THEORY OF COMPUTING, Volume 12 (19), 2016, pp. 1–33
www.theoryofcomputing.org

Locally Checkable Proofs in Distributed Computing

Mika Göös Jukka Suomela

Received August 20, 2012; Revised November 3, 2016; Published November 29, 2016

Abstract: We study *decision problems* related to *graph properties* from the perspective of *nondeterministic distributed algorithms*. For a *yes*-instance there must exist a *proof* that can be verified with a distributed algorithm: all nodes must accept a valid proof, and at least one node must reject an invalid proof. We focus on *locally checkable proofs* that can be verified with a constant-time distributed algorithm. For example, it is easy to prove that a graph is bipartite: the locally checkable proof gives a 2-coloring of the graph, which only takes 1 bit per node. However, it is more difficult to prove that a graph is *not* bipartite—it turns out that any locally checkable proof requires $\Omega(\log n)$ bits per node.

In this paper we classify graph properties according to their local proof complexity, i. e., how many bits per node are needed in a locally checkable proof. We establish tight or near-tight results for classical graph properties such as the chromatic number. We show that the local proof complexities form a natural hierarchy of complexity classes: for many classical graph properties, the local proof complexity is either 0, $\Theta(1)$, $\Theta(\log n)$, or $\mathrm{poly}(n)$ bits per node. Among the most difficult graph properties are proving that a graph is symmetric (has a non-trivial automorphism), which requires $\Omega(n^2)$ bits per node, and proving that a graph is *not* 3-colorable, which requires $\Omega(n^2/\log n)$ bits per node. Any property of connected graphs admits a trivial proof with $O(n^2)$ bits per node.

ACM Classification: C.2.4, F.1.3

AMS Classification: 68M14, 68Q15, 68Q25, 68Q85, 03F20

Key words and phrases: nondeterminism, distributed computing, locality, graphs

A preliminary version of this paper appeared in the Proceedings of the 30th ACM Symposium on Principles of Distributed Computing (PODC 2011) [13].

DOI: 10.4086/toc.2016.v012a019

1 Introduction

This paper studies *decision problems* related to *graph properties* from the perspective of distributed graph algorithms. In distributed graph algorithms, the nodes of an unknown communication network (modeled as a graph) work together in order to solve graph problems related to the structure of the communication graph itself. As argued by Fraigniaud in his PODC 2010 keynote talk [10], the appropriate model for distributed *yes–no* verification is the following:

- For a *yes*-instance, all nodes must output 1.
- For a *no*-instance, at least one node must output 0.

Intuitively, if we have an acceptable input, all nodes will be happy, and if we have an invalid input, at least one node has to raise the alarm.

Our focus is on distributed verification that is made *locally*, by using a *constant-time* distributed algorithm [21, 29]. Throughout this paper, we follow the convention that the running time of a distributed algorithm equals the number of synchronous communication rounds. In particular, in a local algorithm all nodes stop after $O(1)$ communication rounds and announce their outputs. Equivalently, each node makes its *yes–no* decision based on its constant-radius neighborhood in the communication graph.

Recall that a *graph property* is a family of graphs that is closed under isomorphism. We say that a graph property $\mathcal{P}$ is *locally checkable* if it can be checked by a local algorithm. That is, there exists a local algorithm, called *verifier*, that accepts all *yes*-instances $G \in \mathcal{P}$ and rejects all *no*-instances $G \notin \mathcal{P}$.

Examples. An easy example of a locally checkable property is determining if a given connected graph is Eulerian: it is sufficient that each node outputs 1 if its degree is even, and 0 otherwise.

Another example is checking if a given graph is a line graph. If the nodes have unique identifiers, a constant-time verifier can check that the graph does not contain any of the nine forbidden subgraphs in Beineke's [5] characterization of line graphs.

However, local checkability as such does not seem to lead to an interesting complexity theory—for many well-studied graph properties, it is often easy to show that they are *not* locally checkable. The key insight of Korman et al. [15, 16, 18, 19] is to study locally checkable *proofs*.

1.1 Locally checkable proofs

To illustrate the idea of locally checkable proofs, we start with a canonical example. Consider the problem of checking if a given graph is bipartite. This is not a locally checkable property—indeed, if we consider odd vs. even cycles, we can see that any verifier that solves the problem must have the running time $\Omega(n)$. However, if we want to convince a local algorithm that the graph is indeed bipartite, we can augment the graph with a *locally checkable proof*. It is sufficient to give 1 bit of proof per node: if the graph is bipartite, we can give a 2-coloring of the graph as the proof, and a local verifier can check that the proof is correct. Conversely, if the graph is not bipartite, no matter what proof bits we choose, at least one node will detect that the proof is invalid. Hence we say the property of bipartiteness is in the class $\mathsf{LCP}(1)$: for any bipartite graph, there is a locally checkable proof of size 1 bit per node. Precise definitions are given in Section 2.

The concept of locally checkable proofs is related to locally checkable properties as the familiar complexity class NP is related to the class P. If a problem is in NP, then *yes*-instances have a concise proof that can be verified in P. Similarly, if the problem is in $\mathsf{LCP}(f)$, then *yes*-instances have a concise proof with at most $f(n)$ bits per node and the proof itself is checkable with a local algorithm. Equivalently, we can interpret locally checkable proofs as *nondeterministic* local algorithms: in the algorithm, each node can nondeterministically guess $f(n)$ bits.

1.2 Contributions

The main contributions of this paper are the following:

1. We define the class $\mathsf{LCP}(f)$ that consists of graph properties that admit locally checkable proofs of size $f(n)$ bits per node. This model is related to those studied by Korman et al. [15, 16, 18, 19] and Fraigniaud et al. [11], but strictly stronger than both; see Section 3.

2. We catalog graph properties according to their local proof complexities, and we show that the $\mathsf{LCP}(f)$ classes form a natural hierarchy of decision problems. In particular, there are natural graph properties that separate the following levels of the hierarchy: $\mathsf{LCP}(0)$, $\mathsf{LCP}(O(1))$, $\mathsf{LCP}(O(\log n))$, and $\mathsf{LCP}(\mathrm{poly}(n))$.

   We refer to Tables 1 and 2 for a summary. Table 1 catalogs the complexity of verifying *graph properties* (given a graph G, prove that $G \in \mathcal{P}$), while Table 2 catalogs the complexity of verifying *solutions to graph problems* (given a graph G and a solution X to graph problem $\mathcal{P}$, prove that the solution is correct).

3. We argue that $\mathsf{LogLCP} = \mathsf{LCP}(O(\log n))$ is a particularly good candidate for a complexity class of independent interest. The class is robust to variations in the exact definition of the LCP hierarchy. Many graph properties are contained in LogLCP and not contained in $\mathsf{LCP}(o(\log n))$.

4. We present proof techniques that can be used to derive tight and near-tight lower bounds for the local proof complexity. We show how to apply tools from other fields of computer science and mathematics: results in extremal graph theory, fooling set arguments from the field of communication complexity, and gadgets that are typical in NP-hardness proofs. Here we mention, in particular, our tight proof-size lower bound of $\Omega(\log n)$ for non-bipartiteness and related graph properties, and a near-tight lower bound of $\Omega(n^2/\log n)$ for non-3-colorability.

2 Definitions and examples

To begin, we fix a family $\mathcal{F}$ of input graphs. For example, $\mathcal{F}$ might just be the set of all graphs (undirected, unless otherwise noted), or perhaps $\mathcal{F}$ is the set of all *connected* graphs, i. e., we can work under a *promise*. For a graph $G \in \mathcal{F}$, we write $V(G)$ for the set of nodes, $E(G)$ for the set of edges, and $n(G) = |V(G)|$ for the number of nodes in G. If graph G is clear from the context, we simply use the symbols V, E, and n. As is standard in distributed computing, we assume that the nodes of any $G \in \mathcal{F}$ are uniquely identified with $O(\log n)$-bit natural numbers ("identifiers"), that is, $V(G) \subseteq \{1, 2, \ldots, \mathrm{poly}(n(G))\}$. (As

Class	Proof size s	Graph property $\mathcal{P}$	Graph family $\mathcal{F}$	References
$\mathsf{LCP}(0)$	0	Eulerian graphs	connected	§ 1
	0	line graphs	general	§ 1
$\mathsf{LCP}(O(1))$	$\Theta(1)$	s–t reachability	undirected	§ 4.1
	$\Theta(1)$	s–t unreachability	undirected	§ 4.1
	$\Theta(1)$	s–t unreachability	directed	§ 4.1
	$\Theta(1)$	s–t connectivity $= k$	planar	§ 4.2
	$\Theta(1)$	bipartite graphs	general	§ 4.3
	$\Theta(1)$	even $n(G)$	cycles	
$\mathsf{LCP}(O(\log k))$	$O(\log k)$	s–t connectivity $= k$	general	§ 4.2
	$O(\log k)$	chromatic number $\leq k$	general	§ 2
LogLCP	$O(\log n)$	any $\mathsf{coLCP}(0)$ property	connected	§ 7.3
	$O(\log n)$	any monadic Σ_1^1 property	connected	§ 7.5
	$\Theta(\log n)$	odd $n(G)$	cycles	§ 5
	$\Theta(\log n)$	chromatic number > 2	connected	§ 5
$\mathsf{LCP}(\text{poly}(n))$	$\Theta(n)$	fixed-point-free symmetry	trees	§ 6.2
	$\Theta(n^2)$	symmetric graphs	connected	§ 6.1
	$\Omega(n^2/\log n)$	chromatic number > 3	connected	§ 6.3
	$O(n^2)$	any property	connected	§ 6

Table 1: The local proof complexity of verifying graph property $\mathcal{P}$, when we are promised that the input graph is in graph family $\mathcal{F}$. Constant k is a natural number. In reachability problems, nodes s and t are labeled; otherwise the graphs are unlabeled.

Class	Proof size s	Graph problem $\mathcal{P}$	Graph family $\mathcal{F}$	References
$\mathsf{LCP}(0)$	0	maximal matching	general	
	0	any LCL problem	general	§ 3, [21]
	0	any LD problem	connected	§ 3, [11]
$\mathsf{LCP}(O(1))$	$\Theta(1)$	maximum matching	bipartite	§ 4.3
$\mathsf{LCP}(O(\log W))$	$O(\log W)$	max-weight matching	bipartite	§ 4.3
LogLCP	$O(\log n)$	any $\mathsf{coLCP}(0)$ problem	connected	§ 7.3
	$\Theta(\log n)$	leader election	connected	§ 5, [18]
	$\Theta(\log n)$	spanning tree	connected	§ 5, [18]
	$\Theta(\log n)$	maximum matching	cycles	§ 5
	$\Theta(\log n)$	Hamiltonian cycle	connected	§ 5
$\mathsf{LCP}(\infty)$	unlimited	any NLD problem	connected	§ 3, [11]
	unlimited	any $\mathsf{NLD}^{\#n}$ problem	connected	§ 3, [11]

Table 2: The local proof complexity of verifying a solution to graph problem $\mathcal{P}$, when we are promised that the input graph is in graph family $\mathcal{F}$. Here W is the maximum weight of an edge.

an exception, we study *anonymous* models without unique identifiers in Sections 7.1–7.2.) Depending on the problem that we study, nodes and edges may also be associated with weights, colors, labels, etc.; if this is the case, we say that the graph is *labeled*, and otherwise it is *unlabeled*.

2.1 Proofs, verifiers and locality

Proofs. A *proof* P for G is a function $P\colon V(G) \to \{0,1\}^\star$ that associates a binary string with each node of G. The *size* $|P|$ of a proof P is the *maximum* number of bits in any string $P(v)$, over all $v \in V(G)$. We write ε for an empty proof of size 0.

Verifiers. A *verifier* $\mathcal{A}$ is a function that maps each triple (G,P,v) to a binary output 0 or 1. Here $G \in \mathcal{F}$ is a graph, $P\colon V(G) \to \{0,1\}^\star$ is a proof, and $v \in V(G)$ is a node of G. Intuitively, $\mathcal{A}(G,P,v)$ is the *output* of node v if we run the distributed algorithm $\mathcal{A}$ in graph G and each node $u \in V(G)$ is provided with *input* $P(u)$.

Remark 2.1. The issue of whether or not $\mathcal{A}$ is computable (or polynomial-time bounded) is usually immaterial to our considerations. For simplicity, we do not impose any such resource constraints on $\mathcal{A}$; our focus will be on studying the limitations of verifiers arising from *locality*—as defined next.

Local verifiers. For a natural number $r \in \mathbb{N}$ and a node $v \in V(G)$, let $V[v,r] \subseteq V(G)$ be the ball of radius r about v, i. e., set of nodes that are within distance r from v (the shortest path from v to any node in $V[v,r]$ has at most r edges). Let $G[v,r]$ be the subgraph of G induced by $V[v,r]$, and let $P[v,r]\colon V[v,r] \to \{0,1\}^\star$ be the restriction of a proof $P\colon V(G) \to \{0,1\}^\star$ to $V[v,r]$.

