Proof Labeling Schemes[*]

Amos Korman [†] Shay Kutten[‡] David Peleg[§]

April 13, 2007

Abstract

This paper addresses the problem of locally verifying global properties. Several natural questions are studied, such as "how expensive is local verification?" and more specifically "how expensive is local verification compared to computation?" A suitable model is introduced in which these questions are studied in terms of the number of bits a node needs to communicate. In addition, approaches are presented for the efficient construction of schemes, and upper and lower bounds are established on the cost of schemes for multiple basic problems. The paper also studies the role and cost of unique identities in terms of impossibility and complexity.

Previous studies on related questions deal with distributed algorithms that simultaneously compute a configuration and verify that this configuration has a certain desired property. It turns out that this combined approach enables verification to be less costly, since the configuration is typically generated so as to be easily verifiable. In contrast, our approach separates the configuration design from the verification. That is, it first generates the desired configuration without bothering with the need to verify, and then handles the task of constructing a suitable verification scheme. Our approach thus allows for a more modular design of algorithms, and has the potential to aid in verifying properties even when the original design of the structures for maintaining them was done without verification in mind.

[*]A preliminary version of this paper appeared in Proc. of ACM PODC 2005.

[†]Information Systems Group, Faculty of IE&M, The Technion, Haifa, 32000 Israel. E-mail: `pandit@tx.technion.ac.il`. Supported in part at the Technion by an Aly Kaufman fellowship.

[‡]Information Systems Group, Faculty of IE&M, The Technion, Haifa, 32000 Israel. E-mail: `kutten@ie.technion.ac.il`. Supported in part by a grant from the Israel Science Foundation.

[§]Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel. E-mail: `david.peleg@weizmann.ac.il`. Supported in part by a grant from the Israel Science Foundation.

Keywords: distributed networks, proof labels, property verification, self stabilization.

1 Introduction

This paper addresses the problem of locally verifying global properties. This task complements the task of locally computing global functions. Since many functions cannot be computed locally [1, 24, 22], local verification may potentially be even more useful than local computing - one can compute globally and verify locally. We address the natural question of how expensive is local verification. More specifically, we deal with the question of how expensive local verification is in comparison to the computation itself. In terms of both sequential time and distributed communication time, there exists evidence that verification is sometimes easier. For example, verifying that a given color assignment on a given graph is a legal 3 coloring is believed to consume much less time than computing a 3 coloring [21]. In the context of distributed tasks, other measures of complexity are often used, for example, the amount of communication needed. Still, one can ask a similar natural question. Assume that we are given a *distributed representation* of a solution for a problem (for example, each node knows its color but not necessarily the colors of others). It is required to verify the legality of the represented solution (in the example, to verify that the coloring stored by the nodes is a legal 3 coloring). Does the verification consume fewer communication bits than the computation of the solution (e.g., the 3 coloring) itself?

This paper investigates these questions in terms of the number of bits of information that a node needs to convey to its neighbors concerning its state. (It is assumed that local computation is free, although all our schemes use polynomial time for the verification). In particular, we show that the cost of verification is sometimes rather high even for problems that appear simple. On the other hand, for certain other problems the cost of verification is only constant, even for problems that may look complex.

The formal definitions are given in Section 2.1. Informally, we assume that the *state* of every node has already been computed by some algorithm (in the above example, the state may consist of a color). The configuration (formed as the collection of states of all nodes) is supposed to satisfy some predicate (e.g., "the colors of neighboring nodes are different"). Ideally, the number of information bits a node conveys to its neighbors is as small as possible, even smaller than its state. We refer to these information bits as the *proof label* (or simply the *label*) of the node and study the task of computing short labels.

To perform the verification, a node computes some *local* predicate, considering only its own state, as well as the labels of its neighbors (but not their states). In the above example, the local predicate of each node is that the color of the node is different than those of its neighbors. The global configuration predicate (e.g., "the coloring is legal") is implied by the conjunction of the local predicates. One may ask what is the minimum size of a label of a node in such a scheme.

In addition to the theoretical appeal of such problems, one practical justification comes from the area of self stabilization. There, the network should converge to a *legal* configuration (one satisfying the configuration predicate) from *any* initial configuration. One approach is to partition a given self stabilization task into two parts: the first is the detection that the current configuration is illegal (this is enabled by the task we call here "verification", or "proof"). The second part is the computation of a new legal configuration. Again, a natural question is to ask which of these problems is more expensive.

Numerous papers dealt with the task of designing algorithms to compute a legal configuration. A much smaller set of papers dealt with both verifying configurations and with reaching legal ones, and they addressed a very limited set of problems (most notably spanning trees and the problem of a general compiler to transform an algorithm to be self-stabilizing). In this paper we concentrate on the verification part alone, study some of its basic properties, point at impossibilities, develop a wide collection of natural building blocks and problems, and establish some complementary lower bounds.

We note the following major difference between our model and the ones used by the above self stabilization algorithms. There, the design of the computation stage was intertwined with that of the verification stage, and the designers sought to design a computation process that will be easy for verification, and vice versa. This approach may lead to a low cost local verification. However, this approach might also have the disadvantage of making the design process less modular. To simplify the design of algorithms, it is desirable to address these needs separately. In this paper, we assume that the distributed representation of the structure or function at hand is already given, and the labeling we compute is required to verify this specific representation. This allows for more modular algorithm design and frees the algorithm designer to consider other goals when designing the distributed representation. Our approach may sometimes be useful also in verifying properties on existing structures, even when the original design of those structures was done without verification in mind.

To illustrate this difference between the models, let us point out to one of our results, which states that local checking sometimes requires labels that are longer even than the states (such as the states used in previous local checking methods). This occurs in the natural setting where vertices are required to have distinct states. For example, this can happen in an algorithm that hashes unique identities of nodes into shorter unique states. In the case where the underlying graph is an n-vertex path, the size of vertex labels that are required in order to verify that all the states are unique is $\Omega(n)$. This is longer than the state, which is $O(\log n)$. On the other hand, were we allowed to compute the states (rather than prove the given hashing), labels of size zero would have sufficed in the case of unique identities: just have the state equal the identity. (Since the identities are assumed in this example to be unique, the states "computed" in that way are unique too.) We note that in many other cases, "small" labeling schemes exist even for our stronger requirements from a scheme.

Related Work The measure of the label size is related to the problem of the communication complexity [25]. Some of our results are for impossibility of tasks in anonymous networks. Results of that nature concerning computation (rather than verification) were presented in [16] and follow up papers. Some constructions we use simulate a distributed algorithm on every node; a related operation was used in [8]. Self stabilization was introduced in [17]. Self stabilization by local detection, and by the similar variant local checking was introduced in [9, 12, 10, 11]. These papers, as well as many others, e.g. [18, 19, 7, 5, 6]) present self stabilizing algorithms for computing trees using local detection.

Memory efficient distributed representation of a tree (in which, for example, each vertex holds a pointer to its parent) may allow the distributed system to be in an undetected "illegal" configuration (namely, the states of the vertices at a given time may constitute a representation of a structure which is not a tree). For instance, the structure may be "illegal" since the collection of pointers forms a cycle or a collection of cycles, rather than a tree. To overcome such illegal configurations, the algorithms mentioned above were based on adding a variable to each vertex, containing the distance of the vertex from the root of the tree. (The resulting representation, containing both the pointer and the distance variable, is clearly *redundant*). To verify that the distributed redundant representation indeed forms a tree, each vertex compares its own distance variable with the distance variable of its parent. It is easy to see that if the collection of pointers (one per node) induces a cycle, then there must exist a vertex whose distance is not larger than that of its parent. In the example, validity is verified by checking the correctness of a global predicate based on the configuration. The configuration predicate is that the collection of pointers forms a tree. This is based, in turn, on verifying a local predicate at each vertex, saying that if the vertex points at a parent, then the distance variable of the vertex is larger by one than the distance variable of the parent.

In [13], lower bounds for silent stabilization are given. Informally, an algorithm achieves silent stabilization if after stabilization no value is changed in any variable, so the only activity is verifying that the states of the neighbors are the same as the vertex "remembers" them. In a sense, in a state of silent stabilization the vertices must be able to verify that the system is indeed stabilized, and no further changes in the variables are necessary. Hence, the lower bounds for silent stabilization may sometimes translate to lower bounds for proof labeling scheme. This is not always true though, in particular, one can construct proof labeling schemes for problems for which no silent stabilization is possible. In addition, lower bounds in our model do not necessarily imply lower bounds in the model of [13]. Our typical lower bound is for the task of verifying *any* configuration. In contrast, in [13] it is implicitly assumed that the configuration is chosen expressly in such a way that it will have a short label. Another difference from [13] is that in [13], each node had a different labels for each one of its links and every neighbor can 'see' only the labels prepared for it. While here, each node has a single label which can be 'viewed' by all of its neighbors.

There are also some other differences between the models. In [13, 12, 11] it is assumed that a vertex can read a state of a link port of a neighboring vertex. We assume that all the neighbors of a vertex v can see the same *label* of v. One can say that this models a local broadcast media (e.g., radio) versus point to point lines. The more abstract motivation is that our modeling is intended to capture the total number of information bits that need to be conveyed by a node for checking purposes, disregarding the issue of which neighbor they are to be communicated to. Consequently, constructing positive results in our model may be harder than in the point to point model. For example, in the latter model a vertex can easily choose to communicate with just one neighbor. In our model, in contrast, all the neighbors see the label, hence it is harder to tell which of them is the target of the communication. Nevertheless, we show some schemes for problems that seem to need a targeted communication. Still, the size of these schemes is much smaller than a size of the identity of the neighbor.

In [14] it is assumed that a node can read the *output* of near-by nodes. That is, only the part of the state meant to be visible to the outside can be read by other nodes. (The output is the part that appears in the specification of the task to be performed.) As opposed to that, in the current paper the labels often contain information that is not intended as output, and does not appear in the specification of the task. In some sense, this is necessary for efficient solutions, since in [14] it is shown that with their assumption, a very large memory is sometimes needed (e.g. for verifying a spanning tree).

Our Results This paper models and partially answers the question: "how easy is the checking task by itself", with respect to basic building blocks in distributed systems. A suitable model is introduced in Section 2.1 in which this question is studied in terms of the number of bits a node needs to reveal to its neighbors. In Section 2.2 we illustrate our model by presenting some basic proof labeling schemes. In particular, we show a proof labeling scheme for verifying a representation of a spanning tree with size $\Theta(\log n)$, where n in the number of nodes in the graph. We also present a non-trivial constant size proof labeling scheme (even though our model is weaker than previous models as explained above) and show that for any m there is a problem with proof size $\Theta(m)$.

In Section 3 we study factors that affect the cost of checking. We study the role of unique identities in terms of impossibility and complexity. We show that there exist problems and graph families for which no labeling proof scheme exists if no unique identities are assumed. On the other hand, we show a case (specifically, a path of n vertices) in which the transition from anonymous networks to id-based is possible, yet, in order to verify that any such translation is valid, the maximum number of bits used in a label must be $\Omega(n)$. It is interesting to note that even if unique identities are assumed, checking whether the states are also unique can still be as costly as $\Omega(n)$. This is the case when the states are generated by an algorithm that is independent of the labeling scheme. It seems interesting to recall that if the states are assigned in advance in such a way that it is easy to check them, then the size of the labeling scheme is just $O(\log n)$.

Additional evidence to the importance of the role played by identities in proof labeling schemes is provided by a result we present regarding identity invariability. The question under study follows from a result of [1], showing in a particular setting that, intuitively, the actual value of the identities does not matter. More specifically, the result of [1] deals with functions that could be *computed* locally by a vertex, just by looking at states of the neighboring vertices. It is shown therein that if there exists an algorithm to compute a certain function, then there exists an *identities order-invariant* algorithm, with the same complexity. *Order-invariant* algorithms only look at the relative *order* of the identities (i.e., "which identity is higher") rather than at their actual value. Our setting bears a lot of resemblance to that of [1]. Nevertheless, we show that this phenomena does not exist in our model.

In Section 4 we show how to build verification systems systematically and modularly by suggesting two general paradigms for constructing proof labeling schemes. The first is derived by imitating the execution of a distributed algorithm and the second is derived by a modular construction approach based on the notion of composition. This leads to rather efficient schemes for verifying representations of Minimum Spanning Tree, Maximum Matching, $s-t$ Vertex Connectivity and other basic problems. Coming back to the hardness of checking versus the hardness of computing, we show that small proof labeling schemes exist even for NP Hard problems.

