Proof Labeling Schemes

Extended Abstract

Amos Korman
Dept. of Computer Science
The Weizmann Institute
Rehovot 76100, Israel
amos.korman@weizmann.ac.il

Shay Kutten*
Information Systems Group
Fac. of IE&M
The Technion
Haifa 32000, Israel
kutten@ie.technion.ac.il.

David Peleg*
Dept. of Computer Science
The Weizmann Institute
Rehovot 76100, Israel
david.peleg@weizmann.ac.il

ABSTRACT

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

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

Categories & Subject Descriptors:
F.2.2[Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems;
G.2.2[Discrete Mathematics]: Graph Theory;
B.8.1[Performance and Reliability]: Reliability, Testing, and Fault-Tolerance.
General Terms: Algorithms, Reliability, Theory.
Keywords: distributed networks, proof labels, property verification, self stabilization.

*Supported in part by a grant from the Ministry of Science and Technology.

PODC'05, July 17–20, 2005, Las Vegas, Nevada, USA.

1. INTRODUCTION

This paper addresses the problem of locally verifying global properties. This task complements the task of locally computing global functions. Since many functions cannot be computed locally [1, 21, 19], local verification may potentially be even more useful than local computing - one can compute globally and verify locally. We address the natural question of how expensive is local verification. More specifically, we deal with the question of how expensive local verification is in comparison to the computation itself. In terms of both sequential time and distributed communication time, there exists evidence that verification is sometimes easier. For example, verifying that a given color assignment on a given graph is a legal 3 coloring is believed to consume much less time than computing a 3 coloring [18]. In the context of distributed tasks, other measures of complexity are often used, for example the amount of communication needed. Still, one can ask a similar natural question. Assume that we are given a *distributed representation* of a solution for a problem (for example, each node knows its color but not necessarily the colors of others). It is required to verify the legality of the represented solution (in the example, to verify that the coloring stored by the nodes is a legal 3 coloring). Does the verification consume fewer communication bits than the computation of the solution (e.g., the 3 coloring) itself?

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

The formal definitions are given in Section 2.1. Infor-

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

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

In addition to the theoretical appeal of such problems, one practical justification comes from the area of self stabilization. There, the network should converge to a *legal* configuration (one satisfying the configuration predicate) from *any* initial configuration. One approach is to partition a given self stabilization task into two parts: the detection that the current configuration is illegal, and the computation of a new legal configuration. Again, a natural question is to ask which of these problems is more expensive.

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

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

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

Related Work. The measure of the label size is related to the problem of the communication complexity [22]. Some of our results are for impossibility of tasks in anonymous networks. Results of that nature concerning computation (rather than verification) were presented in [14] and follow up papers. Some constructions we use simulate a distributed algorithm on every node; a related operation was used in [7]. Self stabilization was introduced in [15]. Self stabilization by local detection, and by the similar variant local checking was introduced in [8, 11, 9, 10]. These papers, as well as many others, e.g. [16, 17, 6, 4, 5]) present self stabilizing algorithms for computing trees using local detection. Using a memory efficient distributed representation of a tree, in which each vertex holds just a pointer to its parent, might allow the distributed system to be in an undetected "illegal" configuration (namely, the states of the vertices at a given time may constitute a representation of a structure which is not a spanning tree). For instance, the structure may be "illegal" since the collection of pointers forms a cycle or a collection of cycles, rather than a tree.

To overcome such illegal configurations, the algorithms mentioned above were based on adding a variable to each vertex, containing the distance of the vertex from the root of the tree. (The resulting representation, containing both the pointer and the distance variable, is clearly *redundant*). To verify that the distributed redundant representation indeed forms a tree, each vertex compares its own distance variable with the distance variable of its parent. It is easy to see that if the collection of pointers (one per node) induces a cycle, then there must exist a vertex whose distance is not larger than that of its parent. In the example, validity is verified by checking the correctness of a global predicate based on the configuration. The configuration predicate is that the collection of pointers formed a tree. This is based, in turn, on verifying a local predicate at each vertex, saying that if the vertex points at a parent, then the distance variable of the vertex is larger by one than the distance variable of the parent.