A verifier $\mathcal{A}$ is a *local verifier* if there exists a constant $r \in \mathbb{N}$ such that

$$\mathcal{A}(G,P,v) = \mathcal{A}(G[v,r],P[v,r],v) \text{ for all } G,P,v\,.$$

That is, the output of a node v only depends on the input in its radius-r neighborhood. Constant r is the *local horizon* of $\mathcal{A}$. (Alternatively, if we consider Peleg's [25] LOCAL model, a local verifier can be defined as a constant-time distributed algorithm: a local verifier with horizon r can be implemented as a distributed message-passing algorithm that completes in r synchronous communication rounds.)

2.2 Locally checkable proofs

A *graph property* $\mathcal{P} \subseteq \mathcal{F}$ is a subset of graphs that is closed under isomorphism. Examples of graph properties include Hamiltonian graphs, Eulerian graphs, bipartite graphs, connected graphs, line graphs, trees, and cycles.

Definition 2.2. A graph property $\mathcal{P} \subseteq \mathcal{F}$ admits *locally checkable proofs* of size $s\colon \mathbb{N} \to \mathbb{N}$ on family $\mathcal{F}$ if there is a local verifier $\mathcal{A}$ that satisfies the following properties.

(i) *Completeness:* If $G \in \mathcal{P}$ then there exists a proof $P\colon V(G) \to \{0,1\}^\star$ with $|P| \le s(n(G))$ such that $\mathcal{A}(G,P,v) = 1$ for each node $v \in V(G)$.

(ii) *Soundness:* If $G \in \mathcal{F} \setminus \mathcal{P}$ then for every proof $P\colon V(G) \to \{0,1\}^\star$ there is at least one node $v \in V(G)$ with $\mathcal{A}(G,P,v) = 0$.

That is, *yes*-instances have a valid proof that is accepted by all nodes, while *no*-instances are always rejected by at least one node. If f is a function that associates a valid proof $P = f(G)$ with each $G \in \mathcal{P}$, we say that the pair $(f,\mathcal{A})$ is a *proof labeling scheme*.

If a property $\mathcal{P}$ admits locally checkable proofs of size s, we write $\mathcal{P} \in \mathsf{LCP}(s)$. We use $\mathsf{coLCP}(s)$ to denote the class of graph properties whose complement is in $\mathsf{LCP}(s)$; that is, if $\mathcal{F} \setminus \mathcal{P} \in \mathsf{LCP}(s)$, we write $\mathcal{P} \in \mathsf{coLCP}(s)$. The class LogLCP consists of properties that are in $\mathsf{LCP}(s)$ for $s = O(\log n)$; we will see in Section 5 that LogLCP admits many alternative characterizations, which makes it a particularly robust class.

Examples. As we observed in Section 1, Eulerian graphs and line graphs can be verified without a proof, and hence they are in $\mathsf{LCP}(0)$. As further observed in Section 1.1, bipartite graphs are not in $\mathsf{LCP}(0)$ but they are contained in $\mathsf{LCP}(1)$, as they can be verified with one bit of input per node. More generally, if a graph has chromatic number at most k, we can prove it with $\lceil \log k \rceil$ bits per node: simply give a proper k-coloring as the proof.

2.3 Extension: solutions to graph problems

If we consider labeled graphs, we can also define graph properties such as independent sets (*"nodes with label* 1 *form an independent set"*) or spanning trees (*"edges with label* 1 *induce a spanning tree"*). That is, we can interpret the labeling as a *solution to a graph problem*, and we can extend the definitions of locally checkable proofs to verify solutions of graph problems.

Formally, a *graph problem* $\mathcal{P}$ associates any graph $G \in \mathcal{F}$ with a set $\mathcal{P}(G)$ of *solutions*; each solution is a labeled version of graph G. Contrary to the setting of graph properties, there are two natural variants to verifying solutions of graph problem $\mathcal{P}$:

- *Strong* proof labeling scheme $(f, \mathcal{A})$: for any graph $G \in \mathcal{F}$, for any solution $X \in \mathcal{P}(G)$, there is a locally checkable proof $f(X)$ that is accepted by $\mathcal{A}$, and any incorrect labeling $X \notin \mathcal{P}(G)$ is rejected by $\mathcal{A}$.

- *Weak* proof labeling scheme $(f, \mathcal{A})$: for any graph $G \in \mathcal{F}$, there is at least one solution $X \in \mathcal{P}(G)$ such that there is a locally checkable proof $f(X)$ that is accepted by $\mathcal{A}$, and any incorrect labeling $X \notin \mathcal{P}(G)$ is rejected by $\mathcal{A}$.

Put differently, in a strong proof labeling scheme, an adversary can choose both the input and the solution, and we must come up with a locally checkable proof. However, in a weak proof labeling scheme, an adversary chooses the input but we can choose a solution. These two notions are different even for natural verification tasks.

Example 2.3 (Separating Strong from Weak)**.** Consider the task of verifying whether a set of vertices $C \subseteq V(G)$ is a 2-approximation of a minimum vertex cover. It is straightforward to prove that without proof bits, no *strong* proof labeling scheme exists for this task, i. e., the approximation ratio achieved by an arbitrary C cannot be determined locally. On the other hand, on bounded-degree graphs there is a constant-time local algorithm [4] that always outputs a set $C^* \subseteq V(G)$ that is a 2-approximation of the minimum vertex cover—it is this $C = C^*$ that we can verify without proof bits in the *weak* sense.

In contrast to the above example, for the remaining problems studied in this paper, the local proof complexities of strong and weak proof labeling schemes are within a constant factor of each other. All proof labeling schemes that we will exhibit are of the strong variety whereas our lower-bound results preclude not only the existence of strong proof labeling schemes for various problems, but also the existence of weak proof labeling schemes.

3 Comparison with other models

Determinism. Our definition of the LCP hierarchy is an extension of *locally checkable labelings* (LCL) introduced by Naor and Stockmeyer [21] in their seminal work. Naor and Stockmeyer focus on bounded-degree graphs and constant-size labels, but if we generalize the class LCL in a straightforward manner, we arrive at the class $\mathsf{LCP}(0)$.

Our classes $\mathsf{LCP}(f)$ with $f > 0$ thus extend the classical LCL concept by providing $f(n)$ bits of additional information per node. We have chosen the terminology “LCP” to emphasize this connection, admitting that it is a departure from some of the other terminology available in the literature, as outlined next.

Nondeterminism. Our model is not the first, nor the last attempt at introducing nondeterminism in the setting of local decision problems. Indeed, similar extensions have appeared in prior and independent work. Two specific competing models are

- *proof labeling schemes* of Korman et al. [15, 16, 18, 19], and
- *nondeterministic local decision* of Fraigniaud et al. [11].

Our LCP model is the strongest among the trio: positive results in the other models imply positive results in our model; this makes our lower-bound results widely applicable. In short, while an LCP verifier algorithm is limited *only by locality*, the verifiers in the other two models have additional restrictions.

3.1 Proof labeling schemes

The *proof labeling scheme* model of Korman et al. [15, 16, 18, 19] is inspired by the classical notion of *labeling schemes* (see Gavoille and Peleg [12] for a survey). In this model,

(i) the verifier has run-time of 1 communication round, and
(ii) a node cannot see the identifiers nor the input labels of its neighboring nodes.

That is, the output of a single node must be determined on the basis of its own identifier, own input label, own proof label and the proof labels of the neighboring nodes. Thus, an upper bound on the local proof complexity in this model provides an extremely efficient local checking procedure.

Because of these restrictions, verifiers in the proof labeling scheme model are sometimes rather weak. For example, there are simple LCL problems that cannot be solved without proof labels in this model: one example is the *agreement problem* of checking whether all nodes in a connected graph are assigned the same input label [18, Lemma 2.3]; another example is checking whether the input graph is triangle-free. Hence the notion of proof labeling schemes is not a straightforward generalization of the LCL model—something our LCP model strives to be.

However, we note that in some cases the gap between the models can be bridged at the cost of an additive proof-size overhead of $\Theta(\log n)$—or however many bits of local data (node identifiers, input labels) are hidden by the restrictions *(i)–(ii)*. This means, for instance, that the two models admit roughly the same analysis at the $\mathsf{LCP}(\mathrm{poly}(n))$ level (Section 6).

In comparison with prior work related to proof labeling schemes, the main challenges of the present paper can be summarized as follows:

1. Logarithmic lower bounds from prior work do not apply directly. It is harder to fool an LCP verifier that sees all the information available in its constant-radius neighborhood (in particular, $O(\log n)$-bit identifiers). Therefore we need stronger techniques to prove $\Omega(\log n)$ proof-size lower bounds in Section 5.

2. Superlogarithmic lower bounds would apply, but few such bounds are known for graph properties. It seems that the main focus in the prior work has been on problems related to labeled graphs. We can reuse ideas from prior work in Sections 6.1–6.2 in the context of symmetric graphs, but we are not aware of any prior work related to non-3-colorability that we could apply in Section 6.3.

For logarithmic lower bounds, our novel idea is to invoke an extremal result of Bondy and Simonovits [6] in the context of a “gluing” argument. For superlogarithmic lower bounds, the novel idea is to apply the *disjointness* problem to construct a difficult graph problem, while prior work has focused on constructions that are based on *equality*.

3.2 Nondeterministic local decision

Independently of our work, Fraigniaud et al. [11] have also studied nondeterminism (along with randomness) in the context of distributed decision problems. Here, the major difference to our model is that the local proofs $P\colon V(G) \to \{0,1\}^\star$ are not allowed to depend on the identifier assignment on G. This implies, for example, that many leader election problems are not solvable at all in their model, no matter how large P is (cf. Section 7.1).

To connect the results of Fraigniaud et al. to our work, let us define $\mathsf{LCP}'(f)$ similarly to $\mathsf{LCP}(f)$ but restricted to computable properties of connected graphs. With this notation, the class LD of local decision problems defined by Fraigniaud et al. is equal to $\mathsf{LCP}'(0)$. The nondeterministic variants of LD are called NLD and $\mathsf{NLD}^{\#n}$; in the latter class each node knows the total number of nodes in the graph. It turns out that our class $\mathsf{LCP}'(\infty)$ is equal to $\mathsf{NLD}^{\#n}$: both classes contain all computable properties of connected graphs. Hence using the separation results of Fraigniaud et al. we have

$$\mathsf{LCP}'(0) = \mathsf{LD} \subsetneq \mathsf{NLD} \subsetneq \mathsf{NLD}^{\#n} = \mathsf{LCP}'(\infty)\,.$$

While Fraigniaud et al. place one class between the extreme ends of $\mathsf{LCP}'(0)$ and $\mathsf{LCP}'(\infty)$, our work introduces an entire hierarchy of $\mathsf{LCP}(f)$ classes.

4 Problems in $\mathsf{LCP}(O(1))$

As a warm-up, this section gives examples of graph properties and graph problems that admit locally checkable proofs of size $O(1)$ but for which there is no locally checkable proof of size 0. We will see that many fundamental problems related to graph connectivity are in this class.

4.1 Reachability

To ask meaningful questions about connectivity in the LCP model, we require that two nodes s and t are always distinguished in the input graph G; that is, we have the promise that there is exactly one node with label s and exactly one node with label t. It is easy to see that in $\mathsf{LCP}(0)$ we cannot check whether there is a path from s to t in G. However, many questions related to reachability and connectivity are in $\mathsf{LCP}(O(1))$.

Let us first consider the *s–t reachability* problem in an undirected graph G, i. e., proving that there is a path from s to t. This problem admits a locally checkable proof of size 1: we find a shortest path from s to t in G, define that $U \subseteq V$ consists of all nodes on the shortest path, and set $P(v) = 1$ iff $v \in U$. A verifier can locally check that: (i) $s,t \in U$; (ii) s and t have unique neighbors in U; and (iii) every $u \in U \setminus \{s,t\}$ has exactly two neighbors in U [14, p. 130].

Interestingly, the above method breaks down in directed graphs because of back-edges. In graphs of maximum degree Δ, one can still give an easy upper bound of $O(\log\Delta)$ by using edge pointers in the proof labeling to describe a path from s to t, but it is an open problem whether directed s–t reachability is in $\mathsf{LCP}(O(1))$ for general graphs (see also Ajtai and Fagin [1]).

However, it is easy to show that the complement of the above problem, *s–t unreachability*, is in $\mathsf{LCP}(O(1))$ both for undirected and directed graphs. We find a partition $S \cup T$ of V such that $s \in S$, $t \in T$, and there is no (directed) edge from S to T. Such a partition can be encoded with 1 bit per node, and it can be verified locally.

