Error-Sensitive Proof-Labeling Schemes*†

Laurent Feuilloley[1] and Pierre Fraigniaud[2]

1 **Institut de Recherche en Informatique Fondamentale (IRIF), CNRS and University Paris Diderot, Paris, France**
2 **Institut de Recherche en Informatique Fondamentale (IRIF), CNRS and University Paris Diderot, Paris, France**

Abstract

Proof-labeling schemes are known mechanisms providing nodes of networks with *certificates* that can be *verified* locally by distributed algorithms. Given a boolean predicate on network states, such schemes enable to check whether the predicate is satisfied by the actual state of the network, by having nodes interacting with their neighbors only. Proof-labeling schemes are typically designed for enforcing fault-tolerance, by making sure that if the current state of the network is illegal with respect to some given predicate, then at least one node will detect it. Such a node can raise an alarm, or launch a recovery procedure enabling the system to return to a legal state. In this paper, we introduce *error-sensitive* proof-labeling schemes. These are proof-labeling schemes which guarantee that the number of nodes detecting illegal states is linearly proportional to the edit-distance between the current state and the set of legal states. By using error-sensitive proof-labeling schemes, states which are far from satisfying the predicate will be detected by many nodes, enabling fast return to legality. We provide a structural characterization of the set of boolean predicates on network states for which there exist error-sensitive proof-labeling schemes. This characterization allows us to show that classical predicates such as, e.g., acyclicity, and leader admit error-sensitive proof-labeling schemes, while others like regular subgraphs don't. We also focus on *compact* error-sensitive proof-labeling schemes. In particular, we show that the known proof-labeling schemes for spanning tree and minimum spanning tree, using certificates on $O(\log n)$ bits, and on $O(\log^2 n)$ bits, respectively, are error-sensitive, as long as the trees are locally represented by adjacency lists, and not just by parent pointers.

1998 ACM Subject Classification D.1.3 Concurrent Programming – distributed programming, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Fault-tolerance, distributed decision, distributed property testing

Digital Object Identifier 10.4230/LIPIcs.DISC.2017.16

1 Introduction

In the context of fault-tolerant distributed computing, it is desirable that the computing entities in the system be able to detect whether the system is in a legal state (w.r.t. some boolean predicate, potentially expressed in various forms of logics) or not. In the framework of distributed network computing, several mechanisms have been proposed to ensure such a detection (see, e.g., [1, 2, 4, 5, 22]). Among them, *proof-labeling schemes* [22] are mechanisms enabling failure detection based on additional information provided to the nodes. More specifically, a proof-labeling scheme is composed of a *prover*, and a *verifier*. A prover is an oracle that assigns a *certificate* to each node of any given network, and a verifier is a

* Both authors received additional support from ANR project DESCARTES, and Inria project GANG.
† Full version available online at `http://arxiv.org/abs/1705.04144`.

distributed algorithm that locally checks whether the collection of certificates is a *distributed proof* that the network is in a legal state with respect to a given predicate – by "locally", we essentially mean: by having each node interacting with its neighbors only.

The prover is actually an abstraction. In practice, the certificates are provided by a distributed algorithm solving some task (see, e.g., [3, 6, 22]). For instance, let us consider spanning tree construction, where every node must compute a pointer to a neighboring node such that the collection of pointers form a tree spanning all nodes in the network. In that case, the algorithm in charge of constructing a spanning tree is also in charge of constructing the certificates providing a distributed proof allowing a verifier to check that proof locally. That is, the verifier must either accept or reject at every node, under the following constraints. If the constructed set of pointers form a spanning tree, then the constructed certificates must lead the verifier to accept at every node. Instead, if the constructed set of pointers does not form a spanning tree, then, for every possible certificate assignment to the nodes, at least one node must reject. The rejecting node may then raise an alarm, or launch a recovery procedure. Abstracting the construction of the certificates thanks to a prover enables to avoid delving into the implementation details relative to the distributed construction of the certificates, for focussing attention on whether such certificates exist, and on what should be their forms. The reader is referred to [7] for more details about the connections between proof-labeling schemes and fault-tolerant computing.

One weakness of proof-labeling schemes is that they may not allow the system running the verifier to distinguish between a global state which is slightly erroneous, and a global state which is completely bogus. In both cases, it is only required that at least one node detects the illegality of the state. In the latter case though, having only one node raising an alarm, or launching a recovery procedure for bringing the whole system back to a legal state, might be quite inefficient. Instead, if many nodes would detect the errors, then bringing back the system into a legal state will be achieved by a collection of local resets running in parallel, instead of a single reset traversing the whole network sequentially.

Moreover, in several contexts like, e.g., *property-testing* [14, 15], monitoring an error-prone system is implemented via an external mechanism involving a monitor that is probing the system by querying a (typically small) subset of nodes chosen at random. *Non-deterministic* property-testing has been recently investigated in the literature [18, 24], where a certificate is given to the property-testing algorithm. Such a certificate is however global. Instead, we are interested in decentralized certificates, which can also be viewed as, say, annotations provided to the nodes of a network, or to the entries of a database. The correction of the network, or of the database, is then checked by a property-testing algorithm querying nodes at random for recovering the individual states of these nodes, including their certificates. To be efficient, such distributed certificates must guarantee that, if the monitored system is far from being correct, then many nodes are capable to detect the error. Indeed, if just one node is capable to detect the error, then the probability that the monitoring system will query that specific node is very low, resulting in a large amount of time before the error is detected.

In this paper, we aim at designing *error-sensitive* proof-labeling schemes, which guarantee that system states that are far from being correct can be detected by many nodes, providing faster recovery if the error detection mechanism is decentralized, or faster discovery if this error detection mechanism is centralized.

More specifically, the distance between two global states of a distributed system is defined as the *edit-distance* between these two states, i.e., the minimum number of individual states required to be modified in order to move from one global state to the other. A proof-labeling scheme is *error-sensitive* if there exists a constant $\alpha > 0$ such that, for any erroneous system state S, the number of nodes detecting the error is at least $\alpha\, d(S)$, where $d(S)$ is the shortest edit-distance between S and a correct system state. The choice of a linear dependency between the number of nodes detecting the error, and the edit-distance to legal states is not arbitrary, but motivated by the following two observations.

- On the one hand, a linear dependency is somewhat the best that we may hope for. Indeed, let us consider a k-node network G in some illegal state S for which r nodes are detecting the illegality of S — think about, e.g., the spanning tree predicate. Then, let us make n copies of G and its state S, potentially linked by $n-1$ additional edges if one insists on connectivity. In the resulting kn-node network, we get that $O(rn)$ nodes are detecting illegality, which grows linearly with the number of nodes, as n grows.
- On the other hand, while a sub-linear dependency may still be useful in some contexts, this would be insufficient in others. For instance, in the context of property testing, for systems that are ϵ-far from being correct (i.e., essentially, an ϵ fraction of the individual states are incorrect), the linear dependency enables to find a node capable to detect the error after $O(1/\epsilon)$ expected number of queries to random nodes. Instead, a sub-linear dependency would yield an expected number of queries that grows with the size of the system before querying a node capable to detect the error.