In [12], lower bounds for silent stabilization are given. Informally, an algorithm achieves silent stabilization if after stabilization no value is changed in any variable, so the only activity is verifying that the states of the neighbors are the same as the vertex "remembers" them. In a sense, in a state of silent stabilization the vertices must be able to verify that

the system is indeed stabilized, and no further changes in the variables are necessary. Hence, the lower bounds for silent stabilization may sometimes translate to lower bounds for proof labeling scheme. This is not always true though, and in particular, one can construct proof labeling schemes for problems for which no silent stabilization exists In addition, lower bounds in our model do not necessarily imply lower bounds in the model of [12]. Our typical lower bound is for the task of verifying *any* configuration. In contrast, in [12] it is implicitly assumed 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 [12, 11, 10] it is assumed that a vertex can read a state of a link port of a neighboring vertex. We assume that all the neighbors of a vertex v can see the same *label* of v. One can say that this models a local broadcast media (e.g., radio) versus point to point lines. The more abstract motivation is that our modeling is intended to capture the total number of information bits that need to be conveyed by a node for checking purposes, disregarding the issue of which neighbor they are to be communicated to. Consequently, constructing positive results in our model may be harder than in the point to point model. For example, in the latter model a vertex can easily choose to communicate with just one neighbor. In our model, in contrast, all the neighbors see the label, hence it is harder to tell which of them is the target of the communication. Nevertheless, we show some schemes for problems that seem to need a targeted communication. Still, the size of these schemes is much smaller than a size of the identity of the neighbor.

In [13] it is assumed that a node can read the *output* of near-by nodes. That is, only the part of the state meant to be visible to the outside can be read by other nodes. (The output is the part that appears in the specification of the task to be performed.) As opposed to that, in the current paper the labels often contain information that is not intended as output, and does not appear in the specification of the task. In some sense, this is necessary for efficient solutions, since in [13] it is shown that with their assumption, a very large memory is sometimes needed (e.g. for verifying a spanning tree).

Our Results. This paper models and partially answers the question: "how easy is the checking task by itself", with respect to basic building blocks in distributed systems. In Section 2.1 we give some examples of schemes of different information requirements (proof label size) and show that for any size there exists a problem requiring this size.

In Section 3 we study factors that affect the cost of checking. We study the role of unique identities in terms of impossibility and complexity. We show that there exist problems and graph families for which no labeling proof scheme exists if no unique identities are assumed. On the other hand, we show a case (specifically, a path of n vertices) in which the transition from anonymous networks to id-based is possible, yet the cost of the transition in this case is $\Omega(n)$. It is interesting that even if unique identities are assumed, checking whether the states are also unique can be still as costly as $\Omega(n)$. This is higher than the cost of computing unique states in this case.

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

In Section 4 we study the cost of basic building blocks, and show how to build verification systems systematically and modularly. This leads to rather efficient schemes for Minimum Spanning Tree, Maximum Matching, $s-t$ Vertex Connectivity, and some other basic problems. We also present some constant size labeling schemes, even though our model is weaker than previous models as explained above. Coming back to the hardness of checking versus the hardness of computing, we show that small schemes exist even for NP Hard problems.

Due to lack of space, some proofs are deferred to the full paper.

