Locally Checkable Proofs

Mika Göös
mika.goos@cs.helsinki.fi

Jukka Suomela
jukka.suomela@cs.helsinki.fi

Helsinki Institute for Information Technology HIIT, University of Helsinki
P.O. Box 68, FI-00014 University of Helsinki, Finland

ABSTRACT

This work studies *decision problems* 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-colouring 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 work we classify graph problems 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 proof complexities form a natural hierarchy of complexity classes: for many classical graph problems, the proof complexity is either 0, $\Theta(1)$, $\Theta(\log n)$, or $\mathsf{poly}(n)$ bits per node. Among the most difficult graph properties are symmetric graphs, which require $\Omega(n^2)$ bits per node, and non-3-colourable graphs, which require $\Omega(n^2/\log n)$ bits per node—any pure graph property admits a trivial proof of size $O(n^2)$.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes

General Terms

Algorithms, Theory

1. INTRODUCTION

This work studies *decision problems* from the perspective of distributed algorithms. As argued by Fraigniaud in his PODC 2010 keynote talk [8], the appropriate model for yes–no tasks 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 an alarm.

1.1 Locally Checkable Properties

Our focus is on decision tasks that can be solved *locally*, by using a constant-time distributed algorithm [18, 23]. That is, each node must make its decision based on its constant-radius neighbourhood in the communication graph; equivalently, we run the distributed algorithm for $O(1)$ rounds and after that all nodes must stop and announce their outputs.

A trivial example of a decision problem that can be solved locally 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. Such graph properties are called *locally checkable properties*—there exists a local algorithm, called *verifier*, that accepts all Eulerian graphs and rejects all non-Eulerian graphs.

Another example of locally checkable properties is deciding 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 [3] characterisation of line graphs.

1.2 Locally Checkable Proofs

Local checkability as such does not seem to lead to an interesting complexity theory—there are very few locally checkable properties. The key insight of Korman et al. [14–17] is to study locally checkable *proofs*.

To illustrate the idea, consider the problem of deciding 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 distributed algorithm that the graph is indeed bipartite, we can augment the graph with a *locally checkable proof*. In this case, it is sufficient to give 1 bit of proof per node: if the graph is bipartite, we can give a 2-colouring 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,

http://doi.acm.org/10.1145/1993806.1993829

there is a locally checkable proof of size 1 bit per node (refer to Section 2 for a precise definition).

The concepts of locally checkable proofs and locally checkable properties are analogous to, e.g., the familiar pair of complexity classes NP and 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.

1.3 Contributions

In this work, we define the class $\mathsf{LCP}(f)$ that consists of graph problems that admit locally checkable proofs of size $f(n)$ bits per node. The model is similar to that studied by Korman et al. [14–17], but strictly stronger.

We catalogue problems 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 problems that separate the following levels of the hierarchy: $\mathsf{LCP}(0)$, $\mathsf{LCP}(O(1))$, $\mathsf{LCP}(O(\log n))$, and $\mathsf{LCP}(\mathsf{poly}(n))$; see Table 1.

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 (see Section 7.1). Many central graph problems are contained in LogLCP, and they are not contained in $\mathsf{LCP}(o(\log n))$.

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. The same techniques can be applied to derive lower bounds for many other problems in addition to those mentioned in this work.

2. DEFINITIONS AND EXAMPLES

In what follows, $\mathcal{F}$ is a family of simple, undirected graphs. 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. We assume that the nodes of any $G \in \mathcal{F}$ are identified with small natural numbers with $O(\log n)$ bits, that is, $V(G) \subseteq \{1, 2, \ldots, \mathsf{poly}(n(G))\}$. Depending on the problem that we study, nodes and edges may also be associated with weights, colours, labels, etc.

2.1 Proofs and Verifiers

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

A *verifier* $\mathcal{A}$ is a computable 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* $P(u)$.

Table 1: Local Proof Complexities

(a) The local proof complexity of verifying graph property $\mathcal{P}$, assuming that the input graph is in graph family $\mathcal{F}$. Constant k is a natural number. In reachability problems, nodes s and t are labelled; otherwise the graphs are unlabelled (i.e., the focus is on pure graph properties).

Proof size s	Graph property $\mathcal{P}$	Family $\mathcal{F}$	Ref.
$\mathsf{LCP}(0)$:			
0	Eulerian graph	conn.	§1.1
0	line graph	general	§1.1
$\mathsf{LCP}(O(1))$:			
$\Theta(1)$	s–t reachability	undir.	§4.1
$\Theta(1)$	s–t unreachability	undir.	§4.1
$\Theta(1)$	s–t unreachability	directed	§4.1
$\Theta(1)$	s–t connectivity $= k$	planar	§4.2
$\Theta(1)$	bipartite graph	general	§1.2
$\Theta(1)$	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	§2.2
LogLCP:			
$O(\log n)$	coLCP(0) properties	conn.	§7.3
$O(\log n)$	monadic Σ^1_1 properties	conn.	§7.5
$\Theta(\log n)$	odd $n(G)$	cycles	§5
$\Theta(\log n)$	chromatic number > 2	conn.	§5
$\mathsf{LCP}(\mathsf{poly}(n))$:			
$\Theta(n)$	fixpoint-free symmetry	trees	§6.2
$\Theta(n^2)$	symmetric graphs	conn.	§6.1
$\Omega(n^2/\log n)$	chromatic number > 3	conn.	§6.3
$O(n^2)$	computable properties	conn.	§6
—	connected graph	general	

(b) The local proof complexity of verifying a solution of graph problem $\mathcal{P}$, assuming that the input graph is in graph family $\mathcal{F}$. Here W is the maximum weight of an edge.