Our results. We consider boolean predicates on graphs with labeled nodes, as in, e.g., [25]. Given a graph G, a labeling of G is a function $\ell : V(G) \to \{0,1\}^*$ assigning binary strings to nodes. A *labeled graph* is a pair (G,ℓ) where G is a graph, and ℓ is a labeling of G. Given a boolean predicate $\mathcal{P}$ on labeled graphs, the *distributed language* associated to $\mathcal{P}$ is

$$\mathcal{L} = \{(G,\ell) \text{ satisfying } \mathcal{P}\}.$$

It is known that every (Turing decidable) distributed language admits a proof-labeling scheme [17, 22]. We show that the situation is radically different when one is interested in error-sensitive proof-labeling schemes. In particular, not all distributed languages admit an error-sensitive proof-labeling scheme. Moreover, the existence of error-sensitive proof-labeling schemes for the solution of a distributed task is very much impacted by the way the task is specified. For instance, in the case of spanning tree construction, we show that asking every node to produce a single pointer to its parent in the tree cannot be certified in an error-sensitive manner, while asking every node to produce the list of its neighbors in the tree can be certified in an error-sensitive manner.

Our first main result is a structural characterization of the distributed languages for which there exist error-sensitive proof-labeling schemes. Namely, a distributed language admits an error-sensitive proof-labeling scheme if and only if it is *locally stable*. The notion of local stability is purely structural. Roughly, a distributed language $\mathcal{L}$ is locally stable if a labeling ℓ resulting from copy-pasting parts of correct labelings to different subsets $S_1,\dots,S_k$ of nodes in a graph G results in a labeled graph (G,ℓ) that is not too far from being legal, in the sense that the edit-distance between (G,ℓ) and $\mathcal{L}$ is proportional to the size of the boundary of the subsets $S_1,\dots,S_k$ in G, and not to the size of these subsets. This characterization allows us to show that important distributed languages (such as, e.g., acyclicity, leader, etc.) admit error-sensitive proof-labeling schemes, while some very basic distributed languages (such as, e.g., regular subgraph, etc.) do not admit error-sensitive proof-labeling schemes.

Our second main contribution is a proof that the known space-optimal proof-labeling schemes for spanning tree with $O(\log n)$-bit certificates, and for minimum spanning tree (MST) with $O(\log^2 n)$-bit certificates, are both error-sensitive, whenever the trees are encoded at each node by an adjacency list (and not by a single pointer to the parent). Hence, error-sensitivity comes at no cost for spanning tree and MST. Proving this result requires to establish some kind of matching between the erroneously labeled nodes and the rejecting nodes. Establishing this matching is difficult because, for both spanning tree and MST, the rejecting nodes might be located far away from the erroneous nodes. Indeed, the presence of certificates helps local detection of errors, but decorrelates the nodes at which the alarms take place from the nodes at which the errors take place. (See Section 6 for a discussion about *proximity-sensitive* proof-labeling schemes). Moreover, in the case of MST, the known space-optimal proof-labeling scheme uses $O(\log n)$ "layers" of spanning trees (corresponding roughly to the $O(\log n)$ levels of fragments constructed by Borůvka algorithm). It is not a priori clear that errors occurring at different levels are necessarily detected by different nodes, i.e., that k errors are necessarily detected by $\Omega(k)$ nodes, and not just by $O(k/\log n)$ nodes.

Related work. As mentioned before, one important motivation for our work is fault-tolerant distributed computing, with the help of failure detection mechanisms such as proof-labeling schemes. Proof-labeling schemes were introduced in [22]. A tight bound of $\Theta(\log^2 n)$ bits on the size of the certificates for certifying MST was established in [19, 20]. Several variants of proof-labeling schemes have been investigated in the literature, including verification at distance greater than one [17], and the design of proofs with identity-oblivious certificates [12]. Connections between proof-labeling schemes and the design of distributed (silent) self-stabilizing algorithms were studied in [7]. Extensions of proof-labeling schemes for the design of (non-silent) self-stabilizing algorithms were investigated in [21]. In all these work, the number of nodes susceptible to detect an incorrect configuration is not considered, and the only constraint imposed on the error-detection mechanism is that an erroneous configuration must be detected by at least one node. Our work requires the number of nodes detecting an erroneous configuration to grow linearly with the number of errors.

Another important motivation for our work is property testing. Graph property testing was investigated in numerous papers (see [14, 15] for an introduction to the topic), and was recently extended to a non-deterministic setting [18, 24] in which the centralized algorithm is provided with a centralized certificate. Distributed property testing has been introduced in [8], and formalized in [9] (see also [13]). Our work may find applications to centralized property testing, but where the certificate is decentralized. Our error-sensitive scheme guarantees that if the current configuration of the network is ϵ-far from being correct, then probing a constant expected number of nodes is sufficient to detect that this configuration is erroneous.

From a higher perspective, our approach aims at closing the gap between local distributed computing and centralized computing in networks, by studying distributed error-detection mechanisms that perform locally, but generate individual outputs that are related to the global correctness of the system at hand. As such, it is worth mentioning other efforts in the same direction, including especially work in the context of centralized local computing, like, e.g., [10, 16, 26].

Distributed property testing and proof-labeling schemes are different forms of distributed decision mechanisms, which have been investigated under various models for distributed computing. We refer to [11] for a recent survey on distributed decision.

2 Model and definitions

Throughout the paper, all graphs are assumed to be connected and simple (no self-loops, and no parallel edges. Given a node v of a graph G, we denote by $N(v)$ the open neighborhood of v, i.e., the set of neighbors of v in G. In some contexts (as, e.g., MST), the considered graphs may be edge-weighted.

All results in this paper are stated in the classical LOCAL model [27] for distributed network computing, where networks are modeled by undirected graphs whose nodes model the computing entities, and edges model the communication links. Recall that the LOCAL model assumes that nodes are given distinct identities (a.k.a. IDs), and that computation proceeds in synchronous rounds. All nodes simultaneously start executing the given algorithm. At each round, nodes exchange messages with their neighbors, and perform individual computation. There are no limits placed on the message size, nor on the amount of computation performed at each round. Specifically, we are interested in *proof-labeling schemes* [22], which are well established mechanisms enabling to locally detect inconsistencies in the global states of networks with respect to some given boolean predicate. Such mechanisms involve a verification algorithm which performs in just a single round in the LOCAL model. In order to recall the definition of proof-labeling schemes, we first recall the definition of *distributed languages* [12].