2 Definitions and basic examples

2.1 Definitions

We consider distributed systems that are represented by connected graphs. The vertices of the graph $G = \langle V, E \rangle$ correspond to the nodes in the system, and we use the words "vertex" and "node" interchangeably. The edges of G correspond to the links, and we use the words "link" and "edge" interchangeably. Denote $n = |V|$. Every node v has internal ports, each corresponding to one of the edges attached to v. The ports are numbered from 1 to $deg(v)$ (the degree of v) by an internal numbering known only to node v. If G is undirected, then for every vertex v let $N(v)$ denote the set of edges adjacent to v. If G is directed, then for any vertex v let $N(v)$ denote the set of edges outgoing from v. In either case, for every vertex v let $n(v) = |N(v)|$. Unless mentioned otherwise, all graphs considered are undirected. For two vertices u and v in G, let $d_G(u, v)$ denote the unweighted distance between u and v.

Given a vertex v, let s_v denote the state of v and let $v_s = (v, s_v)$. A *configuration graph* corresponding to a graph $G = \langle V, E \rangle$ is a graph $G_s = \langle V_s, E_s \rangle$, where $V_s = \{v_s \mid v \in V\}$ and $(v_s, u_s) \in E_s$ iff $(v, u) \in E$. A *family of configuration graphs* $\mathcal{F}_s$ corresponding to graph family $\mathcal{F}$ consists of configuration graphs $G_s \in \mathcal{F}_s$ for each $G \in \mathcal{F}$. Let $\mathcal{F}_S$ be the largest possible such family when every state s is taken from a given set S. Unless mentioned otherwise, let S denote the set of integers. We sometimes refer to each state s_v of a configuration graph as having two fields: $s_v = (id(v), s'(v))$. Field $id(v)$ is v's *identity* and is encoded using $O(\log n)$ bits. When the context is clear we may refer to $s'(v)$ as the state of v (instead of to $s(v)$). A configuration graph G_s is *id-based* if for every pair of vertices v and u it is given that $id(u) \neq id(v)$. A graph whose identities are arbitrary (including possibly the case where all identities are the same) is termed *anonymous*. An id-based (respectively, anonymous) family is a family of id-based (respectively, anonymous) graphs. Let $\mathcal{F}^{all}$ be the collection of all directed strongly-connected and all undirected connected graphs with $O(n)$ vertices. Let $\mathcal{F}^{undirected}$ be the collection of all undirected connected graphs with $O(n)$ vertices. When it is clear from the context, we use the term "graph" instead of "configuration graph", "id-based graph" or "anonymous graph". We may also use the notation v instead of v_s. Given a family of configuration graphs $\mathcal{F}_s$, let $\mathcal{F}_s(W)$ denote the family of all graphs in $\mathcal{F}_s$ such that, when considered as weighted, the (integral) weight of each edge is bounded from above by W.

Many of our results deal with a distributed representation of subgraphs. Such a representation is encoded in the collection of the nodes' states. There can be many such representations. For simplicity, we focus on the case that an edge is included in the subgraph if it is explicitly pointed at by the state of an endpoint. That is, given a configuration graph G_s, the subgraph (respectively, directed subgraph) induced by the states of G_s, denoted $H(G_s)$ (respectively, $D(G_s)$), is defined as follows. For every vertex $v \in G$, if s_v includes an encoding of one of v's ports pointing to a vertex u, then the edge (respectively, directed edge) (v, u) is an edge in the subgraph. These are the only edges in the subgraph.

Consider a graph G. A distributed problem *Prob* is the task of selecting a state s_v for each vertex v, such that G_s satisfies a given predicate f_{Prob}. This induces the problem *Prob* on a graph family $\mathcal{F}$ in the natural way. We say that f_{Prob} is the *characteristic function* of *Prob* over $\mathcal{F}$.

This paper deals with adding labels to configuration graphs in order to maintain a (locally checkable) distributed proof that the given configuration graph satisfies a given predicate f_{Prob}. Informally, a proof labeling scheme includes a *marker* algorithm M that generates a label for every node, and a *decoder* algorithm that compares labels of neighboring nodes. If a configuration graph satisfies f_{Prob}, then the decoder at every two neighboring nodes declares their labels (produced by marker M) "consistent" with each other. However, if the configuration graph does *not* satisfy f_{Prob}, then for *any possible* marker, the decoder must declare "inconsistencies" between some neighboring nodes in the labels produced by the marker. It is not required that the marker be distributed. However, the decoder is distributed and *local*, i.e., every node can check only the labels of its neighbors (and its own label and state).

More formally, A *marker* algorithm L is an algorithm that given a graph $G_s \in \mathcal{F}_s$, assigns a label $L(v_s)$ to each vertex $v_s \in G_s$. For a marker algorithm L and a vertex $v_s \in G_s$, let $N'_L(v)$ be a set of $n(v)$ fields, one field per neighbor. Each field $e = (v, u)$ in $N'_L(v)$, corresponding to edge $e \in N(v)$, contains the following.

- The port number of e in v.
- The weight of e (if G is unweighted we regard each edge as having weight 1)
- $L(u)$.

Let $N_L(v) = \langle (s_v, L(v)), N'_L(v) \rangle$. Informally, $N'_L(v)$ contains the labels given to all of v's neighbors along with the port number and the weights of the edges connecting v to them. $N_L(v)$ contains v's state and label as well as $N'_L(v)$. A *decoder* algorithm $\mathcal{D}$ is an algorithm which is applied separately at each vertex $v \in G$. When $\mathcal{D}$ is applied at vertex v, its input is $N_L(v)$ and its output, $\mathcal{D}(v, L)$, is boolean.

Let f be some characteristic function of a problem over $\mathcal{F}$. Let $\mathcal{F}_s$ be some family of configuration graphs corresponding to some family $\mathcal{F}$. A *proof labeling scheme* $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ for $\mathcal{F}_s$ and f is composed of a *marker* algorithm $\mathcal{M}$ and a *decoder* algorithm $\mathcal{D}$, such that the following two properties hold.

1. For every $G_s \in \mathcal{F}_s$, if $f(G_s) = 1$ then $\mathcal{D}(v, \mathcal{M}) = 1$ for every vertex $v \in G$.
2. For every $G_s \in \mathcal{F}_s$, if $f(G_s) = 0$ then for every marker algorithm L there exists a vertex $v \in G$ so that $\mathcal{D}(v, L) = 0$.

We note that all the proof labeling schemes constructed in this paper use a polytime decoder algorithm. The *size* of a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ is the maximum number of bits in the label $\mathcal{M}(v_s)$ over all $v_s \in G_s$ and all $G_s \in \mathcal{F}_s$. For a family $\mathcal{F}_s$ and a function f, we say that the *proof size* of $\mathcal{F}_s$ and f is the smallest size of any proof labeling scheme for $\mathcal{F}_s$ and f.

2.2 Basic examples

To illustrate the definitions, we now present basic proof labeling schemes for some id-based and anonymous families. Note that every proof labeling scheme that applies to anonymous families applies also to the corresponding id-based families. The converse in not always true, as shown later. We give examples for problems with different proof sizes. We also show that for any m there is a problem with proof size $\Theta(m)$.

Our first example concerns agreement among all vertices. Note that v's neighbors cannot 'see' the state of v but they can see v's label.

Agreement in anonymous families Problem: Assign all the nodes identical states. Let $S = \{1, 2, \cdots, 2^m\}$.

Lemma 2.1 *The proof size of $\mathcal{F}_S^{all}$ and $f_{Agreement}$ is $\Theta(m)$.*

Proof: We first describe a trivial proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ of the desired size m. Given G_s such that $f_{Agreement}(G_s) = 1$, for every vertex v, let $\mathcal{M}(v) = s_v$. I.e., we just copy the state of node v into its label. Then, $\mathcal{D}(v, L)$ simply verifies that $L(v) = s_v$ and that $L(v) = L(u)$ for every neighbor u of node v. It is clear that π is a correct proof labeling scheme for $\mathcal{F}_S^{all}$ and $f_{Agreement}$ of size m. We now show that the above bound is tight up to a multiplicative constant factor even assuming that $\mathcal{F}_S^{all}$ is id-based. Consider the connected graph G with two vertices v and u. Assume, by way of contradiction, that there is a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ for F_S^{all} and $f_{Agreement}$ of size less than $m/2$. For $i \in S$, let G_s^i be G modified so that both u and v have state $s(u) = s(v) = i$. Obviously, $f_{Agreement}(G_s^i) = 1$ for every i. For a vertex x, let $\mathcal{M}^i(x)$ be the label given to x by marker $\mathcal{M}$ applied on G_s^i. Let $L^i = (\mathcal{M}^i(v), \mathcal{M}^i(u))$. Since the number of bits in L^i is assumed to be less than m, there exist $i, j \in S$ such that $i < j$ and $L^i = L^j$. Let G_s be G modified so that $s_u = i$ and $s_v = j$. Let L be the marker algorithm for G_s in which $L(u) = \mathcal{M}^i(u)$ and $L(v) = \mathcal{M}^j(v)$. Then for each vertex x, $\mathcal{D}(x, L) = 1$, contradicting the fact that $f(G_s) = 0$. ▮

Note that the corresponding computation task, that of assigning every node the same state, requires only states of size 1.

By the above lemma, it is clear that for any m there exists a family $\mathcal{F}_s$ and a function f with proof size $\Theta(m)$. We now present a somehow stronger claim, namely, that a similar result exists also for *graph problems* (namely, problems where the input is only the graph topology).

Corollary 2.2 *For every function $1 \leq m < n^2$, there exists a graph problem on an id-based family with proof size $\Theta(m)$.*

Proof Sketch: Let $\mathcal{F}^m$ be the family of n-node graphs with fixed identities $v_1, v_2, \cdots, v_n$ such that each graph $G \in \mathcal{F}^m$ consists of the following three subgraphs. The first subgraph is a fixed path P containing $v_{n'}, v_{n'+1}, \cdots, v_n$ where $n' = \sqrt{m}+1$ and v_n is an end node of P. Note that $G \backslash P$ contains $\sqrt{m}$ vertices. The other two subgraphs in G, namely G_1 and G_2, are arbitrary connected graphs whose vertex sets are $v_1, v_2, \cdots, v_{\sqrt{m}/2}$ and $v_{\sqrt{m}/2+1}, v_{\sqrt{m}/2+1}, \cdots, v_{\sqrt{m}}$ respectively. Note that both G_1 and G_2 contain $\sqrt{m}/2$ vertices. Graph G contains no edge between G_1 and G_2. The only edges in G are the edges of the subgraphs P,G_1 and G_2 and the edges (v_1, v_n) and $(v_n, v_{\sqrt{m}/2+1})$. The graph problem $f_{Isomorphic}$ characterizes the question of whether the transformation $g(v_i) = v_{\sqrt{m}/2+i}$ maps G_1 to G_2. Let us now briefly describe the proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ for $\mathcal{F}^m$ and $f_{Isomorphic}$. Given a graph $G \in \mathcal{F}^m$ such that $f_{Isomorphic}(G) = 1$, for each i, the label $\mathcal{M}(v_i)$ consists of the index i together with the encoding of both subgraphs G_1 and G_2. The decoder operates similarly to the decoder described in Lemma 3.2. Informally, the decoder at each node v first verifies that v and its neighbors are consistent with v's label, therefore verifying that the encodings of G_1 and G_2 are correct (note that path P is fixed). Then, by looking at v's label, the decoder verifies that the transformation $g(v_i) = v_{\sqrt{m}/2+i}$ maps G_1 to G_2. Clearly, the label size of π is $O(m)$. The lower bound proof uses similar arguments to the ones described in the proof of

Lemma 2.1 together with the fact that there are $2^{\Omega(m)}$ subgraphs G_1 and G_2. ▮

Problems on trees in id-based families: The following example concerns the representation of various spanning trees in the system. The upper bound employs structures and ideas similar to the ones previously used in [8, 6, 7, 9, 19, 5], taking into account the differences between the models and overcoming some technicalities arising in our model. For the lower bounds, we had to establish a proof also for trees and forests that are not spanning.

We consider five different problems, obtained by assigning states to the nodes of G so that $H(G_s)$ (respectively, $D(G_s)$) is a (respectively, directed)

(1) forest;

(2) spanning forest;

(3) tree;

(4) spanning tree;

(5) BFS tree of G (for some root vertex r).

Let $f_{No-cycles}$ (respectively, $f'_{No-cycles}$) be the characteristic function of either one of the five problems above.

