Proof Labeling Schemes*

Amos Korman † Shay Kutten‡ David Peleg§

November 8, 2008

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 vertex needs to communicate. The model includes the definition of a proof labeling scheme (a pair of algorithms- one to assign the labels, and one to use them to verify that the global property holds). In addition, approaches are presented for the efficient construction of schemes, and upper and lower bounds are established on the bit complexity of schemes for multiple basic problems. The paper also studies the role and cost of unique identities in terms of impossibility and complexity, in the context of proof labeling schemes.

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 the verification to be less costly sometimes, 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 it, 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.

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

*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.

1 Introduction

This paper addresses the problem of locally verifying global properties in a distributed system. This task complements the one of locally computing global functions. Since many functions cannot be computed locally [40, 36, 38], local verification may, potentially, be even more useful than local computing - one can compute globally and verify locally.

In terms of sequential time, there is evidence that verification is sometimes easier than computation. 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 legal 3 coloring [19]. As another example, consider the *MST verification* problem, where given a weighted graph $G(V, E)$ together with a tree that spans it, it is required to decide whether this tree is an MST of the graph. This problem was introduced by Tarjan in the sequential model. A deterministic linear ($O(|E|)$) time algorithm for computing an MST is known only in certain cases (although the problem does have a linear time randomized algorithm) [20, 30]. On the other hand, the deterministic sequential verification algorithm of [14] runs in linear time.

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, in the case of MST, suppose each vertex marks one of its incident edges, say, by holding a pointer pointing at it). It is required to verify the legality of the represented solution (in the example, to verify that this collection of marked edges forms an MST). Does the verification consume fewer communication bits than the computation of the solution (e.g., the MST) itself?

Since faults are much more likely to occur in a distributed setting than in a sequential one, the motivation for verification in a distributed setting seems to be even stronger than in a sequential one. A common application of local distributed verification is in the context of self stabilization. See, for example, the notions of *local detection* [2], *local checking* [7] or *silent stabilization* [15]. Self stabilization deals with algorithms that must cope with faults that are rather severe, though of a type that does occur in reality [27, 28]. The faults may cause the states of different vertices to be inconsistent with each other. For example, the collection of marked edges may not be a tree, or may not be an MST. Self stabilizing algorithm thus often use distributed verification repeatedly. If the verification fails, then the output is recomputed in a distributed manner. An efficient verification algorithm thus yields repeated savings in communication.

One may consider several models for local verification. For example, we define one such natural model which assumes a synchronous environment. In this model, all vertices are waken up simultaneously and start a computation. It is required that the represented solution is illegal iff after at most t time units ("rounds"), at least one processor outputs 0 (even if the rest output 1). We refer to this model as the *t-simple model*. Since we want the locality parameter t to be independent of the network, it would be desired to let t be a constant.

Note that even for a constant t (actually, for $t = 1$), many representations can be trivially verified.

For example, in the task of verifying a legal coloring, each vertex just checks that each of its neighbors has a different color than its own. As another example, in a simple distributed representation of a maximal independent set (MIS), each vertex holds a flag indicating whether it belongs to the MIS or not. Clearly, such an MIS representation can be verified in one time unit in the t-simple model.

However, there exist some representations that cannot be verified in one time unit. For example, for the MST verification problem, it can be easily shown that verifying that a collection of marked edges is an MST, or even just a spanning tree, would require t to be $\Omega(n)$. Therefore, this example cannot be captured by the simple model of local verification.

In order to perform local verification tasks such as the verification of spanning trees and the verification of MSTs, we introduce here the concept of *proof labeling schemes*, which is a generalization of the simple model for local verification described above. (The formal definitions are given in Section 2 and an example is given in the next paragraph.) Informally, it is assumed that the *state* of every vertex has already been computed by some algorithm (in the spanning tree or the MST examples, the state may consist of pointers to some incident edges; these edges are referred to as *selected*). The configuration (formed as the collection of states of all vertices) is supposed to satisfy some predicate (e.g., "the set of selected edges form an MST of the underlying graph"). In order to perform the verification, labels are assigned to the vertices in a prepossessing stage. To perform the verification, a vertex computes some *local* predicate, considering only its own state, as well as the labels of its neighbors *but not their states*. The global configuration predicate (e.g., "the set of selected edges form an MST") is implied by the conjunction of the local predicates in the following manner. If the configuration is legal, then given the labels assigned in the preprocessed stage, in one time unit (round), no vertex detects a problem, i.e, each vertex outputs 1. However, if the configuration is illegal, then for *every* possible way of labeling the vertices, at least one vertex detects a problem, i.e., in one time unit, at least one vertex outputs 0. This, in a way, means that if the configuration is illegal, then no the adversary can fool the verifier by changing the labels. The restriction of one time unit can obviously be generalized to t time units (hopefully, t being a constant). However, all our results hold for the case $t = 1$.

Let us demonstrate a proof labeling scheme by returning to the example of verifying that a subgraph (defined by the states) is a tree. For the sake of the example, let us verify an easier predicate, which is that the subgraph is a acyclic (i.e., the subgraph is a forest, which contains one or more trees). Informally, the subgraph is given in a distributed manner by the state of each vertex selecting one incident edge at most. This induces an orientation on the selected edges. If the subgraph is indeed acyclic, then in each connected component (which is a tree), each edge is selected by precisely one of its end-vertices, except maybe one edge (which is selected by both end-vertices). For a given connected component, if each of its edges is selected by precisely one vertex, then precisely one vertex in it does not select any edge. We call this vertex the *root* of the connected component. Otherwise, if one edge is selected by both end-vertices, we choose one of them arbitrary and refer to it as the *root* of the connected component.

The label of a vertex is now its distance (in the subgraph) from the root of the corresponding

connected component. The predicate is verified by checking that at each vertex v, the following three conditions hold. (a) if v does not select one of its edges then its label is zero, (b) if the label at v is zero then either v does not select any of its edges or it selects an edge (v, w) which is also selected by the other end-vertex w and the label at w is 1 , (c) if the label at v is not zero then the label at v is larger than that of its parent (the vertex pointed by v). This construction follows from [2, 9, 7, 8, 16, 17, 6, 5, 3] that present self stabilizing algorithms. Clearly, if every vertex obeys the above rules, then the subgraph is a acyclic.

Let us highlight a major difference between the models used for previous self stabilization algorithms and the model of proof labeling schemes. In the former models, 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 has the advantage that it may lead to low cost local verification. However, it 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 our proof labeling schemes, it is assumed that the distributed representation of the structure or function at hand is already given, and the computed labels are 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. This approach of proof labeling schemes may in some cases be useful also in verifying properties of 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 consider one of the results in this paper. We show that the maximum size (in bits) of a label must sometimes be larger even than the maximum size of a state. (Recall that in previous local checking methods, the states had the same role played by labels here, so there, it could not happen that the label was longer than the state.) This occurs in the natural setting where vertices are required to have distinct states. One of the results in this paper is that, in an n-vertex path, in order to verify that all the states are unique, the maximal number of bits in a label is $\Omega(n)$. (This result still holds even if every vertex is assumed to have some fixed unique identity, but the variable state is not necessarily a function of the identity). This $\Omega(n)$ lower bound is larger than the number of bits in a state, which is $O(\log n)$. On the other hand, had a computation been allowed to choose the states (rather than prove the given states), labels of size $O(\log n)$ bits would have sufficed. In fact, in case it is assumed that each vertex holds some fixed unique identity, there is a trivial proof labeling schemes that does not use a label at all (or, uses a label of sizes zero): 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" proof labeling schemes exist even for our stronger requirements from a verification scheme.

Note that the number of bits in a label is the number of information bits a vertex needs to convey to its neighbors in the verification process. Ideally, this number is as small as possible, even smaller than the number of bits used in a vertex state. Indeed, we evaluate a proof labeling scheme by its *label size*, i.e., the maximum number of bits assigned to a label of a vertex in a graph.

Finally, the notion of proof labeling schemes may be made clearer by comparing it to the notion of a *witness* in complexity theory. Informally, the collection of assigned labels in the prepossessing stage can be considered as a witness. If the configuration is legal, then there exists a witness (labeling assignment) such that the legality of the configuration can be verified in one round. Otherwise, if the configuration is illegal, then no such witness exists, i.e., for any labeling assignment, in one time unit, at least one vertex detects a problem.

Related Work: The measure of the label size is related to the problem of the communication complexity [48]. Some of our results about lower bounds are for impossibility of tasks in anonymous networks. Results of that nature concerning computation (rather than verification) were presented in [4] and follow up papers. Some constructions we use simulate a distributed algorithm on every vertex; a related operation was used in [1]. Self stabilization was introduced in [13]. Self stabilization by local detection, and by the similar variant local checking was introduced in [2, 9, 7, 8]. These papers, as well as many others, e.g., [16, 17, 6, 5, 3], present self stabilizing algorithms for computing trees using local detection. We prove a lower bound for such a verification. A lower bound for the special case that the tree is a spanning one is given in [15] (the proof therein is for a different model, but it may be translated into our model).

