Distributed graph problems through an automata-theoretic lens

Yi-Jun Chang · cyijun@nus.edu.sg · National University of Singapore
Jan Studený · jan.studeny@aalto.fi · Aalto University
Jukka Suomela · jukka.suomela@aalto.fi · Aalto University

Abstract. The *locality* of a graph problem is the smallest distance T such that each node can choose its own part of the solution based on its radius-T neighborhood. In many settings, a graph problem can be solved efficiently with a distributed or parallel algorithm if and only if it has a small locality.

In this work we seek to *automate* the study of solvability and locality: given the description of a graph problem Π, we would like to determine if Π is solvable and what is the asymptotic locality of Π as a function of the size of the graph. Put otherwise, we seek to automatically *synthesize* efficient distributed and parallel algorithms for solving Π.

We focus on *locally checkable* graph problems; these are problems in which a solution is globally feasible if it looks feasible in all constant-radius neighborhoods. Prior work on such problems has brought primarily bad news: questions related to locality are undecidable in general, and even if we focus on the case of *labeled* paths and cycles, determining locality is PSPACE-hard (Balliu et al., PODC 2019).

We complement prior negative results with efficient algorithms for the cases of *unlabeled* paths and cycles and, as an extension, for rooted trees. We study locally checkable graph problems from an automata-theoretic perspective by representing a locally checkable problem Π as a *nondeterministic finite automaton* $\mathcal{M}$ over a *unary alphabet*. We identify polynomial-time-computable properties of the automaton $\mathcal{M}$ that near-completely capture the solvability and locality of Π in cycles and paths, with the exception of one specific case that is co-NP-complete.

1 Introduction

In this work, our goal is to *automate* the design of efficient distributed and parallel algorithms for solving graph problems, as far as possible. In the full generality, such tasks are undecidable: for example, given a Turing machine M, we can easily construct a graph problem Π such that there is an efficient distributed algorithm for solving Π if and only if M halts [33]. Nevertheless, we are here to bring good news.

We focus on so-called *locally checkable* graph problems in *paths, cycles, and rooted trees*, and we show that in many cases, the task of designing efficient distributed or parallel algorithms for such problems can be automated, not only in principle but also in practice.

We study the *locality* of graph problems from an *automata-theoretic perspective*. To introduce the concrete research questions that we study, we first define one specific model of distributed computing, the LOCAL model—through this model we can define the fundamental concept of locality. However, as we will later see, our results are directly applicable in many other synchronous models of distributed and parallel computing as well.

1.1 Prior work

Background: locality and round complexity in distributed computing. In classical centralized sequential computing, a particularly successful idea has been the comparison of

deterministic and nondeterministic models of computing. The question of P vs. NP is a prime example: given a problem in which solutions are easy to verify, is it also easy to solve?

In distributed computing a key computational resource is locality, and hence the distributed analogue of this idea can be phrased as follows: given a problem in which solutions can be verified locally, can it also be solved locally?

This question is formalized in the study of so-called *locally checkable labeling* (LCL) problems in the LOCAL model of distributed computing. LCL problems are graph problems in which solutions are *labelings* of nodes and/or edges that can be *verified locally*: if a solution looks feasible in all constant-radius neighborhoods, then it is also globally feasible [33]. A simple example of an LCL problem is a proper 3-coloring of a graph: if a labeling of the nodes looks like a proper 3-coloring in the radius-1 neighborhood of each node, then it is by definition a feasible solution.

In the LOCAL model of computing [31, 34], we assume that the nodes of the input graph are equipped with unique identifiers from $\{1, 2, \ldots, \mathrm{poly}(n)\}$, where n is the number of nodes. A distributed algorithm with time complexity $T(n)$ is then a function that maps the radius-$T(n)$ neighborhood of each node into its local output. The local output of a node is its own part of the solution, e.g., its own color in the graph coloring problem. Here we say that the algorithm has locality $T(n)$; the locality of a problem is the smallest $T(n)$ such that there exists an algorithm for solving it with locality $T(n)$.

If we interpret the input graph as a computer network, with nodes as computers and edges as communication links, then in $T(n)$ synchronous communication rounds all nodes can gather full information about their radius-$T(n)$ neighborhood. Hence time (number of communication rounds) and distance (how far one needs to see) are interchangeable in the LOCAL model. In what follows, we will primarily use the term *round complexity*.

Prior work: the complexity landscape of LCL problems. Now we have a natural distributed analog of the classical P vs. NP question: given an LCL problem, what is its round complexity in the LOCAL model? This is a question that was already introduced by Naor and Stockmeyer in 1995 [33], but the systematic study of the complexity landscape of LCL questions started only very recently, around 2016 [6–8, 11, 12, 15, 16, 23, 25, 26, 37].

By now we have got a relatively complete understanding of *possible complexity classes*: to give a simple example, if we look at deterministic algorithms in the LOCAL model, there are LCL problems with complexity $\Theta(\log^* n)$, and there are also LCL problems with complexity $\Theta(\log n)$, but it can be shown that there is no LCL problem with complexity between $\omega(\log^* n)$ and $o(\log n)$ [11, 15].