Lemma 2.3 *The proof size of $\mathcal{F}_S^{all}$ and $f_{No-cycles}$ (respectively, $f'_{No-cycles}$) is $\Theta(\log n)$.*

Proof: For proving the upper bound, construct the proof labeling scheme $\pi_{span} = \langle \mathcal{M}_{span}, \mathcal{D}_{span} \rangle$ for $\mathcal{F}_S$ and f being "$H(G_s)$ is a spanning tree". The other cases are constructed in a similar manner. Given G_s so that $f(G_s) = 1$, the marker algorithm $\mathcal{M}_{span}$ operates as follows. If $H = H(G_s)$ is a spanning tree, then it has $n-1$ edges. Therefore, either there is only one vertex r in G_s whose state is not an encoding of one of its port numbers or there exist exactly two vertices whose states point at each other. In the second case let r be the vertex with the smaller identity among the two and in both cases r is considered as the root. Note that the state of each non-root vertex points at its parent in the rooted tree (H, r). Let $\mathcal{M}_{span}(v) = \langle id(r), d_H(v, r) \rangle$. For a vertex v_s and a marker algorithm L, the first field $L(v)$ is denoted by $L_1(v)$ and the second by $L_2(v)$. The decoder $\mathcal{D}_{span}(v, L) = 1$ iff all the following easy to verify events occur.

1. For every neighbor u of v, $L_1(u) = L_1(v) \in S$. I.e., all vertices agree on the identity of the root.

2. If $id(v) = L_1(v)$ then $L_2(v) = 0$.

3. If $id(v)$ is not $L_1(v)$ then s_v is an encoding of a port number of v leading to a vertex u such that $L_2(v) = 1 + L_2(u)$.

4. If $L_2(v) = 0$ then either s_v is not an encoding of a port of v or an encoding of a port of v leading to vertex u and $L_2(u) = 1$.

Obviously the size of π_{span} is $O(\log n)$ so we only need to prove that the scheme is correct. Given G_s so that $f(G_s) = 1$. W show that $D_{span}(v, L) = 1$ for for all $u, v \in V$. The first fields of $\mathcal{M}_{span}(u)$ and $\mathcal{M}_{span}(v)$ are the same since they are both the identity of the root r. If $v \neq r$ then s_v is the identity of v's parent in the tree H, therefore $dist_H(v, r) = 1 + dist_H(s_v, r)$. Also, (2) above holds for r. Hence, $\mathcal{D}(v, \mathcal{M}_{span}) = 1$ for each vertex $v \in G$.

If, for some marker algorithm L ,$\mathcal{D}(v, L) = 1$ for every vertex v, then by (1), all vertices must agree in the first field of their label. Denote this value x. Since the identities of the vertices are disjoint, there can be at most one vertex r satisfying $id(r) = x$. Also, by (3), such a vertex must exist. By (3), for every vertex u such that $id(u) \neq x$ corresponds a directed edge leading to some vertex w and $L(u) - 1 = L(w)$. Therefore all directed paths must reach the special vertex r (satisfying $id(r) = x$). Therefore the edges corresponding to all vertices but r, form a spanning tree T and the only case to be inspected is whether the edge that correspond to r (if this edge exists), belongs to this tree. This is verified by (4). The upper bound for the case of a spanning tree follows.

In the case were f (respectively, f') is a "(respectively, directed) BFS tree", the decoder $\mathcal{D}(v, L)$ also checks that $|L_2(u) - L_2(v)| \leq 1$ for each (respectively, directed) neighbor u of vertex v.

Remark: a similar approach applies also to BFS trees on weighted id-based graphs except that the size of the scheme changes to $O(\log n + \log W)$. Note that in the above schemes if the decoder satisfies $\mathcal{D}(v, L) = 1$ for every v then $L_2(v) = d_G(v, r)$. Therefore, using this scheme we can also prove that each vertex holds its distance to the root.

Let us next prove the lower bound (the proof is essentially the same for all five problems). Let P be the horizontal path of n vertices. For the sake of analysis only, enumerate the vertices of P from left to right, i.e., $P = (1, 2, \cdots, n)$. For $i < n$, let s_i be the port number of the edge leading from vertex i to $i+1$. Obviously, $f(P_s) = 1$ and $f'(P_s) = 1$. Assume, by way of contradiction, that there exists a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ for $\mathcal{F}_s$ and either f or f' which is of size less than $\log(n/2) - 2$. Let $L(i)$ be the label given by $\mathcal{M}$ to vertex i in the above path P_s. Since the number of bits in each $L(i)$ is less than $\log(n/2) - 2$, there exist two pairs of vertices $(i, i+1)$ and $(j, j+1)$

where $1 < i$ and $i+1 < j < n-1$ so that $L(i) = L(j) = L'$ and $L(i+1) = L(j+1) = L''$. We now build the following ring R consisting of $j-i$ vertices whose identities are clockwise ordered from i to $j-1$. For $i \leq k < j-1$ let s_k be the port number of vertex k leading from k to $k+1$ and let s_{j-1} be the port leading from $j-1$ to i. Let us give R_s the same labeling L as $\mathcal{M}$ gives P_s, i.e., each vertex $i \leq k < j$ in R_s is labeled $L(k)$. By the correctness of π on P_s we get that for each vertex $v \in R_s$, $\mathcal{D}(v, L) = 1$. This is a contradiction to the fact that $f(R_s) = 0$ and $f'(R_s) = 0$. ▮

Note that the proof applies to all the cases in the lemma, including the case that a (not necessarily spanning) subgraph does not have a cycle.

Actually, the lower bounds of Lemma 2.3 can be established even for more restricted families. We note that in [13] a lower bound of $\Omega(\log n)$ is shown for the case of spanning trees. It is possible that one may be able to generalize this lower bound to the other cases of Lemma 2.3 and to translate it to our model.

The similarity between the proof method of Lemma 2.3 and previous results for spanning trees may be due to the fact that the corresponding computation task (that of assigning states such that $H(G_s)$ is a spanning tree of G) is not easier than the verification task. That is, the computation also needs to use $\Omega(\log n)$-bit states. Note, however, that as shown in [15], one can choose the states so that the *average* number of bits in a state is $O(\log\log n)$.

Orientation in anonymous trees Problem: We now present a natural problem with a constant proof size whose corresponding computation task requires $\Omega(\log n)$-bit states. Given a tree T, f_{Orient} characterizes the task of assigning states to the nodes of T so that $D(G_s)$ induces an orientation on the edges (towards some root that is not given explicitly). Let $\mathcal{F}^{anon-trees}$ be the family of anonymous trees.

Lemma 2.4 *The proof size of $\mathcal{F}_S^{anon-trees}$ and f_{Orient} is $O(1)$.*

Proof: We describe a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ as required. Let G_s be an anonymous tree s.t. $f_{Orient}(G_s) = 1$. Let r be the unique vertex whose state doesn't encode one of its port numbers. The marker algorithm $\mathcal{M}$ labels each vertex v by its distance from r calculated modulo 3. The decoder returns $\mathcal{D}(v, L) = 1$ iff the following two conditions hold for every neighbor u of v.

1. $|L(u) - L(v)| = 1$.
2. $L(u) + 1 =_{\bmod 3} L(v)$ iff s_v is an encoding of v's port leading to u.

The size of this labeling scheme is $O(1)$ and it is clear that if G_s satisfies $f_{Orient}(G_s) = 1$ then $\mathcal{D}(v, \mathcal{M}) = 1$ at each vertex v. Now suppose that $f_{Orient}(G_s) = 0$. If for every v, s_v is an encoding of one of its port numbers then since the underlying graph G is a tree there must exist two vertices u and v whose states point at each other. Note that no label can be assigned to u and v that will satisfy (2), so we are done. Otherwise, there exists a vertex r whose state is not an encoding of one of its ports. If every v satisfies $\mathcal{D}(v, L) = 1$, then all the states of r's neighbors point at r and by induction we get an orientation of the edges towards r. Therefore $f_{Orient}(G_s) = 1$, which contradicts our assumption. ▮

3 The role and cost of identities

The bounds presented in [13] are similar for id-based and for anonymous families. Our model exhibits a distinction between the two families. Specifically, we show that certain tasks are impossible in some anonymous families but possible in id-based families. We also show that the transition from anonymous to id-based is very costly even on a path, where this transition is possible. Moreover, even for id-based paths, the task of proving whether the states are disjoint is costly. We also separate our model from the one of [1] by showing in Subsection 3.3 that our model is not order-invariant.

3.1 Anonymous versus id-based families

In this subsection we show examples of several problems that do not have proof labeling schemes in certain anonymous families. In contrast, we show that every problem has a proof labeling scheme in any id-based family. Consider the anonymous family $\mathcal{F}^{anon-circles}$ of circles with $O(n)$ vertices. Let f be the characteristic function of either one of the following problems, where it is required to assign states to the vertices of G such that:

1. there exists only one vertex $v \in V$ such that $s_v = 1$; (informally, if the states are considered as identities, then f checks whether the identity 1 is unique)
2. $s_v \neq s_u$ for every pair of vertices $u, v \in V$ (if the states are considered as identities then f checks whether the graph is id-based);
3. the state of each vertex is the number of nodes in G;
4. $H(G_s)$ is a (spanning, BFS) tree of G.

The proof of the following lemma bears some similarities to the proof, given in a different context, that the task of leader election is impossible in anonymous networks [16].

Lemma 3.1 *There is no proof labeling scheme for $\mathcal{F}_S^{anon-circles}$ and f.*

Proof: Assume that there is a proof labeling scheme $\langle \mathcal{M}, \mathcal{D} \rangle$ for $\mathcal{F}_S^{anon-circles}$ and f, where f is the characteristic function of either one of above mentioned problems. Let G be a circle graph with C vertices and let G_s be such that $f(G_s) = 1$. For analysis purposes, starting at some arbitrary vertex, clockwise enumerate the states of the vertices from 1 to C (see the circle on the left in Figure 1). This numbering is defined for ease of reference only, and is not available to the vertices themselves. Moreover, let i also denote the state of the vertex we numbered i.

Since $f(G_s) = 1$, $\mathcal{D}(v, \mathcal{M}) = 1$ for every vertex $v \in G$. Let $\mathcal{M}(i)$ be the label assigned by marker $\mathcal{M}$ to vertex i, the i'th vertex in the enumeration. Let G'_s be the circle with $2C$ vertices and states as shown in the circle on the right in Figure 1. For that circle on the right we gave each reference numbers $1, 2, ...C$ to two vertices, of distance C from each other. Let the state i of these two vertices be the same. We now create a new marker algorithm L for G'_s. For both vertices u, v in G'_s that have state s_i, let $L(u) = L(v) = \mathcal{M}(i)$.

For each vertex v with state s_i in G'_s , $N_L(v)$ in G'_s is the same as $N_{\mathcal{M}}(i)$ in G_s. Therefore, for each $v \in G'_s$ with state s_i, $\mathcal{D}(v, L) = \mathcal{D}(i, \mathcal{M}) = 1$, contradicting the correctness of π since obviously $f(G'_s) = 0$. ▮

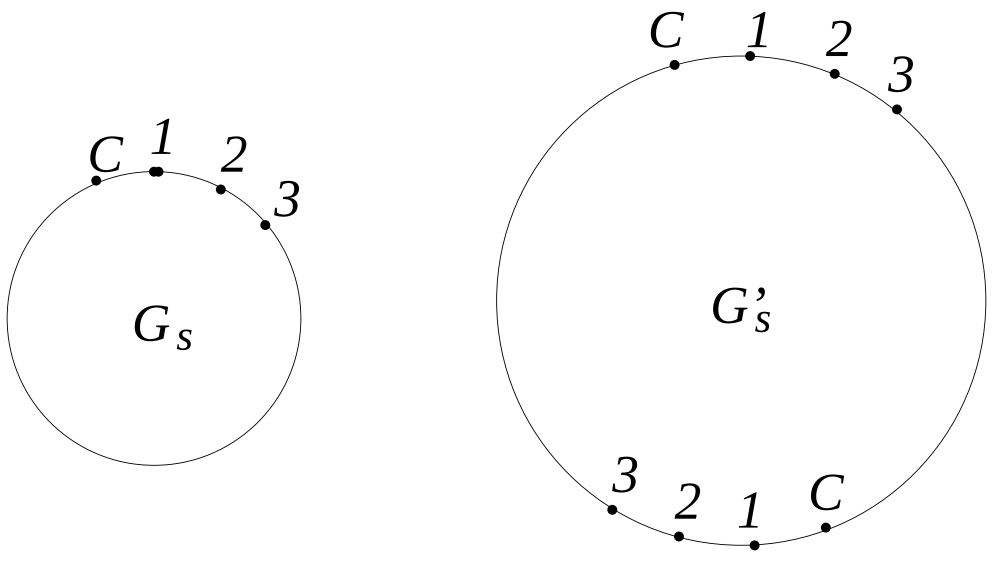

Figure 1: Anonymous circles (the number beside a vertex represents its state).

