by $L(T(u), x_1) = L(T(v), y_1)$ and $L(T(u), x_2) = L(T(v), y_2)$, respectively. Since $T(u) \equiv T(v)$ and $\bar{x}_1 = \bar{x}_2$, the subtrees of $T(v)$ rooted at y_1 and y_2 are similar. On the other hand, arguing as in the proof of Lemma 1, we can show that $\bar{y}_1 \neq w$ and $\bar{y}_2 \neq w$ since $u \neq v$. Hence, \bar{y}_1 and \bar{y}_2 must be the same since $\sigma(G) = 2$. Namely, ψ is a function and $\psi(w) \neq w$.

Next, we show that ψ is an automorphism. Let $e = (w_1, w_2) \in E$. Then, there are adjacent nodes x_1 and x_2 in $T(u)$ such that $\bar{x}_1 = w_1$ and $\bar{x}_2 = w_2$. Define two nodes y_1 and y_2 in $T(v)$ by $L(T(u), x_1) = L(T(v), y_1)$ and $L(T(u), x_2) = L(T(v), y_2)$, respectively. Then $(\psi(w_1), \psi(w_2)) = (\bar{y}_1, \bar{y}_2) \in E$, since y_1 and y_2 are adjacent. \square

7.3 Coverings

A graph K is said to be a *covering* of another graph G if there is a way to label the vertices of K with the names of vertices of G in such a way that, for each vertex u of K labeled v , the set of the labels of u 's neighbors in K coincides with the set of v 's neighbors in G . G and H are said to have a *finite common covering* if and only if there exists a graph K that is a covering of both G and H .

Let G/\mathbf{f} and H/\mathbf{g} be two quotient graphs. If G/\mathbf{f} is isomorphic to H/\mathbf{g} including edge labels, then G and H have isomorphic universal coverings, where a *universal covering* of G is a (possibly infinite) tree which is a covering of G .

Theorem 22 [2, 23] *Two graphs G and H have a finite common covering if and only if they have isomorphic universal coverings.* \square

Using the above theorem, a sufficient condition for a graph to be in D_{size} (FIND-TOPOLOGY), in terms of a common covering, can be derived.

¹⁹ $L(T(u), x)$ was defined in Section 3.

Theorem 23 *Let G be a graph. If there is no graph H with the same size as G such that G and H have a finite common covering, then $G \in D_{size}(\text{FIND-TOPOLOGY})$.*

Proof We give a sketch of an algorithm A that solves `FIND-TOPOLOGY` for graphs satisfying the condition of theorem. When A is executed on a network G of size n under a local edge labeling \mathbf{f} , it constructs the quotient graph G/\mathbf{f} first, and then computes all pairs of a graph H of size n and a local edge labeling \mathbf{g} such that H/\mathbf{g} is isomorphic to G/\mathbf{f} including edge labels. If all H 's are isomorphic, then A outputs H , otherwise, it gives up solving the problem.

To show the correctness of A , suppose that A outputs a graph H of size n , which is not isomorphic to G . Since H/\mathbf{g} is isomorphic to G/\mathbf{f} including edge labels, H and G have isomorphic universal coverings, which implies that H and G have a finite common covering by Theorem 22, a contradiction. \square

Although the above sufficient condition is easier to understand than that of Lemma 14, unfortunately, it is not necessary. Any two r -regular graphs have a finite common covering since their universal coverings are isomorphic. Hence most of the regular graphs do not satisfy the sufficient condition; however, there are likely to be many regular graphs with symmetricity 1, which belong to $D_{size}(\text{FIND-TOPOLOGY})$.

Angluin [1] investigates a kind of topology recognition problem and shows that two graphs having a finite common covering are indistinguishable (Theorem 5.6 in [1]). This might appear to imply that the above sufficient condition would also be necessary. This is due to the difference between the `FIND-TOPOLOGY` and the recognition problem she investigates. Namely, her model assumes a more powerful communication mechanism than ours and the “coin tossing” facility which is not available in our model.

G can be thought of as a kind of covering of G/\mathbf{f} . As we have investigated in detail, the relation between G and G/\mathbf{f} can be described in terms of p -factors. Thus some relation would exist between coverings and factors.