2. DEFINITIONS AND BASIC EXAMPLES

2.1 Definitions

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

Given a vertex v, let s_v denote the state of v and let $v_s = (v, s_v)$. A *configuration graph* corresponding to a graph $G = \langle V, E\rangle$ is a graph $G_s = \langle V_s, E_s\rangle$, where $V_s = \{v_s \mid v \in V\}$ and $(v_s, u_s) \in E_s$ iff $(v, u) \in E$. A *family of configuration graphs* $\mathcal{F}_s$ corresponding to graph family $\mathcal{F}$ consists of configuration graphs $G_s \in \mathcal{F}_s$ for each $G \in \mathcal{F}$. Let $\mathcal{F}_S$ be the largest possible such family when every state s is taken from a given set S. Unless mentioned otherwise, let $S = \{1, 2, \cdots, O(n)\}$. We sometimes refer to each state s_v of a configuration graph as having two fields: $s_v = (id(v), s'(v))$. Field $id(v)$ is v's *identity* and has $O(\log n)$ bits. When the context is clear we may refer to $s'(v)$ as the state of v (instead of to $s(v)$). A configuration graph G_s is *id-based* if for every pair of vertices v and u it is given that $id(u) \neq id(v)$. A family of graphs whose

identities are arbitrary (including possibly graphs where all identities are the same) is termed *anonymous*. An id-based (respectively, anonymous) family is a family of id-based (respectively, anonymous) graphs. Let $\mathcal{F}^{all}$ be the collection of all strongly-connected and all undirected connected graphs with $O(n)$ vertices. Let $\mathcal{F}^{undirected}$ be the collection of all undirected graphs with $O(n)$ vertices. When it is clear from the context, we use the term "graph" instead of "configuration graph", "id-based graph" or "anonymous graph". We may also use the notation v instead of v_s.

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

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

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

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

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

Let $N_L(v) = \langle (s_v, L(v)), N'_L(v) \rangle$. Informally, $N'_L(v)$ contains the labels given to all of v's neighbors along with the port number and the weights of the edges connecting v to them. $N_L(v)$ contains v's state and label as well as $N'_L(v)$.

A *decoder* algorithm $\mathcal{D}$ is an algorithm which is applied separately at each vertex $v \in G$. When $\mathcal{D}$ is applied at vertex v, its input is $N_L(v)$ and its output, $\mathcal{D}(v, L)$, is boolean.

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

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

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

2.2 Basic examples

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

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

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

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

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

and $L^i = L^j$. Let G_s be G modified so that $s_u = i$ and $s_v = j$. Let L be the marker algorithm for G_s in which $L(u) = \mathcal{M}^i(u)$ and $L(v) = \mathcal{M}^j(v)$. Then for each vertex x, $\mathcal{D}(x, L) = 1$, contradicting the fact that $f(G_s) = 0$. □

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

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

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

The following example concerns the representation of various spanning trees in the system. The upper bound employs an idea previously used in [7, 5, 6, 8, 17, 4], but overcoming some technicalities arising from our model. For the lower bounds, we had to establish a proof also for trees and forests that are not spanning.

Trees in id-based families Problems:. We consider five different problems, obtained by assigning states to the nodes of G so that $H(G_s)$ (respectively, $D(G_s)$) is a (respectively, directed) (1) forest; (2) spanning forest; (3) tree; (4) spanning tree; (5) BFS tree of G (for some root vertex r). Let $f_{No-cycles}$ (respectively, $f'_{No-cycles}$) be the characteristic function of either one of the five problems above.

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

Actually, our lower bounds here can be shown even for more restricted families.

Orient*ation in anonymous trees Problem:.* Assign states to the nodes of a tree so that $D(G_s)$ induces an orientation on the edges (towards some root that is not given explicitly). Let $\mathcal{F}^{anon-trees}$ be the family of anonymous trees.

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

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

1. $|L(u) - L(v)| = 1$.
2. $L(u) + 1 = L(v) (\text{mod} 3)$ 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 $f_{Orient}(G_s) = 0$. If for every v, s_v is an encoding of one of its ports then since the underlying graph G is a tree there must exist two vertices u and v whose states point at each other. By (2), either $\mathcal{D}(v, L) = 0$ or $\mathcal{D}(u, L) = 0$ and we are done. Otherwise, there exists a vertex r whose state is not an encoding of one of its ports. If every v satisfies $\mathcal{D}(v, L) = 1$, then all the states of r's neighbors point at r and by induction we get an orientation of the edges towards r. Therefore $f_{Orient}(G_s) = 1$, which contradicts our assumption. □

3. THE ROLE AND COST OF IDENTITIES

The bounds presented in [12] are similar for id-based and anonymous families. Our model exhibits a distinction between the two families. Specifically, we show that certain tasks are impossible in some anonymous families but possible in id-based families. We also show that the transition from anonymous to id-based is very costly even on a path, where this transition is possible. Moreover, even for id-based paths the task of proving whether the states are disjoint is costly. We also separate our model from the one of [1] that was shown to be order invariant. In subsection 3.3, we show that our model is not order invariant.