A distributed language is a collection of labeled graphs, that is, a set $\mathcal{L}$ of pairs (G, ℓ) where G is a graph, and $\ell : V(G) \to \{0,1\}^*$ is a labeling function assigning a binary string to each node of G. Such a labelling may encode just a boolean (e.g., whether the node is in a dominating set or not), or an integer (e.g., in graph coloring), or a collection of neighbor IDs (e.g., for locally encoding a subgraph). A distributed language is said *constructible* if, for every graph G, there exists ℓ such that $(G, \ell) \in \mathcal{L}$. It is *Turing decidable* if there exists a (centralized) algorithm which, given (G, ℓ) returns whether $(G, \ell) \in \mathcal{L}$ or not. All distributed languages considered in this paper are always assumed to be constructible and Turing decidable.

Given a distributed language $\mathcal{L}$, a proof-labeling scheme for $\mathcal{L}$ is a pair prover-verifier $(\mathbf{p}, \mathbf{v})$, where $\mathbf{p}$ is an oracle assigning a certificate function $c : V(G) \to \{0,1\}^*$ to every labeled graph $(G, \ell) \in \mathcal{L}$, and $\mathbf{v}$ is a 1-round distributed algorithm[1] taking as input at each node v its identity $\mathsf{ID}(v)$, its label $\ell(v)$, and its certificate $c(v)$, such that, for every labeled graph (G, ℓ) the following two conditions are satisfied:

- If $(G, \ell) \in \mathcal{L}$ then $\mathbf{v}$ outputs *accept* at every node of G whenever all nodes of G are given the certificates provided by $\mathbf{p}$;
- If $(G, \ell) \notin \mathcal{L}$ then, for every certificate function $c : V(G) \to \{0,1\}^*$, $\mathbf{v}$ outputs *reject* in at least one node of G.

The first condition guarantees the existence of certificates allowing the given legally labeled graph (G, ℓ) to be globally accepted. The second condition guarantees that the verifier cannot be "cheated", that is, an illegally labeled graph will systematically be rejected by at least one node, whatever "fake" certificates are given to the nodes. It is known that every distributed language has a proof-labeling scheme [22].

To define the novel notion of *error-sensitive* proof-labeling schemes, we introduce the following notion of distance between labeled graphs. Let ℓ and ℓ' be two labelings of a same graph G. The *edit distance* between (G, ℓ) and (G, ℓ') is the minimum number of elementary operations required to transform (G, ℓ) into (G, ℓ'), where an elementary operation consists of replacing a node label by another label. That is, the edit distance between (G, ℓ) and (G, ℓ') is simply

$$|\{v \in V(G) : \ell(v) \neq \ell'(v)\}|.$$

The edit-distance from a labeled graph (G, ℓ) to a language $\mathcal{L}$ is the minimum, taken over all labelings ℓ' of G satisfying $(G, \ell') \in \mathcal{L}$, of the edit-distance between (G, ℓ) and (G, ℓ').

[1] That is, every node outputs after having communicated with all its neighbors only once.

Roughly, an error-sensitive proof-labeling scheme satisfies that the number of nodes that reject a labeled graph (G, ℓ) should be (at least) proportional to the distance between (G, ℓ) and the considered language.

▶ **Definition 1.** A proof-labeling scheme $(\mathbf{p}, \mathbf{v})$ for a language $\mathcal{L}$ is *error-sensitive* if there exists a constant $\alpha > 0$, such that, for every labeled graph (G, ℓ),

- If $(G, \ell) \in \mathcal{L}$ then $\mathbf{v}$ outputs *accept* at every node of G whenever all nodes of G are given the certificates provided by $\mathbf{p}$;
- If $(G, \ell) \notin \mathcal{L}$ then, for every certificate function $c : V(G) \to \{0,1\}^*$, $\mathbf{v}$ outputs *reject* in at least $\alpha\,\mathsf{d}$ nodes of G, where d is the edit distance between (G, ℓ) and $\mathcal{L}$, i.e., $\mathsf{d} = \mathrm{dist}\big((G, \ell), \mathcal{L}\big)$.

Note that the at least $\alpha\,\mathsf{d}$ nodes rejecting a labeled graph (G, ℓ) at edit-distance d from $\mathcal{L}$ do not need to be the same for all certificate functions.

3 Basic properties of error-sensitive proof-labeling schemes

Let us first illustrate the notion of error-sensitive proof-labeling scheme by exemplifying its design for a classic example of distributed languages. Let ACYCLIC be the following distributed language:

$$\textsc{acyclic} = \Big\{(G, \ell) : \forall v \in V(G),\ \ell(v) \in N(v) \cup \{\bot\} \\ \text{and} \bigcup_{v \in V(G)\,:\,\ell(v) \neq \bot} \{v, \ell(v)\} \text{ is acyclic}\Big\}$$

That is, the label of a node is interpreted as a pointer to some neighboring node, or to null. Then $(G, \ell) \in$ ACYCLIC if the subgraph of G described by the set of non-null pointers is acyclic. We show that ACYCLIC has an error-sensitive proof-labeling scheme. The proof of this result is easy, as fixing of the labels can be done locally, at the rejecting nodes. Nevertheless, the proposition and its proof serve as a basic example illustrating the notion of error-sensitive proof-labeling scheme.

▶ **Proposition 2.** ACYCLIC *has an error-sensitive proof-labeling scheme.*

Proof. Let $(G, \ell) \in$ ACYCLIC. Every node $v \in V(G)$ belongs to an in-tree rooted at a node r such that $\ell(r) = \bot$. The prover $\mathbf{p}$ provides every node v with its distance $d(v)$ to the root of its in-tree (i.e., number of hops to reach the root by following the pointers specified by ℓ). The verifier $\mathbf{v}$ proceeds at every node v as follows: first, it checks that $\ell(v) \in N(v) \cup \{\bot\}$; second, it checks that, if $\ell(v) \neq \bot$ then $d(\ell(v)) = d(v) - 1$, and if $\ell(v) = \bot$ then $d(v) = 0$. If all these tests are passed, then v accepts. Otherwise, it rejects. By construction, if (G, ℓ) is acyclic, then all nodes accept with these certificates. Conversely, if there is a cycle C in (G, ℓ), then let v be a node with maximum value $d(v)$ in C. Its predecessor in C (i.e., the node $u \in C$ with $\ell(u) = v$) rejects. So $(\mathbf{p}, \mathbf{v})$ is a proof-labeling scheme for ACYCLIC. We show that $(\mathbf{p}, \mathbf{v})$ is error-sensitive. Suppose that $\mathbf{v}$ rejects (G, ℓ) at $k \geq 1$ nodes. Let us replace the label $\ell(v)$ of each rejecting node v by the label $\ell'(v) = \bot$, and keep the labels of all other nodes unchanged, i.e., $\ell'(v) = \ell(v)$ for every node where $\mathbf{v}$ accepts. We have $(G, \ell') \in$ ACYCLIC. Indeed, by construction, the label of every node u in (G, ℓ') has a well-formatted label $\ell'(v) \in N(v) \cup \{\bot\}$. Moreover, let us assume, for the purpose of contradiction, that there is a cycle C in (G, ℓ'). By definition, every node v of this cycle is pointing to $\ell'(v) \in N(v)$. Thus $\ell'(v) = \ell(v)$ for every node of C, from which it follows that no nodes of C was rejecting with ℓ, a contradiction with the fact that, as observed before,