4.2 Connectivity

As a natural generalization of reachability, we can study the *s–t connectivity* of undirected graphs; throughout this text, we focus on the *vertex connectivity*. By extending the techniques of Korman et al. [18] we can show that graphs with s–t connectivity equal to k admit locally checkable proofs of size $O(\log k)$. Here we assume that k is given as input to all nodes (or, equivalently, that k is a global constant).

If and only if the vertex connectivity is exactly k, then by Menger's theorem [7, p. 62] we can find (i) a partition $S \cup C \cup T$ of V such that $s \in S$, $t \in T$, and $|C| = k$, and (ii) k vertex-disjoint s–t paths $p_1, p_2, \ldots, p_k$ such that $|C \cap p_i| = 1$. Without loss of generality, we can assume that each p_i is locally minimal in the sense that it cannot be made shorter without colliding with the other paths p_j, $j \neq i$.

The proof label $P(v)$ encodes whether $v \in S$, $v \in C$, or $v \in T$. Moreover, in the proof label $P(v)$ of a vertex $v \in p_i \setminus \{s,t\}$, we include the path index i (in binary) and also the distance of v from s modulo 3: this allows us to store the orientation on the path p_i. The local verifier can verify that:

1. Nodes s and t have exactly k neighbors labeled with path indices $1, 2, \ldots, k$.
2. Each $v \in p_i \setminus \{s,t\}$ has exactly one predecessor and one successor along p_i.
3. We have $s \in S$, $t \in T$, and there is no edge between S and T.
4. Each $v \in C$ is on a path p_i, its predecessor along p_i is in S and its successor is in T.

If the above checks go through, the structure encoded by the proof P contains exactly k disjoint s–t paths. It may contain some oriented cycles inside S or inside T as well, but this is sufficient to convince the verifier that the connectivity of s and t is at least k. Moreover, if a path crosses C, its color changes from S to T; its color cannot change back to S, and it cannot disappear without reaching t. Hence the above checks are also sufficient to convince the verifier that the size of the s–t separator C is at most k. In summary, s–t-connectivity has to be equal to k.

Finally, we note that the sole source for the $O(\log k)$ label size was the need to store the path indices. However, on planar graphs, only 3 path indices suffice to tell adjacent paths from one another; an adaptation of the above method gives a constant size proof in the case of planar graphs.

4.3 Bipartite matching

While *maximal* matchings can be verified without any proofs, verifying *maximum-cardinality* matchings requires some auxiliary information. In this section we study *bipartite* graphs; later in Section 5.4 we will show that the methods below break down on *non-bipartite* graphs.

Maximum matching. To construct a constant-size proof P for maximum matchings on a bipartite graph G we can use König's theorem [7, p. 35], which states that, on bipartite graphs, the maximum size of a matching equals the minimum size of a vertex cover. Indeed, take any minimum vertex cover $C \subseteq V(G)$, and set $P(v) = 1$ iff $v \in C$. For any matching $M \subseteq E(G)$, we then have, by König's theorem, that M is of maximum size if and only if $|C| = |M|$. To check this condition locally, we simply verify that M is a valid matching, the set C encoded in the proof forms a vertex cover, and each edge of M has exactly one endpoint in C. This proves that maximum matchings in bipartite graphs are in $\mathsf{LCP}(1)$, as the size of P is 1.

Maximum-weight matching. Generalizing the above example, we can use linear programming duality to prove that in an edge-weighted bipartite graph G, a subset of edges $M \subseteq E(G)$ is a maximum-weight matching. Associate a variable $x_e \geq 0$ with each edge $e \in E$, and a dual variable $y_v \geq 0$ with each node $v \in V$. Let $w_e \in \mathbb{N}$ be the weight of edge e, and let A be the vertex-edge incidence matrix of graph G. Recall (e. g., [24, §13.2]) that matrix A and its transpose $A^\top$ are totally unimodular, and hence there are integral vectors x and y that maximize $\sum_e w_e x_e$ subject to $Ax \leq 1$ (primal LP) and minimize $\sum_v y_v$ subject to $A^\top y \geq w$ (dual LP). Each maximum-weight matching M corresponds to an optimal integral solution x of the primal LP, and we can use an optimal dual solution y as a proof; for each node $v \in V$, the proof consists of a binary encoding of the value y_v. To verify the proof, it is sufficient to check that x and y satisfy the complementary slackness conditions. If the weights are integers from $0, 1, \ldots, W$, then we can find an optimal dual solution such that $y_v \in \{0, 1, \ldots, W\}$ for each node v. Hence the size of the proof is $O(\log W)$ bits.

5 Problems in LogLCP

In this section we give examples of graph properties and graph problems that admit locally checkable proofs of size $O(\log n)$ but for which there is no locally checkable proof of size $o(\log n)$. That is, these problems are in LogLCP but not in any lower level of the LCP hierarchy. We begin with positive results that directly build on prior work—a key ingredient is the observation that spanning trees in connected graphs are in LogLCP. After that, we give our new lower-bound results.

5.1 Positive results

A spanning tree is not locally checkable, but Korman et al. [18] show that any spanning tree T can be equipped with a proof of size $O(\log n)$ that, for each vertex v, consists of (i) the identity of a particular vertex a, *the root*, and (ii) the distance from v to a in T. Such a proof can be locally verified by checking that the root-distance at a is 0, and that for each vertex v (i) all neighbors of v agree on the identity of the root, and (ii) the root-distance decreases at exactly one neighbor of v in T and increases at other neighbors.

A spanning tree equipped with a locally checkable proof is a versatile tool. For example, consider the *leader election* problem: there has to be exactly one node that is labeled as the leader. In connected graphs, we can verify any solution to the leader election problem by constructing a spanning tree that is rooted at the leader.

Spanning trees can be also used to prove that the graph is acyclic: we simply show that each component is a tree. Hamiltonian cycles and Hamiltonian paths can be verified by using the same technique: a Hamiltonian path can be interpreted as a spanning tree.

With spanning trees, we can also gather global information about the input graph. For instance, every node can be convinced of the value of $n(G)$ on a connected graph G with the aid of a spanning tree with node counters along the paths towards the root: the leaves have counter value 1, and a non-leaf carries the sum of the counter values of its children plus 1. Hence graph properties such as having an odd number of nodes are also in LogLCP.

In LogLCP, we can also show that the chromatic number of a connected graph is larger than 2 (i. e., the graph is not bipartite). To construct a proof, first find an odd cycle in the graph—such a cycle exists if and only if the graph is non-bipartite. Then select one of the nodes of the cycle as the leader a. Construct a spanning tree rooted at a; this way the verifier can check that there exists exactly one leader. Then propagate a node counter along the cycle, starting and ending at a; this way the leader node can be convinced that it is indeed part of an odd cycle.

5.2 Negative results: overview

We will now prove the following theorem.

Theorem 5.1. *The following graph properties and graph problems do not admit locally checkable proofs of size $o(\log n)$: graphs with odd number of nodes, non-bipartite graphs, spanning trees, and leader election.*

Hence these are examples of problems whose containment in LogLCP is tight: their local proof complexity is exactly $\Theta(\log n)$.

Proof sketch. The negative results build on the same basic idea. We will consider graph properties on cycles. We will assume that there is a proof labeling scheme $(f,\mathcal{A})$ with $o(\log n)$-bit proofs. We will take several *yes*-instances—each of them is a short cycle—and inspect the encoding produced by f. Then we will show that some of the *yes*-instances are necessarily *compatible with each other* in the following sense: we can take several short cycles and *glue them together* to form a longer cycle; the unique identifiers and the proof labels are inherited from the short cycles, and each node of the long cycle will be locally indistinguishable from a node of a short cycle. Hence the verifier will accept the long cycle, as it has to accept all short cycles.

However, even though the short cycles are *yes*-instances, the long cycle will be a *no*-instance. For example, in the case of non-bipartiteness, each short cycle has an odd number of nodes, but the long cycle is composed of an even number of short cycles, and is therefore a *no*-instance. In the case of leader election, each short cycle has one leader node, while the long cycle will contain multiple leaders, and is therefore an invalid solution. Similar ideas can be applied to many other lower bounds.

Subtleties. The only combinatorial hurdle in carrying out the above strategy is in making sure that there are enough proof-labeled neighborhoods among the *yes*-instances accepted by $\mathcal{A}$ that they can be reassembled into the long *no*-instance. To this end we invoke an extremal result of Bondy and Simonovits [6] on a certain bipartite *compatibility* graph. (By contrast, recall that in the model of Korman et al. [18] such gluing operations were made easy by the limited vision of a local verifier—we did not have to worry about node identifiers.)

5.3 The proof

Let $\mathcal{F}$ be a family of graphs that contains (at least) all cycles. In each graph $G \in \mathcal{F}$, we may have a constant number of bits of auxiliary information per node (colors, labels, etc.). Let $\mathcal{P} \subseteq \mathcal{F}$ be a graph property. Assume that $(f, \mathcal{A})$ is a proof labeling scheme for property $\mathcal{P}$ that uses $o(\log n)$-bit proofs. Fix an integer constant $k \geq 2$. Let n be a sufficiently large positive integer. We will assume that n-cycles (with appropriate auxiliary information) are in $\mathcal{P}$.

Our plan is to show that we can always find k *yes*-instances, each of which is an n-cycle, and we can glue them together to form a kn-cycle that inherits the proof labels (and auxiliary information, if any) from the *yes*-instances. Verifier $\mathcal{A}$ will accept each n-cycle, and therefore it will also accept the kn-cycle. For an example, refer to Figures 1 and 2.

Partitioning the space of identifiers. First, split $n = n_A + n_B$, where $n_A = \lfloor n/2 \rfloor$ and $n_B = \lceil n/2 \rceil$. Fix any partition of the set of identifiers $\{1, \dots, n^2\}$ into $2n$ subsets $a_1, \dots, a_n, b_1, \dots, b_n$ so that each $a \in A := \{a_1, \dots, a_n\}$ is of size n_A, and each $b \in B := \{b_1, \dots, b_n\}$ is of size n_B. We denote by $a[i]$ (resp., $b[i]$) the i-th identifier in a (resp., b) in the natural order.

For example, if $n = 10$, we have $n_A = n_B = 5$. We can choose the partition $a_1 = \{1, 2, \dots, 5\}$, $a_2 = \{6, 7, \dots, 10\}$, ..., $b_{10} = \{96, 97, \dots, 100\}$. In this case $a_2[1] = 6$, $b_{10}[2] = 97$, etc.

Family of *yes*-instances. Next, we define a family of *yes*-instances $C(a,b)$ indexed by pairs $(a,b) \in A \times B$. Define $C(a,b)$ to be the n-cycle that contains the nodes a in increasing order followed by the nodes b in decreasing order, completing the cycle. That is, $a[1]$ and $b[1]$ are adjacent, as are $a[n_A]$ and $b[n_B]$; see Figure 1a. Note that $V(C(a,b))$ and $V(C(a',b'))$ are disjoint if $a \neq a'$ and $b \neq b'$.

Augment each $C(a,b)$ with auxiliary information such that $C(a,b)$ is in $\mathcal{P}$, if necessary; let $L_{ab}(v) \in \{0,1\}^\star$ be the bit string associated with node $v \in V(C(a,b))$. For example, if we are interested in the leader election problem, label exactly one node in each $C(a,b)$ as the leader: select a node $u \in V(C(a,b))$ and set $L_{ab}(u) = 1$ and $L_{ab}(v) = 0$ for all $v \neq u$. We can consider either the best-case or the worst-case choice of the leader, thus covering both weak and strong proof labeling schemes.

Recording proof bits. Now we make use of the assumption that property $\mathcal{P}$ admits $o(\log n)$-bit proofs. Apply f to $C(a,b)$ to construct a locally checkable proof P_{ab} of size $o(\log n)$. For each node $v \in V(C(a,b))$, let $P'_{ab}(v) = (L_{ab}(v), P_{ab}(v))$. Finally, define

$$c(a,b) = \big(P'_{ab}(a[2r+1]), P'_{ab}(a[2r]), \dots, P'_{ab}(a[1]), P'_{ab}(b[1]), P'_{ab}(b[2]), \dots, P'_{ab}(b[2r+1])\big).$$

That is, $c(a,b)$ consists of all auxiliary information and all proof bits that are available within distance $2r+1$ from the node $a[1]$ or $b[1]$ in $C(a,b)$; see Figure 1a. By assumption, we have $o(r \log n)$ bits of information in $c(a,b)$.