3.1 Anonymous versus id-based families

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

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

The proof of the following lemma bears some similarities to the proof, given in a different context, that the task of leader election is impossible in anonymous networks [14].

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

LEMMA 3.2. *For every problem there exists a proof labeling scheme in every id-based family.*

3.2 Cost of identities

In the previous subsection we showed that giving a proof labeling scheme for the problem of distinct identities is impossible for a family of anonymous circles. In this subsection we show that although such a proof labeling scheme can be given on anonymous paths, it is very costly. In fact we show

that the proof size of the problem of distinct states on an n-node id-based path is $\Theta(n)$. More formally, let $f_{Distinct}$ be the characteristic function of the following problem: assign states to the nodes of G so that for every pair of vertices u and v we have $s_u \neq s_v$. Let $\mathcal{F}^{path}$ be an id-based family containing a single path G of n vertices.

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

Proof: In this extended abstract we show only the lower bound. Let $\pi = \langle \mathcal{M}, \mathcal{D} \rangle$ be a proof labelling scheme for $\mathcal{F}_S^{path}$ and $f_{Distinct}$. For notational simplicity let us name these vertices from left to right by $\{v_1, v_2, \cdots, v_n\}$ (these are not the nodes' identities). Let us first give a high level description of the proof. We construct a large set X' of configuration graphs, all corresponding to path G such that for each $G_s \in X'$, $f_{Distinct}(G_s) = 1$. We choose X' so that the pair of labels given by $\mathcal{M}$ to the two vertices in the middle of the path must be different in each instance of X'. More formally, a *divided permutation* is a permutation σ on $[1, \cdots, n]$ such that $\sigma(n/2) = n/2$ and $\sigma(n/2+1) = n/2+1$. (Assume without loss of generality that n is even). Fix $s_{v_{n/2}} = n/2$ and $s_{v_{1+n/2}} = 1 + n/2$. For a divided permutation σ, let $\mathcal{M}_\sigma(v_i)$ denote $\mathcal{M}(v_i)$ in the case where for each $1 \leq j \leq n$, $s_j = \sigma(j)$. For a divided permutation σ, let $T_\sigma = \{\sigma(i) \mid 1 \leq i \leq n/2-1\}$ and $Q_\sigma = \{\sigma(i) \mid n/2+2 \leq i \leq n\}$. The proof uses the following claims.

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

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

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

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

Combining the three claims we get an $\Omega(n)$ lower bound for any proof labeling scheme for $\mathcal{F}_s^{path}$ and $f_{Distinct}$. □

We note that if one designs the states and the labels together then the size of the scheme can be much smaller: As mentioned, if $\mathcal{F}_s^{path}$ is id-based, then the label size can be zero. For an anonymous $\mathcal{F}_s^{path}$, one can choose the state of v_i to be i, and the size of these scheme is $\log n$.

3.3 Variability of identities

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

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

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

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

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

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

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

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

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

Consider a $2n$-node cycle C' in which $s_v = n$ for every vertex v in C'. Obviously $f(C'_s) = 0$. Consider the follow-

ing disjoint identity assignment to the vertices of C'.
$(id(v_1), id(v_2), \cdots, id(v_{n-2}), id(v_{n-1}), id(v_n), id(v_{n+1}),$
$id(v_{n+2}), \cdots, id(v_{2n-1}, id(v_{2n})) =$
$(1, 2, , \cdots, n-2, 4n-1, 4n, 2n+1, 2n+2, \cdots, 3n-1, 3n)$.
Now assign to the vertices of C' the labelling $L(v_{i+1(\mathrm{mod}n)}) = l$. Then $\mathcal{D}(v, L) = 1$ for every vertex v, contradicting the correctness of π. □

4. CONSTRUCTION METHODS

It may be unrealistic to expect to find an automatic way for constructing efficient proof labeling schemes. Still, we demonstrate two systematic approaches that can ease the design in many cases. The first is the "distributed method", based on 'imitating' distributed algorithms. The second is a modular construction approach based on the notion of composition. All the graph families in the section are id-based.