Proof size s	Graph problem $\mathcal{P}$	Family $\mathcal{F}$	Ref.
$\mathsf{LCP}(0)$:			
0	maximal matching	general	§2.3
0	LCL problems	general	§3, [18]
0	LD problems	conn.	§3, [10]
$\mathsf{LCP}(O(1))$:			
$\Theta(1)$	maximum matching	bipartite	§2.3
$\mathsf{LCP}(O(\log W))$:			
$O(\log W)$	max-weight matching	bipartite	§2.3
LogLCP:			
$O(\log n)$	coLCP(0) problems	conn.	§7.3
$\Theta(\log n)$	leader election	conn.	§5, [16]
$\Theta(\log n)$	spanning tree	conn.	§5, [16]
$\Theta(\log n)$	maximum matching	cycles	§5
$\Theta(\log n)$	Hamiltonian cycle	conn.	§5
$\mathsf{LCP}(\infty)$:			
unlimited	NLD problems	conn.	§3, [10]
unlimited	$\mathsf{NLD}^{\#n}$ problems	conn.	§3, [10]

For a natural number $r \in \mathbb{N}$ and a node $v \in V(\mathcal{G})$, let $V[v,r] \subseteq V(\mathcal{G})$ be the 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 $\mathcal{G}[v,r]$ be the subgraph of $\mathcal{G}$ induced by $V[v,r]$, and let $P[v,r]\colon V[v,r] \to \{0,1\}^\star$ be the restriction of a proof $P\colon V(\mathcal{G}) \to \{0,1\}^\star$ to $V[v,r]$.

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

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

That is, the output of a node v only depends on the input in its radius-r neighbourhood. Constant r is the *local horizon* of $\mathcal{A}$.

Local verifiers are local algorithms [18, 23]. If we consider Peleg's [19] *local* model, a local verifier is a constant-time distributed algorithm: a local verifier with horizon r can be implemented as a distributed 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 re-assigning the identifiers of the nodes. Put otherwise, if $\mathcal{G}$ and $\mathcal{G}'$ are isomorphic (they have the same structure but possibly different node identifiers), then $\mathcal{G}' \in \mathcal{P}$ if and only if $\mathcal{G} \in \mathcal{P}$. Examples of graph properties include Hamiltonian graphs, Eulerian graphs, bipartite graphs, connected graphs, line graphs, trees, and cycles.

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 $\mathcal{A}$ such that for every $\mathcal{G} \in \mathcal{F}$:

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

(ii) If $\mathcal{G} \notin \mathcal{P}$ then for any proof $P\colon V(\mathcal{G}) \to \{0,1\}^\star$ there is at least one node $v \in V(\mathcal{G})$ such that $\mathcal{A}(\mathcal{G}, P, v) = 0$.

That is, *yes*-instances have a valid proof that is accepted by all nodes, *no*-instances do not have valid proofs, and at least one node detects an invalid proof. If f is a function that associates a valid proof $P = f(\mathcal{G})$ with each $\mathcal{G} \in \mathcal{P}$, we say that the pair $(f, \mathcal{A})$ is a *proof labelling 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 $s = O(\log n)$; see Section 7.1 for an equivalent, alternative characterisation.

As we observed in Section 1, Eulerian graphs and line graphs can be verified without a proof, and hence they are in $\mathsf{LCP}(0)$. 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 $O(\log k)$ bits per node: simply give a proper k-colouring as the proof.

2.3 Extension: Solutions of Graph Problems

If we consider graphs with labelled nodes, 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 extend the definitions of locally checkable proofs to the verification of the solutions of graph problems (see Section 7.2 for two variants of the theme).

For example, maximal matchings can be verified without any proofs, and hence we say that this problem is in $\mathsf{LCP}(0)$. On the other hand, verifying a maximum matching (*"edges with label* 1 *form a maximum-cardinality matching"*) requires some auxiliary information. The general case is non-trivial, but in the case of bipartite graphs we can use König's theorem [5, p. 35] to construct a constant-size proof P: take any minimum vertex cover $C \subseteq V(\mathcal{G})$, and set $P(v) = 1$ iff $v \in C$. Now a local verifier can check that the node labelling encodes a valid matching M, the set C encoded in the proof forms a vertex cover, and each edge of M has exactly one endpoint in C; hence $|C| = |M|$, C is a minimum vertex cover, and M must be a maximum matching. Therefore maximum matchings in bipartite graphs are in $\mathsf{LCP}(1)$, as the size of P is 1.

More generally, we can use linear-programming duality to prove that in an edge-weighted bipartite graph $\mathcal{G}$, a subset of edges $M \subseteq E(\mathcal{G})$ 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 incidence matrix of graph $\mathcal{G}$. Recall that matrix A and its transpose $A^\top$ are totally unimodular, and hence there are integral vectors x and y that maximise $\sum_e w_e x_e$ subject to $Ax \leq 1$ (primal LP) and minimise $\sum_v y_v$ 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.

3. RELATED WORK

Our definition of the LCP hierarchy is an extension of the concept of *locally checkable labellings* introduced by Naor and Stockmeyer [18] in their seminal work. Naor and Stockmeyer focus on bounded-degree graphs and constant-size labels, but if we generalise the class LCL defined by Naor and Stockmeyer in a straightforward manner, we arrive at the class $\mathsf{LCP}(0)$.

Our classes $\mathsf{LCP}(f)$ with $f > 0$ thus extend the classical concept of locally checkable labellings by providing $f(n)$ bits of additional information per node. Similar extensions have been studied in prior work, from two complementary perspectives: local computation with advice and locally checkable proofs.

3.1 Local Computation with Advice

Gavoille and Peleg [11] survey *informative localised labelling schemes* that can be used in the context of distributed algorithms. The idea is to provide each node with a piece of *advice*—a short bit string that helps with local algorithms that solve graph problems. For example, Fraigniaud et al. [9] have recently investigated the following question: how long strings of advice are needed in order to solve classical graph problems such as graph colouring by using local or almost-local algorithms.

The main difference between advice strings and locally checkable proofs is that the advice strings are *assumed* to be correct, while the correctness of a proof can be *verified*. Put otherwise, localised labelling schemes are only applicable in

a friendly and fault-free environment, while a local verifier cannot be fooled even by an adversarial entity.