8 Discussions

`ELECT-LEADER` has been investigated by many researchers, mostly on networks with unique identity numbers. Rosenstiehl, Fiksel and Holliger showed that some problems are not solvable on symmetric graphs [31]. Borge has investigated “symmetry” and “genericity” of distributed systems and derived some necessary or sufficient conditions for `ELECT-LEADER` to be solvable using these concepts [7, 8]. His model is based on *CSP* [16] and is closely related to Angluin’s. In his model, each process is aware of its (unique) name and, can use it in equality testing. As a tool to break symmetry, Itah and Rodeh use a probabilistic method [18], which is studied extensively for solving not only `ELECT-LEADER` but also other problems investigated in this paper (see, e.g., [22, 24, 33]).

We have considered only connected networks. Johnson and Schneider [17] investigate a collection of disjoint networks. After extending the definition of `ELECT-LEADER` (“selection problem” in their terminology) to a collection of disjoint networks, they show that there is a similarity labeling algorithm for such a collection and, therefore, the extended `ELECT-LEADER` is decidable.

Let us extend the other three problems, ELECT-EDGE, SPANNING-TREE and FIND-TOPOLOGY, in the same way, and let \mathcal{P}' denote the set of extended ELECT-LEADER, ELECT-EDGE, SPANNING-TREE and FIND-TOPOLOGY. Thus, for each $P \in \mathcal{P}'$, a solution must be found for each connected component of the network. We now briefly examine these extensions.

Under *topology*, for any problem $P \in \mathcal{P}'$, each processor can decide whether P is solvable for each connected component, using the topology information it has. Then, using A_{EL} (see the proof of Lemma 6), A_{EE} (see the proof of Theorem 2), or A_{ST} (see the proof of Theorem 11), each processor can solve P for the connected component to which it belongs. Solving FIND-TOPOLOGY is trivial. Thus we can conclude the following.

For each problem $P \in \mathcal{P}'$, there is a distributed algorithm in $ALG_{topology}$ which decides whether P is solvable for any given network and solves P if it is solvable.

□

Under *size* or *upbound*, a processor cannot, in general, know about the components to which it does not belong. Therefore, there can be no distributed algorithm for deciding whether P is solvable for a given network.

The problem of recognizing topology has often been investigated as a problem for automata. Wu and Rosenfeld [34] and Rosenfeld [30] looked at the problem using several kinds of cellular graph automata, mainly under the assumption that there is a “leader” among the automata. Shank [32], Mylopolous [26] and Blum and Sakoda [6] also studied models of automata which work on graphs.

Another important problem about anonymous networks is that of computing functions on them. Attiya, Snir and Warmuth [4], Attiya and Snir [3], Moran and Warmuth [25], and Duris and Galil [12] all have recently investigated this problem for anonymous rings. They have characterized the class of functions computable on a ring and showed that these functions are computable with $O(n^2)$ messages, and that any non-constant function computable on a ring has bit complexity $\Omega(n \log n)$ and message complexity $\Omega(n \log^* n)$, where n is the number of processors on the ring.

Yamashita and Kameda [35] have investigated the function computation problem for general anonymous networks. They characterized the class of functions computable on a graph and showed that these functions are computable with $O(mn^2)$ messages, where m is the number of edges of the graph. Norris’ result [27] enables us to reduce it to $O(mn)$. Kranakis, Krizanc, and van den Berg [20] present an algorithm whose bit complexity is $O(D\Delta^2 n^2 \log^2 n)$, where D and Δ are the diameter and the maximum degree of the graph, respectively. As for lower bounds, Yamashita and Kameda [35] present a procedure for deriving a lower bound on the message complexity. For other subclasses of anonymous networks than rings, see, e.g., [5, 19].

Finally, from the fault tolerance’s stand point, the reader may be interested in distributed algorithms which work correctly even if the uniqueness of processor identity numbers is violated. The anonymous network can be considered as a model of the worst case where the uniqueness is completely violated. In most practical situations, only a small number of processors may have the same identity number as a result of some failure or design error, and algorithms for solving different problems could take advantage of the (non-unique) identity numbers. Such algorithms are considered in [37].