Lemma 3.2 *For every problem f there exists a proof labeling scheme in every id-based family $\mathcal{F}_s$ of connected graphs.*

Proof: Let f be a problem and let $\mathcal{F}_s$ be an id-based family of connected graphs. We describe a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ for $\mathcal{F}_s$ and f. Given a graph $G_s \in \mathcal{F}_s$, the marker algorithm $\mathcal{M}$ assigns each vertex v the label $\mathcal{M}(v)$ consisting of two fields. The first field, $\mathcal{M}_1(v)$, contains the entire graph G_s and the second field, $\mathcal{M}_2(v)$, is $id(v)$. The decoder $\mathcal{D}(v, L)$ outputs 1 iff the all following conditions are satisfied. Let $G(v)$ be the graph encoded in $L_1(v)$.

1. For every neighbor u of v, $L_1(u) = L_1(v)$.
2. (a) $L_2(v) = id(v)$.
   (b) There exists a vertex v' in $G(v)$ such that $id(v) = id(v')$.
   (c) The identities of v's neighbors in G and the weights and port numbers of the corresponding edges are the same as the ones associated with v''s neighbors in $G(v)$.
3. $f(G(v)) = 1$.

Clearly, if $f(G_s) = 1$ then $\mathcal{D}(v, \mathcal{M}) = 1$ for every vertex v. Assume that $f(G_s) = 0$ and let L be some marker algorithm. If the first item in the description of the decoder is satisfied for every vertex v then, since G is connected, $G(u) = G(v)$ for every two vertices u and v. If also the second item in the description of the decoder is satisfied for every vertex v then $G(v) = G_s$ for every vertex v. It follows that if the first two items in the description of the decoder are satisfied for every vertex then the third item in the description of the decoder is not satisfied for any vertex v. The lemma follows. ▌

3.2 Cost of identities

In the previous subsection we showed that giving a proof labeling scheme for the problem of distinct identities is impossible for a family of anonymous circles. In this subsection we show that although such a proof labeling scheme can be given on anonymous paths, it is very costly. In fact we show that the proof size of the distinct states problem on an n-node id-based path is $\Theta(n)$. More formally, let $f_{Distinct}$ be the characteristic function of the following problem: assign states to the nodes of G so that for every pair of vertices u and v, $s_u \neq s_v$. Let G be a path with n vertices and disjoint identities and let $\mathcal{F}^{path}$ be the id-based family containing the single path G, i.e., $\mathcal{F}^{path} = \{G\}$. Let $S = \{1, 2, \cdots, m\}$, where $m = O(n)$.

Lemma 3.3 *The proof size of $\mathcal{F}_S^{path}$ and $f_{Distinct}$ is $\Theta(n)$.*

Proof: Let us first show the existence of a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ of size $O(n)$. Given $G_s \in \mathcal{F}_S^{path}$ such that $f_{Distinct}(G_s) = 1$, the marker algorithm $\mathcal{M}$ assigns each vertex v the label $\mathcal{M}(v)$ which consists of two sublabels. Informally, the *orientation* sublabel, $\mathcal{M}_{orient}(v)$, is used in order to prove orientation on the path and the *array* sublabel, $\mathcal{M}_{array}(v)$, is used in order to point for each i, whether the vertex holding state i is to the left or to the right of v. Let r be one of the end-nodes of path G. We say that a vertex u is *to the right* of v if u is closer to r than v. For every vertex v, the orientation sublabel, $\mathcal{M}_{orient}(v)$, is composed of two fields, namely, $\mathcal{M}^1_{orient}(v)$ and $\mathcal{M}^2_{orient}(v)$, where $\mathcal{M}^1_{orient}(v)$ is the port number at v leading to the vertex to the right of v and $\mathcal{M}^2_{orient}(v) =_{\bmod 3} d_G(v, r)$. Let us now describe the array sublabel. For notational simplicity we name these vertices from left to right by $\{v_1, v_2, \cdots, v_n\}$ (these are not the nodes' identities). Recall that $S = \{1, 2, \cdots, m\}$ where $m = O(n)$. For each vertex v, $\mathcal{M}_{array}(v)$ is composed of m fields. For simplicity, we assume $m = n$; the proof for the case $m = O(n)$ is similar. Since $f_{Distinct}(G_s) = 1$, for every $1 \le i \le n$ there exists a unique vertex $v(i)$ such that its state $s_{v(i)}$ is i. Let $index(i)$ denote the index j such that $v_j = v(i)$. For every vertex v and every $1 \le i \le n$, the i'th field in $\mathcal{M}_{array}(v)$, $\mathcal{M}^i_{array}(v)$, contains either $+, -$ or $\circ$. For every vertex v_k and every $1 \le i \le n$, if $index(i) > k$ then $\mathcal{M}^i_{array}(v) = +$, if $index(i) < k$ then $\mathcal{M}^i_{array}(v) = -$ and if $index(i) = k$ then $\mathcal{M}^i_{array}(v) = \circ$. Clearly, π is of size $O(n)$.

The decoder algorithm $\mathcal{D}$ is conceptually composed of two components. In the first component, $\mathcal{D}$ checks whether the first fields in the orientation sublabels of the vertices impose an orientation on the path. This is done similarly to the decoder described in the proof of Lemma 2.4. In the second component, $\mathcal{D}$ uses the orientation imposed by the orientation sublabels in order to verify that the array sublabels are consistent. Formally, the decoder $\mathcal{D}(v, L)$ operates as follows. For every vertex v, denote by $right(v)$ the neighbor u of v such that $L^1_{orient}(v)$ is an encoding of v's port leading to u (if one exists).

1. For every neighbor u of v,
   (a) $|L^2_{orient}(u) - L^2_{orient}(v)| = 1$.
   (b) $L^2_{orient}(u) + 1 =_{\bmod 3} L^2_{orient}(v)$ iff $right(v)$ exists and $u = right(v)$.
2. For every neighbor u of v and every $1 \le i \le n$,
   (a) $s_v = i$ iff $L^i_{array}(v) = \circ$.
   (b) If $u = right(v)$ and $L^i_{array}(v) = \circ$ then $L^i_{array}(u) = -$.

(c) If $u = right(v)$ and $L^i_{array}(v) = -$ then $L^i_{array}(u) = -$.

3. $L^1_{orient}(v)$ is a neighbor of v or v has only one neighbor.

Clearly, if $f(G_s) = 1$ then $\mathcal{D}(v, \mathcal{M}) = 1$ for every vertex v. Let us therefore consider the case where $f(G_s) = 0$. Assume, by contradiction, that there exists a marker algorithm L such that $\mathcal{D}(v, L) = 1$ for every vertex v. Using similar arguments as in the proof of Lemma 2.4, we obtain that the first fields in the orientation sublabels of the vertices impose an orientation on the path towards some root vertex r. By item 3 in the description of the decoder, r must be an end-node vertex of the path. Given this orientation, we obtain that for every vertex $v \neq r$, $right(v)$ exists and is the neighbor to the right of v. Suppose, by way of contradiction, that there are two vertices v_i and v_j such that $i < j$ and $s_{v_i} = s_{v_j} = k$. By item 2.a in the description of the decoder, we have that $L^k_{array}(v_i) = \circ$ and by item 2.b, $L^k_{array}(v_{i+1}) = -$. Therefore, using induction and item 2.c, we obtain that $L^k_{array}(v_p) = -$ for every $p > i$ which contradicts the fact that by item 2.a, $L^k_{array}(v_j) = \circ$. The correctness of π follows.

Let us now turn to prove the lower bound. Let $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ be a proof labeling scheme for $\mathcal{F}^{path}_S$ and $f_{Distinct}$. Let us first give a high level description of our lower bound proof. We construct a large set X' of configuration graphs, all corresponding to path G such that for each $G_s \in X'$, $f_{Distinct}(G_s) = 1$. We choose X' so that the pair of labels given by $\mathcal{M}$ to the two vertices in the middle of the path must be different in each instance of X'. More formally, a *divided permutation* is a permutation σ on $[1, \cdots, n]$ such that $\sigma(n/2) = n/2$ and $\sigma(n/2 + 1) = n/2 + 1$. (Assume without loss of generality that n is even). Fix $s_{v_{n/2}} = n/2$ and $s_{v_{1+n/2}} = 1 + n/2$. For a divided permutation σ, let $\mathcal{M}_\sigma(v_i)$ denote $\mathcal{M}(v_i)$ in the case where for each $1 \leq j \leq n$, $s_j = \sigma(j)$. For a divided permutation σ, let $T_\sigma = \{\sigma(i) \mid 1 \leq i \leq n/2 - 1\}$ and $Q_\sigma = \{\sigma(i) \mid n/2 + 2 \leq i \leq n\}$. The proof uses the following claims.

Claim 1: Let σ_1 and σ_2 be two divided permutations such that $T_{\sigma_1} \cap Q_{\sigma_2} \neq \emptyset$. Then either $\mathcal{M}_{\sigma_1}(v_{n/2}) \neq \mathcal{M}_{\sigma_2}(v_{n/2})$ or $\mathcal{M}_{\sigma_1}(v_{1+n/2}) \neq \mathcal{M}_{\sigma_2}(v_{1+n/2})$.

Proof: Assume that the claim does not hold. Let g be the function over $\{1, \cdots, n\}$ that fixes $n/2$ and $1 + n/2$, identifies with σ_1 on $[1, \cdots, n/2 - 1]$ and identifies with σ_2 on $[2 + n/2, \cdots, n]$. If we let $s_{v_i} = g(i)$, then obviously $f_{Distinct}(G_s) = 0$ since $T_{\sigma_1} \cap Q_{\sigma_2} \neq \emptyset$. Create the following marker algorithm L. For $1 \leq i \leq 1 + n/2$, let $L(v_i) = \mathcal{M}_{\sigma_1}(v_i)$ and for $2 + n/2 \leq i \leq n$, let $L(v_i) = \mathcal{M}_{\sigma_2}(v_i)$. For every divided permutation σ and vertex v, we have $\mathcal{D}(v, \mathcal{M}_\sigma) = 1$. Therefore for all i s.t. $1 \leq i \leq 1 + n/2$, $\mathcal{D}(v_i, L) = \mathcal{D}(v_i, \mathcal{M}_{\sigma_1}) = 1$ and for $2 + n/2 \leq i \leq n$, $\mathcal{D}(v_i, L) = \mathcal{D}(v_i, \mathcal{M}_{\sigma_2}) = 1$, contradicting the correctness of the decoder. ▮

Claim 2: Let X be a set of divided permutations such that for every $\sigma_1, \sigma_2 \in X$, $T_{\sigma_1} \cap Q_{\sigma_2} \neq \emptyset$. There exists some $\sigma \in X$ so that either $\mathcal{M}_\sigma(v_{n/2})$ or $\mathcal{M}_\sigma(v_{1+n/2})$ has at least $\frac{1}{2} \cdot \log |X|$ bits.

Proof: By the previous claim, for each $\sigma \in X$ we obtain a different pair $(\mathcal{M}_\sigma(v_{n/2}), \mathcal{M}_\sigma(v_{1+n/2}))$. Therefore, one pair must consist of at least $\log |X|$ bits, yielding the claim. ▮

Claim 3: There exists a set X of divided permutations so that for every $\sigma_1, \sigma_2 \in X$, $T_{\sigma_1} \cap Q_{\sigma_2} \neq \emptyset$ and $\log |X| = \Omega(n)$.