3.2 Locally Checkable Proofs

The definition of LCP is inspired by the notion of *proof labelling schemes* of Korman et al. [14–17]. While the concepts are closely related to each other, there are subtle differences: In the model of Korman et al., 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 neighbouring nodes. In this model, some trivial problems that are in LCL become unsolvable without proof labels of nonzero size; one example is the *agreement problem* of checking whether all nodes in a connected graph are assigned the same input label [16, Lemma 2.1]. Hence the notion of proof labelling schemes is not a straightforward generalisation of the LCL model—something our LCP model strives to be. The positive results by Korman et al. translate directly to the LCP model, but their lower-bound results do not directly apply, as their model is strictly weaker.

Very recently, Fraigniaud et al. [10] have also studied distributed decision problems. Their focus is on connected graphs; to clarify the relation to our work, let us define that $\mathsf{LCP}'(f)$ is equal to $\mathsf{LCP}(f)$ 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 $\mathsf{LCP}'(0)$. Fraigniaud et al. have also studied two nondeterministic versions of this class, called NLD and $\mathsf{NLD}^{\#n}$; in the latter class each node knows the total number of nodes in the graph. While NLD resembles the class $\mathsf{LCP}'(\infty)$ in our work, there is a major difference: proofs in the NLD model cannot refer to the node identifiers. 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 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.

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 $\mathcal{G}$; 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 decide whether there is a path from s to t in $\mathcal{G}$. However, many questions related to reachability and connectivity are in $\mathsf{LCP}(O(1))$.

4.1 Reachability

Let us first consider the *s–t reachability* problem in an undirected graph $\mathcal{G}$, 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 $\mathcal{G}$, 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 neighbours in U; and (iii) every $u \in U \setminus \{s, t\}$ has exactly two neighbours in U [13, 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 labelling 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 generalisation 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. [16] 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 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 $|C \cap p_i| = 1$. W.l.o.g., we can assume that each p_i is locally minimal in the sense that it can not 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:

(i) Nodes s and t have exactly k neighbours labelled with path indices $1, 2, \ldots, k$.

(ii) Each $v \in p_i \setminus \{s, t\}$ has exactly one predecessor and one successor along p_i.

(iii) We have $s \in S$, $t \in T$, and there is no edge between S and T.

(iv) 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 colour changes from S to T; its colour 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.

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. [16] 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 neighbours of v agree on the identity of the root, and (ii) the root-distance decreases at exactly one neighbour of v in T and increases at other neighbours.

A locally checkable, rooted spanning tree is a versatile tool. For example, it solves the leader election problem in a connected graph: the root of the tree is the leader. Spanning trees can be 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 $n(\mathcal{G})$ on a connected graph $\mathcal{G}$ with the aid of a spanning tree with node counters along the paths towards the root. 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

In what follows, we will show that the following graph properties and graph problems do not admit locally checkable proofs of size $o(\log n)$: graphs with an 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 $\Theta(\log n)$.

The negative results build on the same basic idea. We will focus on cycles. We will assume that there is a proof labelling 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 use extremal results to 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, we will show that the long cycle is a *no*-instance. For example, in the case of non-bipartiteness, each short cycle has an odd number of nodes, but the long cycles 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.

5.3 Gluing Cycles Together

Let $\mathcal{F}$ be a family of graphs that contains (at least) all cycles. In each graph $\mathcal{G} \in \mathcal{F}$, we may have a constant number of bits of auxiliary information per node (colours, labels, etc.). Let $\mathcal{P} \subseteq \mathcal{F}$ be a graph property.

Assume that $(f, \mathcal{A})$ is a proof labelling 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, see Figure 1.

Let us first construct the *yes*-instances. Let $n_1 = \lfloor n/2 \rfloor$ and $n_2 = \lceil n/2 \rceil$; that is, $n_1 + n_2 = n$. Let $A = \{1, 2, \ldots, n\}$ and $B = \{n+1, n+2, \ldots, 2n\}$. For each $a \in A$ and $b \in B$, let $\mathcal{C}(a, b)$ be the n-cycle that contains the following nodes in this order:

$$a,\ a+4n,\ a+6n,\ a+8n,\ \ldots,\ a+2nn_1,$$
$$b+2nn_2,\ \ldots,\ b+8n,\ b+6n,\ b+4n,\ b.$$

Note that $V(\mathcal{C}(a, b))$ and $V(\mathcal{C}(a', b'))$ are disjoint if $a \neq a'$ and $b \neq b'$.

For each a and b, augment $\mathcal{C}(a, b)$ with auxiliary information such that $\mathcal{C}(a, b)$ is in $\mathcal{P}$, if necessary; let $L_{ab}(v) \in \{0, 1\}^\star$ be the bit string associated with node $v \in V(\mathcal{C}(a, b))$. For example, if we focus on the leader election problem, label exactly one node in each $\mathcal{C}(a, b)$ as the leader: select a node $u \in V(\mathcal{C}(a, b))$ and set $L_{ab}(u) = 1$ and $L_{ab}(v) = 0$ for all $v \neq u$. We can consider either best-case or worst-case choice of the leader, thus covering both weak and strong proof labelling schemes (see Section 7.2).

Then we apply f to $\mathcal{C}(a, b)$ to construct a locally checkable proof P_{ab} of size $o(\log n)$. For each node $v \in V(\mathcal{C}(a, b))$, let $P'_{ab}(v) = (L_{ab}(v), P_{ab}(v))$. Finally, define

$$\begin{aligned} c(a,b) = \big(&P'_{ab}(a + 2n \cdot (2r+1)),\ P'_{ab}(a + 2n \cdot 2r),\ \ldots,\\ &P'_{ab}(a + 2n \cdot 2),\ P'_{ab}(a),\ P'_{ab}(b),\ P'_{ab}(b + 2n \cdot 2),\\ &P'_{ab}(b + 2n \cdot 4),\ \ldots,\ P'_{ab}(b + 2n \cdot (2r+1))\big). \end{aligned}$$

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 or b in $\mathcal{C}(a, b)$. By assumption, we have $o(r \log n)$ bits of information in $c(a, b)$.