4.1 The distributed method

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

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

Consider some f and $\mathcal{F}_s$, and a distributed marker algorithm for f and $\mathcal{F}_s$. Assume that for every $G_s \in \mathcal{F}_s$ such that $f(G_s) = 1$, there exists a run A' of A such that: 1) $A'(G) = G_s$, 2) the number of messages a vertex sends in $A'(G)$ is bounded from above by m_0 and 3) each message has $O(\log n)$ bits. Moreover, for every run $A'(G)$, $f(A'(G)) = 1$. If A is synchronous, assume also that for every $G \in \mathcal{F}_s$, the number of rounds in $A(G)$ is bounded from above by p and that the number of messages a vertex sends per round is bounded from above by m_p.

LEMMA 4.1.

1. *There exists a proof labelling 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 labelling scheme for $\mathcal{F}_s$ and f of size $\min\{O(m_0(\log p + \log n)), O(p \cdot m_p \log n)\}$.*

In the following claim, the term "polytime" is used in its sequential time complexity meaning.

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

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

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

The use of the Lemma 4.1 does not suffice to prove Lemma 4.3, since known distributed MST construction algorithms sometimes send $\Omega(Deg(v))$ messages in some round for some v. Substituting this in Lemma 4.1 would have translated this to a size which is higher than the desired. Still, by imitating *most* steps of a distributed algorithm [2], and using some careful changes, we manage to give a proof labeling scheme for MST with the desired size.

Proof Sketch:. Given some $G_s \in \mathcal{F}_s^{undirected}$, similarly to the proof of Lemma 2.3 we can verify that the subgraph induced by the states of G_s is indeed a spanning tree for G, so we only need to prove it is minimum. Denote this tree by T. A subtree of T is a fragment. For each fragment F, an outgoing edge of F is an edge $e = (u, x) \in G$ where $u \in F$ and $x \notin F$. A minimum outgoing edge of F is an outgoing edge of F of minimum weight. It is easy to show that every fragment F has a minimum outgoing edge that also belongs to this specific MST, T. We now build T in p phases that correspond to p fields in the labels. Intuitively, our construction is rather similar to a known distributed algorithm for constructing an MST with the following exceptions. First, the known algorithms construct some MST while we prove specifically the given MST T. Second, in the known distributed construction, each vertex 'wastes' a lot of messages in each round in order to update its neighbors of its new fragment identifier. These update messages do not have to appear in the labels since in our model a vertex v can 'see' its neighbors' labels. Therefore, if for each round each vertex encodes its fragment number, then each vertex v can know at each round to which fragment each of its neighbors belongs to. We now describe the construction more formally. For each vertex v, the field i in $\mathcal{M}(v)$ (that corresponds to phase i) has three subfields denoted $\mathcal{M}(v)_1^i, \mathcal{M}(v)_2^i$ and $\mathcal{M}(v)_3^i$. $\mathcal{M}(v)_1^i$ contains either an identity of one of v's neighbors in T or the identity of v itself. For every vertex v, $\mathcal{M}(v)_1^1$ contains just the identity of v. We inductively show that for each i, $F_i = \{(v, \mathcal{M}(v)_1^i) \mid v \in V\}$ is a collection of distinct fragments where $F_{\log n}$ is T itself. Obviously, F_1 is just the collection V (with no edges). Let F be a fragment in F_i. Let $e(F) = (u, x) \in T$ be a minimum