The concept of silent stabilization and some lower bounds for it are presented in [15]. 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 schemes. This is not always true though, and 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 silent stabilization model of [15]. Our typical lower bound is for the task of verifying *any* configuration. In contrast, in [15] it is assumed implicitly that the configuration is chosen expressly in such a way that it will have a short label.

There are also some other differences between the models. In [15, 9, 8], it is assumed that each neighbor of a vertex u may read a different part of the state of u, that is, the state of a link port of a neighboring vertex. In contrast, our model assumes that all the neighbors of a vertex v can see *the same label* of v. One may say that this models a local broadcast media (e.g., radio) versus point to point lines, which are modeled by the above papers. 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 vertex 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, since all the neighbors see the label, it is harder to tell which of them is the target of the communication. Nevertheless, we show some schemes of very small label size for problems that seem to need targeted communication.

In [10], it is assumed that a vertex can read the *output* of near-by vertices. That is, only the part of the state meant to be visible to the outside can be read by other vertices. (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 [10] it is shown that under their assumption, large amounts of information must sometimes be stored (e.g., for verifying a spanning tree).

Compact label-based representations of spanning trees appear in [11]. The concept of assigning labels in a prepossessing stage in order to ease local queries and computations appeared in various contexts. For example, standard frameworks for routing schemes (e.g., [41, 22, 45, 42]) and informative labeling schemes (e.g., [29, 31, 23, 43, 44]) are based on this concept. Recently, several papers have considered this concept for other tasks such as broadcast [24], exploration [12, 25] and local computation [26]. In this paper, we use this concept for verification. In a recent paper [35], tighter bounds for locally verifying MST were given in some cases.

Our Results: This paper models and partially answers the question: "how easy is the verification task by itself" with respect to basic building blocks in distributed systems. A suitable model is introduced in Section 2, in which this question is studied in terms of the number of bits a vertex needs to reveal to its neighbors. In Section 2.2, we illustrate our model by presenting some basic proof labeling schemes. In particular, we study a proof labeling scheme for verifying a representation of a spanning tree and show that the label size is $\Theta(\log n)$, where n in the number of vertices in the graph. We also present a non-trivial constant size proof labeling scheme for another problem (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 the role of unique identities in terms of impossibility and complexity. We show that there exist problems and graph families for which no proof labeling scheme exists if unique identities are not 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 $\Omega(n)$. We also show that this lower bound on verifying the uniqueness of the states holds even if each vertex is assumed to have a unique identity (that is not considered to be a part of the state).

Additional evidence for the importance of the role played by identities in proof labeling schemes is provided by a result we present regarding identity invariance. The question under study follows a result of [40], showing in a particular setting that, intuitively, the actual value of the identities does not matter. More specifically, the result of [40] deals with functions that can be *computed* locally by a vertex, just by looking at states of the neighboring vertices. The following claim is proved in [40] (for their model): if there exists an algorithm for computing a certain function, then there exists an *identity 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 values. Our setting bears a lot of resemblance to that of [40]. Nevertheless, we show that this claim is not true in our model.

In Section 4, we show how to build verification systems systematically and modularly by suggesting two rather 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.

2 Preliminaries

2.1 Definitions

Let $G = \langle V, E \rangle$ be a connected graph. (For the sake of simplicity, the graphs discussed in this paper have no self loops and no parallel edges.) Some of our results deal with the case where each vertex $v \in V$ has a unique identity. For the sake of unifying the definitions, we assume a (possibly empty) *identity* field $id(v)$ for each vertex $v \in V$. Unless mentioned otherwise, we always assume that for any vertex $v \in V$, $id(v)$ is encoded using $O(\log |V|)$ bits.

In addition to its id, each vertex v is associated with a state $s(v)$ taken from a set S of possible *local states*. (Informally, a state may include an input received by the vertex from the environment, as well as the results of some distributed algorithms executed on the inputs and on the network graph.) The possible *global states* (configurations) are all the elements in the Cartesian product of the local states. We use the following notations. Fix S as some set of possible local states, and let $G = (V, E)$ be a graph. For every vertex $v \in V$, let v_s be a triplet $v_s = (v, s(v), id(v))$, where $s(v) \in S$. Informally, v_s corresponds to the vertex v with state $s(v)$ and identity $id(v)$. A *configuration graph* corresponding to the graph G (and a global state) is a graph $G_s = (V_s, E_s)$, where $V_s = \{v_s \mid v \in V\}$ and $(v_s, u_s) \in E_s$ iff $(v, u) \in E$. Consider any graph family $\mathcal{F}$ (note, $\mathcal{F}$ may contain graphs of different sizes). The *family of configuration graphs* $\mathcal{F}_S$ consists of all configuration graphs $G_s \in \mathcal{F}_S$ corresponding to each $G \in \mathcal{F}$ (note, every state s is taken from the given set S). In what follows, whenever the set S is left unspecified, it is assumed to be the set of integers $\mathbb{N}$. When the context is clear, we may also omit the subscript S altogether, and write simply $\mathcal{F}$ instead of $\mathcal{F}_{\mathbb{N}}$.

A configuration graph G_s is *id-based* if $id(u) \neq id(v)$ for every pair of distinct vertices v and u. 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 $\mathcal{F}^{undirected}$ (respectively, $\mathcal{F}^{directed}$) be the collection of all undirected connected (resp., directed strongly-connected) n-vertex graphs, and let $\mathcal{F}^{all} = \mathcal{F}^{undirected} \cup \mathcal{F}^{directed}$. 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 vertex states. There can be many such representations. For simplicity, we focus on representations where an edge is included in the subgraph if it is pointed at explicitly 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[1] of one of v's ports pointing at 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 and a given set S of local states. A distributed problem $Prob$ is the task of selecting a state $s(v) \in S$ for each vertex v, such that G_s satisfies a given predicate f. This induces the problem $Prob$ on a graph family $\mathcal{F}$ in the natural way. We say that f 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. Informally, a proof labeling scheme includes a *marker* algorithm $\mathcal{M}$ that generates a label for every vertex, and a *decoder* algorithm $\mathcal{D}$ that compares labels of neighboring vertices. If a configuration graph satisfies f, then the decoder at every two neighboring vertices must declare their labels (produced by marker algorithm $\mathcal{M}$) "consistent" with each other. However, if the configuration graph does *not* satisfy f, then for *any possible* labeling assignment, the decoder must discover "inconsistencies" between the labels of *some* neighboring vertices. It is not required that the marker algorithm be distributed. However, the decoder is distributed, i.e., every vertex can check only the labels of its neighbors (and its own label and state). A vertex cannot see the states of its neighbors.

Let us now give a more detailed description. A *labeling* L is an assignment of a label $L(v_s)$ to each vertex $v_s \in G_s$. Informally, each vertex v can "see" its own state and label as well as the labels of each of its neighbors u. A vertex v cannot "see" the state of any neighbor u. The motivation for this is that we would like to see whether it is enough to communicate (to the neighbor) fewer bits than required to describe the state. In other words, our goal is to identify problems in which the number of bits in the label can be smaller than the number of bits used to describe the state. (If desired, however, the model does allow a marker algorithm to include the state of a vertex in its label; this, of course, may lead to large labels). In addition to the label of a neighbor u, vertex v can also see its 'physical connection' to u, i.e., the corresponding port number and weight of the edge (v, u). Informally, this captures the fact that many distributed algorithms make their local decisions at a vertex based also on these parameters.

Formally, for a labeling L and a vertex $v_s \in G_s$, let $N'_L(v)$ be a set of $deg(v)$ fields, where $deg(v)$ is the number of neighbors of v. Each field $e = (v, u)$ in $N'_L(v)$, corresponding to edge $e \in N(v)$, contains the following.

[1] We assume that the states have some particular structure, in which it is known how to extract the encoding.

- The port number of e in v (which is not considered to be a part of the state of v),
- The weight of e (regarded as 1 if G is unweighted),
- $L(u)$.

Let $N_L(v) = ((s(v), id(v), L(v)), N'_L(v))$.

A *marker* algorithm $\mathcal{M}$ is an algorithm that given a graph $G_s \in \mathcal{F}_S$, constructs a labeling for the graph. For notational simplicity, we refer to this labeling as $\mathcal{M}$ as well, namely, we denote the label assigned by the marker algorithm $\mathcal{M}$ to the vertex $v_s \in G_s$ simply by $\mathcal{M}(v_s)$. A *decoder* algorithm $\mathcal{D}$ is an algorithm which is applied separately at each vertex $v \in G$, with local input $N_L(v)$ and Boolean local output $\mathcal{D}(v, N_L(v))$. To simplify the notation, we often use the notation $\mathcal{D}(v, L)$ instead of $\mathcal{D}(v, N_L(v))$.

Let $\mathcal{F}_S$ be some family of configuration graphs corresponding to some family $\mathcal{F}$. Intuitively, $\mathcal{F}_S$ allows one to constrain the domain of the problem to specific assumptions about topology, state space, and identities. Let f be some characteristic function of a problem over $\mathcal{F}_S$. A *proof labeling scheme* $\pi = (\mathcal{M}, \mathcal{D})$ for f over $\mathcal{F}_S$ 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 labeling L there exists a vertex $v \in G$ so that $\mathcal{D}(v, L) = 0$.