Acknowledgements

The authors would like to thank three anonymous referees for their helpful suggestions. Thanks are also due to Prof. P. Hell of Simon Fraser University for contributing Theorem 5 and for extensive and inspiring discussions. The first author would like to thank Professors T. Ae of Hiroshima University, Japan and T. Ibaraki of Kyoto University, Japan, for giving him an opportunity to visit Simon Fraser University.

Appendix

Proof of Lemma 10

This appendix contains the proof of Lemma 10. The proof requires the construction of a local edge labeling \mathbf{f} for G such that $s_{\mathbf{f}} = n/g$, given a graph G of size n satisfying the F-condition with g . Given a graph G which satisfies the F-condition with some g , we first present a procedure for constructing a local edge labeling \mathbf{f} with $s_{\mathbf{f}} = n/g$. Let $\{V_1, \dots, V_g\}$ be a partition of V with which G satisfies the F-condition in Definition 4. Intuitively, we construct \mathbf{f} in such a way that V_1, \dots, V_g are similarity classes induced by \mathbf{f} .

Before presenting our labeling procedure ST-LABEL, we first define three *standard labelings*.

1. Let $F = (V, E')$ be a 1-factor. A labeling for F is called the *standard labeling for F by label p* if, for each edge $e = (u, v)$ in E' , $f_u(e) = f_v(e) = p$.
2. Let $F = (V, E')$ be a 2-factor. A labeling for F is called the *standard labeling for F by labels p and q ($p \neq q$)* if the edges $e_0 = (u_0, u_1), e_1 = (u_1, u_2), \dots, e_h = (u_h, u_0)$ in each cycle of the 2-factor are labeled as p, q, p, q, \dots along the cycle, i.e., for $i = 0, 1, \dots, h$, $f_{u_i}(e_i) = p$ and $f_{u_{i+1}}(e_i) = q$, where $u_{h+1} = u_0$.
3. Let $F = (V_1 // V_2, E')$ be a 1-regular bipartite graph. A labeling for F is called the *standard labeling for F by labels p and q* (p and q may be the same) if for any edge $e = (u, v) \in E' \subseteq V_1 \times V_2$, $f_u(e) = p$ and $f_v(e) = q$.

For $i = 1, 2, \dots, g$, since G_{V_i} is $\{1,2\}$ -factorable, let the 1-factors, $F_1^i, \dots, F_{\ell_i}^i$, and 2-factors, $\overline{F}_1^i, \dots, \overline{F}_{m_i}^i$, constitute a $\{1,2\}$ -factorization of G_{V_i} . Since $B_{V_i, V_j} = (V_i // V_j, E \cap (V_i \times V_j))$, $i \neq j$, is a regular bipartite graph, it is 1-factorable [14, 15]. Let B_{V_i, V_j} be m_{ij} -regular, and let the 1-factors, $F_1^{ij}, F_2^{ij}, \dots, F_{m_{ij}}^{ij}$, constitute a 1-factorization of B_{V_i, V_j} . Note that the degree of any vertex u in V_i is $\sum_{j=1}^g m_{ij}$, where $m_{ii} = \ell_i + 2m_i$. Vertex u is a part of each 2-factor, and therefore two ports of u “belong” to each 2-factor. This accounts for $2m_i$ in the expression for m_{ii} given above.

We now present a Standard-Labeling procedure, ST-LABEL, which constructs a local edge labeling for any given graph G satisfying the F-condition with g . (The labeling in Figure 5(a) was obtained by ST-LABEL, starting with graph G without labeling.)

[ST-LABEL]

For each $i = 1, 2, \dots, g$, carry out steps (1) and (2):

- (1) Label the edges in $E \cap (V_i \times V_i)$ as follows.

- For each $j = 1, 2, \dots, \ell_i$, carry out the standard labeling for F_j^i by label j .
- For each $j = 1, 2, \dots, m_i$, carry out the standard labeling for \overline{F}_j^i by labels j' and j'' , where $j' = j + \ell_i$ and $j'' = j + \ell_i + m_i$.

[So far $\ell_i + 2m_i$ port numbers have been allocated to each vertex in V_i .]