$\mathbf{v}$ rejects every cycle. Therefore $(G, \ell') \in$ ACYCLIC. Hence the edit-distance between (G, ℓ) and ACYCLIC is at most k. It follows that $(\mathbf{p}, \mathbf{v})$ is error-sensitive, with sensitivity parameter $\alpha \geq 1$. ◀

The definition of error-sensitiveness is based on the existence of a proof-labeling scheme for the considered language. However, two different proof-labeling schemes for the same language may have different sensitivity parameters α. In fact, we show that every non-trivial language admits a proof-labeling schemes which is *not* error-sensitive. That is, the following result shows that demonstrating the existence of a proof-labeling scheme that is *not* error-sensitive for a language does not prevent that language to have another proof-labeling scheme which *is* error-sensitive. We say that a distributed language is *trivially approximable* if there exists a constant d such that every labeled graph (G, ℓ) is at edit-distance at most d from $\mathcal{L}$. The proof of the following result can be found in the full version.

▶ **Proposition 3.** *Let $\mathcal{L}$ be a distributed language. Unless $\mathcal{L}$ is trivially approximable, there exists a proof-labeling scheme for $\mathcal{L}$ that is not error-sensitive.*

Recall that the fact that every distributed language has a proof-labeling scheme can be established by using a *universal* proof-labeling scheme $(\mathbf{p}_{univ}, \mathbf{v}_{univ})$ (see [17]). Given a distributed language $\mathcal{L}$, and a labeled graph $(G, \ell) \in \mathcal{L}$ on an n-node graph G, a universal certificate $c : V(G) \to \{0,1\}^*$ for that labeled graph is defined for every node $u \in V(G)$ by the triple $c(u) = (T, M, L)$ where nodes are ordered from 1 to n in arbitrary order, T is a vector with n entries indexed from 1 to n where $T[i]$ is the ID of the ith node v, $L[i]$ is the label $\ell(v)$ of the ith node v, and M is the adjacency matrix of G. The prover $\mathbf{p}_{univ}$ assigns $c(u)$ to every node $u \in V(G)$. The verifier $\mathbf{v}_{univ}$ then checks at every node u that its certificate is consistent with the certificates given to its neighbors (i.e., they all have the same T, L, and M, the indexes matches with the IDs, and the actual neighborhood of v is as it is specified in T, L and M). If this test is not passed, then $\mathbf{v}_{univ}$ outputs *reject* at u, otherwise it outputs *accept* or *reject* according to whether the labeled graph described by (M, L) is in $\mathcal{L}$ or not. It is easy to check that $(\mathbf{p}_{univ}, \mathbf{v}_{univ})$ is indeed a proof-labeling scheme for $\mathcal{L}$. The universal proof-labeling scheme has the following nice property, that we state as a lemma for further references in the text (see proof in the full version).

▶ **Lemma 4.** *If a distributed language $\mathcal{L}$ has an error-sensitive proof-labeling scheme, then the universal proof-labeling scheme applied to $\mathcal{L}$ is error-sensitive.*

While every distributed language has a proof-labeling scheme, we show, using Lemma 4, that there exist languages for which there are no error-sensitive proof-labeling schemes (see the full version for the proof).

▶ **Proposition 5.** *There exist languages that do not admit any error-sensitive proof-labeling scheme.*

▶ Remark. The language REGULAR used in the proof of Proposition 5 to establish the existence of languages that do not admit any error-sensitive proof-labeling schemes actually belongs to the class LCL of locally checkable labelings [25]. Therefore, the fact that a language is easy to check locally does not help for the design of error-sensitive proof-labeling schemes.

We complete this warmup section by some observations regarding the encoding of distributed data structures. Let us consider the following two distributed languages, both corresponding to spanning tree. The first language, ST_p, encodes the spanning trees using pointers to parents, while the second language, ST_l, encodes the spanning trees by listing all the incident edges of each node in these tree.

$$\textsc{st}_p = \Big\{(G,\ell) : \quad \forall v \in V(G),\, \ell(v) \in N(v) \cup \{\bot\} \text{ and } \Big(\bigcup_{v\in V(G)\,:\,\ell(v)\neq\bot} \{v,\ell(v)\}\Big) \text{ forms a spanning tree}\Big\}$$

$$\textsc{st}_l = \Big\{(G,\ell) : \quad \forall v \in V(G),\, \ell(v) \subseteq N(v) \text{ and } u \in \ell(v) \text{ iff } v \in \ell(u), \text{ and } \Big(\bigcup_{v\in V(G)} \bigcup_{u\in\ell(v)} \{u,v\}\Big) \text{ forms a spanning tree}\Big\}.$$

Obviously, $\textsc{st}_p$ is just a compressed version of $\textsc{st}_l$ as the latter can be constructed from the former in just one round. However, note that $\textsc{st}_p$ cannot be recover from $\textsc{st}_l$ in a constant number of rounds, because $\textsc{st}_p$ provides a consistent orientation of the edges in the tree. It follows that $\textsc{st}_p$ is an encoding of spanning trees which is actually strictly richer than $\textsc{st}_l$. This difference between $\textsc{st}_p$ and $\textsc{st}_l$ is not anecdotal, as we shall prove later that $\textsc{st}_l$ admits an error-sensitive proof-labeling scheme, while we show hereafter that $\textsc{st}_p$ is not appropriate for the design of error-sensitive proof-labeling schemes.

▶ **Proposition 6.** $\textsc{st}_p$ *does not admit any error-sensitive proof-labeling scheme.*

Proof. Let P_n be the n-node path $u_1, u_2, \ldots, u_n$ with n even. Let ℓ_0, ℓ_1, and ℓ_2 be labelings defined by $\ell_1(u_i) = u_{i+1}$ for all $1 \leq i < n$, and $\ell_1(u_n) = \bot$; $\ell_2(u_i) = u_{i-1}$ for all $1 < i \leq n$, and $\ell_2(u_1) = \bot$; and $\ell_3(u_i) = u_{i-1}$ for all $1 < i \leq \frac{n}{2}$, $\ell_3(u_i) = u_{i+1}$ for all $\frac{n}{2}+1 \leq i < n$, and $\ell_3(u_1) = \ell_3(u_n) = \bot$. We have $(P_n,\ell_1) \in \textsc{st}_p$ and $(P_n,\ell_2) \in \textsc{st}_p$, while the distance from (P_n,ℓ_3) to $\textsc{st}_p$ is at least $\frac{n}{2}$. Let $(\mathbf{p},\mathbf{v})$ be a proof-labeling scheme for $\textsc{st}_p$. Consider the case of (P_n,ℓ_3) where every u_i, $i = 1,\ldots,\frac{n}{2}$, is given the certificate assigned by $\mathbf{p}$ to u_i in (P_n,ℓ_2), and every u_i, $i = \frac{n}{2}+1,\ldots,n$, is given the certificate assigned by $\mathbf{p}$ to u_i in (P_n,ℓ_1). With such certificates, (P_n,ℓ_3) is rejected by $\mathbf{v}$ at $u_{\frac{n}{2}}$ and $u_{\frac{n}{2}+1}$ only. ◀