outgoing edge of F such that $u \in F$. Let $SPAN(F)$ be a spanning tree for F rooted at u. For each vertex $v \in F$ such that $v \neq u$ let $\mathcal{M}(v)_2^i = (id(u), id(y), d_{SPAN(F)}(v,u))$ where y is v's parent in $SPAN(F)$ and let $\mathcal{M}(u)_2^i = id(x)$ (where $e(F) = (u,x)$). For each vertex $v \in F$ let $\mathcal{M}(v)_3^i = w(e)$ where $w(e)$ is e's weight. Let $\mathcal{M}(v)_1^{i+1}$ = the second field of $\mathcal{M}(v)_2^i$. Then following claim is trivial.

CLAIM 4.4. *If F_i is a collection of distinct fragments then so is F_{i+1}.*

Each fragment in F_{i+1} contains at least two fragments in F_i. Moreover, 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 for $p = \log n$, $|F_p| = n$. It is easy to show by induction that for all i, if F_i is contained in some MST then there exists an MST T' such that F_{i+1} is contained in T'. Therefore, $F_{\log n}$ is an MST tree.
After verifying that G_s is indeed a spanning tree for G using $\mathcal{D}_{span}$, the decoder $\mathcal{D}_{mst}$ needs to verify that the construction of T is as it should be.
For a vertex v and marker algorithm L, $\mathcal{D}_{mst}(v, L)$ verifies that $L(v)_1^i$ contains an identity of either one of v's neighbors in T or the identity of v itself and that for every vertex v, $L(v)_1^1$ contains just the identity of v. The first subfield of $L(u)_2^i$ for all vertices induces a partition of the vertices into maximal connected components $A_1, A_2, \cdots$ so that for each j and for each $u \in A_j$ the first subfield of $L(u)_2^i$ is the same. We denote this value by $r(A)$. Let A be the maximal connected component that v belongs to. The decoder uses similar operations as $\mathcal{D}_{span}$ to verify that the second subfield of $L(u)_2^i$ for all vertices $u \in A$ forms a spanning tree to the subgraph of G induces by A rooted at the vertex $r(A)$. Then the decoder verifies that all vertices $u \in A$ agree on $L(u)_3^i$ denoted $w(A)$, and that for every $u \in A$, $w(A) \leq \min\{w(u,t) \mid (u,t) \in E, t \notin A\}$. For a vertex v such that $v = r(A)$, the decoder verifies that $w(A) = \omega(v, L(v)_2^i)$. □

LEMMA 4.5. *Every proof labelling scheme for $\mathcal{F}_S^{undirected}$ and f_{MST} has size $\Omega(\log n + \log W)$.*

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

4.2 Composition

When constructing a proof labeling scheme for some problem $Prob_1$, we want sometimes to use modularly a solution to a different problem, $Prob_2$. An explanation of the arising technicalities in doing so appears in the full version. We remark that in all the cases in this paper, the size of the composition of proofs is asymptotically the same as the sum of their sizes.

We now give proof labeling schemes for several problems, some of which are used as components for others defined through composition as described above. All graph families mentioned below are id-based.

Compositions using a scheme for semi-group functions:. The following proof labeling scheme is an important building block for many other schemes. Out of the following lemmas, the proof for Lemma 4.9 is given as an example for a proof for a composition. Let $G = \langle V, E \rangle$ be a graph. A function gr is a semi-group function if it is well-defined for every subset of V_s, and is associative and commutative. For example, $gr(S)$ can be $|S|$ or $\sum_{v \in S} s_v$. Let gr be a semi-group function defined on all graphs $G_s \in \mathcal{F}_s$. We assume that the values of gr over all subsets of V_s and all graphs $G_s \in \mathcal{F}_s$ are bounded from above by l. (For the lower bound the values may be larger than l). Let f_{gr} be the characteristic function of the following problem: assign states to the nodes of G so that the state of every vertex is the value $gr(V_s)$. The proofs of the following lemma uses the scheme for a rooted spanning tree as a building block, and the value $gr(V_s)$ is recursively verified on the tree. In every vertex, we verify gr of the subtree rooted at the vertex.