However, much less is known about *how to decide the complexity* of a given LCL problem. Many such questions are undecidable in general, and undecidability already holds in relatively simple settings such as LCLs on 2-dimensional grids and tori [12, 33]. We will zoom into graph classes in which no such obstacle exists.

1.2 New results

Our focus: cycles, paths, and rooted trees. Throughout this work, our main focus will be on paths and cycles. This may at first seem highly restrictive, but as we will show in Section 7, once we understand LCL problems in paths and cycles, through reductions we will also gain understanding on so-called edge-checkable problems in rooted trees.

In cycles and paths, there are only three possible round complexities: $O(1)$, $\Theta(\log^* n)$, or $\Theta(n)$ [1]. Randomness does not help in cycles and paths—this is a major difference in comparison with trees, in which there are LCL problems in which randomness helps exponentially [9, 15, 35].

If our input is a *labeled* path or cycle (nodes and edges may be assigned labels from an alphabet of finite size), the round complexity is known to be decidable, but unfortunately it is at

least PSPACE-hard [1]. On the other hand, the round complexity of LCLs on *unlabeled directed cycles* (nodes and edges are unlabeled) has a simple graph-theoretic characterization [12].

However, many questions are left open by prior work, and these are the questions that we will resolve in this work:

- What happens in *undirected* cycles?
- What happens if we study *paths* instead of cycles?
- Can we also characterize the *existence* of a solution for all graphs in a graph class?

To illustrate these questions, consider the following problems that can be expressed as LCLs:

- Π_{2col}: finding a proper 2-coloring,
- Π_{orient}: finding a globally consistent orientation (i.e., an orientation of edges such that it does not contain a node with two incoming or outgoing edges).

The round complexity of Π_{2col} is $\Theta(n)$ both in cycles and paths, regardless of whether they are directed or undirected, while the complexity of Π_{orient} is $\Theta(n)$ in the undirected setting but it becomes $O(1)$ in the directed setting. Problems Π_{2col} and Π_{orient} are always solvable on paths, and Π_{orient} is always solvable on cycles, but if we have an odd cycle, then a solution to Π_{2col} does not exist. In particular, for Π_{2col} there are infinitely many solvable instances and infinitely many unsolvable instances. Our goal in this work is to develop a framework that enables us to make these kind of observations *automatically* for any given LCL problem.

Contributions. We study LCL problems in unlabeled cycles and paths, both with and without consistent orientation. For each of these settings, we show how to answer the following questions in a mechanical manner, for any given LCL problem Π:

- How many unsolvable instances there are (none, finitely, or infinitely many)?
- How many solvable instances there are (none, finitely, or infinitely many)?
- What is the round complexity of Π for solvable instances ($O(1)$, $\Theta(\log^* n)$, or $\Theta(n)$)?

We show that all such questions are not only decidable but they are in NP or co-NP, and almost all such questions are in P, with the exception of a couple of specific questions that are NP-complete or co-NP-complete. We also give a complete classification of all possible case combinations—for example, we show that if there are infinitely many unsolvable instances, then the complexity of the problem for solvable instances cannot be $\Theta(\log^* n)$.

We give a uniform automata-theoretic formalism that enables us to study such questions, and that makes it possible to leverage prior work on automata theory (see Section 1.3). We also develop new efficient algorithms for some automata-theoretic questions that to our knowledge have not been studied before.

Finally, we show that our results can be also used to analyze a family of LCL problems in rooted trees. This demonstrates that the automata-theoretic framework considered here is also applicable beyond the seemingly restrictive case of cycles and paths.

Our main result—the complete classification of the solvability and distributed round complexity of all LCL problems in undirected and directed cycles and paths is presented in Table 1. The classification shows that all questions about the round complexity and solvability of any LCL problem in undirected and directed cycles and paths can be characterized by six properties of the automaton representing the LCL problem, except for the solvability of undirected paths. We will show that all these six properties can be decided in polynomial time, so all these questions about the round complexity and solvability can be answered in polynomial time.

Extensions to other models of distributed and parallel computing. While we use the LOCAL model of distributed computing throughout this work, our results are also directly applicable in many other models of distributed and parallel computing.