Note that the first property above concerns only the labeling produced by the marker algorithm $\mathcal{M}$, while the second speaks of arbitrary labelings L, including ones that could have been produced by an adversary attempting to fool the decoder.

The *size* of a proof labeling scheme $\pi = (\mathcal{M}, \mathcal{D})$ is the maximum number of bits in the label $\mathcal{M}(v_s)$ assigned by the marker algorithm $\mathcal{M}$ 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 f over $\mathcal{F}_S$.

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 $\Theta(m)$.

Let us start with a simple and natural problem that has a constant proof size while the corresponding computation task requires $\Omega(\log n)$-bit states.

The anonymous tree orientation problem: Given a tree T, f_{orient} characterizes the task of assigning states to the vertices of T so that $D(G_s)$ (as defined in Section 2.1) induces an orientation on the edges (towards some root whose identity is not given to the vertices). Let $\mathcal{F}^{anon-trees}$ be the family of anonymous trees. In particular, assume that the orientation is represented by storing at each vertex the number of its port that points at its parent. Clearly, the distributed representation of the orientation (that is, the computation task) requires $\Omega(\log n)$ bits per vertex in the worst case (i.e., when port numbers are assigned to ports by an adversary).

Lemma 2.1 *The proof size of f_{orient} over $\mathcal{F}^{anon-trees}$ is $O(1)$.*

Proof: We describe a proof labeling scheme $\pi = (\mathcal{M}, \mathcal{D})$ as required. Let G_s be an anonymous tree s.t. $f_{orient}(G_s) = 1$. Let r be the intended root, namely, the unique vertex whose state does not encode any of its port numbers. The marker algorithm $\mathcal{M}$ assigns each vertex v a label $\mathcal{M}(v)$ consisting of v's distance from r modulo 3. On a labeling L, 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. $L(u) + 1 \equiv L(v) \bmod 3$ 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 $f_{orient}(G_s) = 0$. One case is that 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 Condition 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. ▌

The spanning tree problem in id-based families: Let us consider some variants of an example that is known in the area of self stabilization, and discuss some of the differences. Let $f_{acyclic}$ characterize the task of assigning states to the vertices of G so that $D(G_s)$ induces an acyclic graph.

Lemma 2.2 *The proof size of $f_{acyclic}$ over $\mathcal{F}^{all}$ is $\Theta(\log n)$.*

Proof: The upper bound is proven by adapting the ideas of [5, 3, 6, 1, 2, 17] to our model. Let us just sketch that proof for upper bound; the reader is referred to [34] for the details.

Note that an acyclic graph is simply a collection of trees. To verify the acyclicity property, each vertex is labeled by its distance from its root in the corresponding tree. The decoder verifies that the distance of the child is larger (by one) than the distance of the parent. Moreover, the label of a vertex

is zero iff it doesn't have a parent. It is easy to get convinced that if this property holds everywhere, then there are no cycles.

Let us next prove the lower bound. Let P be the path of n vertices. For the simplicity of the description only, assume that the path is embedded in the plain, such that vertex i lies left of vertex $i+1$ for $i = 0, 1, ...n-1$. (The names i, $i+1$, etc. are used for the sake of the analysis only, and are not accessible to the vertices). Let us term such a path a *horizontal* path. For $i < n$, let $s(i)$ be the port number of the edge leading from vertex i to $i+1$. Obviously, $f_{acyclic}(P_s) = 1$. Assume, by way of contradiction, that there exists a proof labeling scheme $\pi = (\mathcal{M}, \mathcal{D})$ for $f_{acyclic}$ over $\mathcal{F}$ (the family of such paths), which is of size less than $\frac{1}{2}\log(\frac{n}{2} - 2)$. Let $\mathcal{M}(i)$ be the label given by $\mathcal{M}$ to vertex i in the above path P_s. Since the number of bits in each $\mathcal{M}(i)$ is less than $\frac{1}{2}\log(\frac{n}{2} - 2)$, there exist two pairs of vertices $(i, i+1)$ and $(j, j+1)$ where $1 < i$ and $i+1 < j < n-1$ so that $\mathcal{M}(i) = \mathcal{M}(j) = \mathcal{M}'$ and $\mathcal{M}(i+1) = \mathcal{M}(j+1) = \mathcal{M}''$. We now build the following ring R consisting of $j - i$ vertices whose identities are clockwise ordered from $i+1$ to j. For $i < k \leq j$, let $s(k)$ be the port number of vertex k leading from k to $k+1$ and let $s(j)$ be the port leading from j to $i+1$. Let us give R_s the same labeling L as $\mathcal{M}$ gives P_s, i.e., each vertex $i+1 \leq k \leq j$ in R_s is labeled $\mathcal{M}(k)$. By the correctness of π on P_s we get that for each vertex $v \in R_s$, $\mathcal{D}(v, \mathcal{M}) = 1$. This is a contradiction to the fact that $f(R_s) = 0$ and $f'(R_s) = 0$. ▮

Remark: There are a number of variants for the problem. For example, in maintaining a spanning acyclic graph, the scheme needs to ensure also that the subgraph indeed includes all the vertices. In addition, when maintaining a spanning tree, the scheme needs to ensure also that the subgraph is connected. These somewhat different requirements imply some differences in the proof labeling schemes. Still, it is easy to modify the scheme used in our upper bound so that it can be used for the other variants mentioned here. Similarly, the lower bound proof also holds. Another variant, for which a similar scheme can be constructed, is the problem of shortest path trees over weighted id-based graphs (one difference is that the size of the scheme changes to $O(\log n + \log W)$).

Note that our lower bound is stronger than the one previously presented in the self stabilization context (cf. [15]); that lower bound showed that the verification of a *spanning tree* needs a large state (recall that in [15] the states played the role played here by the label). In contrast, our lower bound applies to the size of labels required for verifying the cycle freedom property alone (even, say, for a tree is not required to span the entire network).

Agreement in anonymous families Problem: The next example is still very simple (especially given the known linear lower bound on Equality in communication complexity [48]). We present it since we believe that it makes an important general point. Indeed, we make use of it later in this paper. Note that v's neighbors cannot 'see' the state of v but they can see v's label (that is, the input of the decoder at a neighbor u of v does not include $s(v)$, but it does include $L(v)$).

Lemma 2.3 *Let $f_{agreement}$ be the characteristic function of the problem of assigning all the vertices identical states from some finite set of integers S. The proof size of $f_{agreement}$ over $\mathcal{F}^{all}$ is $\Theta(\log |S|)$.*

Proof: Let $m = \lceil \log |S| \rceil$. We first describe a trivial proof labeling scheme $\pi = (\mathcal{M}, \mathcal{D})$ of size m. For each i, let s_i denote the i'th smallest element in S, and for each element $s \in S$, let $index(s)$ be such $s = s_{index(s)}$. Given G_s such that $f_{agreement}(G_s) = 1$, for every vertex v, let $\mathcal{M}(v) = index(s(v))$, where $s(v)$ is the state of v. Then, $\mathcal{D}(v, L)$ simply verifies that $L(v) = index(s(v))$ and that $L(v) = L(u)$ for every neighbor u of vertex v (the fact that $L(v) = index(s(v))$ can be verified since the decoder knows the set S). It is clear that π is a correct proof labeling scheme for $f_{agreement}$ over $\mathcal{F}^{all}$ 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 $\pi = (\mathcal{M}, \mathcal{D})$ for $f_{agreement}$ over F_S^{all} of size less than $m/2$. For $i \in S$, let G_s^i be G with states $s(u) = s(v) = i$. 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 the marker algorithm $\mathcal{M}$ applied on G_s^i. Let $L^i = (\mathcal{M}^i(v), \mathcal{M}^i(u))$. Since the number of bits in $\mathcal{M}^i(v)$ and also in $\mathcal{M}^i(u))$ is less than $m/2$, the number of bits in L^i is less than m. Hence, there exist $i, j \in S$ such that $i < j$ and $L^i = L^j$. Let G_s be G with states $s(u) = i$ 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 $f(G_s) = 0$. ▮

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

Corollary 2.4 *For every value $1 \leq m < n^2$, there exists a graph problem over an id-based family of n vertices graphs with proof size $\Theta(m)$.*

The proof of Corollary 2.4 can be found in [34]. The question of whether two bit strings are identical (in Lemma 2.3) is transformed to the question of whether two given subgraphs of a given graph are identical.

3 The role and cost of identities

The bounds presented in [15] are similar for id-based and for anonymous families. Our model exhibits a difference 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 distinct is costly. This is an example where verifying a given configuration (as we do in this paper) may require many more bits of communication than the combined approach, that of designing the configuration in such a way that it will be easy to verify. In this section, we also separate our model from the one of [40] by showing in Subsection 3.3 that our model is not order-invariant.