Proof: Let $\hat{V}$ be the collection of all $(n-2)!$ divided permutations, and let $\Gamma = \langle \hat{V}, \hat{E} \rangle$ be the graph over $\hat{V}$ in which, for two permutations σ_1 and σ_2, $(\sigma_1, \sigma_2) \in \hat{E}$ iff $T_{\sigma_1} \cap Q_{\sigma_2} = \emptyset$. The degree of each vertex in Γ is $((n/2-1)!)^2 - 1$. Relying on the well known fact that every graph G with maximum degree d has an independent set of size $|V(G)|/d$, we conclude that Γ has an independent set X of size $\frac{(n-2)!}{((n/2-1)!)^2}$. Hence $\log(\frac{(n-2)!}{((n/2-1)!)^2}) = \Omega(n)$. Since X is an independent set of Γ, by definition of $\hat{E}$, $T_{\sigma_1} \cap Q_{\sigma_2} \neq \emptyset$ for every $\sigma_1, \sigma_2 \in X$. ▮

Combining the three claims, an $\Omega(n)$ lower bound is obtained on the size of any proof labeling scheme for $\mathcal{F}_s^{path}$ and $f_{Distinct}$. This concludes the lemma. ▮

We note that if one designs the states and the labels together then the size of the scheme can be much smaller: As mentioned, if $\mathcal{F}_s^{path}$ is id-based, then the label size can be zero (That is, if we are allowed to choose the states, let the state of every vertex be the identity of that vertex; the decoder just needs to check that this is the case, and no label is necessary). For an anonymous $\mathcal{F}_s^{path}$, one can choose the state of v_i to be i, and the size of these scheme is $\log n$.

3.3 Variability of identities

Two assignments of identities to vertices of a graph are *order preserving* if for every pair of vertices u,v, either $id(u) > id(v)$ in both id assignments, or $id(v) > id(u)$ in both. An algorithm is *order-invariant* if its output is the same for every two order preserving id assignments to the vertices of G. The following is shown in [1] for the model used there. Let A be a local algorithm for an id-based family $\mathcal{F}_s$ which computes a locally verified function f. Then there exists an *order-invariant* algorithm A' with the same complexities as A. I.e., A' uses only the relative order of the identities. We now investigate the question of whether every problem with a proof labeling scheme has also an order-invariant proof labeling scheme. We formalize our question as follows.

For a graph $G = \langle V, E \rangle \in \mathcal{F}_s$ with n vertices, a *legal assignment* to G is an assignment of disjoint identities to V in the range $\{1, \cdots, O(n)\}$. An *invariant proof labeling scheme* is a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ such that if $G_s \in \mathcal{F}_s$ satisfies $f(G_s) = 1$ for every legal assignment to G, then for every two order preserving (legal) assignments and every vertex v, $\mathcal{M}(v)$ is the same under both assignments.

Lemma 3.4 *There exists an id-based family $\mathcal{F}_s$ and a boolean function f over $\mathcal{F}_s$ for which there exists a proof labeling scheme but no order-invariant proof labeling scheme.*

Proof: Let $\mathcal{F}$ be the collection of cycles and let $S = \{1, 2, \cdots, O(n)\}$. Let f be a boolean function over $\mathcal{F}_S$ such that $f(G_s) = 1$ iff the number of vertices in G is s_v for all $v_s \in G_s$. I.e., f checks whether the states of the vertices of a cycle truly represent the number of vertices in it. It is easy to see that there exists a proof labeling scheme for $\mathcal{F}_S$ and f. Assume, by way of contradiction, that there also exists an invariant proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ for this problem. Consider an n-node cycle C so that for every vertex v in C, $s_v = n$. Consider the following four order equivalent identity assignments to C_s.

$$\begin{aligned}
(id_1(v_1), id_1(v_2), \cdots, id_1(v_{n-2}), id_1(v_{n-1}), id_1(v_n)) &= (1, 2, \cdots, n-2, 4n-1, 4n). \\
(id_2(v_1), id_2(v_2), \cdots, id_2(v_{n-2}), id_2(v_{n-1}), id_2(v_n)) &= (2n+1, 2n+2, \cdots, 3n-2, 4n-1, 4n). \\
(id_3(v_1), id_3(v_2), \cdots, id_3(v_{n-2}), id_3(v_{n-1}), id_3(v_n)) &= (1, 2, \cdots, n-2, 3n-1, 3n). \\
(id_4(v_1), id_4(v_2), \cdots, id_4(v_{n-2}), id_4(v_{n-1}), id_4(v_n)) &= (2n+1, 2n+2, \cdots, 3n-2, 3n-1, 3n).
\end{aligned}$$

Obviously, for every legal assignment to C_s, we have $f(C_s) = 1$. Since π is an invariant proof labeling scheme to this problem then for every vertex $v_i \in C_s$, marker $\mathcal{M}$ gives the same label in all these four assignments. Denote this label by $l(i)$.

Consider a $2n$-node cycle C' in which $s_v = n$ for every vertex v in C'. Obviously $f(C'_s) = 0$. Consider the following disjoint identity assignment to the vertices of C'.
$(id(v_1), id(v_2), \cdots, id(v_{n-2}), id(v_{n-1}), id(v_n), id(v_{n+1}), id(v_{n+2}), \cdots, id(v_{2n-1}, id(v_{2n})) =$
$(1, 2, , \cdots, n-2, 4n-1, 4n, 2n+1, 2n+2, \cdots, 3n-1, 3n)$.
Now assign to the vertices of C' the labeling $L(v_{i(\bmod n)}) = l(i)$. Then $\mathcal{D}(v, L) = 1$ for every vertex v, contradicting the correctness of π. ▮

4 Constructing proof labeling schemes

It may be unrealistic to expect to find an automatic way for constructing efficient proof labeling schemes. Still, we demonstrate two systematic approaches that can ease the design in many cases.

The first is the “distributed method”, based on ‘imitating’ distributed algorithms. The second is a modular construction approach based on the notion of composition. All the graph families in the section are id-based.

4.1 The distributed method

If there exists a distributed algorithm that generates exactly the configurations satisfying some characteristic function f, then we show an upper bound on the proof size of f.

¿From the more practical point of view, recall that a motivation for the model is a modular approach, i.e., given a configuration, we need to verify it. In many cases, the given configuration is generated by a distributed algorithm. Below, we show how to generate a labeling scheme in every such case. This demonstrates that ‘imitating’ an algorithm is sometimes useful in generating a proof labeling scheme of a small size. The approach is demonstrated in this section by building a proof labeling scheme of size $O(\log^2 n + \log n \log W)$ for minimum spanning trees, where W is the maximum weight of an edge. This construction is based on imitating the steps of a distributed algorithm for constructing an MST. After deriving the scheme from the algorithm, we make some changes in the scheme to reduce its size. In addition, we modify the algorithm so that it generates every possible MST (nondeterministically), rather than just a specific one. (This is needed in order to satisfy our requirement that every legal configuration has a proof labeling scheme; this means that here every MST must have a proof labeling scheme). We note that we do not have a better proof labeling scheme for MST using a different method. Let us also comment that we do not expect that the distributed method turns out to be the best approach for every problem. This is because the distributed method bases the verification on the computation, while there exists some evidence that the computation is sometimes harder, see e.g. Claim 4.2.

Consider some $\mathcal{F}_s$ and f, and let A be a non-deterministic distributed algorithm on $\mathcal{F}$ such that for every $G_s \in \mathcal{F}_s$ satisfying $f(G_s) = 1$ there exists a run A' of A such that the following hold.

1. $A'(G) = G_s$,
2. the number of messages a vertex sends in $A'(G)$ is bounded from above by m_0,
3. each message is encoded using $O(\log n)$ bits.

Moreover, assume that for every run $A'(G)$, $f(A'(G)) = 1$. If A is synchronous, assume also that for every $G \in \mathcal{F}_s$, the number of rounds in $A(G)$ is bounded from above by p and that the number of messages a vertex sends per round is bounded from above by m_p.

Lemma 4.1

1. *There exists a proof labeling scheme for $\mathcal{F}_s$ and f of size $O(m_0(\log m_0 + \log n))$.*
2. *If A is synchronous then there exists a proof labeling scheme for $\mathcal{F}_s$ and f of size* $\min\{O(m_0(\log p + \log n)), O(p \cdot m_p \log n)\}$.

Proof: We first describe a proof labeling scheme π_A for $\mathcal{F}_s$ and f of size $O(m_0(\log m_0 + \log n))$. Given G_s such that $f(G_s) = 1$, let $A'(G)$ be a specific run of $A(G)$ as described above. Note that at most $n \cdot m_0$ messages are sent by $A'(G)$. We enumerate these messages according to their chronological order, braking ties arbitrary. The label of every node v consists of at most m_0 fields, each corresponding to a message of $A'(G)$ incident to v. The i'th field of v's label contains three subfields. The first subfield contains the content of the i'th message v sends in $A'(G)$. The second contains the identity of the node receiving this message and the third subfield contains j, the number of the message in the above enumeration. The decoder is described later in this proof.

If $A(G)$ is synchronous then we show two proof labeling schemes for $\mathcal{F}_s$ and f, denoted π^1_A and π^2_A, where π^1_A has size $O(m(\log p + \log n))$ and π^2_A has size $O(p \cdot m_p \log n)$. For each vertex v, $\mathcal{M}^1_A(v)$ has a field for each message sent by v in A'. Each such field contains three subfields. The first subfield contains the content of the corresponding message, the second consists of the identity of the node receiving this message and the third contains the round number in which the message was sent. For each vertex v, $\mathcal{M}^2_A(v)$ has p fields. Each field i contains at most m_p subfields. Each subfield corresponds to a message that v sent in the i'th round. If the corresponding message was sent from v to u then the content of the message followed by the identity of u appear in that subfield.

For a marker algorithm L and a vertex v, all three decoders verify the following.

- There exists some run of A at v such that if the messages sent to v (in the corresponding order) are as indicated in the labels of v's neighbors, then the messages appearing in v's label are the messages sent by v (in the corresponding order).
- There exists some run of A at v such that if the messages sent to v are as indicated in the labels of v's neighbors and the messages appearing in v's label are the messages sent by v, then the output of A at v is s_v.

If at each node v the above items are verified, then the labels implicitly describe a distributed algorithm which is a legitimate run of the algorithm A on G whose output at each node v is s_v. Since $f(A(G)) = 1$ for every run of A, the lemma follows. ▮

A *polytime distributed algorithm* is a distributed algorithm that runs in sequential time (including the time needed for calculations) polynomial in the size of the graph.

Claim 4.2 *If $NP \neq P$ then there exist problems that have proof labeling schemes with polylog size and polytime decoder algorithm but do not have a polysize proof labeling scheme (with polytime decoder algorithm) that is constructed in the distributed method.*

Proof: Consider the Hamiltonian path problem. Using ideas similar to those of Lemma 2.3, it can be easily shown that it has an $O(\log n)$ size proof labeling scheme with polytime decoder algorithm. However if $NP \neq P$ then there is no polytime distributed algorithm that proves whether G has an Hamiltonian path. Therefore, given a graph G_s such that $H(G_s)$ is a Hamiltonian path of G, constructing the Hamiltonian path $H(G_s)$ from G, using a distributed algorithm, would either take more than a polynomial number of rounds or more than polynomial calculation time. In the first case the size of the labels constructed via the distributed method must be more than polynomial, and in the second case the decoder cannot be polynomial. ▮

Minimum Spanning Tree (*MST*) Problem: Assign states to the nodes of G so that $H(G_s)$ is an MST for G.

Lemma 4.3 *There exists a proof labeling scheme $\pi_{mst} = \langle \mathcal{M}_{mst}, \mathcal{D}_{mst} \rangle$ for $\mathcal{F}_S^{underected}(W)$ and f_{MST} of size $O(\log^2 n + \log n \log W)$.*

The use of the Lemma 4.1 does not suffice to prove Lemma 4.3, since the known distributed MST construction algorithms sometimes require some vertex v to send $\Omega(Deg(v))$ messages in some round. Substituting this in Lemma 4.1 yields a proof labeling scheme of higher size than desired. Still, by imitating *most* steps of the distributed algorithm described in [2], and using some careful changes, we manage to construct a proof labeling scheme for $\mathcal{F}_S^{underected}$ and f_{MST} of the desired size. (The resulting scheme can also be viewed as a scheme based on [3].)