LEMMA 4.6. *For every semi group function gr, there exists a proof labeling scheme π_{gr} for $\mathcal{F}_s^{all}$ and f_{gr} of size $O(\log n + \log l)$.*

LEMMA 4.7. *There exists a semi group function gr such that* every *proof labeling scheme for family $\mathcal{F}_s^{all}$ and function f_{gr} must have size $\Omega(\log n + \log l)$.*

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

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

LEMMA 4.8. *The proof size of $\mathcal{F}_S^{undirected}$ and both functions f_{Clique} and $f_{Indepenent}$ is $\Theta(\log n)$.*

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

LEMMA 4.9. *The proof size of family $\mathcal{F}_S^{s_0-t_0}$ and Function f_{v-conn} is $\Theta(\log n)$.*

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

- At the special vertex s_0, the decoder verifies that s_0 has k neighbors, namely $u_1, u_2, \cdots, u_k$, s.t. $L_1(u_i) = i$.
- For every vertex v with nonempty label (in particular, $v \neq s_0, \quad v \neq t_0$), the decoder verifies that there is an edge connecting v to $u = L_3(v)$ and that either $u = t_0$ or (1) $L_2(u) = L_2(v) + 1$ and (2) $L_1(u) = L_1(v)$.

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

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

We now give a proof labeling scheme π_{f_2} for $\mathcal{F}_S^{s_0-t_0}$ and f_2. Let us first assume that s_0 and t_0 are not neighbors. Then, by Menger's theorem, there exists k vertices u_1, u_2, $\cdots$, u_k whose deletion from the graph G (along with their edges) disconnects t_0 from s_0. We now mark each u_i by $*$, each vertex in the connected component of s by 0, and the rest of the vertices by 1. Using π_{gr} (Lemma 4.2) 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 edge (s_0, t_0) and f_2 is with parameter $k-1$ instead of k. Again, it is easy to show that π_{f_2} is a correct proof labeling scheme for $\mathcal{F}_S^{s_0-t_0}$ and f_2 of size $O(\log n)$. Combing π_1 and π_2 we get a proof labeling scheme for $\mathcal{F}_S^{s_0-t_0}$ and f_{v-conn} of size $O(\log n)$.

If k is large then the lower bound follows from Lemma 2.1. Otherwise, the lower bound follows from Lemma 2.3 □

In the definition of the problem we intentionally assumed that s_0, and t_0 are not given as a part of the problem but rather as a part of the definition of the graph family $\mathcal{F}_S^{s_0-t_0}$. Otherwise the lower bound would have immediately followed from lemma 2.1.

Note that the upper bound for k vertex connectivity is independent of k. Using similar methods we can construct proof labeling scheme for k- the flow between s and t, with size $O(k \log n)$. Let W be the weight of the "heaviest" edge in a graph (1 for unweighted graphs). We also have a tight bound of size $\theta(\log n + \log W)$ for schemes to verify the diameter on trees, and to approximate the diameter from below in general graphs.

To demonstrate a combination of the distributed method and of a composition of proof labeling schemes we first present the following scheme (itself both a composition and a building block), in order to use it in a later composition. The following example uses techniques based on dynamic programming.

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

Assign states to the nodes of G so that $H(G_s)$ is a maximum matching of G. Let $\mathcal{F}^{paths}$ be the collection of all weighted paths with $O(n)$ vertices and let $S = \{1, 2\}$. Let $m = \max\{\omega(H) \mid$ H is a matching of E$\}$.

LEMMA 4.10. *The proof size of $\mathcal{F}_S^{paths}$ and f_{Match} is* $\Theta(\log n + \log m)$.