3.1 Anonymous versus id-based families

Consider the family $\mathcal{F}^{anon-cycles}$ of anonymous cycles with 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 either,

1. there exists exactly 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); or
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); or
3. the state of each vertex encodes the number of vertices in G; or
4. the subgraph $H(G_s)$ induced by the states of G_s (recall, this is the subgraph that contains precisely the selected edges) 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 [4].

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

Since the proof is rather standard, let us just outline the main idea. (A detailed proof is found in [34].) **Sketch of proof:** Assume, for example, that there is a proof labeling scheme for the problem where the state of each vertex is the number of vertices. Use the marker algorithm $\mathcal{M}$ of this scheme to label some cycle G with some C vertices, where the state of each vertex is the number n. Now, consider another cycle G' with $2C$ vertices, where the state of each vertex is n. Obviously, the decoder should output zero at some vertex, since the state does not encode the size of the cycle. However, it is easy to show that the decoder can be fooled by a standard argument of showing that it is hard to break symmetry in antonymous networks. That is, assign some consecutive C vertices in G' the same labels produced by $\mathcal{M}$ for the C vertices of G. Then, label the remaining C vertices of G' by the same labeling. It is easy to show that the decoder must return 1 in every vertex of G'_S because it has the same input as in G_S. This means that the decoder in ${G'}_S$ fails to detect that the state does not represent the number of vertices. ▌

Theorem 3.2 *For every computable problem f, there exists a proof labeling scheme on every id-based family $\mathcal{F}$ of connected graphs.*

Proof: The proof is based on a full information approach, namely, encoding all the global information in the label of every vertex, so that it can be checked locally. That is, let f be a computable problem and let $\mathcal{F}$ be an id-based family of connected graphs. We describe a proof labeling scheme $\pi = (\mathcal{M}, \mathcal{D})$ for f over $\mathcal{F}$.

Each vertex is given a label $G(v)$ which is the description of the whole network graph, together with all the inputs of every vertex. In addition (as a separate field in the label) the label of each vertex includes also its identity. Given a characteristic function f, a decoder at each vertex can verify that $f(G(v)) = 1$ for the graph $G(v)$ in the label of v. More formally, given a graph $G_s \in \mathcal{F}$, the marker algorithm $\mathcal{M}$ assigns each vertex v the label $\mathcal{M}(v)$ consisting of two sublabels (fields). The first field, $\mathcal{M}_1(v)$, that contains the entire graph G_s and the second field, $\mathcal{M}_2(v)$, is $id(v)$. The decoder $\mathcal{D}(v, L)$ verifies that $G(v)$ (of the label) is the same as the network graph. For that, the decoder verifies first that all the vertices agree on the label (see Lemma 2.3), and then, the decoder at each vertex v verifies that v and its neighbors are represented correctly in the label. That is, let $G(v)$ be the graph encoded in $L_1(v)$). The decoder outputs 1 iff all the following conditions hold.

1. For every neighbor u of v, $L_1(u) = L_1(v)$.
2. (a) $L_2(v) = id(v)$.
   (b) There exists a unique 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 $G(v)$. (To check that, the decoder at v uses the second sublabels L_2 of v's neighbors.
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 labeling. 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 some family of anonymous cycles. 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 over an n-vertex id-based path is $\Theta(n)$. More formally, let $f_{distinct}$ be the characteristic function of the following problem: assign states to the vertices 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 distinct identities and let $\mathcal{F}^{path}$ be the id-based family containing the single path G, i.e., $\mathcal{F}^{path} = \{G\}$. Assume, that the states assigned to vertices in a path of n vertices are in the set $S_1 = \{1, 2, \ldots, n\}$.

Lemma 3.3 *The proof size of $f_{distinct}$ over $\mathcal{F}^{path}_{S_1}$ is $\Theta(n)$.*

Proof: We prove here the lower bound. The (rather simple) upper bound proof is described in [34]. Informally, the idea behind the upper bound proof is the following. First, we let the vertices agree on a certain orientation using the proof labeling scheme described in Lemma 2.1. Then, the marker algorithm encodes an array containing n fields in the label of each vertex. The i'th field in the array of v indicates whether the identity of v is i, whether the identity i belongs to a vertex to the 'right' of v, or whether the identity i belongs to a vertex to the 'left' of v. Clearly, the size of a label is $O(n)$. The decoder at each vertex v simply verifies that the array of v is consistent with the arrays of its neighbors (and with v's identity).

Let us now turn to prove the lower bound. (It may be possible to derive an alternative proof based on the lower bound for 2-party communication complexity on element distinctness, cf. [46, 37]). Let $\pi = (\mathcal{M}, \mathcal{D})$ be a proof labeling scheme for $f_{distinct}$ over $\mathcal{F}_{S_1}^{path}$. 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'. (Intuitively, the vertices in the middle must be able to compare the set of the vertices on the right to the set on the left in order for their decoders to be correct).

More formally, a *divided permutation* is a permutation σ on $[1, \ldots, n]$ 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 $\{1, \ldots, n\}$ that fixes $n/2$ and $1 + n/2$, identifies with σ_1 on $[1, \ldots, n/2 - 1]$ and identifies with σ_2 on $[2 + n/2, \ldots, n]$. 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 labeling L. Let

$$L(v_i) = \begin{cases} \mathcal{M}_{\sigma_1}(v_i), & 1 \leq i \leq 1 + n/2, \\ \mathcal{M}_{\sigma_2}(v_i), & 2 + n/2 \leq i \leq n. \end{cases}$$

For every divided permutation σ and vertex v, we have $\mathcal{D}(v, \mathcal{M}_\sigma) = 1$. Therefore for all $1 \leq i \leq 1+n/2$, $\mathcal{D}(v_i, L) = \mathcal{D}(v_i, \mathcal{M}_{\sigma_1}) = 1$ 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 $\Gamma = \left(\hat{V}, \hat{E}\right)$ 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 $\lfloor |V(G)|/(d+1) \rfloor$, we conclude that Γ has an independent set X of size $\lfloor \frac{(n-2)!}{((n/2-1)!)^2} \rfloor$. Note that $\log(\frac{(n-2)!}{((n/2-1)!)^2}) = \Omega(n)$. Since X is an independent set of Γ, by the 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 $\mathcal{F}_{S_1}^{path}$ 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 $\mathcal{F}_{S_1}^{path}$, one can choose the state of v_i to be i. Clearly, this can be verified by a scheme whose size is $O(\log n)$ (let the label of each vertex v be equal to v state and verify that, also, agree on an orientation and verify this agreement; finally, the decoder also verifies that the state of the neighbor on the "left" is smaller by 1).

3.3 Variability of identities

Two assignments of identities to vertices of a graph are *order consistent* 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 consistent id assignments to the vertices of G. For example, an algorithm that outputs "true" iff the identity of some vertex is 6, is *not* order invariant. The following is shown in [40] for the model used there. Informally, let A be a local algorithm for an id-based family $\mathcal{F}$ which performs a computational task $\mathcal{C}$ on $\mathcal{F}$, where $\mathcal{C}$ does not depend on the identities of the vertices (see [40] for details regarding their model). Then, there exists an *order-invariant* algorithm A' that solves $\mathcal{C}$ on $\mathcal{F}$ 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 (defined below).

Informally, consider a computational task $\mathcal{C}$ that does not depend on the identities of a vertex. Perhaps surprisingly, the labelings used by any proof labeling scheme for such a task may have to be different for different identities. Moreover, different labelings are needed even for such a $\mathcal{C}$ if an id assignment is replaced by another order consistent id assignment. This may prove useful for establishing lower bounds.

More formally, for a graph $G = (V, E) \in \mathcal{F}$, a *legal id assignment* to G is an assignment of distinct identities to V. An *id independent* function is a function f such that $f(G_s) = 1$ for every legal id assignment. (Note that the above example- the function which outputs "true" iff the number 6 is an id

of a vertex in the graph- is not id independent). An *order invariant* proof labeling scheme $\pi = (\mathcal{M}, \mathcal{D})$ is a proof labeling scheme for which, for every two order consistent (legal) assignments and every vertex v, the label $\mathcal{M}(v)$ is the same under both assignments.