Proof: Let $G_s \in \mathcal{F}_s^{undirected}(W)$ be such that $f_{MST}(G_s) = 1$. Denote the tree $H(G_s)$ by T. We refer to a subtree of T as a fragment. For each fragment F, an outgoing edge of F is an edge

$e = (u, x) \in G$ where $u \in F$ and $x \notin F$. A minimum outgoing edge of F, $e(F)$, is an outgoing edge of F of minimum weight. It is easy to show that every fragment F has a minimum outgoing edge that also belongs to this specific MST, T. We now build T in p phases that correspond to p fields in the labels. Intuitively, our construction is rather similar to the construction in the distributed algorithm of [2, 3] with the following adjustments. First, the algorithm in [2] constructs some MST while we construct specifically the given MST T. Second, in the algorithm of [2], each vertex 'wastes' a lot of messages in each round in order to update its neighbors of its new fragment identifier. These update messages do not have to appear in the labels since in our model a vertex v can 'see' its neighbors' labels. Therefore, if in each round each vertex encodes its fragment number, then each vertex v can know at each round to which fragment each of its neighbors belongs to. We now describe the construction more formally. For every vertex v, the i'th field in $\mathcal{M}(v)$ (that corresponds to phase i) has three subfields denoted $\mathcal{M}_1^i(v), \mathcal{M}_2^i(v)$ and $\mathcal{M}_3^i(v)$. The first subfield $\mathcal{M}_1^i(v)$ contains either an identity of one of v's neighbors in T or the identity of v itself. For every vertex v, $\mathcal{M}_1^1(v)$ contains just the identity of v. We inductively show that for each i, $F_i = \{(v, \mathcal{M}_1^i(v)) \mid v \in V\}$ is a collection of distinct fragments where $F_{\log n}$ is T itself. Obviously, F_1 is just the collection V (with no edges). Let F be a fragment in F_i and let $e(F) = (u, x) \in T$ be a minimum outgoing edge of F such that $u \in F$. Let $SPAN(F)$ be a spanning tree for F rooted at u. For each vertex $v \in F$ such that $v \neq u$ let $\mathcal{M}_2^i(v) = (id(u), id(y), d_{SPAN(F)}(v, u))$ where y is v's parent in $SPAN(F)$ and let $\mathcal{M}_2^i(u) = (id(u), id(x))$ (where $e(F) = (u, x)$). Informally, these second subfields are used to encode and verify the spanning tree $SPAN(F)$ of F (rooted at u). For each vertex $v \in F$ let $\mathcal{M}_3^i(v) = w(e(F))$ where $w(e(F))$ is the weight of a minimum outgoing edge of F (which also belongs to T). Let $\mathcal{M}_1^{i+1}(v)$ be the second field of $\mathcal{M}_2^i(v)$. The proof of the following claim is straightforward.

Claim: If F_i is a collection of distinct fragments then so is F_{i+1}.

Each fragment in F_{i+1} contains at least two fragments in F_i. In particular, the number of vertices in each fragment in F_{i+1} is at least double the number of vertices belonging to the smallest fragment in F_i. Therefore, $|F_{\log n}| = n$ yielding $F_{\log n} = T$.

Consider the case where T is not known and for each fragment F, $e(F)$ is chosen to be an arbitrary minimum outgoing edge of F. Then for all i, if F_i is contained in some MST then there exists an MST T' such that F_{i+1} is contained in T'. Therefore, using this construction we obtain that $F_{\log n}$ is an MST tree. If follows that the decoder $\mathcal{D}_{mst}$ needs only to verify that the construction of T is as it should be. This is done as follows.

For a vertex v and a marker algorithm L, $\mathcal{D}_{mst}(v, L)$ verifies that $L_1^i(v)$ contains an identity of either v itself or one of v's neighbors in T and that for every vertex v, $L_1^1(v)$ contains just the identity of v. The first subfield of $L_2^i(u)$ for all vertices u induces a partition of the vertices into maximal connected components $A_1, A_2, \cdots$ so that for each j and for each $u \in A_j$ the first subfield of $L_2^i(u)$ is the same. We denote this value by $r(A)$. Let A be the maximal connected component that v belongs to. The decoder uses similar operations as $\mathcal{D}_{span}$ (for verifying spanning trees, see Lemma 2.3) to verify that the second subfield of $L_2^i(u)$ for all vertices $u \in A$ forms a spanning tree to the subgraph of G induces by A rooted at the vertex $r(A)$. Moreover, the decoder verifies that this subtree contains the same set of vertices as the corresponding subtree implicitly described by the set of edges $\{(v, L_1^i(v)) \mid v \in A\}$. This is achieved by verifying that each edge in one subtree is pointed at in the other subtree by one of its endpoints. Then the decoder verifies that all vertices $u \in A$ agree on $L_3^i(u)$ denoted $w(A)$, and that for every $u \in A$, $w(A) \leq \min\{w(u,t) \mid (u,t) \in E, t \notin A\}$. For a vertex v such that $v = r(A)$, the decoder verifies that $w(A) = \omega(v, L_2^i(v))$. The correctness of the scheme follows. Since each label contains $O(\log n)$ fields and since each such field contains $O(\log n + \log W)$ bits, the lemma follows. ∎

Lemma 4.4 *Every proof labeling scheme for $\mathcal{F}_S^{undirected}(W)$ and f_{MST} has size $\Omega(\log n + \log W)$.*

Proof: Since an MST is also a spanning tree we get by Lemma 2.3 that every proof labeling scheme on $\mathcal{F}_S^{undirected}(W)$ and f_{MST} has size $\Omega(\log n)$. Therefore, we only need to show that every proof labeling scheme on $\mathcal{F}_S^{undirected}(W)$ and f_{MST} has size $\Omega(\log W)$. Assume, by way of contradiction, that there exists a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ on $\mathcal{F}_S^{undirected}(W)$ and f_{MST} of size less than $\frac{1}{6} \cdot \log \frac{W-1}{2}$. Let C^i be the 6-vertex cycle whose identities are ordered clockwise from 1 to 6. Let the weight of each edge in C^i be 1 except that $\omega(3,4) = 2i$ and $\omega(6,1) = 2i+1$. For each vertex $1 \leq j < 6$, let s_j be the port number in vertex j leading from j to $j+1$. Obviously, $f_{MST}(C_s^i) = 1$ for every i. Therefore, for every i, marker $\mathcal{M}$ assigns a labeling assignment L^i to the vertices of C_s^i so that for each vertex v in C_s^i, $\mathcal{D}(v, L^i) = 1$. Since the size of π is less than $\frac{1}{6} \cdot \log(\frac{W-1}{2})$, we get that there exist two cycles C^i and C^k so that $i < k \leq (W-1)/2$ and $L^i = L^j$. Let C_s be the same as C_s^i except that $\omega(3,4) = 2k$. Obviously, $f_{MST}(C_s) = 0$. However, by the correctness of π on C_s^i and C_s^k we get that $\mathcal{D}(v, L^i) = 1$ for every vertex $v \in C_s$, contradiction. ∎

4.2 Composition

When constructing a proof labeling scheme for some problem $Prob_1$, we sometimes want to use modularly a solution to a different problem, $Prob_2$. We illustrate the notion of composition for labeling schemes through an example. Let $\mathcal{F}$ be the family of all connected $O(n)$-vertex graphs. Assume that we want to prove that the states of all nodes in a given graph $G_s \in \mathcal{F}_s$ are identical and equal to the number of nodes in G. Let f_{SUM} be the characteristic function of this problem. To construct a desired proof labeling scheme, it is useful to have a spanning tree T on G. (See, for example, the proof of Lemma 4.5 below.) To utilize the spanning tree, we may want to use the proof labeling scheme π_{span} for spanning trees. However, π_{span} checks whether $H(G_s)$ is a spanning tree for G, which is clearly not the case, since the states in G_s are supposed to be the number of nodes in G and not encodings of port numbers. In order to overcome this technicality, we define a new graph $G_{\tilde{s}}$ to be the graph G except that the state $\tilde{s}_v$ of each non-root node is an encoding of v's port leading to its parent in the desired spanning tree T. We now label each node $v_s \in G_s$ with a label consisting of three parts, namely, $L_1(v)$, $L_2(v)$, and $L_3(v)$. Part $L_1(v)$ is the state of $G_{\tilde{s}}$; part $L_2(v)$ is the label given to v by $\mathcal{M}_{span}$ for graph $G_{\tilde{s}}$. Part $L_3(v)$ is now designed to verify f_{SUM} on the tree graph defined by L_1. The decoder then simulates a decoder for a spanning tree (for that it uses L_1 and L_2) and then simulates a decoder for f_{SPAN} on $\mathcal{F}_S^{trees}$, for the tree defined by L_1 and the labels given by L_3.

It follows that when using a proof labeling scheme π_1 (for some graph family $\mathcal{F}_s$) in a composition, and then applying π_2, the size of the resulted composed scheme is bounded from above by the sum of the sizes (of π_1 and π_2) plus the maximum size of a state in $\mathcal{F}_s$.

In all of the cases described hereafter, the number of bits in a state is $O(\log n)$ and the sizes of every proof labeling scheme which is used in a composition is $\Omega(\log n)$. Hence the size of the composed proofs are asymptotically the same as the sum of the sizes of the corresponding proof labeling schemes used for the composition. We now give proof labeling schemes for several problems, some of which are used as components for others defined through composition as described above. All graph families mentioned below are id-based.

4.2.1 Compositions using a scheme for semi-group functions:

The following proof labeling scheme is an important building block for many other schemes. Let $G = \langle V, E \rangle$ be a graph. A function g is a semi-group function if it is well-defined for every subset U_s of V_s, and is associative and commutative. For example, $g(U)$ can be $|U|$ or $\sum_{v \in U} s_v$. Let g be a semi-group function defined on all graphs $G_s \in \mathcal{F}_s$. We assume that the values of g over all subsets of V_s and all graphs $G_s \in \mathcal{F}_s$ are bounded from above by some l. Let f_g be the characteristic function of the following problem: assign a value $val(u)$ to each node u of G so that for every node u in G, $val(u) = g(V_s)$. The proof of the following lemma uses the scheme for a rooted spanning tree as a building block, and the value $g(V_s)$ is recursively verified on the tree.

Lemma 4.5 *For every semi-group function g, there exists a proof labeling scheme π_g for $\mathcal{F}_s^{all}$ and f_g of size $O(\log n + \log l)$.*

Proof: We construct a desired scheme $\pi_g = \langle \mathcal{M}, \mathcal{D} \rangle$ as follows. Using composition of proofs and Lemma 2.3, we can assume a spanning tree T is known. I.e., each vertex knows which of its edges leads to its parent in the tree and the root knows it is the root. Furthermore, given $N_L(v)$, each vertex v knows which of its edges connects it to a child of v in T. $\mathcal{M}_g(v)$ consists of two fields: the first field is $g(V_s)$ and the second consists of $g(T_v)$, where T_v is the set of descendants of v (including v itself) in T. Given a marker algorithm L and a vertex $v \in G$, Let $L_i(v)$ denote the i'th field of $L(v)$. The decoder $\mathcal{D}_g(v, L) = 1$ iff all the following hold.

1. For each vertex v, $L_1(v) = val(v)$.
2. For each neighbor u of v, $L_1(v) = L_1(u)$. I.e., all the vertices have the same candidate for $g(V_s)$.
3. Let $C(v)$ be the set of children u of v in T. Then $L_2(v) = g(v_s, \{L_2(u) \mid u \in C(v)\})$.
4. If v is the root of T then $L_1(v) = L_2(v)$.
5. The decoder for f_{span} gives an answer 1.

The size bounds are easy to verify, yielding the lemma. ▮

Lemma 4.6 *There exists a semi-group function g such that* every *proof labeling scheme for family $\mathcal{F}_s^{undirected}$ and function f_g must have size $\Omega(\log n + \log l)$.*

Proof: By letting $g_1(U) = |U|$ for every subset U of V, a similar proof as the proof of the lower bound in Lemma 2.3 shows that every proof labeling scheme for $\mathcal{F}$ and f_{g_1} must have size $\Omega(\log n)$. Let $g_2(U) = \max\{s_v \mid v \in U\}$ for every subset U of V, and let G be the connected graph containing two vertices v and u. For $i = 1, 2, \cdots, l$, let G_s^i be the graph G such that $s_v = s_u = (i, i)$. Then,

$g_2(G_s^i) = 1$ for each i. A proof along similar lines as the proof for the lower bound in Lemma 2.1 shows that every proof labeling scheme for G and f_{g_2} must have size $\Omega(\log l)$. For every subset U of V, let $g(U) = \langle g_1(U), g_2(U) \rangle$. If follows that every proof labeling scheme for $\mathcal{F}_s^{all}$ and f_g must have size $\Omega(\log n + \log l)$. ▮

4.2.2 Clique and independent set

Let f_{Clique} (respectively, $f_{Independent}$) be the characteristic function of the following problem: assign states to the vertices of G so that: 1) All the states have the same value k, and 2) there exists a clique (respectively, independent set) of size k in G.