(2) For each $j = 1, 2, \dots, g$, $j \neq i$, label the edges in $E \cap (V_i \times V_j)$ as follows. For each $d = 1, \dots, m_{ij}$, carry out the standard labeling for F_d^{ij} by labels p and q , where

- $p = m_{ii} + \sum_{h=1, h \neq i}^{j-1} m_{ih} + d$,
- $q = m_{jj} + \sum_{h=1, h \neq j}^{i-1} m_{hj} + d$,
- $m_{ii} = \ell_i + 2m_i$, and
- $m_{jj} = \ell_j + 2m_j$.

[End of ST-LABEL]

In step (2) of ST-LABEL, m_{ij} consecutive port numbers are allocated to the ports of $u \in V_i$ on the edges which join u and the vertices in V_j .

Lemma 16 *Procedure ST-LABEL correctly constructs a local edge labeling \mathbf{f} such that each V_i is contained in a similarity class under \equiv .*

Proof We observe that

- (1) for any $u \in V$, all port numbers at u are distinct, i.e., $f_u(e) \neq f_u(e')$ for any two links e and e' incident to u , and
- (2) for any two vertices u and u' in any V_i , if $f_u(e) = p$ and $f_{u'}(e) = q$ for an edge $e = (u, v)$ incident to u , then there exists another edge $e' = (u', v')$ such that $f_{u'}(e') = p$ and $f_{v'}(e') = q$, and that v and v' are both in the same partition V_j for some j .

It follows from (2) that $T(u) \equiv T(v)$ holds for any two vertices u and v in V_i , i.e., u and v belong to the same similarity class under \equiv . \square

Note that Lemma 16 does not rule out the possibility that two distinct sets V_i and V_j belong to the same similarity class.

[Proof of Lemma 10]

Let G satisfy the F-condition with g , with a partition of V , $\{V_1, \dots, V_g\}$. Then by Lemma 16, the local edge labeling \mathbf{f} for G constructed by ST-LABEL for the partition $\{V_1, \dots, V_g\}$ has the property that, for any $i = 1, \dots, g$, V_i is contained in a similarity class under \equiv , which implies $s_{\mathbf{f}} \geq n/g$. \square

References

- [1] D. Angluin, “Local and global properties in networks of processors,” *Proc. 12th ACM Symp. Theory of Computing*, pp. 82–93, 1980.
- [2] D. Angluin and A. Gardiner, “Finite common covering of pairs of regular graphs,” *Journal of Combinatorial Theory, Ser. B* 30, pp. 184–187, 1981.

- [3] H. Attiya and M. Snir, “Better computing on the anonymous ring,” *Journal of Algorithms*, vol.12, pp. 204–238, 1991.
- [4] H. Attiya, M. Snir, and M.K. Warmuth, “Computing on the anonymous ring,” *Journal of ACM*, vol. 35, no. 4, pp. 845–875, 1988.
- [5] P.W. Beame and H.L. Bodlaender, “Distributed computing on transitive networks: The torus,” *Proc. 6th Symp. Theoretical Aspects of Computer Science, Lecture Notes in Computer Science*, vol. 349, Springer-Verlag, Berlin, Germany, pp. 294–303, 1989.
- [6] M. Blum and W. Sakoda, “On the capability of automata in 2 and 3 dimensional space,” *Proc. 18th IEEE Symp. Foundations of Computer Science*, Providence, RI, pp. 147–161, 1977.
- [7] L. Borge, “Symmetry and genericity for CSP distributed systems,” *Tech. Report 85–32*, Laboratory of Information Theory and Programming, University of Paris, Paris, 1985.
- [8] L. Borge, “On the existence of symmetric algorithms to find leaders in network of communicating sequential processes,” *Tech. Report 86–18*, Laboratory of Information Theory and Programming, University of Paris, Paris, 1986.
- [9] E. Chang, “Echo algorithms: Depth parallel operations on general graphs,” *IEEE Trans. Software Engineering*, vol. 8, no. 4, pp. 391–401, 1982.
- [10] E. Chang and R. Roberts, “An improved algorithm for decentralized extrema-finding in circular configurations of processes,” *Communications of ACM*, vol. 22, no. 5, pp. 281–283, 1979.
- [11] E.W. Dijkstra and C.S. Sholten, “Termination detection in diffusing computations,” *Information Processing Letters*, vol. 11, no. 1, pp. 1–4, 1980.
- [12] P. Duris and Z. Galil, “Two lower bounds in asynchronous distributed computation,” *Journal of Computer and System Sciences*, vol. 42, pp. 245–266, 1991.
- [13] G.N. Frederickson and N. Lynch, “Electing a leader in a synchronized ring,” *Journal of ACM*, vol. 34, no. 1, pp. 98–115, 1987.
- [14] P. Hall, “On representatives of subsets,” *Journal of London Mathematics Society*, vol. 10, pp. 26–30, 1935.
- [15] A.F. Harary, *Graph Theory*. Reading, MA: Addison–Wesley, 1969.
- [16] C.A.R. Hoare, “Communicating sequential processes,” *Communications of ACM*, vol. 21, no. 8, pp. 666–677, 1978.
- [17] R.E. Johnson and F.B. Schneider, “Symmetry and similarity in distributed systems,” *Proc. 4th ACM Symp. Principles of Distributed Computing*, Minaki, Ontario, pp. 13–22, 1985.
- [18] A. Itah and M. Rodeh, “The lord of the ring, or probabilistic methods for breaking symmetry in distributed networks,” *Tech. Report RJ 3110*, IBM, Yorktown Heights, New York, 1981.