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

Proof: Let $\mathcal{F}$ be the collection of cycles of size at most $2n$ and let $S = \{1, 2, \ldots, 4n\}$. Let f be a Boolean function over $\mathcal{F}$ 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. Clearly, f is id independent. It is easy to see that there exists a proof labeling scheme for f over $\mathcal{F}$. Assume, by way of contradiction, that there also exists an order invariant proof labeling scheme $\pi = (\mathcal{M}, \mathcal{D})$ for this problem. Consider an n-vertex cycle C so that for every vertex v in C, $s(v) = n$. Consider the following four order equivalent identity assignments to C_s.

$$(id_1(v_1), id_1(v_2), \ldots, id_1(v_{n-2}), id_1(v_{n-1}), id_1(v_n)) = (1, 2, \ldots, n-2, 4n-1, 4n).$$

$$(id_2(v_1), id_2(v_2), \ldots, id_2(v_{n-2}), id_2(v_{n-1}), id_2(v_n)) = (2n+1, 2n+2, \ldots, 3n-2, 4n-1, 4n).$$

$$(id_3(v_1), id_3(v_2), \ldots, id_3(v_{n-2}), id_3(v_{n-1}), id_3(v_n)) = (1, 2, \ldots, n-2, 3n-1, 3n).$$

$$(id_4(v_1), id_4(v_2), \ldots, id_4(v_{n-2}), id_4(v_{n-1}), id_4(v_n)) = (2n+1, 2n+2, \ldots, 3n-2, 3n-1, 3n).$$

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

Consider a $2n$-vertex cycle C' in which $s(v) = n$ for every vertex v in C'. Obviously, $f(C'_s) = 0$. Consider the following legal identity assignment to the vertices of C'.
$(id(v_1), id(v_2), \ldots, id(v_{n-2}), id(v_{n-1}), id(v_n), id(v_{n+1}), id(v_{n+2}), \ldots, id(v_{2n-1}), id(v_{2n})) =$
$(1, 2, , \ldots, n-2, 4n-1, 4n, 2n+1, 2n+2, \ldots, 3n-1, 3n)$.
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 this 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 can show an upper bound on the proof size of f. The idea is to generate a scheme that verifies each configuration by imitating an execution of the algorithm that generated this specific configuration. (Hence, an algorithm that implements f but does not generate *all* the configurations.) Informally, we record in the label of a vertex its whole history in the above execution, and verify with the neighbors of the vertex that this history is consistent with their histories. (If the vertex label "remembers" that the vertex sent a message to a neighbor, then the label of the neighbor must "remember" receiving the message).

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, no matter how did the network reach this configuration. In many cases, though, 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 benefit of this 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. Note that this is much smaller than the label constructed in the proof of Theorem 3.2. This construction is based on imitating the steps of a distributed algorithm for constructing an MST. The MST construction algorithm we imitate is a modification of known algorithms. First we needed to modify those algorithms so that the resulting algorithm generates every possible MST (non-deterministically), 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 here that every MST must have a proof labeling scheme). In addition, the method explained below generates smaller labels from the modified algorithm then the labels it would have generated from previous algorithms. (However, the modifications so that the algorithm implies a shorter label, made it impractical to actually use it as a distributed MST algorithm.)

Let us comment that we do not expect that the distributed method to be the best approach for every problem. One reason is because the distributed method bases the verification on the computation, while there exists some evidence that the computation is sometimes harder.

Consider some graph family $\mathcal{F}_{S'}$ and a problem f, and let A be a non-deterministic distributed algorithm on $\mathcal{F}_{S'}$ 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 $A'(G)$, $f(A'(G)) = 1$. In the specific case that A is synchronous we make additional assumptions bounding the complexity of A for any $G \in \mathcal{F}_{S'}$. The total number of

rounds of A in this case is assumed to be bounded from above by some τ and the number of messages per round and the number of messages a vertex sends per round is assumed to be bounded from above by some m_p.

Theorem 4.1 *Let $\mathcal{F}_{S'}$ and f be a graph family and be a problem, respectively, and let A be a non-deterministic distributed algorithm as described above. Then the following holds.*

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 $\min\{O(m_0(\log \tau + \log n)), O(\tau \cdot m_p \log n)\}$.*

Proof: Let us first construct the labeling for the schemes claimed in the theorem. Consider a graph G_s such that $f(G_s) = 1$ and let $A'(G)$ be a specific run of $A(G)$ as described above. For the scheme in the first part of the lemma, enumerate the (up to nm_0) messages according to their chronological order, breaking ties arbitrary. The marker algorithm $\mathcal{M}_A^0$ assigns to each vertex v a label $\mathcal{M}_A^0(v)$ with up to m_0 fields, each containing three subfields, as follows. The first subfield of field i contains the content of the i'th message v sends in $A'(G)$. The second subfield contains the identity of the vertex 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.

For the second part of the lemma, construct two schemes π_A^1 and π_A^2. The labeling in scheme π_A^1 is the same as the labeling $\mathcal{M}_A^0(v)$ above, except that the third subfield in every field i contains the round number in which v's i-th message (corresponding to that field) was sent. The label $\mathcal{M}_A^2(v)$ of the third scheme has τ fields. Each field i contains one subfield for each 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.

Given a labeling L, all three decoders verify the following in each vertex v.

(i) 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).

(ii) There exists some run of A at v such that the one that is described in (i) above, and the result of A at v is $s(v)$.

Clearly, if at each vertex v the above items are verified, then the labels implicitly describe a legitimate run of the algorithm A on G, such that the run ends with the output $s(v)$ at each vertex v. Since $f(A(G)) = 1$ for every run of A, the lemma follows. ∎

Comment 4.1 *We assumed that the length of a message is $O(\log n)$ (see item 3 above) to simplify the results. If messages are shorter, the $O(\log n)$ terms in the statement of the theorem remain unchanged.*

This is because the labels of the schemes constructed in the above proof have fields each containing a message and an identity. Recall, that we assumed that the length of an identity is $O(\log n)$. If one assumes that the number of bits in a message is some X which is not $O(\log n)$, then the term $\log n$ in the results above must be replaced by X.

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

Theorem 4.2 *There exists a proof labeling scheme $\pi_{mst} = (\mathcal{M}_{mst}, \mathcal{D}_{mst})$ for f_{MST} over $\mathcal{F}^{undirected}(W)$ of size $O(\log^2 n + \log n \log W)$.*

Here, we do not assume that the edge weights are unique. Recall, however, that in this section, we assume unique vertex identities. Hence, verifying an MST is possible by Theorem 4.1. However, the use of Theorem 4.1 does not suffice to prove Theorem 4.2, since the known distributed MST construction algorithms sometimes require some vertex v to send $\Omega(\triangle(v))$ messages in many rounds. Substituting this in Theorem 4.1 yields a proof labeling scheme of higher size than desired.

Let us first give a high level description for the proof of Theorem 4.2. Informally, it turns out that a minor change in Theorem 4.1 (and in its proof) suffices to prove Theorem 4.2. The label of a vertex v constructed in the proof of Theorem 4.1 includes one $O(\log n)$ bit field per message sent by v to a neighbor. In the case that v sends identical messages to all its neighbors, multiple fields are added to the label of v. We want to add just one field in such a case. In the following, we use the property of our model that allows a vertex to "see" the labels of *all* its neighbors at no cost.

In the revised scheme, only one field is added to the label of v in such a case that v sends the same message at the same time to all of its neighbors. (If v sends this message to many of them, we assume it sends the message to all of them at no additional cost). The field corresponding to such a message (a *local broadcast* message) is marked as a *local broadcast* field. Let us term a message sent to a single neighbor a *local unicast* message. The size of the label generated by the distributed method for one round now includes the following (1) $O(\log n)$ bits per round for the local broadcast message (of $O(\log n)$ bits) that v can send in this round, plus (2) $O(\log n)$ bits per round for each local unicast message that v sends in that round (these second kind of labels are the same as the labels produced for the synchronous cases in Theorem 4.1).

We now use a simplified synchronous version of the classical asynchronous distributed MST construction algorithm of [21], termed Algorithm Sync_GHS. This algorithm, presented in [42], operates in $O(\log n)$ phases. In each phase, each vertex sends $O(1)$ local unicast messages plus $O(1)$ local broadcast messages. To prove Theorem 4.2, it is then left (1) to prove the above complexity of algorithm Sync_GHS, and (2) to show that algorithm Sync_GHS can generate all possible MSTs, rather than just some specific MST.

The message complexity analysis of algorithm Sync_GHS, as presented in [42], distinguishes between two message classes, one referred to hereafter as local broadcast messages, namely, messages used by

a vertex to inform its neighbors of the identity of its component, and the other, hereafter regarded as *local unicast messages*, consisting of the rest of the messages. Since the (asynchronous) GHS algorithm is widely known, we shall not give a complete description of the algorithm or its analysis here. (A more detailed proof of Theorem 4.2 appears in [34].) However, for completeness we provide a high level description of the algorithm.

The Sync_GHS algorithm starts when each vertex is a (non-spanning) tree in itself, termed a *fragment*. So, the network is covered by a forest of fragments. During the execution, fragments are merged, such that the property that each is directed towards a root is kept. An edge whose one endpoint belongs to a fragment F and the other belongs to another fragment is termed *outgoing edge*. In each *phase*, each fragment merges with other fragments by adding its minimum weight outgoing edge. Finally, one fragment (the MST) spans the network.

For each phase, each vertex v maintains the *name* of its fragment, that is the identity of the root of the fragment in that phase. Vertex v informs its neighbors of this fragment name. Hence, each neighbor can detect whether its edge to v is outgoing (this is the case that the root of the neighbor is different than the root of v). The messages informing the neighbors of the fragment's root identity are those that we call here the local broadcasts. In the complexity analysis there, those messages are counted separately from the other messages.