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

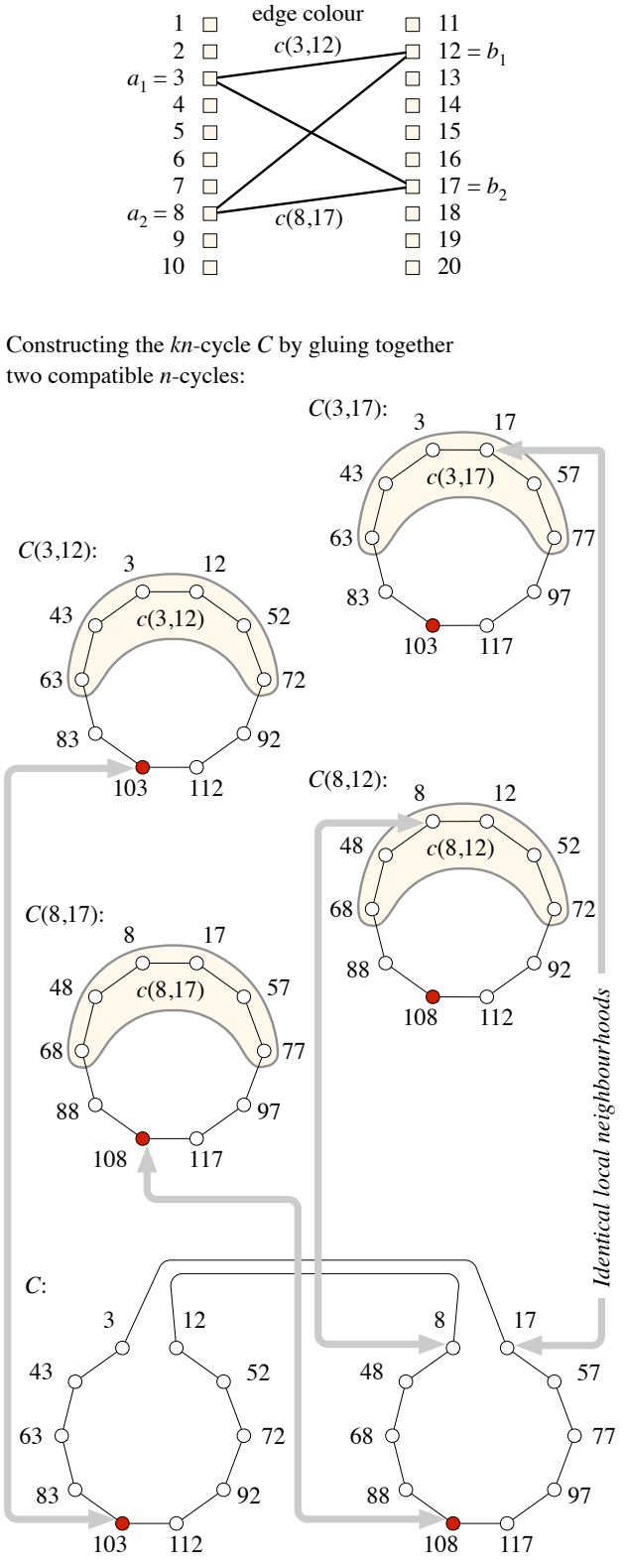

Figure 1: Gluing cycles together.

colouring of $K_{n,n}$ as follows: the colour 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 $\log(n)/3$, and the number of distinct colours 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 colour.

Now we can apply a result due to Bondy and Simonovits [4]: 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 colour, 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 $a_{k+1} = a_1$.

Now we construct a kn-cycle $\mathcal{C}$ by gluing together n-cycles $\mathcal{C}(a_1, b_1), \mathcal{C}(a_2, b_2), \ldots, \mathcal{C}(a_k, b_k)$. That is, we take the node-disjoint graphs $\mathcal{C}(a_i, b_i)$, remove the edges $\{a_i, b_i\}$ for each i, and add $\{b_{i-1}, a_i\}$ for each $i > 1$. For each node $v \in V(\mathcal{C})$, we inherit the auxiliary information $L(v)$ and the proof bits $P(v)$ from the cycles $\mathcal{C}(a_i, b_i)$.

It remains to argue that the computation of $\mathcal{A}$ on $\mathcal{C}$ with the labels L and proof P is accepting. To see this, pick a vertex $v \in V(\mathcal{C})$. Then there is an i such that $v \in V(\mathcal{C}(a_i, b_i))$. If v is far from a_i and b_i, then the local neighbourhood of v looks identical in $\mathcal{C}$ and $\mathcal{C}(a_i, b_i)$; as $\mathcal{C}(a_i, b_i)$ is a *yes*-instance, v accepts the input. If v is near b_i, then the local neighbourhood of v looks identical in $\mathcal{C}$ and $\mathcal{C}(a_{i+1}, b_i)$, which is another *yes*-instance. Similarly, if v is near a_i, then its local neighbourhood looks identical in $\mathcal{C}$ and $\mathcal{C}(a_i, b_{i-1})$, which is also a *yes*-instance. In all cases, v accepts the input.

Thus the kn-cycle $\mathcal{C}$ is accepted by all nodes. If $\mathcal{C} \notin \mathcal{P}$, 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 $\mathcal{C} \notin \mathcal{P}$, 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 $\mathcal{C}(a, b)$ contains exactly one node labelled as a leader and $\mathcal{C}$ contains two nodes.
- Spanning trees: Again, we can choose $k = 2$. The spanning tree in each $\mathcal{C}(a, b)$ contains all edges of $E(\mathcal{C}(a, b))$ except one, i.e., it is a spanning path. The solution encoded in $\mathcal{C}$ 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 $n(\mathcal{C}(a, b))$ is odd while $n(\mathcal{C})$ is even.