4 Characterization

We now define the notion of *local stability*, which allows us to characterize the distributed languages admitting an error-sensitive proof-labeling scheme. This notion naturally pops up in the context of proof-labeling schemes [22] and locally checkable proofs in general [17]. Indeed, in these latter frameworks, languages that are "hard" to prove, in the sense that they require certificates of large size (typically of $\Omega(n^2)$ bits), are not locally stable, in the sense that glueing together two legal labeled graphs, say by connecting them by an edge, results in a labeled graph which can be very far from being legal. Local stability also naturally pops up in the context of the classical construction tasks which admit local algorithms, such as $(\Delta+1)$-coloring and MIS [23]. Indeed, those tasks share the property that any partial solution can be extended to a larger solution without modifying the already assigned labels. Extending the partial solution actually only depends on the "border" of the current partial solution.

More specifically, let G be a graph, and let H be a subgraph of G, that is, a graph H such that $V(H) \subseteq V(G)$, and $E(H) \subseteq E(G)$. We denote by $\partial_G H$ the set of nodes at the *boundary* of H in G, that is, which belongs to $V(H)$, and are incident to an edge in $E(G) \setminus E(H)$.

Given a labeling ℓ of a graph G, and a subgraph H of G, the labeling ℓ_H denotes the labeling of H induced by ℓ restricted to the nodes of H:

$$\ell_H(v) = \begin{cases} \ell(v) & \text{if } v \in V(H) \\ \emptyset & \text{otherwise (where } \emptyset \text{ denotes the empty string).} \end{cases}$$

Roughly, a distributed language $\mathcal{L}$ is locally stable if, by copy-pasting parts of legal labelings with small cuts between these parts, the resulting labeled graph is not too far from being legal. More precisely, let G be a graph, and let $H_1, \dots, H_k$ be a family of connected subgraphs of G such that $(V(H_i))_{i=1,\dots,k}$ is a partition of $V(G)$. For every $i \in \{1, \dots, k\}$, let us consider a labeled graph $(G_i, \ell_i) \in \mathcal{L}$ such that H_i is a subgraph of G_i. Let ℓ be the labeling of G defined as $\ell = \sum_{i=1}^{k} \ell_i$, i.e. for every $v \in V(G)$, $\ell(v) = \ell_i(v)$ where i is such that $v \in V(H_i)$. We say that such a labeled graph (G, ℓ) is *induced* by the labeled graphs (G_i, ℓ_i), $i = 1, \dots, k$, *via* the subgraphs $H_1, \dots, H_k$.

▶ **Definition 7.** A language $\mathcal{L}$ is *locally stable* if there exists a constant $\beta > 0$, such that, for every labeled graph (G, ℓ) induced by labeled graphs $(G_i, \ell_i) \in \mathcal{L}$, $i = 1, \dots, k$, via subgraphs $H_1, \dots, H_k$, the edit-distance between (G, ℓ) and $\mathcal{L}$ is at most $\beta \, | \cup_{i=1}^{k} \partial_G H_i \cup \partial_{G_i} H_i|$.

That is, the labeled graph resulting from cut-and-pasting parts of legally labeled graphs (G_i, ℓ_i), $i = 1, \dots, k$, is at edit-distance from $\mathcal{L}$ upper bounded by the number of nodes at the boundary of the subgraphs H_i in G and G_i, and is independent of the number of nodes in each of these subgraphs H_i, $i = 1, \dots, k$.

We have now all ingredients to state our characterization result:

▶ **Theorem 8.** *Let $\mathcal{L}$ be a distributed language. $\mathcal{L}$ admits an error-sensitive proof-labeling scheme if and only if $\mathcal{L}$ is locally stable.*

Proof. We first show that if a distributed language $\mathcal{L}$ admits an error-sensitive proof-labeling scheme then $\mathcal{L}$ is locally stable. So, let $\mathcal{L}$ be a distributed language, and let $(\mathbf{p}, \mathbf{v})$ be an error-sensitive proof-labeling scheme for $\mathcal{L}$ with sensitivity parameter α. Let (G, ℓ) be a labeled graph induced by labeled graphs $(G_i, \ell_i) \in \mathcal{L}$, $i = 1, \dots, h$, via the subgraphs $H_1, \dots, H_h$ for some $h \geq 1$. Since, for every $i \in \{1, \dots, h\}$, $(G_i, \ell_i) \in \mathcal{L}$, there exists a certificate function c_i such that $\mathbf{v}$ accepts at every node of (G_i, ℓ_i) provided with the certificate function c_i. Now, let us consider the labeled graph (G, ℓ), with certificate $c_i(u)$ on every node $u \in V(H_i)$ for all $i = 1, \dots, h$. With such certificates, the nodes in $V(H_i)$ that are not in $\partial_G H_i \cup \partial_{G_i} H_i$ have the same close neighborhood in (G, ℓ) and in (G_i, ℓ_i). Therefore, they accept in (G, ℓ) the same way they accept in (G_i, ℓ_i). It follows that the number of rejecting nodes is bounded by $| \cup_{i=1}^{h} \partial_G H_i \cup \partial_{G_i} H_i|$, and therefore (G, ℓ) is at edit-distance at most $\frac{1}{\alpha} | \cup_{i=1}^{h} \partial_G H_i \cup \partial_{G_i} H_i|$ from $\mathcal{L}$. Hence, $\mathcal{L}$ is locally stable, with parameter $\beta = \frac{1}{\alpha}$.

It remains to show that if a distributed language is locally stable then it admits an error-sensitive proof-labeling scheme. Let $\mathcal{L}$ be a locally stable distributed language with parameter β. We prove that the universal proof-labeling scheme $(\mathbf{p}_{univ}, \mathbf{v}_{univ})$ for $\mathcal{L}$ (cf. Section 3) is error-sensitive for some parameter α depending only on β. Let $(G, \ell) \notin \mathcal{L}$, and let us fix some certificate function c. The verifier $\mathbf{v}_{univ}$ rejects in at least one node. We show that if $\mathbf{v}_{univ}$ rejects at k nodes, then the edit-distance between (G, ℓ) and $\mathcal{L}$ is at most k/α for some constant $\alpha > 0$ depending only on β. For this purpose, let us consider the outputs of $\mathbf{v}_{univ}$ applied to (G, ℓ) with certificate c, and let us define the graph G' as the graph obtained from G by removing all edges for which $\mathbf{v}_{univ}$ rejects at both extremities. Note that the graph G' may not be connected.