Finding compatible cycles. Now let $K_{n,n} = (A \cup B, E)$ be the complete bipartite graph with $E = \{\{a,b\} : a \in A, b \in B\}$. We define an edge coloring of $K_{n,n}$ as follows: the color of the edge $\{a,b\} \in E$ is $c(a,b)$. For a sufficiently large n, the number of bits of information in $c(a,b)$ is smaller than $\log(n)/3$,

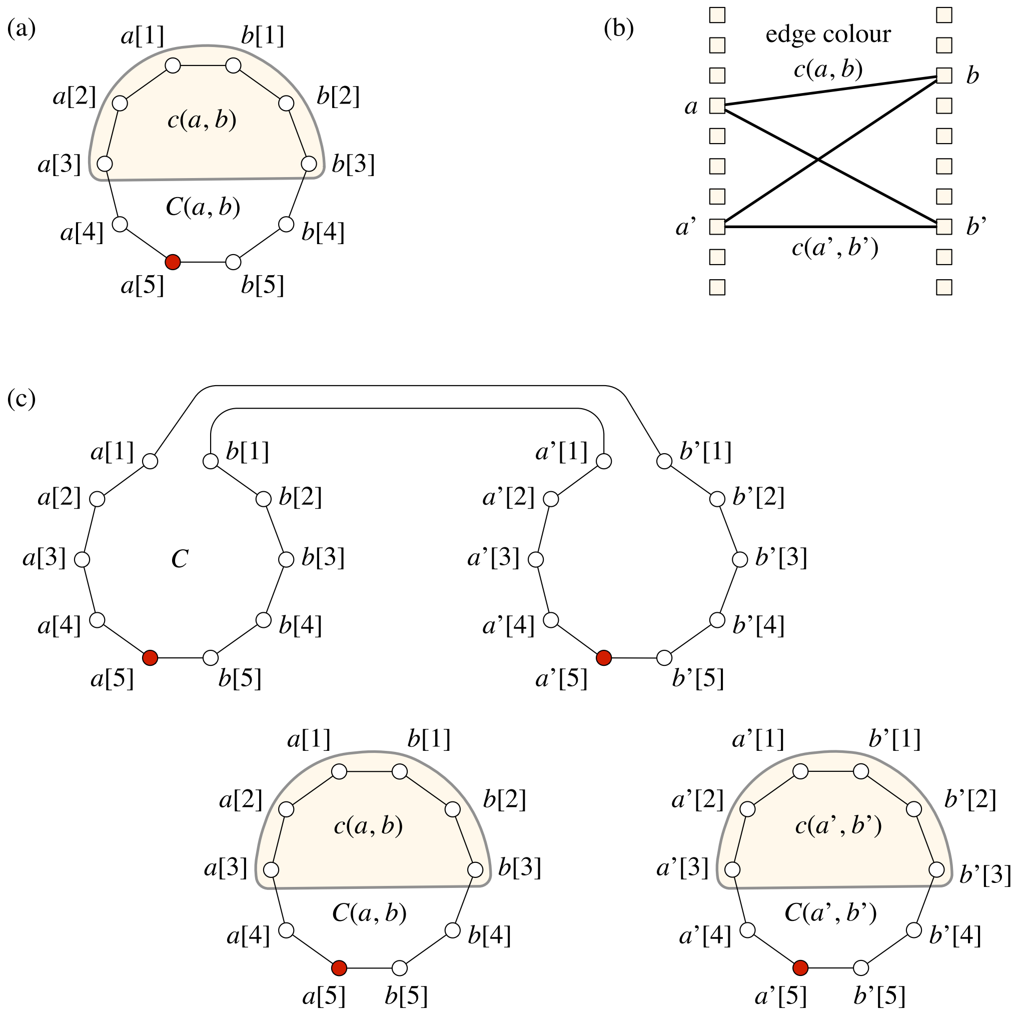


Figure 1: An illustration of the $\Omega(\log n)$ lower-bound construction; here $n = 10$, $r = 1$, and $k = 2$. (a) Construction of the cycle $C(a,b)$. We consider the leader election problem in this example; hence precisely one node is highlighted to indicate the leader node. (b) A monochromatic $2k$-cycle a, b, a', b' in $K_{n,n}$. (c) Constructing the kn-cycle C by gluing together two compatible n-cycles, $C(a,b)$ and $C(a',b')$.

and the number of distinct colors in $K_{n,n}$ is therefore smaller than $\sqrt[3]{n}$. Hence there is a subset of edges $H \subseteq E$ such that $|H| > |E|/\sqrt[3]{n} = n^{5/3}$ and all edges of H have the same color.

Now we can apply the result due to Bondy and Simonovits [6]: for any $k \geq 2$ and for a sufficiently large n, the subgraph $(A \cup B, H)$ necessarily contains a $2k$-cycle. Let the nodes of the cycle be $a_1, b_1, a_2, b_2, \ldots, a_k, b_k$ in this order, such that $a_i \in A$ and $b_i \in B$ for each i. As all edges of the cycle have the same color, we have

$$c(a_1,b_1) = c(a_2,b_2) = \cdots = c(a_k,b_k) = c(a_1,b_k) = c(a_2,b_1) = \cdots = c(a_k,b_{k-1})\,.$$

For convenience, define $b_0 = b_k$ and $a_{k+1} = a_1$.

Gluing. Next, we glue together the n-cycles $C(a_1,b_1), C(a_2,b_2), \ldots, C(a_k,b_k)$ to construct a kn-cycle C. That is, we take the node-disjoint graphs $C(a_i,b_i)$, remove the edges $\{a_i[1], b_i[1]\}$ for each i, and add $\{b_{i-1}[1], a_i[1]\}$ for each $i > 1$. For each node $v \in V(C)$, we inherit the auxiliary information $L(v)$ and the proof bits $P(v)$ from the cycles $C(a_i,b_i)$. The gluing process is illustrated in Figure 1c for the case $k = 2$. We have two compatible n-cycles, $C(a,b)$ and $C(a',b')$, and we connect $a[1]$ to $b'[1]$ and $a'[1]$ to $b[1]$.

Analysis. It remains to argue that the computation of $\mathcal{A}$ on C with the labels L and proof P is accepting. To see this, pick a vertex $v \in V(C)$. Then there is an i such that $v \in V(C(a_i,b_i))$. If v is far from $a_i[1]$ and $b_i[1]$, then the local neighborhood of v looks identical in C and $C(a_i,b_i)$; as $C(a_i,b_i)$ is a *yes*-instance, v accepts the input. If v is near $b_i[1]$, then the local neighborhood of v looks identical in C and $C(a_{i+1},b_i)$, which is another *yes*-instance. Similarly, if v is near $a_i[1]$, then its local neighborhood looks identical in C and $C(a_i,b_{i-1})$, which is also a *yes*-instance (see Figure 2 for an illustration). In all cases, v accepts the input. Thus the kn-cycle C is accepted by all nodes. If $C \notin \mathcal{P}$, we have a contradiction, and we can conclude that the graph property $\mathcal{P}$ does not admit locally checkable proofs of size $o(\log n)$.

5.4 Implications

Now we can give concrete examples of graph properties and graph problems $\mathcal{P}$ for which $C \notin \mathcal{P}$, provided that we choose the parameters k and n properly:

- Non-bipartite graphs: We can select an odd n and $k = 2$.
- Leader election: It is sufficient to choose $k = 2$. Then each $C(a,b)$ contains exactly one node labeled as a leader and C contains two nodes.
- Spanning trees: Again, we can choose $k = 2$. The spanning tree in each $C(a,b)$ contains all edges of $E(C(a,b))$ except one, i. e., it is a spanning path. The solution encoded in C consists of two disjoint paths, and is therefore not a spanning tree.

We can also apply the same construction to counting problems: to give a simple example, if we choose an odd n and an even k, then $n(C(a,b))$ is odd while $n(C)$ is even.

We can also prove lower bounds for optimization problems. Consider, for example, the problem of finding a maximum matching in a cycle. If n is odd, then each $C(a,b)$ has necessarily one unmatched

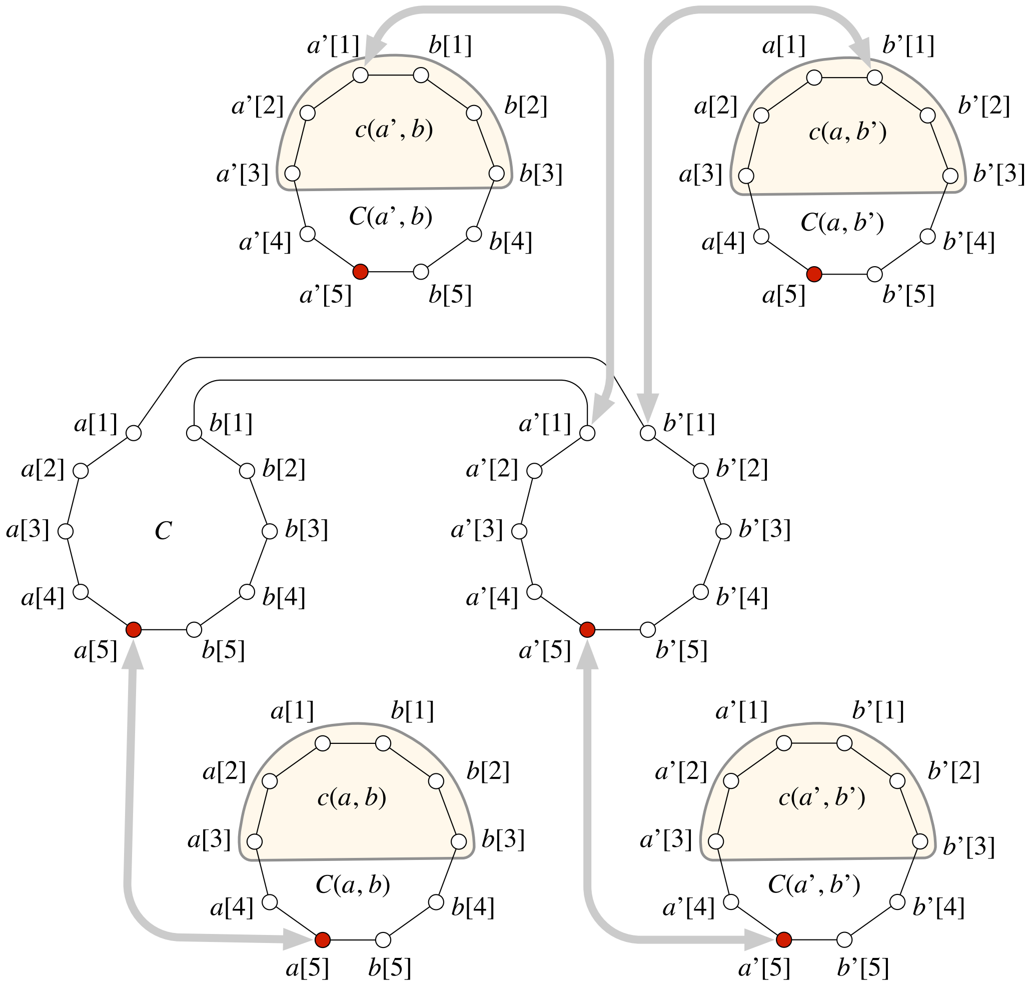


Figure 2: Cycle C was constructed by gluing together two compatible n-cycles, $C(a,b)$ and $C(a',b')$. The grey arrows indicate pairs of nodes that have identical local neighborhoods: some parts of C look locally identical to $C(a,b)$ or $C(a',b')$, while other parts look locally identical to $C(a,b')$ or $C(a',b)$.

node. The solution inherited from $C(a,b)$ to C has therefore k unmatched nodes, and cannot be optimal. Hence all of these problems require proofs of size $\Omega(\log n)$ and the lower bound is tight. This concludes the proof of Theorem 5.1.

6 Problems in $\mathsf{LCP}(\mathrm{poly}(n))$

In the previous sections, we have seen problems that admit locally checkable proofs of size $O(\log(n))$. Now we turn our attention to the problems that require much larger proofs. If the nodes can have arbitrary labels (e. g., weights), it is easy to come up with *artificial* problems that require arbitrarily large proofs. However, in this section we will focus on finding *natural* examples of graph properties that are related to *unlabeled* graphs: we do not have any additional information besides the structure of the graph G and the unique node identifiers.

In connected graphs, *any* property of unlabeled graphs admits locally checkable proofs of size $O(n^2)$. We can encode the structure of G and the unique node identifiers in $O(n^2)$ bits; the nodes can verify that their neighbors agree on the structure of G and that their local neighborhoods match those in G. Finally, the nodes can solve the problem by brute force.

In this section, we will show that there are natural graph properties that require $\Omega(n^2)$-bit proofs. Such problems are the most difficult problems from the LCP perspective—we can only save a constant factor in comparison with the brute-force solution.

6.1 Symmetric graphs

In this section, we fix $\mathcal{F}$ to be the family of connected graphs. We say that a graph G is *symmetric* if it has a non-trivial automorphism, that is, there is an automorphism $g\colon V \to V$ that is not the identity function; otherwise it is *asymmetric*.

Theorem 6.1. *Symmetricity of connected graphs requires locally checkable proofs of size* $\Omega(n^2)$.