Finally, it is well known that if the weights of the edges are unique, then there exists only one spanning tree. If weights are not unique, then a fragment may have several minimum outgoing edges. It is known that each different MST results from different selections of the minimum outgoing edges, and vice versa, if only minimum outgoing edges are selected, then the result is an MST. Hence, the Sync_GHS algorithm can be made to output any MST, as required here (when a fragment has several outgoing edges of identical weight, one possible execution chooses the one that is included in the given MST). This completes the sketch of the proof of Theorem 4.2.

For the sake of completeness, we describe a proof labeling scheme explicitly. The description of the proof labeling scheme that results from the Sync_GHS algorithm explicitly would be somewhat long (though shorter than that of a scheme that would result from the original asynchronous algorithm of [21]). The following is based on an "expedited" version of algorithm Sync_GHS. This version "expedites" each phase of the algorithm to just one round, using some oracle that cannot be implemented by a "real" distributed algorithm. The oracle will "save" the distributed computation for finding a minimum outgoing edge in Sync_GHS. Still, the verification by the proof labeling scheme will show that a minimum outgoing edge was identified correctly by the oracle. The correctness will then follow from that of the known MST construction.

Proof of Theorem 4.2: By the remark after Lemma 2.2, we know that the size of a scheme verifying that the subgraph induced by the states is a spanning tree is $O(\log n)$. Therefore, in the remaining, we assume that the subgraph induced by the states is a spanning tree T and focus on verifying that T is in fact an MST.

For every vertex v, the i'th field in $\mathcal{M}(v)$ (that corresponds to round i in the algorithm) has a subfield $\mathcal{M}_1^i(v)$ that contains the identity of $parent(v)$, the parent of v in the i'th round of algorithm Sync_GHS. (If v is the root of its fragment, then we say that v is its own parent and therefore, $\mathcal{M}_1^i(v)$ simply contains the identity of v; In particular, for every vertex v, $\mathcal{M}_1^1(v)$ contains just the identity of v.) For each i, $F_i = \{(v, \mathcal{M}_1^i(v)) \mid v \in V\} \equiv \{(v, parent(v)) \mid v \in V\}$ is a collection of distinct fragments where $F_{\log n}$ is T itself.

Obviously, F_1 is just the collection V (with no edges). Now, given F_i, let us compute F_{i+1}. 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 $u \in F$. The algorithm redirects the edges of F towards u. This action is taken also in the algorithm of [42] (and of [21]), but there it requires a not very short distributed computation. Instead, here we assume that some oracle detects immediately which of the edges of all the vertices in the fragment is a minimum outgoing one. (However, the correctness of this detection will be verified by the proof labeling scheme). We call the redirected fragment $SPAN(F)$. Note that the parent of v in $SPAN(F)$ may be different from its parent in F. We term v's parent in $SPAN(F)$ the *min-edge direction* of v.

To imitate the algorithm, for each vertex $v \in F$ where $v \neq u$, let $\mathcal{M}_2^i(v) = (id(u), id(y), d_{SPAN(F)}(v, u))$, where y is v's parent in $SPAN(F)$ (that is, the second field in $\mathcal{M}_2^i(v)$ is the min-edge direction of v). We also 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). Also, let $\mathcal{M}_3^i(v) = w(e(F))$ (this is used to verify that $e(F)$ is indeed minimum).

Now, the algorithm connects each $SPAN(F)$ to the tree on the other side of $e(F)$. This is imitated in the labels as follows. Let $\mathcal{M}_1^{i+1}(v)$ be the second field of $\mathcal{M}_2^i(v)$. (That is, v's parent in the next round $i+1$ is the next vertex on the way to the minimum outgoing edge; or, if $v = u$ is the endpoint of that edge, then its parent is the other endpoint, unless the other endpoint has a lower identity and its min-edge direction is u itself).

This completes the details of how the marker algorithm imitates the MST construction algorithm. Given a labeling L, the decoder at each vertex v checks for each round, that (1) that $\mathcal{L}_1^1(v)$ is indeed v's identity and $\mathcal{L}_3^1(v)$ is v's minimum weight edge; (2) that $\mathcal{L}^{i+1}(v)$ is consistent with $\mathcal{L}^i(v)$ (v's parent in the $i+1$-st round is v's min-edge direction in the i-th round as described above); (3) that the relation (vertex, parent) (that is, $(v, \mathcal{L}_1^i(v))$) forms a directed tree (this is verified similarly to the proof of Lemma 2.2); (4) that v agrees with its min-edge direction vertex on the weight of the minimum outgoing edge $e(F)$; (5) that v does not have an outgoing edge that is lighter than $e(F)$ (as claimed for the weight of $e(F)$ in the third subfield); (6) that $SPAN(F)$ is a tree (as in Lemma 2.2) rooted in the endpoint of $e(F)$; it also verifies (7) that $SPAN(F)$ contains the same set of vertices as the corresponding subtree described by the set of edges $\{(v, L_1^i(v)) \mid v \in A\}$ (by verifying that each edge in one subtree is pointed at in the other subtree by one of its endpoints); finally, the decoder verifies (8) that if v is the root of $SPAN(F)$ then the edge to the min-edge direction is indeed the outgoing edge $e(F)$ whose weight is as agreed in the third subfields of each vertex.

As said above, the proof is now similar to that of Theorem 4.1. The proof that the oracle algorithm is correct, is basically, the proof of the algorithm of [42] (and is similar to proofs of quite a few previous MST algorithms), so we skip it here. Hence, if T is the result of an execution of the algorithm, then T is indeed a minimum spanning tree. It is rather easy to see that the decoder verifies that labels contain the history of a valid execution. Hence, if the decoder outputs 1, then the subgraph is indeed a minimum spanning tree.

On the other hand, assume that an MST is given. It is rather easy to see that the oracle algorithm as described, constructs exactly this MST. Again, the decoder indeed verifies that a valid history of the algorithm is given. Hence, the decoder outputs 1.

Each fragment in F_{i+1}, the set of fragments that correspond to the $i+1$-st round, 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, $|F_{\log n}| \geq n$ yielding $F_{\log n} = T$. Hence, each label contains $O(\log n)$ fields. Since each such field contains $O(\log n + \log W)$ bits, the theorem follows. ▮

Lemma 4.3 *Every proof labeling scheme for f_{MST} over $\mathcal{F}^{undirected}(W)$ has size $\Omega(\log n + \log W)$.*

Proof: Since an MST is also a spanning tree, we get by Lemma 2.2 that every proof labeling scheme on $\mathcal{F}^{undirected}(W)$ and f_{MST} has size $\Omega(\log n)$. Therefore, we only need to show that every proof labeling scheme for f_{MST} over $\mathcal{F}^{undirected}(W)$ has size $\Omega(\log W)$. Assume, by way of contradiction, that there exists a proof labeling scheme $\pi = (\mathcal{M}, \mathcal{D})$ for f_{MST} over $\mathcal{F}^{undirected}(W)$ 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, $f_{MST}(C^i_s) = 1$ for every i. Therefore, for every i, every correct marker algorithm $\mathcal{M}$ assigns a labeling assignment L^i to the vertices of C^i_s so that for each vertex v in C^i_s, $\mathcal{D}(v, L^i) = 1$. 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 $L^i = L^k$. Let C_s be the same as C^i_s except that $\omega(3,4) = 2k$. Obviously, $f_{MST}(C_s) = 0$. However, by the correctness of π on C^i_s and C^k_s we get that $\mathcal{D}(v, L^i) = 1$ for every vertex $v \in C_s$, a contradiction. ▮

4.2 Composition

When constructing a proof labeling scheme for some problem $Prob_1$, we sometimes want to use modularly a solution for 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 n-vertex graphs. Assume that we want to prove that the states of all vertices in a given graph $G_s \in \mathcal{F}$ are identical and equal to the number of vertices 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 4.4 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 vertices 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 vertex is an encoding of v's port leading to its parent in the desired spanning tree T. We now label each vertex $v_s \in G_s$ 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} over $\mathcal{F}_{S'}^{trees}$, 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 $\mathcal{F}$.

In all of the cases described hereafter, the number of bits in a state is $O(\log n)$ and the size of every proof labeling scheme which is used in a composition is $\Omega(\log n)$. Hence, the sizes of the constructed proofs are asymptotically the same as the sum of the sizes of the corresponding proof labeling schemes used for the composition. Let us now present proof labeling schemes for several problems, some of which are used as components for others defined through composition as described above.

4.2.1 Compositions using a scheme for semi-group functions

The following simple proof labeling scheme is an important building block for many other schemes. Let $G = (V, E)$ be a graph. A real valued function g is a semi-group function if it is defined for every subset U_s of V_s, and is associative and commutative (see [42] page 36 for details). 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 $G_s \in \mathcal{F}$. We assume that the values of g over all subsets of V_s and all graphs $G_s \in \mathcal{F}$ 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 vertex u of G so that for every vertex 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 verified on the tree recursively.