Let C be a connected component of G', with at least one node u at which $\mathbf{v}_{univ}$ accepts. Let $c(u) = (T, M, L)$ be the certificate of node u, as it should be in the universal proof-labeling scheme as described in section 3. Since $\mathbf{v}_{univ}$ accepts at u, node u shares the same triple

(T, M, L) with all its neighbors in G', as $\mathbf{v}_{univ}$ would reject at u otherwise. Similarly, for every neighbor v of u, it must be the case that v agrees on (T, M, L) with each of its neighbors w in G', as otherwise $\mathbf{v}_{univ}$ would have rejected at both v and w, and the edge $\{v, w\}$ would have been removed from G. It follows that all nodes in C share the same triple (T, M, L) as the one given to the accepting node u. Also (M, L) coincides with the local structure of C and its labeling ℓ at all accepting nodes in C. Moreover, since $\mathbf{v}_{univ}$ accepts at u, we have $(M, L) \in \mathcal{L}$. We denote by (G_C, ℓ_C) this labeled graph in $\mathcal{L}$.

Let C be a connected component of G' where all nodes reject. In fact, by construction, such a component is composed of just one isolated node. For every such isolated rejecting node u, let us denote by (G_C, ℓ_C) a labeled graph composed of a unique node, with ID equal to the ID of u, and with labeling $\ell_C(u)$ such that $(G_C, \ell_C) \in \mathcal{L}$.

Let $\mathcal{C}$ be the set of all connected components of G'. Let (G, ℓ') be the graph induced by labeled graphs (G_C, ℓ_C) via the subgraphs $C \in \mathcal{C}$. Note that (G, ℓ) and (G, ℓ') coincide, but for the isolated rejecting nodes. By local stability, (G, ℓ') is at edit-distance at most $\beta\, |\cup_{C\in\mathcal{C}}\, \partial_G C \cup \partial_{G_C} C|$ from $\mathcal{L}$. Now, the nodes in $\cup_{C\in\mathcal{C}} \partial_G C \cup \partial_{G_C} C$ are exactly the rejecting nodes. Thus the number k of rejecting nodes satisfies $k = |\cup_{C\in\mathcal{C}}\, \partial_G C \cup \partial_{G_C} C|$, and the edit-distance from (G, ℓ') to $\mathcal{L}$ is at most $\beta\, k$. On the other hand, by construction, the edit-distance between (G, ℓ') and (G, ℓ) is at most the number of isolated rejecting nodes, that is, at most k. Therefore, the edit-distance between (G, ℓ) and $\mathcal{L}$ is at most $(\beta + 1)\, k$. Thus, the universal proof-labeling scheme is error-sensitive, with parameter $\alpha = \frac{1}{\beta+1}$. ◀

Proposition 5 can be viewed as a corollary of Theorem 8 as it is easy to show that REGULAR is not locally stable. Nevertheless, local stability may not always be as easy to establish, because it is based on merging an arbitrary large number of labeled graphs. We thus consider another property, called *strong local stability*, which is easier to check, and which provides a sufficient condition for the existence of an error-sensitive proof-labeling scheme. Given two labeled graphs (G, ℓ) and (G', ℓ'), and a subgraph H of both G and G', the labeling $\ell - \ell_H + \ell'_H$ for G is the labeling such that, for every node $v \in V(G)$, $(\ell - \ell_H + \ell'_H)(v) = \ell'_H(v)$ if $v \in V(H)$, and $(\ell - \ell_H + \ell'_H)(v) = \ell(v)$ otherwise.

▶ **Definition 9.** A language $\mathcal{L}$ is *strongly locally stable* if there exists a constant $\beta > 0$, such that, for every graph H, and every two labeled graphs $(G, \ell) \in \mathcal{L}$ and $(G', \ell') \in \mathcal{L}$ admitting H as a subgraph, the labeled graph $(G, \ell - \ell_H + \ell'_H)$ is at edit-distance at most $\beta\, |\partial_{G'} H + \partial_G H|$ from $\mathcal{L}$.

The following lemma states that strong local stability is indeed a notion that is at least as strong as local stability (see proof in the full version).

▶ **Lemma 10.** *If a language $\mathcal{L}$ is strongly locally stable, then it is locally stable.*

In fact, strong local stability is a notion strictly stronger than local stability, although they coincide on bounded-degree graphs (cf. the full version). Thanks to the characterization in Theorem 8, and to the sufficient condition of Lemma 10, we immediately get error-sensitiveness for the language

$$\begin{aligned}\textsc{leader} = \big\{(G, \ell) :& \forall v \in V(G),\ \ell(v) \in \{0, 1\}\\ &\text{and there exists a unique } v \in V(G) \text{ for which } \ell(v) = 1\big\}.\end{aligned}$$

▶ **Corollary 11.** LEADER *admits an error-sensitive proof-labeling scheme.*

Also, one can show that the language ST_l of spanning trees, whenever encoded by adjacency lists, admits an error-sensitive proof-labeling scheme.

This is in contrast to Proposition 6 (see proof in the full version).

▶ **Corollary 12.** ST_l *admits an error-sensitive proof-labeling scheme.*

Also, Theorem 8 allows us to prove (see proof in the full version) that minimum-weight spanning tree (MST) is error-sensitive (whenever the tree is encoded locally by adjacency lists). More specifically, let

$$\text{MST}_l = \Big\{(G,\ell) : \forall v \in V(G),\, \ell(v) \subseteq N(v) \text{ and } \Big(\bigcup_{v\in V(G)} \bigcup_{u\in\ell(v)} \{u,v\}\Big) \text{ forms a MST}\Big\}. \quad (1)$$

▶ **Corollary 13.** MST_l *admits an error-sensitive proof-labeling scheme.*

5 Compact error-sensitive proof-labeling schemes

The characterization of Theorem 8 together with Lemma 4 implies an upper bound of $O(n^2)$ bits on the certificate size for the design of error-sensitive proof-labeling schemes for locally stable distributed languages. In this section, we show that the certificate size can be drastically reduced in certain cases. It is known that spanning tree and minimum spanning tree have proof-labeling schemes using certificates of polylogarithmic size $\Theta(\log n)$ bits [4, 22], and $\Theta(\log^2 n)$ bits [20], respectively. We show the proof-labeling schemes for spanning tree and MST in [4, 20, 22] are actually error-sensitive.

Recall that Proposition 6 proved that spanning tree does not admit any error-sensitive proof-labeling schemes whenever the tree is encoded at each node by a pointer to its parent: ST_p does not have any error-sensitive proof-labeling scheme. However, we show that ST_l, i.e., the language of spanning trees encoded by adjacency lists, does have a very compact error-sensitive proof-labeling scheme.