Proof: We show the existence of a proof labeling scheme $\pi_A = \langle \mathcal{M}_A, \mathcal{D}_A \rangle$ for $\mathcal{F}_S^{paths}$ and f_{Match} of size $O(\log n + \log m)$. The proof of the lower bound is omitted from this abstract. Let $P = (v_1, v_2, \cdots, v_n)$ be the given path. Let G_s be such that $f_{Match}(G_s) = 1$. First, by composing with the proof labeling scheme presented in the proof of Lemma 2.4, we may assume an orientation on the path. I.e., v_1 'knows' it is the leftmost vertex and each other vertex 'knows' which outgoing edge connects it to a vertex on its left. Second, by composing with π_{gr} and with the proof of Lemma 2.1, we may assume that all vertices 'know' and agree on the value $\omega(H)$, where $H = H(G_s)$. Since it is easy to verify locally that H is indeed a matching, it is enough to show that we can design a proof labeling scheme proving $\omega(H) = m$. We calculate m in the following manner. Let P_i be the prefix subpath $(v_1, v_2, \cdots, v_i)$. Let A_i be the maximum value over all matchings P_i that exclude edge e_{i-1}. Let B_i be the maximum value over all matchings in P_i that include edge e_{i-1}. Note that $A_{i+1} = \max\{A_i, B_i\}$, $B_{i+1} = A_i + \omega(e_{i+1})$ and $m = \max\{A_n, B_n\}$. Let $\mathcal{M}(v) = (A_i, B_i)$. 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 $\omega(H) = \max\{A_n, B_n\}$ and we are done. □

The following example illustrates a combination of the distributed method and a composition of proof labeling schemes. Let $T = \langle V, E \rangle$ be a weighted tree. Let $\mathcal{F}^{trees}$ be the collection of all weighted trees with n vertices. Denote by an *approximation function*, a function *Approx* over $\mathcal{F}^{trees}$, such that $Approx(T)$ is a nonempty set of matchings of T with the property that for every $H \in Approx(T)$, $\omega(H) \geq m/2$.

A 2-approximation for Maximum matching on a tree Problem. Assign states to the nodes of the given tree T such that the subgraph induced by them is in $Approx(T)$. The following lemma doesn't assert that every 2-approximation maximum matching on a tree can be proven but rather that we know how to prove a specific 2-approximate maximum matching. This demonstrates the need in our model to adapt the labeling to given states assigned by existing algorithms. The construction also demonstrates both modular composition and the distributed method.

LEMMA 4.11. *There exists an approximation function H and a proof labeling scheme $\pi_A = \langle \mathcal{M}_A, \mathcal{D}_A \rangle$ for $\mathcal{F}_S^{trees}$ and f_{Approx} of size $O(\log n + \log m)$.*

Proof: In [3], given a tree T, they show how to distributely build a collection $\mathcal{P} = \mathcal{P}(T)$ of vertex disjoint paths with a constant number of rounds and a constant number of messages each vertex sends per round, such that $m \leq \sum_{P \in \mathcal{P}} \omega(P)$. For every path $P \in \mathcal{P}$, a maximum matching $MAX(P)$ for P satisfies $\omega(MAX(P)) \geq \omega(P)/2$. Therefore the collection of edges $MAX(T) = \cup_{P \in \mathcal{P}} MAX(P)$ satisfies $\omega(MAX(T)) \geq m/2$. Therefore $\mathcal{H}(T) = \{\cup_{P \in \mathcal{P}} MAX(P) \mid MAX(P)$ is a maximum matching on P$\}$ is an approximation function. By Lemma 4.1, there exists a proof labeling scheme of size $O(\log n)$ so that each vertex in T knows which of its edges belong to $\mathcal{P}$. On each path $P \in \mathcal{P}$ (recall that these paths are vertex disjoint), we compose with the proof labeling scheme shown in Lemma 4.10 applied to path P. Altogether, we have a proof labeling scheme of size $O(\log n + \log m)$ for $\mathcal{F}_S^{trees}$ and f_{Approx} as desired. □

5. REFERENCES

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

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

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

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

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

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

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

[8] 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).

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

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

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

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

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

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

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

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

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

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

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

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