We can also prove lower bounds for optimisation problems. Consider, for example, the problem of finding a maximum matching in a cycle. If n is odd, then each $\mathcal{C}(a, b)$ has necessarily one unmatched node. The solution inherited from $\mathcal{C}(a, b)$ to $\mathcal{C}$ 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.

6. PROBLEMS IN LCP(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 *pure graph properties*: we do not have any additional information besides the structure of the graph $\mathcal{G}$ and the unique node identifiers.

In connected graphs, *any* computable pure graph property admits locally checkable proofs of size $O(n^2)$. We can encode the structure of $\mathcal{G}$ and the unique node identifiers in $O(n^2)$ bits; the nodes can verify that their neighbours agree on the structure of $\mathcal{G}$, and then they can solve the problem by brute force.

In this section, we will show that there are pure 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

We will focus on the family $\mathcal{F}$ of connected graphs. In what follows, we say that a graph $\mathcal{G}$ 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*.

Let $\mathcal{P} \subset \mathcal{F}$ consist of symmetric graphs. We will show that property $\mathcal{P}$ does not admit locally checkable proof of size $o(n^2)$. That is, a proof with $o(n^2)$ bits per node is not sufficient to convince a local verifier that a given connected graph is symmetric. To reach a contradiction, assume that there exists a proof labelling 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 $C(\mathcal{G})$ with any graph $\mathcal{G}$. Graphs $\mathcal{G}$ and $C(\mathcal{G})$ are isomorphic; moreover, whenever $\mathcal{G}$ and $\mathcal{H}$ are isomorphic, their canonical forms $C(\mathcal{G})$ and $C(\mathcal{H})$ are equal. We assume that the node identifiers of a canonical form are $V(C(\mathcal{G})) = \{1, 2, \ldots, n(\mathcal{G})\}$. We also define a graph with *shifted identifiers* as follows: for an integer i, graph $C(\mathcal{G}, i)$ has

$$V(C(\mathcal{G}, i)) = \{i + 1, i + 2, \ldots, i + n(\mathcal{G})\}$$

as the set of node identifiers. Moreover, we assume that $g\colon v \mapsto i + v$ is an isomorphism from $C(\mathcal{G})$ to $C(\mathcal{G}, i)$. In particular, $C(\mathcal{G}, 0) = C(\mathcal{G})$.

Now we are ready to give the lower-bound construction. Given connected graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ with $n(\mathcal{G}_1) = n(\mathcal{G}_2) = k$, we construct a graph $\mathcal{G} = \mathcal{G}_1 \circledcirc \mathcal{G}_2$ with $V(\mathcal{G}) = \{1, 2, \ldots, 3k\}$ as follows: $\mathcal{G}$ consists of a copy of $C(\mathcal{G}_1, k)$, a copy of $C(\mathcal{G}_2, 2k)$, and the path $(k + 1, 1, 2, \ldots, k, 2k + 1)$. That is, $\mathcal{G}$ consists of a path that joins graphs that are isomorphic to $\mathcal{G}_1$ and $\mathcal{G}_2$.

Assume that $\mathcal{G}_1$ and $\mathcal{G}_2$ are asymmetric. If $\mathcal{G}_1$ and $\mathcal{G}_2$ are isomorphic, then by construction $\mathcal{G} = \mathcal{G}_1 \circledcirc \mathcal{G}_2$ is symmetric: there is a non-trivial automorphism that maps, e.g., $1 \mapsto k$ and $k + 1 \mapsto 2k + 1$. Conversely, if $\mathcal{G}_1$ and $\mathcal{G}_2$ are not isomorphic, then $\mathcal{G}$ must be asymmetric.

Let $\mathcal{F}_k$ be a family containing a representative from each isomorphism class of asymmetric connected graphs with k nodes. For any $\mathcal{G}_1 \in \mathcal{F}_k$, the local verifier $\mathcal{A}$ has to accept $\mathcal{G}_1 \circledcirc \mathcal{G}_1$, as it is symmetric. Since almost all graphs are connected [5, Cor. 11.3.3] and asymmetric [6], we have

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

Now assume that $k \geq 2r + 1$, and consider the proof labels of the nodes in $U = \{1, 2, \ldots, 2r + 1\}$. 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 $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{F}_k$ such that the labelling scheme assigns the same proof bits to the nodes $1, 2, \ldots, 2r + 1$ in both $\mathcal{G}_1 \circledcirc \mathcal{G}_1$ and $\mathcal{G}_2 \circledcirc \mathcal{G}_2$.

Now we can construct the asymmetric graph $\mathcal{G} = \mathcal{G}_1 \circledcirc \mathcal{G}_2$. For the nodes $k + 1, k + 2, \ldots, 2k$ we inherit the proof labels from $f(\mathcal{G}_1 \circledcirc \mathcal{G}_1)$, and for the nodes $2r + 2, 2r + 3, \ldots, k, 2k + 1, 2k + 2, \ldots, 3k$ we inherit the proof labels from $f(\mathcal{G}_2 \circledcirc \mathcal{G}_2)$. For the nodes $1, 2, \ldots, 2r + 1$ we use the common labelling of $f(\mathcal{G}_1 \circledcirc \mathcal{G}_1)$ and $f(\mathcal{G}_2 \circledcirc \mathcal{G}_2)$. Now the radius-$r$ neighbourhood of any node in $\mathcal{G}$ looks identical to the neighbourhood of a node in $\mathcal{G}_1 \circledcirc \mathcal{G}_1$ or $\mathcal{G}_2 \circledcirc \mathcal{G}_2$. Hence all nodes will accept the input even though $\mathcal{G}$ is not symmetric, a contradiction.

In conclusion, in order to verify that a given connected graph is symmetric, we need proofs of size $\Theta(n^2)$.