Proof. To reach a contradiction, assume that there exists a proof labeling scheme $(f,\mathcal{A})$ with $o(n^2)$-bit proofs. To facilitate the proof, we will use *canonical forms* of graphs. We associate a canonical form $C(G)$ with any graph G. Graphs G and $C(G)$ are isomorphic; moreover, whenever G and H are isomorphic, their canonical forms $C(G)$ and $C(H)$ are equal. We assume that the node identifiers of a canonical form are $V(C(G)) = \{1,2,\dots,n(G)\}$. We also define a graph with *shifted identifiers* as follows: for an integer i, graph $C(G,i)$ has $V(C(G,i)) = \{i+1, i+2, \dots, i+n(G)\}$ as the set of node identifiers. Moreover, we assume that $g\colon v \mapsto i+v$ is an isomorphism from $C(G)$ to $C(G,i)$. In particular, $C(G,0) = C(G)$.

Now we are ready to give the lower-bound construction. Given two connected graphs G_1 and G_2 with $n(G_1) = n(G_2) = k$, we construct a graph $G = G_1 \circledcirc G_2$ with $V(G) = \{1,2,\dots,3k\}$ as follows: G consists of a copy of $C(G_1,k)$, a copy of $C(G_2,2k)$, and the path $(k+1,1,2,\dots,k,2k+1)$. That is, G consists of a path that joins graphs that are isomorphic to G_1 and G_2.

Assume that G_1 and G_2 are asymmetric. If G_1 and G_2 are isomorphic, then $G = G_1 \circledcirc G_2$ is symmetric: there is a non-trivial automorphism that maps $1 \mapsto k$ and $k+1 \mapsto 2k+1$ and so on. Conversely, if G admits a non-trivial automorphism φ, then, by the construction of G, we must have that φ swaps G_1 and G_2. This way, φ induces an isomorphism of G_1 and G_2.

Let $\mathcal{F}_k$ be a family containing a representative from each isomorphism class of asymmetric connected graphs with k nodes. For any $G_1 \in \mathcal{F}_k$, the local verifier $\mathcal{A}$ has to accept $G_1 \circledcirc G_1$, as it is symmetric. Since almost all graphs are connected [7, Cor. 11.3.3] and asymmetric [8], we have

$$|\mathcal{F}_k| = (1-o(1))2^{\binom{k}{2}}/k! \qquad \text{and} \qquad \log|\mathcal{F}_k| = \Theta(k^2)\,.$$

Now assume that $k \geq 2r+1$, where r is the local horizon of $\mathcal{A}$. Consider the proof labels of the nodes in $U = \{1,2,\dots,2r+1\}$. There are only $o(rn^2)$ proof bits in U. As $r = O(1)$ and $n = 3k$, we have only $o(k^2)$ proof bits in U; for sufficiently large k this is less than $\log|\mathcal{F}_k|$. Hence we must have two different graphs $G_1, G_2 \in \mathcal{F}_k$ such that the labeling scheme assigns the same proof bits to the nodes $1,2,\dots,2r+1$ in both $G_1 \circledcirc G_1$ and $G_2 \circledcirc G_2$.

Now we can construct the asymmetric graph $G = G_1 \circledcirc G_2$. For the nodes $k+1, k+2, \dots, 2k$ we inherit the proof labels from $f(G_1 \circledcirc G_1)$, and for the nodes $2r+2, 2r+3, \dots, k, 2k+1, 2k+2, \dots, 3k$ we inherit the proof labels from $f(G_2 \circledcirc G_2)$. For the nodes $1,2,\dots,2r+1$ we use the common labeling of $f(G_1 \circledcirc G_1)$ and $f(G_2 \circledcirc G_2)$. Now the radius-r neighborhood of any node in G looks identical to the neighborhood of a node in $G_1 \circledcirc G_1$ or $G_2 \circledcirc G_2$. Hence all nodes will accept the input even though G is not symmetric, a contradiction. □

Remark 6.2. The basic idea of the above proof is similar to Korman et al. [18, Corollary 2.4]. However, Korman et al.'s construction gives a problem related to graphs that are labeled with unique identifiers; it is not a graph property in the usual sense (closed under re-assigning the identifiers).

6.2 Fixed-point-free symmetry on trees

In this section, we fix $\mathcal{F}$ to be the family of connected trees. Here, any graph property $\mathcal{P} \subseteq \mathcal{F}$ admits a locally checkable proof of size $O(n)$: for each node v of the tree $G \in \mathcal{P}$, we encode the structure of G and an index that identifies which node of G is v; the structure of a tree can be encoded in $\Theta(n)$ bits, and the index requires $\Theta(\log n)$ bits.

Now we will show that there are properties of unlabeled trees that require $\Theta(n)$-bit proofs. We will use the following (artificial) problem as an example. We say that a graph G has a *fixed-point-free symmetry* if there is an automorphism that fixes no nodes, i. e., there is an automorphism $g\colon V(G) \to V(G)$ such that $g(v) \neq v$ for all $v \in V(G)$.

Theorem 6.3. *Trees with a fixed-point-free symmetry require locally checkable proofs of size* $\Theta(n)$.

The proof is analogous to the case of symmetric graphs. The only difference is that we let $\mathcal{F}_k$ consist of rooted trees with k nodes; if $G_1, G_2 \in \mathcal{F}_k$ and k is even, then $G_1 \circledcirc G_2$ has a fixed-point-free symmetry if and only if $G_1 = G_2$. We have $\log|\mathcal{F}_k| = \Theta(k)$ [28, Seq. A000081]; hence a proof of size $o(n)$ bits leads to a contradiction.

6.3 Non-3-colorability

Now we turn our attention to the classical problem of graph coloring. In Section 5 we have already seen that in the case of 2-colorability, the complement of the problem is strictly more difficult: to show that a graph can be colored with 2 colors a $\Theta(1)$-bit proof is sufficient, but to show that a graph cannot be colored with 2 colors we need $\Theta(\log n)$-bit proofs.

In the case of 3-colorings, the difference between the problem and its complement is even more dramatic. Again, constant-size proofs are enough to show that a graph can be colored with 3 colors, as we can give a 3-coloring as a proof. However, to prove that a graph cannot be colored with 3 colors, we need very large proofs, with polynomially many bits per node.

Theorem 6.4. *Non-3-colorability requires locally checkable proofs of size $\Omega(n^2/\log n)$.*

Let $\mathcal{F}$ be the family of connected graphs, and let $\mathcal{P} \subseteq \mathcal{F}$ consist of graphs that have chromatic number larger than 3. We will show that property $\mathcal{P}$ does not admit locally checkable proofs of size $o(n^2/\log n)$. Recall that any property of unlabeled graphs admits proofs of size $O(n^2)$; hence the result is almost tight, and shows that non-3-colorability does not have a proof labeling scheme that is substantially better than the brute-force approach.

Proof overview. The template for our lower-bound proof will be the *disjointness problem* in two-party communication complexity: two players, Alice and Bob, are given sets A and B, respectively, and they need to communicate to find out whether $A \cap B = \varnothing$. It is well-known that this problem has high communication complexity even in the nondeterministic setting where the players seek to prove that $A \cap B = \varnothing$; see Example 1.23 and Lemma 2.4 in Kushilevitz and Nisan [20]. Recall also that proving $A \cap B \neq \varnothing$ is easy: simply guess an element in $A \cap B$.

Our lower bound graph $G = G_{A,B}$ will have two subgraphs G_A and G_B that are owned by Alice and Bob, respectively. These subgraphs are connected by some $\Theta(\log n)$ *wires*, that, intuitively, allow nondeterministic communication between the players (via the proof labels). The subgraphs G_A and G_B are constructed in such a way that if we are given any partial 3-coloring c of G defined only on the wires of G, then c can be extended to 3-coloring of G_A (resp., G_B) if and only if c encodes, in a certain way, an element in A (resp., B). This means that $G_{A,B}$ will be 3-colorable iff $A \cap B \neq \varnothing$—or in other words $G_{A,B}$ will be non-3-colorable iff $A \cap B = \varnothing$. A communication complexity argument then implies that the wires must carry large proof labels.

Properties of G_A. Let k be a positive integer. Define $I = \{0, 1, \ldots, 2^k - 1\}$. Given a set $A \subseteq I \times I$, we construct a graph G_A, with the following properties:

(i) The total number of nodes in G_A is $\Theta(2^k)$.

(ii) The set of nodes $V(G_A)$ contains the following nodes: T, F, N, $x_0, x_1, \ldots, x_{k-1}$, and $y_0, y_1, \ldots, y_{k-1}$.

Moreover, proper 3-colorings of G_A have the following properties:

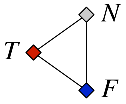
(iii) The nodes T, F, and N have three different colors. The nodes with the same color as T are said to be *true*, those with the same color as F are *false*, and others are *neutral*.

(iv) Each of x_i and y_i has to be true or false. Hence we can interpret the coloring of the nodes x_i as a binary encoding of an integer $x \in I$; similarly the coloring of the nodes y_i is a binary encoding of an integer $y \in I$.

(v) In any 3-coloring, we must have $(x,y) \in A$. Conversely, we can find a proper 3-coloring that encodes any $(x,y) \in A$.

An explicit construction of G_A follows. In the subsequent analysis the details of the construction are not used outside of the above properties (i)–(v).

Gadgets. We begin with the basic building blocks of G_A. In graph G_A, nodes T, F, and N form a triangle, ensuring that in any 3-coloring these nodes have different colors.



Recall that the nodes with the same color as T are said to be *true*, those with the same color as F are *false*, and others are *neutral*. Nodes T, F, and N are extensively used in the construction of the following gadgets.

A *Boolean gadget* has two *terminals*; see Figure 3. In a proper 3-coloring, exactly one terminal is true and the other terminal is false; conversely, any such assignment is a proper 3-coloring.

A *color changer* has one *input node* and one *output node*; see Figure 4. In any proper 3-coloring, a true input implies a true output while a neutral input implies a false output (and a false input is not possible); conversely, any such assignment can be realized as a proper 3-coloring.

A *disjunction gadget* has a number of *output nodes*; see Figure 5. In any proper 3-coloring, at least one output node is true and all other output nodes are neutral; conversely, any such assignment can be realized as a proper 3-coloring. The construction uses Boolean gadgets.

Finally, we construct a *binary decoder* as illustrated in Figure 6. There are k input nodes, labeled with $x_0, x_1, \ldots, x_{k-1}$, and 2^k output nodes, labeled with $0, 1, \ldots, 2^k - 1$, i. e., with a number in I. In any 3-coloring, each input node is either true or false, while each output node is either true or neutral. The color assignment of the input nodes can be interpreted as a binary encoding of a number $i \in I$, and the decoder guarantees that only the output node with label i is true. Conversely, any such assignment can be realized as a proper 3-coloring. In the construction of the binary decoder, a disjunction gadget ensures that at least one output node is true, while the edges between inputs and outputs ensure that if an output node is true, its index matches the truth assignment of the input nodes. Now we have all basic ingredients for constructing graph G_A.

Construction of G_A. The high-level structure of graph G_A is shown in Figure 7. There are $2k+3$ *terminals*: T, F, N, $x_0, x_1, \ldots, x_{k-1}$, and $y_0, y_1, \ldots, y_{k-1}$. There are also *non-terminal* nodes a_i, b_i, and c_i for $i \in I$.

The construction contains two binary decoders: one mapping inputs x_j to outputs a_i, and the other mapping inputs y_i to outputs c_j. Moreover, there are 2^k color changers, each of them mapping input a_i to output b_i. Finally, we have an edge $\{b_i, c_j\}$ if and only if $(i, j) \notin A$.

Now in any proper 3-coloring, nodes x_j and y_j have to be either true or false; we can interpret the coloring of nodes x_j as the binary encoding of a number $x \in I$, and the coloring of nodes y_j as the binary encoding of a number $y \in I$. By construction, we have

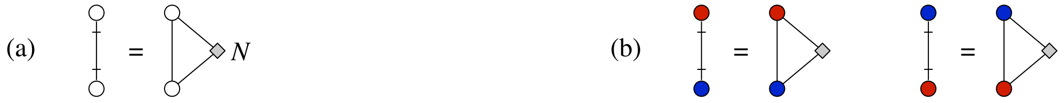


Figure 3: (a) Boolean gadget: one terminal is true, one terminal is false. (b) Proper 3-colorings.

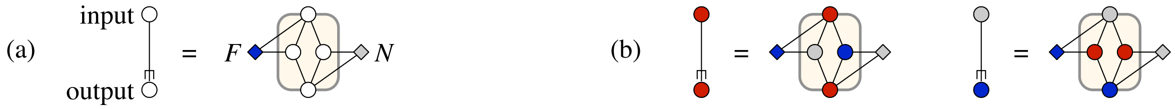


Figure 4: (a) Color changer: true ↦ true but neutral ↦ false. (b) Proper 3-colorings.