In the following lemmas, we use a composition with π_g (itself a composition with the scheme for spanning trees) to count the number of nodes (in a clique, or an independent set, or a minimum cut).

Lemma 4.7 *The proof size of $\mathcal{F}_S^{undirected}$ and both functions f_{Clique} and $f_{Indepenent}$ is $\Theta(\log n)$.*

Proof: For the upper bound, we construct $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ for $\mathcal{F}_S^{undirected}$ and f_{Clique} as follows. For G_s such that $f_{Clique}(G_s) = 1$, let K be a clique with k vertices (where k is the state of all the nodes). If $v \in K$, then let $\mathcal{M}(v) = 1$, otherwise let $\mathcal{M}(v) = 0$. By using Lemma 4.5, we can verify that $|\{v \mid L_1(v) = 1\}| = k$. We further verify that the set $\{v \mid L_1(v) = 1\}$ forms a clique by checking at each vertex v such that $L_1(v) = 1$ that v has $k-1$ neighbors u satisfying $L_1(u) = 1$. A similar construction give a proof labeling scheme for $\mathcal{F}$ and $f_{Independent}$ with size $O(\log n)$.

The proof for the lower bound resembles the proof for the lower bound in Lemma 2.1. A direct reduction from the problem discussed in Lemma 2.1 does not seem immediate.

We prove the lower bound for f_{Clique}, however, a similar proof also applies to $f_{Independent}$. Let $\{G_k\}_{k=2}^{n-1}$ be a family of n-node graphs with fixed vertices $v_1, v_2, \cdots, v_n$ such that for every $1 < k < n$, the only edge in G_k incident to v_1 is (v_1, v_2) and the maximum clique in the subgraph of G_k induced by $v_2, v_3, \cdots, v_n$, is of size k. For every $1 < k < n$, let k be the state of each vertex in G_k. Therefore, for every $1 < k < n$, $f_{Clique}(G_k) = 1$. Assume, by way of contradiction, that there is a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ for $F_S^{undirected}$ and f_{Clique} of size less than $-1 + \frac{1}{2}\log n$. For a vertex x, let $\mathcal{M}_k(x)$ be the label given to x by marker $\mathcal{M}$ in G_k. Let $L_k = (\mathcal{M}_k(v_1), \mathcal{M}_k(v_2))$. Since the number of bits in L_k is assumed to be less than $\log(n-2)$, there exist $1 < i < j < n-1$ such that $L^i = L^j$. Let G' be G_j modified so that $s_{v_1} = i$. Let L be the marker algorithm for G' in which for every vertex x, $L(x) = \mathcal{M}_j(x)$. Then, for each vertex x, $\mathcal{D}(x, L) = 1$, contradicting the fact that $f(G') = 0$ (since the state of v_1 is different than the one of v_2. The lemma follows. ▮

4.2.3 Vertex-connectivity and flow

Let $\mathcal{F}_S^{s_0-t_0}$ be the graph family containing all undirected n-node graphs with two given nodes s_0 and t_0. Let f_{v-conn} be the characteristic function of the following problem: assign states to the nodes of G so that each state contains a field $k \in S$ that is the vertex connectivity between s_0 and t_0.

Lemma 4.8 *The proof size of $\mathcal{F}_S^{s_0-t_0}$ and f_{v-conn} is $\Theta(\log n)$.*

Proof: We first construct a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ for $\mathcal{F}_S^{s_0-t_0}$ and f_{v-conn} of size $O(\log n)$. Let f_1 and f_2 be two boolean functions from $\mathcal{F}_S^{s_0-t_0}$ so that $f_1(G) = 1$ iff there exists at least k vertex disjoint paths connecting s_0 and t_0, and $f_2(G) = 1$ iff there exists at most k vertex disjoint paths connecting s_0 and t_0. Obviously, $f_{v-conn}(G) = 1$ iff $f_1(G) = 1$ and $f_2(G) = 1$. We first give a proof labeling scheme π_{f_1} for $\mathcal{F}_S^{s_0-t_0}$ and f_1. The part of the proof that all the nodes agree on the value of k is the same as in Lemma 2.1. We concentrate on proving the other parts of f_1. Let G be such that $f_1(G) = 1$. Let $P_1, P_2, \cdots, P_k$ be k vertex disjoint paths connecting s_0 and t_0. For every i, denote the vertices of P_i by $(p_i^0, p_i^1 \cdots, p_i^{l_i})$, where $s_0 = p_i^0$, $t_0 = p_i^{l_i}$ and p_i^{j+1} is the next vertex after p_i^j in P_i. For a vertex p_i^j where $1 \leq j \leq l_i - 1$, set $\mathcal{M}_{f_1}(p_i^j) = (i, j, id(p_i^{j+1})) \equiv \langle L_1(v), L_2(v), L_3(v) \rangle$. For every other vertex v set $\mathcal{M}_{f_1}(v) = \emptyset$. Assume first that s_0 is not a neighbor of t_0. The decoder $\mathcal{D}_{f_1}$ verifies the following.

- At the special vertex s_0, the decoder verifies that s_0 has precisely k neighbors with nonempty labels.
- For every vertex v with nonempty label such that $v \neq s_0$ and $v \neq t_0$, the decoder verifies that there is an edge connecting v to $u = L_3(v)$ and that either $u = t_0$ or (1) $L_2(u) = L_2(v) + 1$ and (2) $L_1(u) = L_1(v)$.

If is easy to show that π_{f_1} is a correct proof labeling scheme for $\mathcal{F}_S^{s_0-t_0}$ and f_1 of size $O(\log n)$.

If s_0 is a neighbor of t_0 then the label of s_0 is the identity of s_0, the label of t_0 is the identity of t_0. In this case the decoder in s_0 needs to verify that t_0 is a neighbor, and acts as above for $k - 1$, and for G minus the edge (s_0, t_0).

We now give a proof labeling scheme π_{f_2} for $\mathcal{F}_S^{s_0-t_0}$ and f_2. Let us first assume that s_0 and t_0 are not neighbors. Then, by Menger's theorem (cf. [20]), there exists k vertices u_1, u_2, $\cdots$, u_k whose deletion from the graph G (along with their edges) disconnects t_0 from s_0. We now mark each u_i by $*$, each vertex in the connected component of s_0 by 0, and the rest of the vertices by 1. Using π_g (Lemma 4.5) we first verify that there are exactly k vertices marked with $*$. The decoder then verifies the following.

- s_0 is marked with 0.
- t_0 is marked with 1.
- For every neighbor u of a vertex v, if both u and v are not marked with $*$ then they are both marked the same.

Now, if s_0 and t_0 are neighbors (which is locally detectable), we do the same as above assuming our graph is G minus the edge (s_0, t_0) and f_2 is with parameter $k-1$ instead of k. Again, it is easy to show that π_{f_2} is a correct proof labeling scheme for $\mathcal{F}_S^{s_0-t_0}$ and f_2 of size $O(\log n)$. Combing π_1 and π_2 we get a proof labeling scheme for $\mathcal{F}_S^{s_0-t_0}$ and f_{v-conn} of size $O(\log n)$.

As for the lower bound, a similar proof to the proof of the lower bound in Lemma 4.7 can be constructed, showing that the above mentioned upper bound is asymptotically tight. ∎

Let $\mathcal{F}_S^{s_0-t_0}$ be as before and let f_{flow} be the characteristic function of the following problem: assign states to the nodes of G so that each state contains a field $k \in S$ which is the flow between s_0 and t_0.

Lemma 4.9 *The proof size of $\mathcal{F}_S^{s_0-t_0}$ and f_{flow} is $O(k \log n)$.*

Proof: Let f_1 and f_2 be two boolean functions from $\mathcal{F}_S^{s_0-t_0}$ so that $f_1(G) = 1$ iff the flow between s_0 and t_0 is at most k and $f_2(G) = 1$ iff the flow between s_0 and t_0 is at least k. Obviously, $f_{flow}(G) = 1$ iff $f_1(G) = 1$ and $f_2(G) = 1$. We first give a proof labeling scheme for $\mathcal{F}_S^{s_0-t_0}$ and f_1. By the Max-flow min-cut theorem (cf. [20]), the flow between s_0 and t_0 is at most k iff there exists a set H of (directed) edges such that:

- The capacity of H is at most k ,i.e., $\omega(H) := \sum_{e \in H} \omega(e) \leq k$.
- Deleting the edges of H from G separates t from s.

Let G be such that $f(G) = 1$ and let H be a set of (directed) edges as described above. Since we assume that all the weights are integers, we get that $|H| \leq k$. For each vertex u, let H_u be the set of (directed) edges $(u, v) \in H$. For every vertex u such that H_u is not empty, let $\mathcal{M}(u)$ consist of $|H_u|$ fields where each field correspond to a different edge $e \in H_u$ and contains $id(e) = (u, v)$. After deleting the edges of H we mark each vertex that is reachable from s_0, by 0, and every other vertex, by 1. (Note that t_0 is marked by 1). By Lemma 4.2.1, we can verify that $\omega(H) \leq k$. In order to check that the deletion of H separates t from s, the decoder $\mathcal{D}_{flow}$ verifies the following items.

- s_0 is marked with 0.
- t_0 is marked with 1.
- For every vertex u, and (directed) edge $(u, v) \notin H_u$, vertex v is marked the same as u.

It is easy to show that this is a correct proof labeling scheme for $\mathcal{F}_S^{s_0-t_0}$ and f_1 of size $O(k \log n)$.

A proof labeling scheme for $\mathcal{F}_S^{s_0-t_0}$ and f_2 of size $O(k \log n)$ can easily be constructed similar to f_1 in the proof of Lemma 4.8. That is, it is enough to by specify at each vertex v the direction and quantity of the (integer) positive flow on each edge incident to v. The decoder then checks that Kirchoff laws are conserved in every vertex as required for a legal flow (see e.g. [20]). By combining the proof labeling scheme for $\mathcal{F}_S^{s_0-t_0}$ and f_1 with the one for $\mathcal{F}_S^{s_0-t_0}$ and f_2, we obtain a proof labeling scheme for $\mathcal{F}_S^{s_0-t_0}$ and f_{flow} with size $O(k \log n)$, as desired. ▮

4.2.4 Estimating the diameter

Let $f_{Low-diam}$ be the characteristic function of the following problem: assign states to the nodes of G so that the value of each state is the same and this value bounds the diameter from below.

Lemma 4.10 *The proof size of $\mathcal{F}_S^{undirected}$ and $f_{Low-diam}$ is $\Theta(\log n)$.*

Proof sketch: We construct a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ for $\mathcal{F}_S^{undirected}$ and $f_{Low-diam}$ as follows. Let G_s such that $f_{Low-diam}(G_s) = 1$. Let x, y be two vertices in G whose distance from each other is the diameter of G. Let $T(x)$ be the BFS tree rooted at x and let $T(y)$ be the BFS tree rooted at y. We first copy the state at each vertex into its label in order to verify that all states are the same. This is done using the proof labeling scheme described in Lemma 2.1. We continue by composing with the proof labeling scheme for rooted BFS trees (Lemma 2.3) encoding and verifying that $T(x)$ (respectively, $T(y)$) is a BFS tree rooted at x (resp., y). By the remark at the end of the proof of the upper bound for Lemma 2.3, using the proof labeling scheme for $T(x)$, we can assume that each vertex knows its distance to x. Using the proof labeling scheme for $T(y)$, we verify that y exists and it is therefore left for the decoder to verify at y that $d(y,x) \geq s_y$. The correctness of the scheme follows and the upper bound is obtained using Lemmas 2.1 and 2.3. The lower bound follows using a similar proof to the proof of the lower bound in Lemma 2.3. ▮

Note that Lemma 4.10 also for $\mathcal{F}_S^{all}(W)$, only that the bound changes to $O(\log n + \log W)$. Note also that by a similar construction we can prove that the diameter is at most $2c$ for some constant c.

4.2.5 Diameters of trees

We start by proving the following graph theoretical claim.

Claim 4.11 *Let T be a tree rooted at r and let D be the diameter of T. Let a be a vertex maximizing the function $d(r,\cdot)$ and let b be the vertex maximizing $d(a,\cdot)$, then $d(a,b) = D$.*

Proof: Let x, y be vertices such that $d(x,y) = D$ and let $t = nca(x,y)$ (where $nca(x,y)$ is the *nearest common ancestor* of x and y). For vertices u and v, denote by $[u,v]$ the (shortest) path connecting u and v in T. If vertex a is a descendent of t, then either $[x,t] \cap [a,t] = \{t\}$ or $[y,t] \cap [a,t] = \{t\}$. Assume without loss of generality that $[x,t] \cap [a,t] = \{t\}$. Let $g = nca(a,y)$. Since g is in $[t,y]$ and since $d(r,y) \leq d(r,a)$ then $d(g,y) \leq d(g,a)$. Therefore $d(x,y) \leq d(x,a) \leq d(a,b)$ and the claim follows in this case.