Lemma 4.4 *For every semi-group function g, there exists a proof labeling scheme π_g for f_g over $\mathcal{F}^{all}$ of size $O(\log n + \log l)$.*

Proof sketch: Using a composition of proofs and Lemma 2.2, we can assume the existence of a spanning tree T where each vertex knows which of its edges leads to its parent in the tree and which leads to a child (and the root knows it is the root). In the scheme, each vertex v is labeled by (1) $g(T)$ and (2) $g(T_v)$ (the subtree rooted at v). The vertex then verifies that it agrees with all of its neighbors on $g(T)$, and that its $g(T_v)$ agrees with applying g on T_{u_i} for each of its children u_i. In addition, the root r checks that $g(T_r) = g(T)$. (This proof, in more details, appears in [34].)

Lemma 4.5 *There exists a semi-group function g such that* every *proof labeling scheme for function f_g over family $\mathcal{F}^{undirected}$ 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 f_{g_1} over $\mathcal{F}$ 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. Again, similarly to the proof of the lower bound in Lemma 2.3, every proof labeling scheme for G and f_{g_2} must have size $\Omega(\log l)$. For every subset U of V, let $g(U) = (g_1(U), g_2(U))$. If follows that every proof labeling scheme for f_g over $\mathcal{F}^{all}$ must have size $\Omega(\log n + \log l)$. ▮

4.2.2 Clique and independent set

Let us now demonstrate the use of Lemma 4.4 (itself using a composition with a scheme for spanning trees) for further proofs by compositions.

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 $k \in \mathcal{N}$ (where $\mathcal{N}$ is the set of natural numbers), and 2) there exists a clique (respectively, independent set) of size k in G.

Theorem 4.6 *The proof size of both functions f_{clique} and $f_{independent}$ over $\mathcal{F}_{\mathcal{N}}^{undirected}$ is $\Theta(\log n)$.*

Proof: For the upper bound, we construct $\pi = (\mathcal{M}, \mathcal{D})$ for f_{clique} over $\mathcal{F}_{\mathcal{N}}^{undirected}$ 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 vertices). If $v \in K$, then let $\mathcal{M}(v) = 1$, otherwise let $\mathcal{M}(v) = 0$. By using Lemma 4.4, 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 yields a proof labeling scheme for $f_{independent}$ over $\mathcal{F}_{\mathcal{N}}^{undirected}$ with size $O(\log n)$.

The proof for the lower bound resembles the proof for the lower bound in Lemma 2.3. However, a direct reduction from the problem discussed in Lemma 2.3 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-vertex graphs with fixed vertices $v_1, v_2, \ldots, v_n$ 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 $v_2, v_3, \ldots, v_n$, is of size k. Also, 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 $\pi = (\mathcal{M}, \mathcal{D})$ for f_{clique} over $F^{undirected}$, 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 the marker algorithm $\mathcal{M}$ 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 a labeling 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 theorem follows. ▮

4.3 Additional examples for the composition method and for a combination of the methods

Let us now present some additional results for graph theory. They demonstrate the use of composition, together with the use of known facts in graph theory and with the use of techniques we have shown earlier, such as those used in Lemma 2.3 and Lemma 2.2.

For the sake of completeness, we present one detailed proof- that of Lemma 4.7. For the later lemmas we present much shorter proofs. We still point at the main ideas of the proof, but assume that at this point, the reader is already familiar with the technique of constructing proof labeling schemes.

4.3.1 Vertex-connectivity and flow

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

Lemma 4.7 *The proof size of f_{v-conn} over $\mathcal{F}^{s_0-t_0}$ is $\Theta(\log n)$.*

Proof: We first construct a proof labeling scheme $\pi = (\mathcal{M}, \mathcal{D})$ for $\mathcal{F}^{s_0-t_0}$ and f_{v-conn} of size $O(\log n)$. Let f_1 and f_2 be two Boolean functions from $\mathcal{F}^{s_0-t_0}$ so that $f_1(G) = 1$ iff there exist at least k vertex disjoint paths connecting s_0 and t_0, and $f_2(G) = 1$ iff there exist 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 $\mathcal{F}^{s_0-t_0}$ and f_1. The part of the proof that all the vertices agree on the value of k is the same as in Lemma 2.3. We concentrate on proving the other parts of f_1. Let G be such that $f_1(G) = 1$. Consider, first, the case that s_0 and t_0 are not neighbors. Let $P_1, P_2, \ldots, P_k$ be k vertex disjoint paths connecting s_0 and t_0. For every i, denote the vertices of P_i by $(p_i^0, p_i^1 \ldots, p_i^{l_i})$, 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 $\mathcal{M}_{f_1}(p_i^j) = (i, j, id(p_i^{j+1})) \equiv (L_1(v), L_2(v), L_3(v))$. For every other vertex v, set $\mathcal{M}_{f_1}(v) = \emptyset$. 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) $L_1(u) = L_1(v)$.

It is easy to show that π_{f_1} is a correct proof labeling scheme for f_1 over $\mathcal{F}^{s_0-t_0}$, 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 present a proof labeling scheme π_{f_2} for f_2 over $\mathcal{F}^{s_0-t_0}$. Let us, first, assume that s_0 and t_0 are not neighbors. Then, by Menger's theorem (cf. [18]), there exist k vertices u_1, u_2, ..., 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 4.4), 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 $\mathcal{F}^{s_0-t_0}$ and f_2 of size $O(\log n)$. Combining π_1 and π_2, we get a proof labeling scheme for $\mathcal{F}^{s_0-t_0}$ and f_{v-conn} of size $O(\log n)$.

As for the lower bound, a proof that is similar to the one of the lower bound in Theorem 4.6 can be constructed, showing that the above mentioned upper bound is asymptotically tight. ▮

An alternative formulation of the problem would output the connectivity only at s_0 and t_0. Proving the lower bound then may be more difficult.

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

Lemma 4.8 *The proof size of f^k_{flow} over $\mathcal{F}^{s_0-t_0}$ is $O(k \log n)$.* ▮

It is not hard to construct such a scheme, that is rather similar to the one appearing in the upper bound proof of the previous lemma. Like in that proof, we use the labels to construct paths. On each path, again, a vertex verifies that its distance to t_0 is consistent with the distance of its neighbors on the path. The main difference is that in the previous lemma, a vertex (except for s_0 and t_0) could participate in at most one path, since paths had to be vertex disjoint. Here, up to k paths may pass a vertex, and the vertex may need to keep (in the label) a different distance (to t_0) on each of them. Hence, the size of the label is $O(k \log n)$ rather than $O(\log n)$ (that was the size in the previous lemma). The vertex also keeps in the label the amount of flow on each of its paths. In addition, instead of the condition that only one path passes via a vertex, here, a vertex verifies the Kirchoff law for the flows it receives and sends to neighbors. Vertex s_0 verifies that it has k paths. For completeness, the detailed proof appears in [34].

4.3.2 Maximum matching on paths

Let P be the horizontal path of n vertices (such as the one in the proof of Lemma 2.2) $v_1, v_2, \ldots, v_n$. (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 $e_i = (v_i, v_{i+1})$ in E, 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 vertex. For such a set, denote by $\omega(H) = \sum_{e \in H} \omega(e)$. Let $m = \max\{\omega(H) \mid \text{H is a matching of E}\}$. 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 vertices 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 n vertices and let $S = \{1, 2\}$. The following proof uses techniques based on dynamic programming.

Lemma 4.9 *The proof size of f_{match} over $\mathcal{F}^{paths}(W)$ is $O(\log n + \log W)$.*

Proof: We show the existence of a proof labeling scheme $\pi_A = (\mathcal{M}_A, \mathcal{D}_A)$ for f_{match} over $\mathcal{F}^{paths}(W)$ of size $O(\log n + \log W)$. Let $P = (v_1, v_2, \ldots, v_n)$ be the given path, with edges $\{e_1, e_2, \ldots e_{n-1}\}$. Let G_s be such that $f_{match}(G_s) = 1$. First, by composing with the proof labeling scheme presented in Lemma 2.1, 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 $(v_1, v_2, \ldots, v_i)$. 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 $\mathcal{M}(i) = (A_i, B_i)$. 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 $m = O(n \cdot W)$. ∎

4.3.3 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. An *approximation function* is a function *Approx* over $\mathcal{T}(W)$ such that for every tree $T \in \mathcal{T}(W)$, *Approx*(T) is a nonempty set of matchings of T with the property that for every $H \in$ *Approx*(T), $\omega(H) \geq m/2$. We consider the problem of assigning states to the vertices 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 does not assert that *every* 2-approximation for maximum matching over a tree can be proven, but rather that we can prove a specific 2-approximate maximum matching. This leaves open the problem of efficiently proving any given 2 approximation for maximum matching.