▶ **Theorem 14.** ST_l *has an error-sensitive proof-labeling scheme with certificates of size* $O(\log n)$ *bits.*

For figures that illustrate the construction of the following proof see the full version.

Proof. We show that the classical proof-labeling scheme $(\mathbf{p},\mathbf{v})$ for ST_l is error-sensitive. On instances of the language, i.e., on labeled graphs (G,ℓ) where ℓ encodes a spanning tree T of G, the prover $\mathbf{p}$ chooses an arbitrary root r of T, and then assigns to every node u a certificate $(I(u),P(u),d(u))$ where $I(u) = \mathsf{ID}(r)$, $P(u)$ is the ID of the parent of u in the tree (or $ID(u)$ if u is the root), and $d(u)$ the hop-distance in the tree from u to r. The verifier $\mathbf{v}$ at every node u first checks that:

- the adjacency lists are consistent, that is, if u is in the list of v, then v is in the list of u;
- there exists a neighbor of u with ID $P(u)$, we denote it $p(u)$;
- the node u has the same root-ID $I(u)$ as all its neighbors in G;
- $d(u) \geq 0$.

Then, the verifier checks that:

- if $\mathsf{ID}(u) \neq I(u)$ then $d(p(u)) = d(u)-1$, and for every other neighbor w in ℓ, $d(w) = d(u)+1$ and $p(w) = u$;
- if $\mathsf{ID}(u) = I(u)$ then $P(u) = ID(u)$, $d(u) = 0$, and every neighbor w of u in ℓ satisfies $d(w) = d(u) + 1$ and $p(w) = u$.

By construction, if $(G,\ell) \in \text{ST}_l$, then $\mathbf{v}$ accepts at every node. Also, it is easy to check that if $(G,\ell) \notin \text{ST}_l$, then, for every certificate function c, at least one node rejects.

To establish error-sensitivity for the above proof-labeling scheme, let us assume that $\mathbf{v}$ rejects at $k \geq 1$ nodes with some certificate function c. Then, let (G',ℓ') be the labeled graph coinciding with (G,ℓ) except that all edges for which $\mathbf{v}$ rejects at both endpoints are removed both from G, and from the adjacency lists in ℓ of the endpoints of these edges. Note that modifying ℓ into ℓ' only requires to edit labels of nodes that are rejecting. The graph G' may be disconnected. Let (C, ℓ'_C) be a connected component of (G', ℓ').

We claim that the edges of ℓ'_C form a forest in C. First note that if there is a cycle in the edges of ℓ'_C, then this cycle already existed in ℓ because we have added no edges when transforming ℓ into ℓ'. Consider such a cycle in ℓ, and the certificates given by $\mathbf{p}$. Either an edge is not oriented, that is no node uses this edge to point to its parent, or the cycle is consistently oriented and then distances are not consistent. In both cases two adjacent nodes of the cycle would reject when running $\mathbf{v}$. Then this cycle cannot be present in ℓ'_C, as at least one edge has been removed. As a consequence ℓ'_C form a forest of C. In C, if a node is connected to no other node by an edge of ℓ'_C, we will consider it as a tree of one node. With this convention, ℓ' is a spanning forest of G'.

We will now bound the number of trees in ℓ' by a function of k. The number of trees in ℓ' is equal to the sum of the number of trees in each component (C, ℓ'_C).

Let us run $\mathbf{v}$ on graph (C, ℓ'_C), and let k_C be the number of rejecting nodes. Observe that for every two nodes u and v in a component C, it holds that $I(u) = I(v)$. Indeed, otherwise, there would exist two adjacent nodes u and v in C with $I(u) \neq I(v)$, resulting in $\mathbf{v}$ rejecting at both nodes, which would yield the removal of $\{u, v\}$ from G. Consequently, at most one tree of ℓ'_C has a root whose ID corresponds to the ID given in the certificate. Then the number of trees in ℓ'_C is bounded by $k_C + 1$, and the total number of trees is bounded $\sum_C k_C + 1 = (\sum_C k_C) + |C|$.

Note that because of the design of the proof-labeling scheme, the nodes that accept when running $\mathbf{v}$ on (G, ℓ) also accept in (G', ℓ'). Then $\sum_C k_C \leq k$.

Let V_C be the set of nodes of C. It is easy to see that for all C, there exists a node of V_C that rejects when we run $\mathbf{v}$ on (G, ℓ). Indeed if there is no rejecting node, then no edge between C and the rest of the graph is removed, and then there is only one component in the graph. But then all node accept, which contradict the fact that $k \geq 1$. Then $|C| \leq k$.

So overall all ℓ' encodes a spanning forest with at most $2k$ trees. Such a labeling can thus be modified to get a spanning tree by modifying the labels of at most $4k$ nodes. That is, $(\mathbf{p}, \mathbf{v})$ is error-sensitive with parameter $\alpha \geq \frac{1}{4}$. ◀

Finally, we show that the compact proof-labelling scheme in [20, 22] for minimum-weight spanning tree, as specified in Eq. (1) of Section 4 is error-sensitive when the edge weights are distinct.

▶ **Theorem 15.** MST$_l$ *admits an error-sensitive proof-labeling scheme with certificates of size* $O(\log^2 n)$ *bits.*

Hereafter, we provide a sketch of proof for Theorem 15 (the complete proof is deferred to the full version).

Sketch of proof. A classic proof-labeling scheme for MST (see, e.g., [19, 20, 22]) consists in encoding a run of Borůvka algorithm. Recall that Borůvka algorithm maintains a spanning forest whose trees are called fragments, starting with the forest in which every node forms a fragment. The algorithm proceeds in a sequence of steps. At each step, it selects the lightest outgoing edge from every fragment of the current forest, and adds all these edges to the MST, while merging the fragments linked by the selected edges. This algorithm eventually produces a single fragment, which is a MSTof the whole graph, after at most a logarithmic number of steps.

At each node u, the certificate of the scheme consists of a table with a logarithmic number of fields, one for each round of Borůvka algorithm. For each step of Borůvka algorithm, the corresponding entry of the table provides a proof of correctness for the fragment including u, plus the certificate of a tree pointing to the lightest outgoing edge of the fragment. The verifier verifies the structures of the fragments, and the fact that no outgoing edges from each fragment have smaller weights than the one given in the certificate. It also checks that the different fields of the certificate are consistent (for instance, it checks that, if two adjacent nodes are in the same fragment at step i, then they are also in the same fragment at step $i+1$).