6.2 Fixpoint-Free Symmetry on Trees

Let us now focus on the family $\mathcal{F}$ of connected trees. Then any pure graph property $\mathcal{P} \subseteq \mathcal{F}$ admits a locally checkable proof of size $O(n)$: for each node v of the tree $\mathcal{G} \in \mathcal{P}$, we encode the structure of $\mathcal{G}$ and an index that identifies which node of $\mathcal{G}$ 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 pure graph properties that require $\Theta(n)$-bit proofs. We will use the following (artificial) problem as an example. We say that a graph $\mathcal{G}$ has a *fixpoint-free symmetry* if there is an automorphism that fixes no nodes, i.e., there is an automorphism $g\colon V(\mathcal{G}) \to V(\mathcal{G})$ such that $g(v) \neq v$ for all $v \in V(\mathcal{G})$.

Let $\mathcal{P} \subset \mathcal{F}$ consists of those connected trees that have a fixpoint-free symmetry. We will show that $\mathcal{P}$ does not admit proofs of size $o(n)$, and is therefore among the most difficult properties of trees.

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 $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{F}_k$ and k is even, then $\mathcal{G}_1 \circledcirc \mathcal{G}_2$ has a fixpoint-free symmetry if and only if $\mathcal{G}_1 = \mathcal{G}_2$. We have $\log |\mathcal{F}_k| = \Theta(k)$ [22, Seq. A000081]; hence a proof of size $o(n)$ bits leads to a contradiction. We conclude that trees with a fixpoint-free symmetry require locally checkable proofs of size $\Theta(n)$.

6.3 Non-3-Colourability

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

In the case of 3-colourings, 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 coloured with 3 colours, as we can give a 3-colouring as a proof. However, to prove that a graph cannot be coloured

with 3 colours, we need very large proofs, with polynomially many bits per node.

More specifically, let us again focus on the family $\mathcal{F}$ of connected graphs, and let $\mathcal{P} \subset \mathcal{F}$ 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 pure graph property admits proofs of size $O(n^2)$; hence the result is almost tight, and shows that non-3-colourability does not have a proof labelling scheme that is substantially better than the brute-force approach.

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 $\mathcal{G}_A$, with the following properties:

(i) The total number of nodes in $\mathcal{G}_A$ is $\Theta(2^k)$.

(ii) The set of nodes $V(\mathcal{G}_A)$ 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, valid 3-colourings of $\mathcal{G}_A$ have the following properties:

(iii) The nodes T, F, and N have three different colours. The nodes with the same colour as T are said to be *true*, those with the same colour 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 colouring of the nodes x_i as a binary encoding of an integer $x \in \{0, 1, \ldots, 2^k - 1\}$; similarly the colouring of the nodes y_i is a binary encoding of an integer y.

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

Such graphs exist; for an explicit construction of $\mathcal{G}_A$ as well as additional illustrations, see the extended version of this work [12].

We denote by $\mathcal{G}'_A$ an isomorphic copy of $\mathcal{G}_A$; we use the primed symbols T', F', N', x'_i, and y'_i to refer to the nodes of $\mathcal{G}'_A$.

In addition to graphs $\mathcal{G}_A$ and $\mathcal{G}'_A$, we will need *wires* that propagate colours. A wire w consists of $9r$ nodes, labelled $w(i, j)$ for $i = 1, 2, \ldots, 3r$ 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-colouring of a wire, the nodes $w(i, 1)$, $w(i, 2)$ and $w(i, 3)$ must have different colours, and $w(i, j)$ must have the same colour as $w(i + 1, j)$.

Given two sets $A, B \subseteq I \times I$, we construct a graph $\mathcal{G} = \mathcal{G}_{A,B}$ that consists of $\mathcal{G}_A$ and $\mathcal{G}'_B$ that are connected to each other by $2k+1$ wires. The wires are labelled with w_T, $w_1^x, w_2^x, \ldots, w_k^x$, and $w_1^y, w_2^y, \ldots, w_k^y$. The endpoints of the wires are identified with the nodes of the subgraphs $\mathcal{G}_A$ and $\mathcal{G}'_B$ as follows:

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

We make the following observations of $\mathcal{G}$:

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

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

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

Moreover, 3-colourings of $\mathcal{G}$ have the following properties:

(iv) Nodes N and N' have the same colour, nodes T and T' have the same colour, and nodes F and F' have the same colour. Hence the concepts of true, false, and neutral nodes are well-defined in $\mathcal{G}$.

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

It follows that $\mathcal{G}_{A,B}$ has a 3-colouring 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 $\mathcal{G}_{A,\bar{A}}$ does not have a 3-colouring; 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 $\mathcal{G}_{A,\bar{A}}$ and $\mathcal{G}_{B,\bar{B}}$.

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). W.l.o.g., assume that $A \cap \bar{B} \neq \emptyset$. Hence $\mathcal{G}_{A,\bar{B}}$ admits a 3-colouring, and it is therefore not in $\mathcal{P}$. But we can construct a proof as follows: the proof bits of $\mathcal{G}_A$ are inherited from the proof of $\mathcal{G}_{A,\bar{A}}$, the proof bits of $\mathcal{G}'_{\bar{B}}$ are inherited from the proof of $\mathcal{G}_{B,\bar{B}}$, and the proof bits of wires are the same as in $\mathcal{G}_{A,\bar{A}}$ and $\mathcal{G}_{B,\bar{B}}$. Hence each node of $\mathcal{G}_{A,\bar{B}}$ has a local neighbourhood that looks identical to a node of $\mathcal{G}_{A,\bar{A}}$ or $\mathcal{G}_{B,\bar{B}}$. As $\mathcal{G}_{A,\bar{A}}$ and $\mathcal{G}_{B,\bar{B}}$ are *yes*-instances, all nodes will accept $\mathcal{G}_{A,\bar{B}}$, a contradiction.

We conclude that non-3-colourability requires proofs of size $\Omega(n^2/\log n)$, and proofs of size $O(n^2)$ are trivially sufficient.

7. DISCUSSION

We conclude this work by discussing alternative definitions and extensions of the LCP hierarchy, and by relating the LCP classes to each other as well as other complexity classes.

7.1 Alternative Characterisations of LogLCP

Throughout this work, we use the assumption that the local verifier can access the unique identifiers of the nodes, and our definition of the class LogLCP builds on this assumption as well. However, we can use spanning trees to show that the definition of 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. For the sake of simplicity, we will focus on connected graphs.