- [19] E. Kranakis and D. Krizanc, “Computing boolean functions on anonymous hypercube networks (Extended Abstract),” *Tech. Report CS-R9040*, Center for Mathematics and Computer Science, The Netherlands, 1990.
- [20] E. Kranakis, D. Krizanc, and J. van den Berg, “Computing boolean functions on anonymous networks,” *Proc. 17th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science*, vol. 443, Springer–Verlag, Berlin, Germany, pp. 254–267, 1990.
- [21] Lamport, L., and Lynch, N.A., “Distributed computing: Models and methods” in *Hand Book of Theoretical Computer Science, volume B: Formal Models and Semantics (J. van Leeuwen, Ed.)* Cambridge, MA: The MIT Press, pp. 1157–1200, 1990.
- [22] I. Lavallée and C. Lavault, “Spanning tree construction for nameless networks,” *Proc. 4th International Workshop on Distributed Algorithms, Lecture Notes in Computer Science*, vol. 486, Springer–Verlag, Berlin, Germany, pp. 41–56, 1990.
- [23] F.T. Leighton, “Finite common coverings of graphs,” *Journal of Combinatorial Theory*, Ser. B 33, pp. 231–238, 1982.
- [24] Y. Matias and Y. Afek, “Simple and efficient election algorithms for anonymous networks,” *Proc. 3rd. International Workshop on Distributed Algorithms, Lecture Notes in Computer Science*, vol. 392, Springer–Verlag, Berlin, Germany, pp. 183–194, 1989.
- [25] S. Moran and M.K. Warmuth, “Gap theorems for distributed computation,” *SIAM Journal on Computing*, vol. 22, no. 2, pp. 379–394, 1993.
- [26] J. Mylopoulos, “On the relation of graph grammars and graph automata,” *Proc. 13th IEEE Symp. Switching and Automata Theory*, College Park, Md, pp. 108–120, 1972.
- [27] N. Norris, “Universal covers of edge–labeled digraphs: Isomorphism to depth $n-1$ implies isomorphism to all depths,” *Discrete Applied Math.*, To appear.
- [28] J. Pachl, E. Korach, and D. Rotem, “Lower bounds for distributed maximum–finding algorithms,” *Journal of ACM*, vol. 31, no. 4, pp. 905–918, 1984.
- [29] J. Petersen, “Die Theorie der regulären Graphs,” *Acta Mathematica*, vol. 15, pp. 193–220, 1891.
- [30] A. Rosenfeld, “Networks of automata: some applications,” *IEEE Trans. on Systems, Man and Cybernetics*, vol. SMC–5, no. 3, pp. 380–383, 1975.
- [31] P. Rosenstiehl, J.R. Fiksel, and A. Holliger, “Intelligent graphs,” in *Graph Theory and Computing* (R. Read, Ed.). New York, NY: Academic Press, pp. 219–265, 1982.
- [32] H.S. Shank, “Graph property recognition machines,” *Mathematical Systems Theory*, vol. 5, pp. 45–49, 1971.
- [33] B. Schieber and M. Snir, “Calling names on nameless networks,” *Proc. 8th ACM Symp. Principles of Distributed Computing*, Edmonton, Alberta, Canada, pp. 319–328, 1989.