To prove that this classic scheme is error-sensitive, we perform the same decomposition as in the proof of Theorem 14, removing the edges that have both endpoints rejecting. We then consider each connected component C of the remaining graph, together with the subgraph S of that component described by the edges of the given labeling. In general, S is not a MST of the component C (S can even be disconnected). Nevertheless, we can still make use of the key property that the subgraph S is not arbitrarily far from a MST of the component C. Indeed, the edges of S form a forest, and these edges belong to a MST of the component. As a consequence, it is sufficient to add a few edges to S for obtaining a MST. To show that S is indeed not far from being a MST of C, we define a relaxed version of Borůvka algorithm, and show that the labeling of the nodes corresponds to a proper run of this modified version of Borůvka algorithm. We then show how to slightly modify both the run of the modified Borůvka algorithm, and the labeling of the nodes, to get a MST of the component. Finally, we prove that the collection of MSTs of the components can be transformed into a MST of the whole graph, by editing a few node labels only. ◀

6 Conclusion

In this paper, we consider on a stronger notion of proof-labeling scheme, named *error-sensitive* proof-labeling scheme, and provides a structural characterization of the distributed languages that can be verified using such a scheme in distributed networks. This characterization highlights the fact that some basic network properties do *not* have error-sensitive proof-labeling schemes, which is in contrast to the fact that every property has a proof-labeling scheme. However, important network properties, like acyclicity, leader, spanning tree, MST, etc., do admit error-sensitive proof-labeling schemes. Moreover, these schemes can be designed with the same certificate size as the one for the classic proof-labeling schemes for these properties.

Our study of error-sensitive proof-labeling schemes raises intriguing questions. In particular, we observed that every distributed languages seems to fit in one of the following two scenarios: either it does not admit error-sensitive proof-labeling schemes, or it admits error-sensitive proof-labeling schemes with the same certificate size as the most compact proof-labeling schemes known for this language. We do not know whether there exists a distributed language admitting error-sensitive proof-labeling schemes, but such that all error-sensitive proof-labeling schemes for that language use certificates larger than the ones used for the most compact proof-labeling schemes for that language.

Proximity-sensitivity. Another desirable property for a proof-labeling scheme is *proximity-sensitivity*, requiring that every error is detected by a node close to that error. Proximity-sensitivity appears to be a very demanding notion, even stronger than error-sensitivity, for the former implies the later whenever the errors are spread out in the network. It would be informative to provide a structural characterization of the distributed languages that can be verified using proximity-sensitive proof-labeling schemes, and at which cost in term of label size.

References

1 Yehuda Afek and Shlomi Dolev. Local stabilizer. *J. Parallel Distrib. Comput.*, 62(5):745–765, 2002. doi:10.1006/jpdc.2001.1823.

2 Yehuda Afek, Shay Kutten, and Moti Yung. The local detection paradigm and its application to self-stabilization. *Theor. Comput. Sci.*, 186(1-2):199–229, 1997. doi:10.1016/S0304-3975(96)00286-1.

3 Baruch Awerbuch, Shay Kutten, Yishay Mansour, Boaz Patt-Shamir, and George Varghese. Time optimal self-stabilizing synchronization. In *25th ACM Symposium on Theory of Computing (STOC)*, pages 652–661, 1993. doi:10.1145/167088.167256.

4 Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction (extended abstract). In *32nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 268–277, 1991. doi:10.1109/SFCS.1991.185378.

5 Joffroy Beauquier, Sylvie Delaët, Shlomi Dolev, and Sébastien Tixeuil. Transient fault detectors. *Distributed Computing*, 20(1):39–51, 2007. doi:10.1007/s00446-007-0029-x.

6 Lélia Blin and Pierre Fraigniaud. Space-optimal time-efficient silent self-stabilizing constructions of constrained spanning trees. In *35th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 589–598, 2015. doi:10.1109/ICDCS.2015.66.

7 Lélia Blin, Pierre Fraigniaud, and Boaz Patt-Shamir. On proof-labeling schemes versus silent self-stabilizing algorithms. In *16th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 18–32, 2014. doi:10.1007/978-3-319-11764-5_2.

8 Zvika Brakerski and Boaz Patt-Shamir. Distributed discovery of large near-cliques. *Distributed Computing*, 24(2):79–89, 2011.

9 Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. In *30th International Symposium on Distributed Computing (DISC)*, pages 43–56, 2016. doi:10.1007/978-3-662-53426-7_4.

10 Guy Even, Moti Medina, and Dana Ron. Best of two local models: Local centralized and local distributed algorithms. Technical report, arXiv CoRR abs/1402.3796, 2014.

11 Laurent Feuilloley and Pierre Fraigniaud. Survey of distributed decision. *Bulletin of the EATCS*, 119, 2016.

12 Pierre Fraigniaud, Amos Korman, and David Peleg. Towards a complexity theory for local distributed computing. *J. ACM*, 60(5):35:1–35:26, 2013. doi:10.1145/2499228.

13 Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. Distributed testing of excluded subgraphs. In *30th Int. Symposium on Distributed Computing (DISC)*, volume 9888 of *LNCS*, pages 342–356. Springer, 2016.

14 Oded Goldreich. A brief introduction to property testing. In *Studies in Complexity and Cryptography – Miscellanea on the Interplay between Randomness and Computation*, number 6650 in LNCS, pages 465–469. Springer, 2011. doi:10.1007/978-3-642-22670-0_31.

15 Oded Goldreich. Introduction to testing graph properties. In *Studies in Complexity and Cryptography – Miscellanea on the Interplay between Randomness and Computation*, number 6650 in LNCS, pages 470–506. Springer, 2011. doi:10.1007/978-3-642-22670-0_32.

16 Mika Göös, Juho Hirvonen, Reut Levi, Moti Medina, and Jukka Suomela. Non-local probes do not help with many graph problems. In *30th International Symposium on Distributed Computing (DISC)*, pages 201–214, 2016. doi:10.1007/978-3-662-53426-7_15.

17 Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory of Computing*, 12(19):1–33, 2016. doi:10.4086/toc.2016.v012a019.

18 Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. In *6th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 133–142, 2015. doi:10.1145/2688073.2688079.

19 Liah Kor, Amos Korman, and David Peleg. Tight bounds for distributed MST verification. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 69–80, 2011. doi:10.4230/LIPIcs.STACS.2011.69.

20 Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4):253–266, 2007. doi:10.1007/s00446-007-0025-1.

21 Amos Korman, Shay Kutten, and Toshimitsu Masuzawa. Fast and compact self-stabilizing verification, computation, and fault detection of an MST. *Distributed Computing*, 28(4):253–295, 2015. doi:10.1007/s00446-015-0242-y.

22 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010. doi:10.1007/s00446-010-0095-3.

23 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992. doi:10.1137/0221015.

24 László Lovász and Katalin Vesztergombi. Non-deterministic graph property testing. *Combinatorics, Probability & Computing*, 22(5):749–762, 2013. doi:10.1017/S0963548313000205.

25 Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

26 Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Compututer Science*, 381(1-3):183–196, 2007. doi:10.1016/j.tcs.2007.04.040.

27 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.