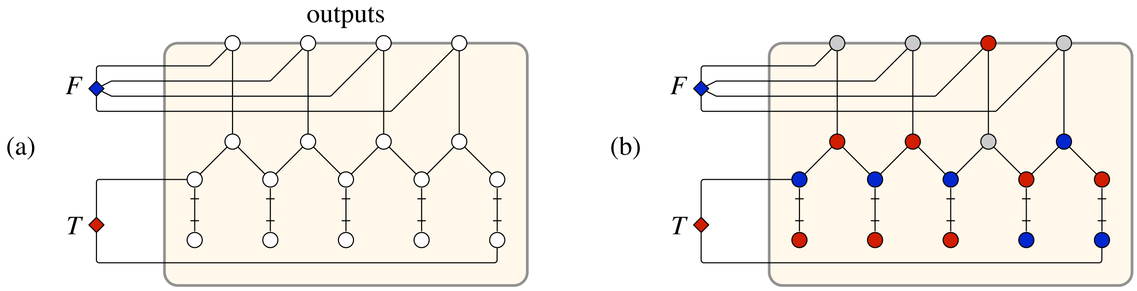


Figure 5: (a) Disjunction gadget with four output nodes. (b) An example of a proper 3-coloring.

(i) a_i is true iff $i = x$; otherwise a_i is neutral,
(ii) b_i is true iff $i = x$; otherwise b_i is false,
(iii) c_i is true iff $i = y$; otherwise c_i is neutral.

In particular, both b_x and c_y are true, and therefore we must have $(x,y) \in A$. Conversely, for any pair of integers $(x,y) \in A$, we can construct a proper 3-coloring of G_A such the coloring of x_j forms the binary encoding of x and the coloring of y_j forms the binary encoding of y.

Construction of $G = G_{A,B}$. We have now completed the construction of G_A for any given set $A \subseteq I \times I$. In what follows, we denote by G'_A an isomorphic copy of G_A; we use the primed symbols T', F', N', x'_i, and y'_i to refer to the terminal nodes of G'_A.

We will need one more gadget: a *wire* that propagates colors; see Figure 8. A wire w consists of $9r$ nodes, labeled $w(i,j)$ for $i = 1,2,\dots,3r$ and $j = 1,2,3$. For each i, the nodes $w(i,1)$, $w(i,2)$, and $w(i,3)$ form a triangle. For each $i < 3r$ and $j \neq j'$, the nodes $w(i,j)$ and $w(i+1,j')$ are connected with an edge. It follows that in any 3-coloring of a wire, the nodes $w(i,1)$, $w(i,2)$ and $w(i,3)$ must have different colors,

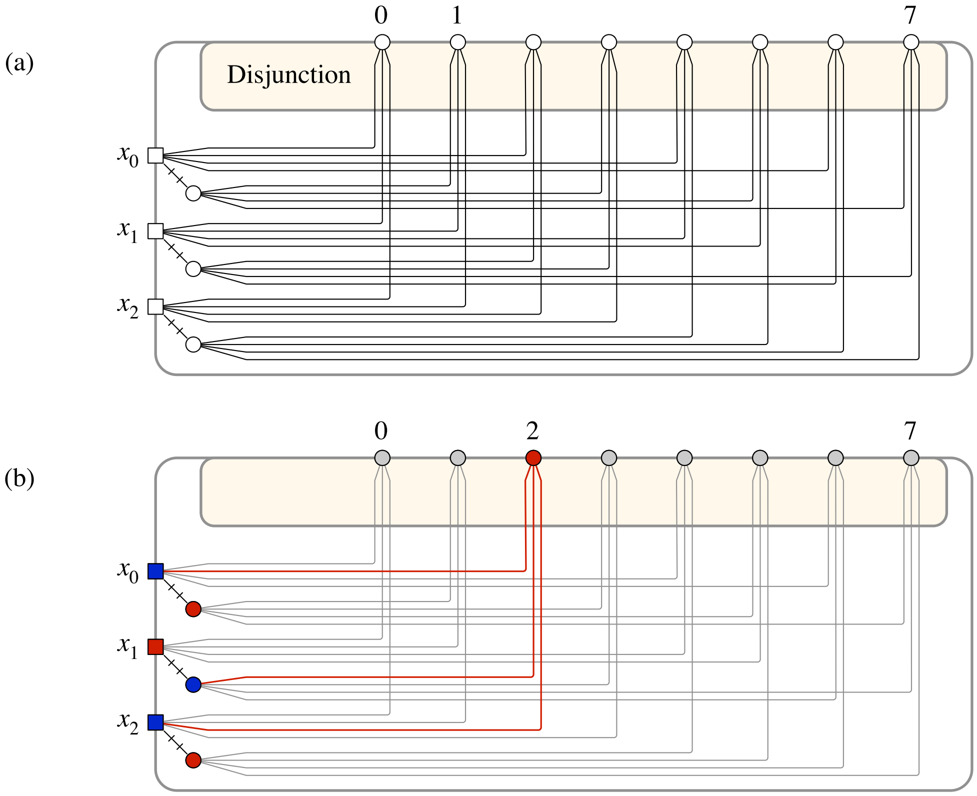


Figure 6: (a) A binary decoder for $k = 3$. (b) An example: $x = 2$.

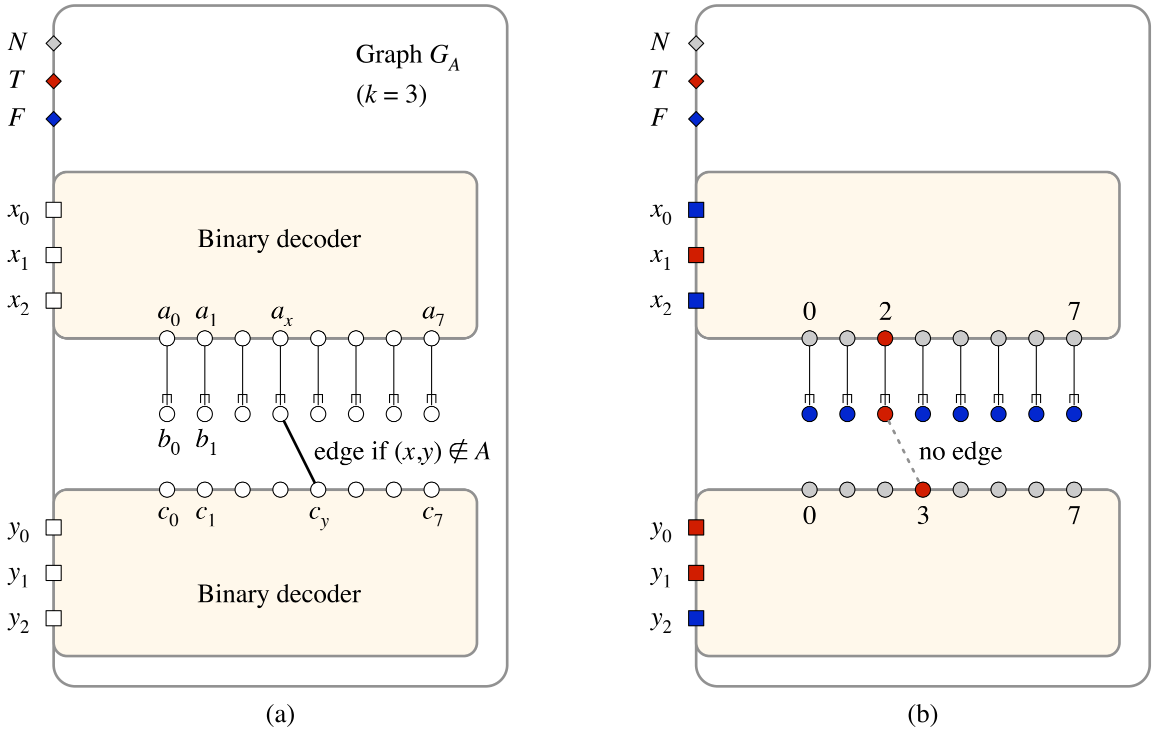


Figure 7: (a) The high-level structure of graph G_A. (b) An example: $(x,y) = (2,3) \in A$.

and $w(i, j)$ must have the same color as $w(i+1, j)$.

Given two sets $A, B \subseteq I \times I$, we construct a graph $G = G_{A,B}$ that consists of subgraphs G_A and G'_B that are connected to each other by $2k+1$ wires; see Figure 9. The wires ensure that G_A and G'_B agree on the coloring of the terminals.

In more detail, we label the $2k+1$ wires with w_T, $w^x_1, w^x_2, \dots, w^x_k$, and $w^y_1, w^y_2, \dots, w^y_k$. The endpoints of the wires are identified with the nodes of the subgraphs G_A and G'_B as follows:

$$\begin{aligned}
w(1,1) &= N && \text{and} & w(3r,1) &= N' && \text{for each wire } w,\\
w_T(1,2) &= T && \text{and} & w_T(3r,2) &= T',\\
w^x_i(1,2) &= x_i && \text{and} & w^x_i(3r,2) &= x'_i && \text{for each } i = 0,1,\dots,k-1,\\
w^y_i(1,2) &= y_i && \text{and} & w^y_i(3r,2) &= y'_i && \text{for each } i = 0,1,\dots,k-1.
\end{aligned}$$

Analysis. We make the following observations of G:

(i) The total number of nodes in G is $n = \Theta(2^k)$.

(ii) Let $W \subseteq V(G)$ consist of the nodes that are not in G_A or G'_B; these are internal nodes of the wires. The number of nodes in W is $\Theta(rk) = \Theta(r \log n)$.

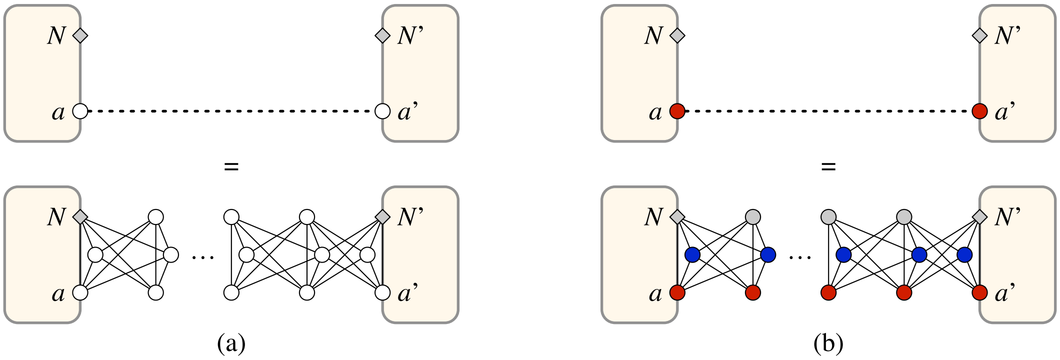


Figure 8: (a) A wire between terminals a and a'. (b) An example of a proper 3-coloring.

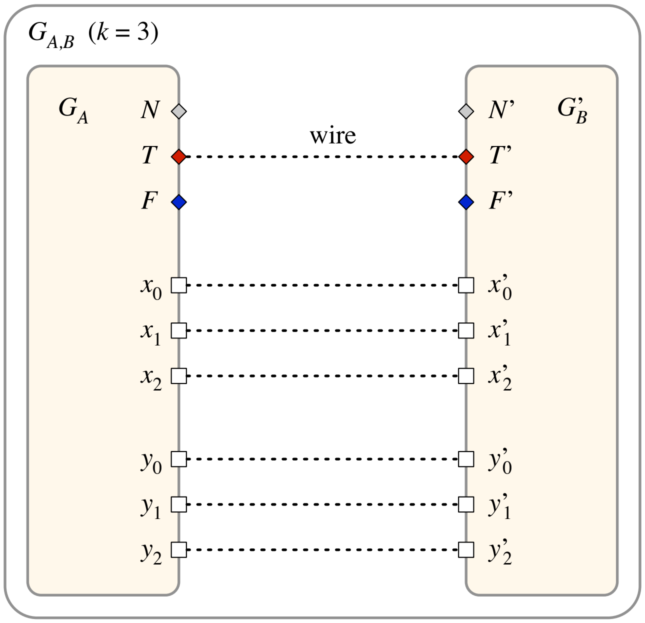


Figure 9: Graph $G_{A,B}$ consists of graphs G_A and G'_B that are connected by $2k+1$ wires.

(iii) The shortest path from a node of G_A to a node of G'_B has length at least $3r-1$. In particular, for sufficiently large r, the local neighborhood of any node is a subset of $W \cup V(G_A)$ or a subset of $W \cup V(G'_B)$.

Moreover, 3-colorings of G have the following properties:

(iv) Nodes N and N' have the same color, nodes T and T' have the same color, and nodes F and F' have the same color. Hence the concepts of true, false, and neutral nodes are well-defined in G.

(v) Nodes x_i and x'_i have the same color for each i, and nodes y_i and y'_i have the same color for each i. In particular, both G_A and G'_B agree on the encoding of the same pair (x,y), and we must have $(x,y) \in A \cap B$.

It follows that $G_{A,B}$ has a 3-coloring if and only if $A \cap B \neq \emptyset$.