If vertex a is not a descendent of t, then let $z = gcd(t,a)$ and let $q = gcd(a,b)$. Consider two cases.

- **Case 1**: q is on $[z,a]$.
  Then, $d(x,y) \leq d(x,z) + d(z,y) \leq d(x,z) + d(z,a) = d(x,a) \leq d(a,b)$.
- **Case 2**: q is not on $[z,a]$.
  Then, $d(x,y) \leq d(x,z) + d(z,y) \leq d(a,z) + d(z,b) = d(a,b)$.

In either case, we obtain that $d(x,y) \leq d(a,b)$ and the claim follows. ▮

Let $c \geq 1$ be a constant integer. Let $\mathcal{T}$ be the collection of all connected (anonymous) trees with $O(n)$ vertices. Let $f_{Diameter}$ be a boolean function over $\mathcal{T}_S$ so that $f_{Diameter}(G_s) = 1$ iff the diameter of T is s_v for every vertex v.

Lemma 4.12 *The proof size of $\mathcal{T}_S$ and $f_{Diameter}$ is $\Theta(\log n)$.*

Proof: We first construct a proof labeling scheme as claimed. Given a tree T, our proof labeling scheme works as follows. Following the same steps as in the proof of the upper bound for Lemma 4.10, we can agree and verify that some vertex x maximizes the distance to the root r and that some vertex y maximizes the distance to x. In addition, we may assume that vertex y knows $d(x, y)$. Using Claim 4.11, it is therefore left to verify that the state at each vertex is the same and that this value is $d(x, y)$. The upper bound follows.

We prove the lower bound using an argument that is similar to the one used in the proof of the lower bound of Lemma 2.3. Let P be the horizontal path of n vertices. Enumerate the vertices of P from left to right. i.e., $P = (1, 2, \cdots, n)$ (the enumeration is only for the analysis). For every vertex v, let $s_v = n$. Obviously, $f_{Diameter}(P_s) = 1$. Assume, by way of contradiction, that there exists a proof labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ for $\mathcal{F}_s$ and $f_{Diameter}$ which is of size less than $\log(n/2) - 2$. Let $L(i)$ be the label given by $\mathcal{M}$ to vertex i in the above path P_s. Since the number of bits in each $L(i)$ is less than $\log(n/2) - 2$, there exist two pairs of vertices $(i, i+1)$ and $(j, j+1)$ where $1 < i$ and $i+1 < j < n-1$ so that $L(i) = L(j)$ and $L(i+1) = L(j+1)$. We now construct the following path P' consisting of $n - (j - i)$ vertices: $(1, 2, \cdots, i, i+1, j+2, j+3, \cdots, n)$. For every vertex $v \in P'$ let $s_v = n$. Let us give P'_s the same labeling L as $\mathcal{M}$ gives P_s, i.e., each vertex i in P'_s is labeled $L(i)$. By the correctness of π on P_s we get that for each vertex $v \in P'_s$, $\mathcal{D}(v, L) = 1$ which contradict the fact that $f_{Diameter}(P'_s) = 0$. ▮

Again, Lemma 4.12 holds also for $\mathcal{T}_S(W)$ only that the upper bound changes to $O(\log n + \log W)$.

4.2.6 Maximum Matching on paths

Let P be the horizontal path of n vertices $v_1, v_2, \cdots, v_n$. (The enumeration is from left to right and does not necessarily correspond to the identities of the vertices; i.e., $id(v_i)$ is not necessarily i). For every edge $e_i = (v_i, v_{i+1})$, let $\omega(e_i)$ denote the weight of e_i. A set H of edges of P is called a matching if no two edges $e, e' \in H$ share a common node. For such set, denote by $\omega(H) = \sum_{e \in H} \omega(e)$. Let $m = \max\{\omega(H) \mid \text{H is a matching of P}\}$. A matching H of P is called a maximum matching if $\omega(H) = m$.

Let f_{Match} be the characteristic function of the following problem: assign states to the nodes of G so that $H(G_s)$ is a maximum matching of G. Let $\mathcal{F}^{paths}$ be the collection of all weighted paths with $O(n)$ vertices and let $S = \{1, 2\}$. Let $m = \max\{\omega(H) \mid \text{H is a matching of E}\}$. The following proof uses techniques based on dynamic programming.

Lemma 4.13 *The proof size of $\mathcal{F}_S^{paths}(W)$ and f_{Match} is $O(\log n + \log W)$.*

Proof: We show the existence of a proof labeling scheme $\pi_A = \langle \mathcal{M}_A, \mathcal{D}_A \rangle$ for $\mathcal{F}_S^{paths}(W)$ and f_{Match} of size $O(\log n + \log W)$. Let $P = (v_1, v_2, \cdots, v_n)$ be the given path. Let G_s be such that $f_{Match}(G_s) = 1$. First, by composing with the proof labeling scheme presented in Lemma 2.4, we may assume an orientation on the path. I.e., v_1 'knows' it is the leftmost vertex and each other vertex 'knows' which outgoing edge connects it to a vertex on its left. Second, by composing with the proof labeling scheme described in Subsection 4.2.1, we may assume that all vertices 'know' and agree on the value $\omega(H)$, where $H = H(G_s)$. Since it is easy to verify locally that H is indeed a matching, it is enough to show that we can design a proof labeling scheme proving $\omega(H) = m$. We calculate m in the following manner. Let P_i be the prefix subpath $(v_1, v_2, \cdots, v_i)$. Let A_i be the maximum value over all matchings P_i that exclude edge e_{i-1}. Let B_i be the maximum value over all matchings in P_i that include edge e_{i-1}. Note that $A_{i+1} = \max\{A_i, B_i\}$, $B_{i+1} = A_i + \omega(e_{i+1})$ and $m = \max\{A_n, B_n\}$. Let $cM(i) = (A_i, B_i)$. By the above observations, given the label of the vertex to its left and the weight of the corresponding edge, each vertex v_i can verify that its label is indeed (A_i, B_i). In addition, the rightmost vertex v_n verifies that $\omega(H) = \max\{A_n, B_n\}$. The lemma follows since $m = O(n \cdot W)$. ▮

4.2.7 A 2-approximation for Maximum matching on a trees

The following example illustrates a combination of the distributed method and a composition of proof labeling schemes. Denote by an *approximation function*, a function *Approx* over $\mathcal{T}_S(W)$, such that for every tree $T \in \mathcal{T}_S(W)$, *Approx*(T) is a nonempty set of matchings of T with the property that for every $H \in$ *Approx*(T), $\omega(H) \geq m/2$.

Consider the following problem: assign states to the nodes of the given tree T such that the subgraph induced by them is in *Approx*(T).

In contrast to most of our results, the following lemma doesn't assert that every 2-approximation maximum matching on a tree can be proven but rather that we know how to prove a specific 2-approximate maximum matching. This leaves open a problem: how to prove efficiently any given 2 approximation for maximum matching.

Lemma 4.14 *There exists an approximation function H and a proof labeling scheme $\pi_A = \langle \mathcal{M}_A, \mathcal{D}_A \rangle$ for $\mathcal{F}_S^{trees}$ and f_{Approx} of size $O(\log n + \log W)$.*

Proof: In [4], given a tree T, they show how to distributively build a collection $\mathcal{P} = \mathcal{P}(T)$ of vertex disjoint paths with a constant number of rounds and a constant number of messages each vertex sends per round, such that $m \leq \sum_{P \in \mathcal{P}} \omega(P)$. For every path $P \in \mathcal{P}$, a maximum matching $MAX(P)$ for P satisfies $\omega(MAX(P)) \geq \omega(P)/2$. Therefore the collection of edges $MAX(T) = \cup_{P \in \mathcal{P}} MAX(P)$ satisfies $\omega(MAX(T)) \geq m/2$. Therefore $\mathcal{H}(T) = \{\cup_{P \in \mathcal{P}} MAX(P) \mid MAX(P)$ is a maximum matching on P$\}$ is an approximation function. By Lemma 4.1, there exists a proof labeling scheme of size $O(\log n)$ so that each vertex in T knows which of its edges belong to $\mathcal{P}$. On each path $P \in \mathcal{P}$ (recall that these paths are vertex disjoint), we compose with the proof labeling scheme shown in Lemma 4.13 applied to the path P. Altogether, we have a proof labeling scheme of size $O(\log n + \log m)$ for $\mathcal{F}_S^{trees}$ and f_{Approx}. The lemma follows as $m = O(n \cdot W)$. ▮

References

[1] M. Naor and L. Stockmeyer. What can be computed locally? *Proc. 25th ACM Symp. on Theory of Computing*, pages 184–193, 1993.

[2] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, 2000.

[3] R.G. Gallager, P.A. Humblet, P.M. Spira. A distributed algorithm for minimum-weight spanning trees. ACM Trans. on Programming Languages and Systems, 5 (1983) 66-77.

[4] M. Wattenhofer and R. Wattenhofer. Distributed Weighted Matching. *Proc. DISC*, pages 335–348, 2004.

[5] A. Arora and M. Gouda. Distributed reset (extended abstract) *Proc. 10th Conf on Foundations of Software Technology and Theoretical Computer Science*, 316-331, November 1990, Bangalore, India.

[6] S. Aggarwal and S. Kutten. Time Optimal Self-Stabilizing Spanning Tree Algorithms. *Proc. 13th Conf on Foundations of Software Technology and Theoretical Computer Science*, 1993, 400-410.

[7] B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, and G. Varghese. Time optimal self-stabilizing synchronization. *Proc. STOC* 1993, 652-661.

[8] Y. Afek and S. Dolev. Local Stabilizer. J. Parallel Distrib. Comput. 62(5)s. 745-765 (2002).

[9] Y. Afek, S. Kutten, and M. Yung. The Local Detection Paradigm and Its Application to Self-Stabilization. *Theor. Comput. Sci.* 186(1-2): 199-229 (1997).

[10] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-Stabilization By Local Checking and Correction. *Proc. FOCS* 1991, 268-277.

[11] B. Awerbuch, B. Patt-Shamir, G. Varghese, and S. Dolev. Self-Stabilization by Local Checking and Global Reset. *Proc. WDAG*, 1994, 326-339.

[12] Baruch Awerbuch, George Varghese. Distributed Program Checking: a Paradigm for Building Self-stabilizing Distributed Protocols. In *Proc. FOCS*, 1991, 258-267.

[13] S. Dolev, M. Gouda, and M. Schneider. Requirements for silent stabilization. *Acta Informatica*, 36(6): 447-462, 1999.

[14] Beauquier, J., Delaet, S., Dolev, S., and Tixeuil, S., "Transient Fault Detectors". Proc. of the 12th International Symposium on DIStributed Computing, Springer-Verlag LNCS:1499, pp. 62-74, 1998.

[15] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman and D. Peleg. "Labeling Schemes for Tree Representation". *Proc. 7'th Int. Workshop on Distributed Computing.* Dec. 2005.

[16] D. Angluin. Local and global properties in networks of processes. In *Proc. 12th ACM Symp. on Theory of Computing*, 82–93, May 1980.

[17] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Comm. ACM*, 17(11):643–644, November 1974.

[18] S. Dolev, A. Israeli, and S. Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Computing Journal*, 7,1, 3-16, 1993.

[19] S. Dolev, A. Israeli, and S. Moran. Uniform dynamic self-stabilizing leader election. *IEEE Trans. Parallel Distrib. Syst.* 8(4): 424-440 (1997).

[20] S. Even. *Graph Algorithms.* Computer Science Press, 1979.

[21] M. Garey and D. Johnson. Computers and Intractability. W.H. Freeman and Company, New York, 1979.

[22] Nathan Linial. Distributive Graph Algorithms-Global Solutions from Local Data. FOCS 1987: 331-335.

[23] Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, 3:609–678, 1993.

[24] Fabian Kuhn, Thomas Moscibroda, Roger Wattenhofer. What cannot be computed locally! PODC 2004: 300-309.

[25] Andrew C. Yao. Some Complexity Questions Related to Distributed Computing. Proc. of the 11th ACM Symposium on Theory of Computing (STOC), 1979, 209-213.