Lemma 4.10 *There exists an approximate matching H and a proof labeling scheme $\pi_A = (\mathcal{M}_A, \mathcal{D}_A)$ of size $O(\log n + \log W)$ for verifying that H is in f_{approx} over $\mathcal{F}^{trees}(W)$.*

Proof: We rely on a distributed algorithm due to [47], for constructing a collection $\mathcal{P} = \mathcal{P}(T)$ of vertex disjoint paths for a given tree T, such that $m \leq \sum_{P \in \mathcal{P}} \omega(P)$. The algorithm operates in $O(1)$ rounds and with each vertex sending $O(1)$ messages per round. For every path $P \in \mathcal{P}$, a maximum matching $MM(P)$ for P satisfies $\omega(MM(P)) \geq \omega(P)/2$. Therefore, the collection of edges $MM(T) = \cup_{P \in \mathcal{P}} MM(P)$ satisfies $\omega(MM(T)) \geq m/2$. Therefore,

$$\mathcal{H}(T) \;=\; \{ \bigcup_{P \in \mathcal{P}} MM(P) \mid MM(P) \quad \text{is a maximum matching on P} \}$$

is an approximation function. By Theorem 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 4.9 applied to the path P. Altogether, we have a proof labeling scheme of size $O(\log n + \log m)$ for verifying that H is in f_{approx} over $\mathcal{F}^{trees}$. The lemma follows as $m = O(n \cdot W)$. ▌

5 Conclusion

In this paper, we present a formal setting for distributed verification as a separate task from that of the computation. In the sequential context, such a separation proved to be useful for multiple fields such as complexity theory, cryptography, and software engineering. In the area of distributed computing, the application to self stabilization is clear. It would be interesting to see whether additional applications arise. It would also be interesting to find additional relations between the cost of computing and the cost of verifying.

We have presented quite a few examples. Still, these are just a few of the possible functions over graphs (and over configuration graphs). Hence, the number of functions for which proof labeling schemes can still be studied is huge. Moreover, for many problems addressed in this paper (for example flow and maximum matching), no optimal solutions were presented. Finding tight bounds for the size of proof labeling schemes for such problems is challenging. In addition, it is natural to also try to find proof labeling schemes for various implicit labeling schemes such as the ones in [23, 31, 32].

We proved that for any m there exists a problem for which the optimal size of a proof labeling scheme is $\Theta(m)$. Still, the natural problems for which we found efficient schemes are only from a limited set of sizes. It seems interesting to find natural problems whose schemes require, say, a size of $\sqrt{n}$. One can also seek more general results, for example, characterizations of problems with small sizes for their proof labeling schemes.

For the problem of assigning a unique state to every vertex, we have shown a large difference in the results for two cases. For the case that one is allowed to select the algorithm that assigns the states, the size of the proof labeling schemes can be very small (even zero for id-based families). If the (unique)

states are arbitrary, then the size is much larger ($\Omega(n)$). It seems interesting to identify additional such cases, and to characterize them.

We presented two somewhat general methods for constructing proof labeling schemes. Still, they cover only limited cases. In particular, the composition method uses only the composition of a logical "and" between two predicates. It would be interesting to define and find uses for more general compositions.

We have addressed here only the static case of a function, and did not address the more general case of verifying interactive programs efficiently. Even for the case of a function, we have not presented efficient distributed algorithms to compute schemes of small sizes. In particular, it would be interesting to compute the labels in parallel to the execution of the algorithm that computes the function. (Some ideas in our description of the distributed method for constructing small sizes proof labeling schemes may prove useful for that task).

6 Acknowledgement

We would like to thank the anonymous reviewers for their contribution to improve the write-up of this paper significantly.

References

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

[2] 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).

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

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

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

[6] B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, and G. Varghese. Time optimal self-stabilizing synchronization. In *Proc. ACM Symposium on the Theory of Computer Science*, 1993, 652-661.

[7] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-Stabilization By Local Checking and Correction. In *Proc. IEEE Symposium on the Foundations of Computer Science*, 1991, 268-277.

[8] B. Awerbuch, B. Patt-Shamir, G. Varghese, and S. Dolev. Self-Stabilization by Local Checking and Global Reset. In *Proc. Workshop on Distributed Algorithms*, Lecture Notes in Computer Science 857, Springer, 1994, 326-339.

[9] B. Awerbuch, G. Varghese. Distributed Program Checking: a Paradigm for Building Self-stabilizing Distributed Protocols. In *Proc. IEEE Symposium on the Foundations of Computer Science*, 1991, 258-267.

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

[11] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. "Labeling Schemes for Tree Representation". In *Proc. 7'th Int. Workshop on Distributed Computing.* Dec. 2005, Springer LNCS 3741, 13-24, 2005.

[12] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-Guided Graph Exploration by a Finite Automaton. In *Proc. 32nd Int. Colloq. on Automata, Languages, and Programming*, 2005, 335-346.

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

[14] B. Dixon, M. Rauch, and R. E. Tarjan. Verification and Sensitivity Analysis of Minimum Spanning Trees in Linear Time. *SIAM Journal on Computing*, Vol. 21, No 6, 1184-1192, December 1992.

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

[16] 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.

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

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

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

[20] M.L. Fredman and D.E. Willard. Trans-Dichotomous algorithms for minimum spanning trees and shortest paths. In *Proc. 31st Annual Symposium on Foundations of Computer Science*, Los Alamitos, CA, 1990, 719-725.

[21] 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.

[22] C. Gavoille. Routing in distributed networks: Overview and open problems. *ACM SIGACT News-Distributed Computing Column* **32**, (2001), 36–52.

[23] C. Gavoille, D. Peleg, S. Pérennes and R. Raz. Distance labeling in graphs. *J. of Algorithms*, **53(1)**, (2004), 85–112.

[24] P. Fraigniaud, D. Ilcinkas, and A. Pelc. Oracle size: a new measure of difficulty for communication tasks. In *Proc. 25th Ann. ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, 2006, 179-187.

[25] P. Fraigniaud, D. Ilcinkas, and A. Pelc. Tree Exploration with an Oracle. In *Proc. 31st Int. Symp. on Mathematical Foundations of Computer Science*, 2006, 24-37.

[26] P. Fraigniaud, A. Korman, and E. Lebhar. Local MST Computation with Short Advice. In *Proc. 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, June 2007, 154-160.

[27] G.M. Jayaram and G. Varghese. Crash failures can drive protocols to arbitrary states. In *Proc. 15th Ann. ACM Symp. on Principles of Distributed Computing*, 1996, 247-256.

[28] G.M. Jayaram and G. Varghese. The Complexity of Crash Failures. In *Proc. 16th Ann. ACM Symp. on Principles of Distributed Computing*, 179-188.

[29] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. In *SIAM J. on Descrete Math* **5**, (1992), 596–603.

[30] D. R. Karger, P.N. Klein, and R.E. Tarjan. A Randomized Linear-Time Algorithm to Find Minimum Spanning Trees. *J. of ACM* Vol. 42, No. 2, 321-328, 1955.

[31] M. Katz, N.A. Katz, A. Korman and D. Peleg. Labeling schemes for flow and connectivity. *SIAM Journal on Computing* **34** (2004), 23–40.

[32] A. Korman. Labeling Schemes for Vertex Connectivity. In *Proc. 34th Int. Colloq. on Automata, Languages and Prog.*, July 2007, 102-109.

[33] A. Korman, S. Kutten, and D. Peleg. Proof labeling schemes. In *Proc. 24th Ann. ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, 2005, Las Vegas, Nevada, USA, July 2005, 9-18.

[34] A. Korman, S. Kutten, and D. Peleg. Proof Labeling Schemes. A detailed version. http://ie.technion.ac.il/ kutten/pdf/ProofLabelingSchemes.pdf

[35] A. Korman and S. Kutten. Distributed Verification of Minimum Spanning Trees. In *Proc. 25th Ann. ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, 2006, Denver, Colorado, USA, July 2006, 26-34.

[36] Fabian Kuhn, Thomas Moscibroda, Roger Wattenhofer. What cannot be computed locally! In *Proc. ACM Symp. on the Principles of Distributed Computing*, 2004, 300-309.

[37] E. Kushilevitz and N. Nissan. *Communication complexity.* Cambridge University Press, 1997.

[38] Nathan Linial. Distributive Graph Algorithms-Global Solutions from Local Data. In *IEEE Symposium on the Foundations of Computer Science*, 1987, 331-335.

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

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

[41] D. Peleg and E. Upfal. A tradeoff between size and efficiency for routing tables. *J. ACM* **36**, (1989), 510–530.

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

[43] D. Peleg. Proximity-Preserving Labeling Schemes. *J. Graph Theory* **33**, (2000), 167–176.

[44] D. Peleg. Informative Labeling Schemes for Graphs, *Theoretical Computer Science* **340**, *Special Issue of MFCS'00 papers*, (2005), 577–593.

[45] N. Santoro and R. Khatib. Labeling and implicit routing in networks. *The Computer Journal* **28**, (1985), 5–8.

[46] P. Tiwari. Lower bounds on communication complexity in distributed computer networks. *J. ACM* **34**, (1987), 921–938.

[47] M. Wattenhofer and R. Wattenhofer. Distributed Weighted Matching. In *Proc. 18th International Symposium on Distributed Computing*, Springer LNCS 3274, 2004, 335-348.

[48] Andrew C. Yao. Some Complexity Questions Related to Distributed Computing. In *Proc. 11th ACM Symposium on Theory of Computing*, 1979, 209-213.