Let $A \subseteq I \times I$ and let $\bar{A}$ be its complement. Now $A \cap \bar{A} = \emptyset$ and $G_{A,\bar{A}}$ does not have a 3-coloring; hence it is in $\mathcal{P}$. If we had locally checkable proofs of size $o(n^2/\log n)$, the total number of proof bits in W would be $o(rn^2)$; on the other hand, there are $\Theta(n^2)$ elements in $I \times I$. Hence for sufficiently large n there are two different sets $A, B \subseteq I \times I$ such that we have the same proof bits in W for both $G_{A,\bar{A}}$ and $G_{B,\bar{B}}$.

Now we are ready to apply a fooling set argument. As $A \neq B$, we have $A \cap \bar{B} \neq \emptyset$ or $\bar{A} \cap B \neq \emptyset$ (or both). Without loss of generality, assume that $A \cap \bar{B} \neq \emptyset$. Hence $G_{A,\bar{B}}$ admits a 3-coloring, and it is therefore not in $\mathcal{P}$. But we can construct a proof as follows: the proof bits of G_A are inherited from the proof of $G_{A,\bar{A}}$, the proof bits of $G'_{\bar{B}}$ are inherited from the proof of $G_{B,\bar{B}}$, and the proof bits of wires are the same as in $G_{A,\bar{A}}$ and $G_{B,\bar{B}}$. Hence each node of $G_{A,\bar{B}}$ has a local neighborhood that looks identical to a node of $G_{A,\bar{A}}$ or $G_{B,\bar{B}}$. As $G_{A,\bar{A}}$ and $G_{B,\bar{B}}$ are *yes*-instances, all nodes will accept $G_{A,\bar{B}}$, a contradiction.

This completes the proof of Theorem 6.4 that non-3-colorability requires proofs of size $\Omega(n^2/\log n)$. For comparison, proofs of size $O(n^2)$ are trivially sufficient.

7 Structural properties

In the previous sections we have discussed *concrete examples* of graph properties and their local proof complexities. By contrast, the focus of this final section is on the *structural properties* of the LCP hierarchy: We study the sensitivity of our LCP model to the definitional choices made in Section 2. In a similar vein, we will seek alternative characterizations of the class LogLCP. Finally, we give some observations relating our LCP classes to other complexity classes.

7.1 Port numbering algorithms

Throughout this paper, we have used the assumption that the local verifier can access the unique identifiers of the nodes (in accordance with Peleg's [25] LOCAL model). Furthermore, we have assumed that the proof can depend on the identifier assignment (recall that this convention is not followed by Fraigniaud et al. [11]). In this section we ask what one can locally prove in a setting where node identifiers are not available.

The standard model for computing on networks without unique identifiers is the *port numbering model* (PN) of Angluin [3]. In the PN model the nodes are considered *anonymous*; they do not have unique identifiers. There is only a port numbering available in the network, i. e., a node of degree d can refer to its neighbors by integers $1,2,\dots,d$. We refer to Angluin [3] and Suomela [29] for a formal definition of the PN model.

Recall that if $\varphi\colon V(H)\to V(G)$ is a *covering map* (i. e., an onto graph homomorphism that for each $v\in V(H)$ maps the neighbors of v in H bijectively onto the neighbors of $\varphi(v)$ in G), then H is said to be a *lift* of G [2, 3]. For example, any kn-cycle is a lift of an n-cycle. Furthermore, we say that a property $\mathcal{P}\subseteq\mathcal{F}$ is *closed under lifts* if $G\in\mathcal{P}$ implies $H\in\mathcal{P}$ for every lift $H\in\mathcal{F}$ of G.

A fundamental limitation of a PN algorithm $\mathcal{A}$ is that it is fooled by a lift $\varphi\colon V(H)\to V(G)$ in the following sense [3, 29]: for every local input $P\colon V(G)\to\{0,1\}^\star$ we have that

$$\mathcal{A}(H,P\circ\varphi,v)=\mathcal{A}(G,P,\varphi(v))\quad\text{for all } v\in V(H)$$

for suitable port numberings of H and G. Thus—regardless of the size of P—this directly implies that any property that can be locally checked using a PN verifier must be closed under lifts: if each node outputs 1 on (G,P), then each node outputs 1 on $(H,P\circ\varphi)$. In fact, the converse holds on connected graphs:

Theorem 7.1. *Suppose that $\mathcal{P}\subseteq\mathcal{F}$ is a property of connected graphs that is closed under lifts. Then $\mathcal{P}$ admits locally checkable proofs of size $O(n^2)$ that can be verified using a PN algorithm.*

Proof. On input a connected graph $H\in\mathcal{F}$, the PN verifier for $\mathcal{P}$ expects the proof label $P(v)$ of a node $v\in V(H)$ to encode a pair $(G,\varphi(v))$, where G is a connected graph (without port numbers) having identifiers $V(G)=\{1,\dots,n(G)\}$, and $\varphi(v)\in V(G)$ is a distinguished node. Naturally, an honest $O(n^2)$-bit proof will have $P(v)=(H,v)$ for all v.

We let the PN verifier perform the following checks at a node $v\in V(H)$. First, we check that all the neighbors of v hold the same encoding of G in their proof labels. This ensures that all nodes agree on the encoding of G since the input graph is connected. Then v checks that $\deg_H(v)=\deg_G(\varphi(v))$ and that

$$\{\varphi(u): u \text{ is a neighbor of } v \text{ in } H\}=\{u: u \text{ is a neighbor of } \varphi(v) \text{ in } G\}\,.$$

These checks guarantee that the map $\varphi\colon V(H)\to V(G)$ is a covering map. Finally, the PN verifier accepts iff $G\in\mathcal{P}$. The soundness of the checking procedure now follows from the assumption that $\mathcal{P}$ is closed under lifts. □

7.2 Alternative characterizations of LogLCP

A LogLCP proof can address individual vertices by their $O(\log n)$ identifiers. Consequently—and in contrast to the discussion above—LogLCP contains properties that are not closed under lifts (e. g., non-bipartiteness). Next, we show how a little additional input structure can enable a PN verifier to check properties in LogLCP. This shows that the class LogLCP is robust in the sense that we can change our underlying model of distributed computation and yet arrive at exactly the same class of graph properties. We note that the model (3) below is related to the class $\mathsf{NLD}^{\#n}$ considered by Fraigniaud et al. [11].

Theorem 7.2. *On connected graphs G, the class* LogLCP *coincides with properties that can be locally verified with $O(\log n)$ bits in each of the following models:*

1. *PN with* unique identifiers*, i. e., the model used to define* LogLCP *in Section 2.*

2. *PN with an* elected leader *in G.*
3. *PN with each node being given $n(G)$ as input.*

Proof. We show that the above models are equivalent by describing how to pass from one model to another with only an additive $O(\log n)$ overhead in proof size.

($1 \Rightarrow 2$ & 3): Using spanning tree methods [18] we already saw in Section 5 that (on a connected graph) proving the presence of a unique leader and verifying the number of nodes is in LogLCP.

(2 & $3 \Rightarrow 1$): Consider the following attempt at encoding a spanning tree T of G in the PN model: the $O(\log n)$ proof encodes T together with the root-distances. The problem with this encoding is that since we cannot talk about the identifier of the anonymous root of T, we cannot verify that such an encoding describes a *spanning tree* rather than a *spanning forest*. However, we can overcome this tree/forest problem by employing either one of the assumptions (2) or (3): Under (2) we can require that the unique elected leader in G is the root of the spanning tree. Under (3) we can equip T with node counters along the paths towards the root and thus verify that all $n(G)$ nodes are contained in the same tree. Hence, in either of the models (2) or (3) we can build and verify a spanning tree T of G.

Using T we can generate unique identifiers for the nodes of G as follows (cf. *ancestor labeling schemes* [12]): Do a depth-first traversal on T starting at the root and record in the proof label of a node v its discovery time $x(v)$ and finishing time $y(v)$; the identifier of a node v is then an encoding of the pair $(x(v), y(v))$. We note that the pairs $(x(v), y(v))$ can be locally checked to be consistent with a depth-first traversal on T—it follows that the node identifiers are globally unique. □

Remark 7.3. Fraigniaud et al. [11] describe how a *randomized* local verifier can check—in a certain probabilistic sense—whether a graph contains *at most one leader*. This technique provides yet another means to check for the uniqueness of a rooted spanning tree.

7.3 Complementing in $\mathsf{LCP}(0)$

We will now show that $\mathsf{coLCP}(0) \subseteq \mathsf{LogLCP}$ on connected graphs. The key idea is that we can employ spanning trees to reverse the decision made by an $\mathsf{LCP}(0)$ verifier.

Let $\mathcal{F}$ be the family of connected graphs, let $\mathcal{P} \subseteq \mathcal{F}$ be a graph property in $\mathsf{LCP}(0)$, and let $\bar{\mathcal{P}} = \mathcal{F} \setminus \mathcal{P}$ be the complement of $\mathcal{P}$. We will show that $\bar{\mathcal{P}}$ admits a locally checkable proof of size $O(\log n)$ in graph family $\mathcal{F}$.

Let $\mathcal{A}$ be a local verifier that checks $\mathcal{P}$; we will use $\mathcal{A}$ to design a proof labeling scheme for $\bar{\mathcal{P}}$. For any *yes*-instance $G \in \bar{\mathcal{P}}$, we construct a locally checkable proof P of size $O(\log n)$ as follows: Select a root node a with $\mathcal{A}(G, \varepsilon, a) = 0$, i. e., a is a node that rejects the input G. Then choose an arbitrary spanning tree T rooted at a. Let P consist of an encoding of (T, a) and a proof of its correctness.

The proof is checked by a local verifier $\bar{\mathcal{A}}$ as follows. First, $\bar{\mathcal{A}}$ verifies that T is valid spanning tree rooted at a; in particular, there is a finite path from any $v \in V(G)$ to a. Second, at the root node a, verifier $\bar{\mathcal{A}}$ simulates $\mathcal{A}$ and checks that $\mathcal{A}(G, \varepsilon, a) = 0$.

By construction, a valid proof is accepted by $\bar{\mathcal{A}}$. Let us now show that we cannot fool $\bar{\mathcal{A}}$. Consider a *no*-instance $G \notin \bar{\mathcal{P}}$ and any proof P. We have one of the following cases:

1. P is not an encoding of a rooted spanning tree. At least one node detects this and the input is rejected by $\bar{\mathcal{A}}$.

2. P is an encoding of a spanning tree T rooted at a. However, $G \in \mathcal{P}$ and therefore $\mathcal{A}(G, \varepsilon, a) = 1$. Hence $\bar{\mathcal{A}}$ outputs 0 at the root node a.

We conclude that for each $G \in \bar{\mathcal{P}}$ there is a proof P that is accepted by $\bar{\mathcal{A}}$, and for each $G \notin \bar{\mathcal{P}}$ any proof P is rejected by $\bar{\mathcal{A}}$. Hence $\bar{\mathcal{P}}$ is in LogLCP.

7.4 Containment in NP and NP/poly

Comparing classes such as LogLCP and NP is not straightforward. To define the LCP hierarchy, we have used the LOCAL model, which allows unlimited local computation. Hence if we have unbounded node degrees in G (or unbounded amount of additional information per node in the form of colors or weights), we can easily come up with artificial problems that are in LCP(0) but not in NP.

However, the situation becomes much more interesting if we study bounded-degree graphs; moreover, we will consider properties of unlabeled graphs, i. e., there is no additional information besides the node identifiers and the topology of the graph.

In this restricted case, we can still show that there are problems in LogLCP that are not contained in NP. Once again, we can resort to spanning tree methods: without loss of generality, we can assume that a LogLCP verifier has access to $n = n(G)$ in any connected graph G. Hence the verifier can solve arbitrarily hard problems concerning the integer n, including those that are not in NP (which exist by the hierarchy theorems [23, §7.2]).

However, if $\mathcal{P} \in$ LogLCP is a graph property related to unlabeled bounded-degree graphs, we *can* show that $\mathcal{P}$ is in NP/poly, i. e., NP with a polynomial-size non-uniform advice. In a bounded-degree graph, the number of nodes inside the local horizon is bounded by a constant, and hence a LogLCP verifier $\mathcal{A}$ uses only $O(\log n)$ bits of input in total. Thus verifier $\mathcal{A}$ can be encoded as a lookup table of size $2^{O(\log n)}$, which is polynomial in n. We can provide the entire lookup table as the advice string S to an NP/poly machine M. Then M merely guesses the $O(n \log n)$-bit proof $P \colon V(G) \to \{0,1\}^\star$, and uses the advice string S to verify the guess.