Let us consider two different models of distributed computation, M_1 and M_2. Model M_1 has *unique identifiers*, while model M_2 has a *port numbering* and a *leader*. In more detail, model M_1 is the one defined in Section 2: each node has a unique identifier of size $O(\log n)$ bits. Model M_2 is defined as follows: The nodes do not have unique identifiers. There is only port numbering [2] available in the network, i.e., a node

of degree d can refer to its neighbours by integers $1, 2, \ldots, d$. In addition to the port numbering, we know that there is exactly one node $a \in V(\mathcal{G})$ that is designated as a leader.

From the perspective of properties that can be verified without auxiliary information, the two models are very different. To give an example, in M_1 it is easy to verify that the graph is triangle-free, while this is not solvable in M_2. However, it turns out that the class of properties that can be verified with $O(\log n)$ bits in M_1 is equal to the class of properties that can be verified with $O(\log n)$ bits in M_2.

To see this, assume that $(f_2, \mathcal{A}_2)$ is a proof labelling scheme for a graph property $\mathcal{P}$ in model M_2. Then we can construct a scheme $(f_1, \mathcal{A}_1)$ for model M_1 as follows: Consider a graph $\mathcal{G} \in \mathcal{P}$ with an arbitrary choice of unique identifiers. We assign the port numbers as follows: for each node the neighbour number i is the neighbour with the ith smallest unique identifiers. We choose an arbitrary leader node $a \in \mathcal{G}$ and an arbitrary spanning tree T rooted at a. Now f_1 constructs a proof that consists of $f_2(\mathcal{G})$ and an encoding of the spanning tree (T, a). Then $\mathcal{A}_1$ can first verify the encoding of (T, a), and then simulate $\mathcal{A}_2$.

Conversely, if $(f_1, \mathcal{A}_1)$ is a proof labelling scheme for $\mathcal{P}$ in model M_1, we can construct a scheme $(f_2, \mathcal{A}_2)$ for M_2 as follows: Consider a graph $\mathcal{G} \in \mathcal{P}$ with an arbitrary port numbering and a leader a. Let T be an arbitrary spanning tree rooted at a. Generate unique identifiers for $V(\mathcal{G})$ by doing a depth-first traversal on T starting at a and recording, for every node v, its discovery time $x(v)$ and finishing time $y(v)$; the unique identifier of a node v is an encoding of the pair $(x(v), y(v))$. Now f_2 constructs a proof that consists of $f_1(\mathcal{G})$, an encoding of (T, a), and the unique identifier of each node. Then $\mathcal{A}_2$ can first verify the encoding of (T, a), and it can check that the pairs $(x(v), y(v))$ are locally consistent with a depth-first traversal on the rooted spanning tree—it follows that the node identifiers must be globally unique. Finally, $\mathcal{A}_2$ can simulate $\mathcal{A}_1$ using the unique identifiers that were encoded in the proof.

In essence, we can use a locally checkable spanning tree [16] and a simple *ancestor labelling scheme* [11] to translate proof labelling schemes between models M_1 and M_2 with only $O(\log n)$ overhead. Hence the class LogLCP can be defined equally well using either of these models.

7.2 Weak and Strong Schemes

For graph problems, we can consider two variants of proof labelling schemes:

- *Strong* proof labelling schemes: in any graph $\mathcal{G}$, for any feasible solution X, there is a proof that shows that X is a correct solution.
- *Weak* proof labelling schemes: in any graph $\mathcal{G}$, there is at least one feasible solution X, such that we can prove that X is a correct solution.

Put otherwise, in a strong proof labelling 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 labelling scheme, an adversary chooses the input but we can choose a solution.

Intuitively, solving the weak version of the problem might be easier, and a weak proof labelling scheme could admit smaller proofs. For example, in the leader election problem, we could focus on convenient solutions: perhaps we could select the node with the smallest identifier as the leader, and come up with a small locally checkable proof for such a selection.

However, for many natural problems studied in this work, the proof complexities of strong and weak proof labelling schemes are within a constant factor of each other. In Section 5 we saw that problems such as leader election and spanning trees admit locally checkable proofs of size $O(\log n)$. All of these results are *strong* proof labelling schemes: for example, we can take *any* spanning tree and augment it with a proof of size $O(\log n)$. We also saw that these problems do not admit locally checkable proofs of size $o(\log n)$. The lower-bound result precludes not only the existence of strong proof labelling schemes, but it also shows that there is no weak proof labelling scheme.

7.3 Complement of LCP(0)

On connected graphs, one can employ spanning trees to reverse the decision made by an $\mathsf{LCP}(0)$ verifier $\mathcal{A}$ as follows. Let $\mathcal{P}$ be a graph property in $\mathsf{LCP}(0)$. If we have a *no*-instance $\mathcal{G} \notin \mathcal{P}$, then we can construct a proof P of size $O(\log n)$ that convinces a local verifier $\bar{\mathcal{A}}$ of $\mathcal{G} \notin \mathcal{P}$.

To construct the proof P, select a root node a with $\mathcal{A}(\mathcal{G}, \epsilon, a) = 0$, i.e., a is a node that rejects the input $\mathcal{G}$. 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. Then a local verifier $\bar{\mathcal{A}}$ can verify that T is valid spanning tree rooted at a; in particular, there is a finite path from any $v \in V(\mathcal{G})$ to a. Moreover, at the root node, $\bar{\mathcal{A}}$ can simulate $\mathcal{A}$ and verify that $\mathcal{A}(\mathcal{G}, \epsilon, a) = 0$. We conclude that $\mathsf{coLCP}(0) \subseteq \mathsf{LogLCP}$ on connected graphs.

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 $\mathcal{G}$ (or unbounded amount of additional information per node in the form of colours or weights), we can easily come up with artificial problems that are in $\mathsf{LCP}(0)$ but not in NP.