- [34] A. Wu and A. Rosenfeld, “Cellular graph automata, I and II,” *Information and Control*, vol. 42, pp. 305–353, 1979.
- [35] M. Yamashita and T. Kameda, “Computing functions on an anonymous network,” *Mathematical Systems Theory*, To appear.
- [36] M. Yamashita and T. Kameda, “Computing on anonymous networks,” *Proc. 7th ACM Symp. Principles of Distributed Computing*, Toronto, Ontario, Canada, pp. 117–131, 1988.
- [37] M. Yamashita and T. Kameda, “Electing a leader when processor identity numbers are not distinct,” *Proc. 3rd International Workshop on Distributed Algorithms, Lecture Notes in Computer Science*, vol. 392, Springer–Verlag, Berlin, Germany, pp. 303–314, 1989.

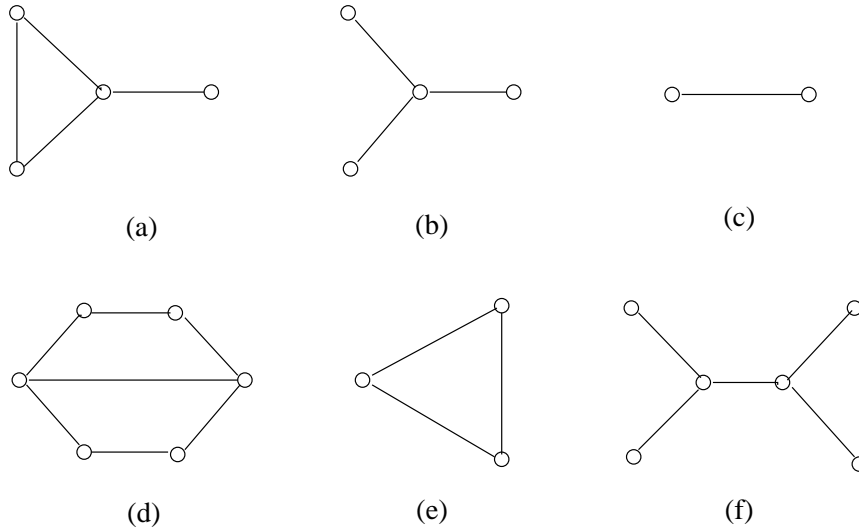
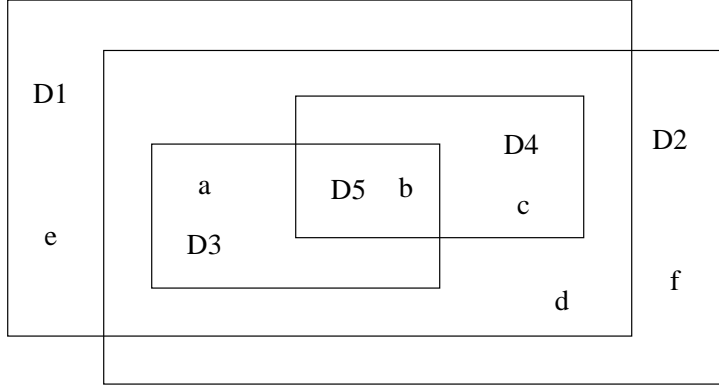


Figure 8: Inclusion relationships among $D_{topology}(P)$'s, $D_{size}(P)$'s, $D_{upbound}(P)$'s and $D_{noinfo}(P)$'s. $D_1 = D_{size}(\text{FIND-TOPOLOGY})$. $D_2 = D_I(P)$ for any $I \in \{topology, size\}$ and $P \in \{\text{ELECT-EDGE}, \text{SPANNING-TREE}\}$. $D_3 = D_I(\text{ELECT-LEADER})$ for any $I \in \{topology, size\}$. $D_4 = D_I(P)$ for any $I \in \{upbound, noinfo\}$ and $P \in \{\text{ELECT-EDGE}, \text{SPANNING-TREE}, \text{FIND-TOPOLOGY}\}$. $D_5 = D_I(\text{ELECT-LEADER})$ for any $I \in \{upbound, noinfo\}$.