7.5 Connections to descriptive complexity

A central result in descriptive complexity theory and one that began the field is Fagin's [9], [14, Ch. 7] characterization of the class NP as graph properties expressible by *existential second-order formulas* (Σ_1^1)—this is the extension of first-order logic that allows existential quantification over relation symbols $X_1, X_2, \ldots$ (of any arity). Some NP-complete graph properties (e. g., 3-colorability) are even expressible by *monadic* Σ_1^1 formulas that only quantify over *unary* relation symbols [1, 26]. In this section, we make observations of a connection between the LogLCP class and the class of graph properties that are expressible by monadic Σ_1^1 formulas.

In the study of *first-order* expressibility, locality is a thematic subject; this is illustrated by Hanf's theorem and the work of Gaifman [14, Ch. 6]. Building on this work, Schwentick and Barthelmann [27] have shown that on connected graphs, every monadic Σ_1^1 formula is equivalent to a formula of the form

$$\vartheta = \exists X_1 \exists X_2 \ldots \exists X_k \exists x \forall y : \varphi(X_1, \ldots, X_k, x, y),$$

where φ is first-order (using relation symbols $X_1, \ldots, X_k$) and *local around* y. Here, a formula φ is local around y if there is a constant r so that for all graphs G and all interpretations of $X_1, X_2, \ldots, X_k, x, y$ it can be determined whether G satisfies $\phi(X_1, X_2, \dots, X_k, x, y)$ on the basis of the r-radius neighborhood of y in G. (More formally, the quantifications in φ are always of the form $\exists z : (\mathrm{dist}(z, y) \leq r \land \psi)$ or $\forall z : (\mathrm{dist}(z, y) \leq r \to \psi)$, where "$\mathrm{dist}(z, y) \leq r$" simply stands for "the distance between z and y is at most r".)

Let us consider the family $\mathcal{F}$ of connected graphs. If and only if a graph $G \in \mathcal{F}$ has property ϑ, there are unary relations $A_1, A_2, \dots, A_k$ and a node $a \in V$ such that G satisfies $\forall y : \varphi(A_1, A_2, \dots, A_k, a, y)$. For each node v and each relation A_i, encoding $A_i(v)$ takes 1 bit. To prove the existence of the node a, we can use a spanning tree rooted at a; a locally checkable spanning tree requires $O(\log n)$ bits per node (recall Section 5). To check the proof, the verifier $\mathcal{A}$ first checks the spanning tree, and then evaluates $\varphi(A_1, A_2, \dots, A_k, a, y)$ for each node y. As φ is local around y, the verifier $\mathcal{A}$ is a local algorithm. Hence in connected graphs, any monadic Σ_1^1 graph property $\mathcal{P}$ admits locally checkable proofs of size $O(\log n)$, i. e., $\mathcal{P} \in \mathsf{LogLCP}$.

8 Open problems

We conclude by summarizing some open problems raised by our paper. While the local proof complexity as a function of the number of nodes n is now fairly well understood, we left a small gap in the case of non-3-colorability, which is probably a proof artifact.

1. Prove that non-3-colorability is not in $\mathsf{LCP}(o(n^2))$.

There are also many open questions related to the local proof complexity as a function of the maximum degree Δ:

2. In Section 4.1 we saw that directed s–t reachability admits locally checkable proofs of size $O(\log \Delta)$. Is there a proof labeling scheme with $O(1)$-bit proofs?

3. Eulerian tours admit a straightforward proof of size $O(\Delta \log n)$, and the lower-bound techniques of Section 5 imply a lower bound of $\Omega(\log n)$. Is there a proof labeling scheme with $O(\log n)$-bit proofs?

If we increase the local horizon of a verifier, can we decrease the proof size? Korman et al. [17] show how this can be done when verifying minimum weight spanning trees. Do such trade-offs exist for all properties in $\mathsf{LCP}(O(1))$, or not:

4. Do we have $\mathsf{LCP}(k) \subsetneq \mathsf{LCP}(k+1)$ for all constants $k \in \mathbb{N}$?

The analogous question has been answered affirmatively for monadic Σ_1^1 expressibility [22].

Perhaps our most nagging question is one that involves another structural property of the class $\mathsf{LCP}(O(1))$. Recall that in Section 7.4 it was straightforward to see that $\mathsf{LogLCP} \not\subseteq \mathsf{NP}$. The analogous question for $\mathsf{LCP}(O(1))$ is the following:

5. If $\mathcal{P} \subseteq \mathcal{F}$ is a graph property in $\mathsf{LCP}(O(1))$ related to *connected*, unlabeled, bounded-degree graphs, do we necessarily have $\mathcal{P} \in \mathsf{NP}$?

It should be noted that in case $\mathcal{F}$ is closed under disjoint unions of graphs, a single component $C \subset G$ in a graph $G \in \mathcal{F}$ can have node identifiers that are not bounded by a polynomial in $n(C)$. In this case, a modification of the Ramsey argument of Naor and Stockmeyer [21, §3] applies to get rid of the identifier dependencies and proves $\mathcal{P} \in \mathsf{NP}$.

Acknowledgments

We thank the anonymous reviewers for their helpful comments and suggestions. This work was supported in part by the Academy of Finland, Grants 132380 and 252018, the Finnish Cultural Foundation, and the Research Funds of the University of Helsinki. Most of this work was conducted while the authors were affiliated with the University of Helsinki.

References

[1] MIKLÓS AJTAI AND RONALD FAGIN: Reachability is harder for directed than for undirected finite graphs. *J. Symbolic Logic*, 55(1):113–150, 1990. Preliminary version in FOCS'88. JSTOR. 9, 28

[2] ALON AMIT, NATHAN LINIAL, JIŘÍ MATOUŠEK, AND EYAL ROZENMAN: Random lifts of graphs. In *Proc. 12th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'01)*, pp. 883–894. ACM Press, 2001. Abstract based on papers in Combinatorica '02, Combinatorics, Probability and Computing '06, Random Struct. Algorithms '02, and Combinatorica '05. [ACM:365801] 26

[3] DANA ANGLUIN: Local and global properties in networks of processors. In *Proc. 12th STOC*, pp. 82–93. ACM Press, 1980. [doi:10.1145/800141.804655] 25, 26

[4] MATTI ÅSTRAND AND JUKKA SUOMELA: Fast distributed approximation algorithms for vertex cover and set cover in anonymous networks. In *Proc. 22nd Ann. ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'10)*, pp. 294–302. ACM Press, 2010. [doi:10.1145/1810479.1810533] 7

[5] LOWELL W. BEINEKE: Characterizations of derived graphs. *J. Combinatorial Theory*, 9(2):129–135, 1970. [doi:10.1016/S0021-9800(70)80019-9] 2

[6] JOHN A. BONDY AND MIKLÓS SIMONOVITS: Cycles of even length in graphs. *J. Combin. Theory Ser. B*, 16(2):97–105, 1974. [doi:10.1016/0095-8956(74)90052-5] 8, 12, 15

[7] REINHARD DIESTEL: *Graph Theory*. Springer, Berlin, Germany, 3rd edition, 2005. Official website. 10, 18

[8] PAUL ERDŐS AND ALFRÉD RÉNYI: Asymmetric graphs. *Acta Mathematica Hungarica*, 14(3-4):295–315, 1963. [doi:10.1007/BF01895716] 18

[9] RONALD FAGIN: Generalized first-order spectra and polynomial-time recognizable sets. In R. M. KARP, editor, *Complexity of Computation*, volume 7, pp. 43–73. ams, 1974. Available at author's home page. 28

[10] PIERRE FRAIGNIAUD: Distributed computational complexities: are you Volvo-addicted or NASCAR-obsessed? In *Proc. 29th Ann. ACM Symp. on Principles of Distributed Computing (PODC'10)*, pp. 171–172. ACM Press, 2010. [doi:10.1145/1835698.1835700] 2

[11] PIERRE FRAIGNIAUD, AMOS KORMAN, AND DAVID PELEG: Local distributed decision. In *Proc. 52nd FOCS*, pp. 708–717. IEEE Comp. Soc. Press, 2011. To appear in J. ACM. J. ACM version available at author's home page. [doi:10.1109/FOCS.2011.17] 3, 5, 8, 9, 25, 26, 27

[12] CYRIL GAVOILLE AND DAVID PELEG: Compact and localized distributed data structures. *Distributed Computing*, 16(2-3):111–120, 2003. [doi:10.1007/s00446-002-0073-5] 8, 27

[13] MIKA GÖÖS AND JUKKA SUOMELA: Locally checkable proofs. In *Proc. 30th Ann. ACM Symp. on Principles of Distributed Computing (PODC'11)*, pp. 159–168. ACM Press, 2011. [doi:10.1145/1993806.1993829] 1

[14] NEIL IMMERMAN: *Descriptive Complexity*. Graduate Texts in Computer Science. Springer, Berlin, Germany, 1999. 9, 28

[15] AMOS KORMAN AND SHAY KUTTEN: On distributed verification. In *Proc. 8th Internat. Conf. on Distributed Computing and Networking (ICDN'06)*, pp. 100–114. Springer, 2006. [doi:10.1007/11947950_12] 2, 3, 8

[16] AMOS KORMAN AND SHAY KUTTEN: Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4):253–266, 2007. Preliminary version in PODC'06. [doi:10.1007/s00446-007-0025-1] 2, 3, 8

[17] AMOS KORMAN, SHAY KUTTEN, AND TOSHIMITSU MASUZAWA: Fast and compact self stabilizing verification, computation, and fault detection of an MST. In *Proc. 24th Ann. ACM Symp. on Principles of Distributed Computing (PODC'05)*, pp. 311–320, 2011. To appear in Distributed Computing. [doi:10.1145/1993806.1993866] 29

[18] AMOS KORMAN, SHAY KUTTEN, AND DAVID PELEG: Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010. Preliminary version in PODC'05. [doi:10.1007/s00446-010-0095-3] 2, 3, 5, 8, 10, 11, 12, 18, 27

[19] AMOS KORMAN, DAVID PELEG, AND YOAV RODEH: Constructing labeling schemes through universal matrices. *Algorithmica*, 57(4):641–652, 2010. Preliminary version in ISAAC'06. [doi:10.1007/s00453-008-9226-7] 2, 3, 8

[20] EYAL KUSHILEVITZ AND NOAM NISAN: *Communication Complexity*. Cambridge Univ. Press, Cambridge, UK, 1997. [ACM:264772] 19

[21] MONI NAOR AND LARRY J. STOCKMEYER: What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995. Preliminary version in STOC'93. [doi:10.1137/S0097539793254571] 2, 5, 7, 30

[22] MARTIN OTTO: A note on the number of monadic quantifiers in monadic Σ^1_1. *Inform. Process. Lett.*, 53(6):337–339, 1995. [doi:10.1016/0020-0190(94)00219-O] 29

[23] CHRISTOS H. PAPADIMITRIOU: *Computational Complexity*. Addison-Wesley Publishing Company, 1994. 28

[24] CHRISTOS H. PAPADIMITRIOU AND KENNETH STEIGLITZ: *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Inc., Mineola, NY, USA, 1998. [ACM:31027] 11

[25] DAVID PELEG: *Distributed Computing: A Locality-Sensitive Approach*. Volume 5 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, 2000. [ACM:355459] 6, 25

[26] THOMAS SCHWENTICK: On winning Ehrenfeucht games and monadic NP. *Ann. Pure Appl. Logic*, 79(1):61–92, 1996. Preliminary version in FOCS'94. [doi:10.1016/0168-0072(95)00030-5] 28

[27] THOMAS SCHWENTICK AND KLAUS BARTHELMANN: Local normal forms for first-order logic with applications to games and automata. *Discrete Mathematics and Theoretical Computer Science*, 3(3):109–124, 1999. Preliminary version in STACS'98. DMTCS. 28

[28] NEIL J. A. SLOANE: The On-Line Encyclopedia of Integer Sequences, 2010. `http://oeis.org`. 18

[29] JUKKA SUOMELA: Survey of local algorithms. *ACM Computing Surveys*, 45(2):24, 2013. [doi:10.1145/2431211.2431223] 2, 26

AUTHORS

Mika Göös
Postdoctoral fellow
Harvard University
Cambridge, MA
mika@seas.harvard.edu
`http://www.cs.utoronto.ca/~mgoos/`

Jukka Suomela
Assistant professor
Helsinki Institute for Information Technology HIIT
Department of Computer Science
Aalto University, Finland
jukka.suomela@aalto.fi
`http://users.ics.aalto.fi/suomela/`

ABOUT THE AUTHORS

MIKA GÖÖS is a postdoctoral fellow at the Theory of Computing group at Harvard. He obtained his Ph. D. from the University of Toronto (2016) under the supervision of Toniann Pitassi. He also holds a M. S. from University of Oxford (2011) and a B. S. from Aalto University (2010). His research interests revolve around computational complexity theory.

JUKKA SUOMELA is an assistant professor in the Department of Computer Science at Aalto University. He received his Ph. D. from the University of Helsinki in 2009, and the title of Docent in Computer Science in 2012. Suomela's research group conducts basic research related to the theoretical foundations of distributed computing, with particular emphasis on the concept of locality in the context of distributed graph algorithms.