However, the situation becomes much more interesting if we focus on bounded-degree graphs; moreover, we will focus on pure graph properties, 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: w.l.o.g., we can assume that a LogLCP verifier has access to $n(\mathcal{G})$ in any connected graph $\mathcal{G}$. Hence the verifier can solve arbitrarily hard computable problems concerning the integer $n(\mathcal{G})$, including those that are not in NP.

However, if $\mathcal{P} \in \mathsf{LogLCP}$ is a pure graph property related to bounded-degree graphs, we *can* show that $\mathcal{P}$ is in $\mathsf{NP}_{/\mathsf{poly}}$, 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 $\mathsf{NP}_{/\mathsf{poly}}$ machine M. Then M merely guesses the $O(n \log n)$-bit proof $P\colon V(\mathcal{G}) \to \{0,1\}^\star$, 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 [7], [13, Ch. 7] characterization of the class NP as graph problems expressible by *existential second-order formulas* (Σ_1^1). Some NP-complete graph properties are even expressible by *monadic* Σ_1^1 formulas that only quantify over *unary* relation symbols [1, 20]. 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 [13, Ch. 6]. Building on this work, Schwentick and Barthelmann [21] 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 and *local around* y. Here, a formula φ is local around y if there is a constant r so that for all graphs $\mathcal{G}$ and all interpretations of $X_1, X_2, \ldots, X_k, x, y$ it can be determined whether $\mathcal{G} \models \phi(X_1, X_2, \ldots, X_k, x, y)$ on the basis of the r-radius neighbourhood of y in $\mathcal{G}$. More specifically, the quantifications in φ are always of the form $\exists z : (\mathrm{dist}(z, y) \leq r \wedge \psi)$ or $\forall z : (\mathrm{dist}(z, y) \leq r \rightarrow \psi)$.

Let us focus on the family $\mathcal{F}$ of connected graphs. If and only if a graph $\mathcal{G} \in \mathcal{F}$ has property ϑ, there are monadic relations $A_1, A_2, \ldots, A_k$ and a node $a \in V$ such that $\mathcal{G} \models \forall y : \varphi(A_1, A_2, \ldots, A_k, a, y)$. 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 $\varphi(A_1, A_2, \ldots, A_k, a, y)$ 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., $\mathcal{P} \in$ LogLCP.

8. ACKNOWLEDGEMENTS

This work was supported in part by the Academy of Finland, Grant 132380, the Finnish Cultural Foundation, and the Research Funds of the University of Helsinki.

9. REFERENCES

[1] Miklos Ajtai and Ronald Fagin. Reachability is harder for directed than for undirected finite graphs. *The Journal of Symbolic Logic*, 55(1):113–150, 1990.

[2] Dana Angluin. Local and global properties in networks of processors. In *Proc. 12th Symposium on Theory of Computing (STOC 1980)*, pages 82–93. ACM Press, 1980.

[3] Lowell W. Beineke. Characterizations of derived graphs. *Journal of Combinatorial Theory*, 9(2):129–135, 1970.

[4] John A. Bondy and Miklós Simonovits. Cycles of even length in graphs. *Journal of Combinatorial Theory, Series B*, 16(2):97–105, 1974.

[5] Reinhard Diestel. *Graph Theory*. Springer, 3rd edition, 2005.

[6] Paul Erdős and Alfréd Rényi. Asymmetric graphs. *Acta Mathematica Hungarica*, 14:295–315, 1963.

[7] Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. Karp, editor, *Complexity of Computation*, volume 7, pages 43–73. 1974.

[8] Pierre Fraigniaud. Distributed computational complexities: are you Volvo-addicted or NASCAR-obsessed? In *Proc. 29th Symposium on Principles of Distributed Computing (PODC 2010)*, pages 171–172. ACM Press, 2010.

[9] Pierre Fraigniaud, Cyril Gavoille, David Ilcinkas, and Andrzej Pelc. Distributed computing with advice: Information sensitivity of graph coloring. In *Proc. 34th International Colloquium on Automata, Languages and Programming (ICALP 2007)*, volume 4596 of *LNCS*, pages 231–242. Springer, 2007.

[10] Pierre Fraigniaud, Amos Korman, and David Peleg. Local distributed decision, 2010. Manuscript, arXiv:1011.2152 [cs.DC].

[11] Cyril Gavoille and David Peleg. Compact and localized distributed data structures. *Distributed Computing*, 16(2–3):111–120, 2003.

[12] Mika Göös and Jukka Suomela. Locally checkable proofs. `http://www.iki.fi/jukka.suomela/lcp`, 2011. Manuscript.

[13] Neil Immerman. *Descriptive Complexity*. Springer, 1999.

[14] Amos Korman and Shay Kutten. On distributed verification. In *Proc. 8th International Conference on Distributed Computing and Networking (ICDCN 2006)*, volume 4308 of *LNCS*, pages 100–114. Springer, 2006.

[15] Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4):253–266, 2007.

[16] Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. In *Proc. 24th Symposium on Principles of Distributed Computing (PODC 2005)*, pages 9–18. ACM Press, 2005.

[17] Amos Korman, David Peleg, and Yoav Rodeh. Constructing labeling schemes through universal matrices. *Algorithmica*, 57(4):641–652, 2010.

[18] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.

[19] David Peleg. *Distributed Computing – A Locality-Sensitive Approach*. SIAM, 2000.

[20] Thomas Schwentick. Graph connectivity and monadic NP. In *Proc. 35th Symposium on Foundations of Computer Science (FOCS 1994)*, pages 614–622. IEEE, 1994.

[21] Thomas Schwentick and Klaus Barthelmann. Local normal forms for first-order logic with applications to games and automata. *Discrete Mathematics and Theoretical Computer Science*, 3:109–124, 1999.

[22] N. J. A. Sloane. The on-line encyclopedia of integer sequences. `http://oeis.org`, 2010.

[23] Jukka Suomela. Survey of local algorithms. `http://www.iki.fi/jukka.suomela/local-survey`, 2011. Manuscript submitted for publication.