Type	A	B	C	D	E	F	G	H	I	J	K
Def. 2.3: symmetric problem	yes	yes	yes	no	yes	yes	no	yes	no	yes	no
Def. 3.1: repeatable state	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	no
Def. 3.2: flexible state [12]	yes	yes	yes	yes	yes	yes	yes	no	no	no	no
Def. 3.3: loop [12]	yes	yes	yes	yes	no	no	no	no	no	no	no
Def. 3.4: mirror-flexible state	yes	yes	no	—	yes	no	—	no	—	no	—
Def. 3.6: mirror-flexible loop	yes	no	no	—	no	no	—	no	—	no	—
Number of instances: 0 = zero < = finite ∞ = infinite ? = NP-complete to decide											
· solvable cycles	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	0
· solvable paths	∞	∞	∞	∞	∞	∞	∞	∞	∞	<	<
· unsolvable cycles	0	0	0	0	<	<	<	∞	∞	∞	∞
· unsolvable paths	<	<	<	<	<	<	<	?	?	∞	∞
Distributed round complexity: □ = $O(1)$ ⊞ = $\Theta(\log^* n)$ ■ = $\Theta(n)$ × = N/A											
· directed cycles [12]	□	□	□	□	⊞	⊞	⊞	■	■	×	×
· directed paths	□	□	□	□	⊞	⊞	⊞	■	■	□	□
· undirected cycles	□	⊞	■	×	⊞	■	×	■	×	×	×
· undirected paths	□	⊞	■	×	⊞	■	×	■	×	□	×

Table 1: Classification of LCL problems in cycles and paths. This table defines 11 types, labeled with A–K, based on six properties of the automaton representing the LCL problem (Definitions 2.3, 3.1–3.6); see Figure 3 for examples of problems of each type. For each problem type, we show what is the number of solvable instances, the number of unsolvable instances, and the distributed round complexity for both directed and undirected paths and cycles. The cases marked with "×" refer to problems that are not well-defined or that are never solvable. For the cases labeled with "?" deciding the number of unsolvable instances is NP-complete (or co-NP-complete depending on the way one defines the decision problem); see Section 4. However, for all other cases the type directly determines both solvability, and all these cases are also decidable in polynomial time; see Section 5. The correctness of this classification is proved in Section 6.

In distributed computing we usually assume that the input graph represents the communication network; each node is a computer, each edge is a communication link, and the nodes can communicate by passing messages to each other. However, in parallel computing we usually take a very different perspective: we assume that the input graph is stored as a linked data structure somewhere in the shared memory, and we have multiple processors that can access the memory. In such a setting, directed paths and rooted trees are particularly relevant families of input, as they correspond to linked lists and tree data structures.

While the settings are superficially different, our *upper bounds* apply directly in all such settings. All of our distributed algorithms are based on the observation that there are two canonical problems: *distance-k anchoring* (Definition 3.9) and *distance-k orientation* (Definition 3.10). Both of the canonical problems can be solved in the message-passing setting with small messages and with little local memory. Furthermore, when we look at rooted trees (Section 7), our algorithms are "one-sided": each node only needs to receive information from its parent. It also follows that our algorithms work e.g. in the CONGEST model [34] of distributed computing, and they can be efficiently simulated e.g. in the classic PRAM model, as well as various modern models of massively parallel computing.

Our *lower bounds* are also broadly applicable, as they hold in the LOCAL model, which is a very strong model of distributed computing (unbounded message size; unlimited local storage; unbounded local computation; nodes can talk to all of their neighbors in parallel). In particular, the lower bounds trivially hold also in the CONGEST model. Adapting the lower bounds to

shared-memory models takes more effort, but it is also possible—see Fich and Ramachandran [22] for an example of how to turn $\Omega(\log^* n)$ lower bounds for the LOCAL model into $\Omega(\log \log^* n)$ lower bounds for variants of the PRAM model.

Comparison with prior work. In comparison with [1, 12, 15, 16, 33], our work gives a more fine-grained perspective: instead of merely discussing decidability, we explore the question of which of the decision problems are in P, NP, and co-NP.

In comparison with the discussion of directed cycles in [12], our work studies a much broader range of settings. Previously, it was not expected that the simple characterization of LCLs on directed cycles could be extended in a straightforward manner to paths or undirected cycles. For example, we can define an infinite family of orientation problems that can be solved in undirected cycles in $O(1)$ rounds but that require a nontrivial algorithm; such problems do not exist in directed cycles, as $O(1)$-round solvability implies trivial 0-round solvability.

Furthermore, we study the graph-theoretic question of the existence of a solution in addition to the algorithmic question of the complexity of finding a solution, and relate solvability with complexity in a systematic manner; we are not aware of prior work that would do the same in the context of LCLs in the LOCAL model.

Our work also takes the first steps towards an effective (i.e., polynomial-time computable) characterization of LCL problems in trees, by showing how to characterize so-called edge-checkable problems in rooted trees.

For general LCL problems on bounded-degree trees, previous work [6, 14, 16] showed that it is decidable to distinguish between the complexity pairs $O(\log n)$ – $n^{\Omega(1)}$ and $O(n^{1/(k+1)})$ – $\Omega(n^{1/k})$ for any constant $k \geq 1$. These algorithms are not efficient, as these are EXPTIME-hard problems [14].

The previous work [1, 6, 14, 16] studying the complexity landscape of LCL problems on paths, cycles, and bounded-degree trees *with input labels* uses a different connection to automata theory. In their proofs, they classified paths and trees into a finite number of classes satisfying certain properties using the *pumping lemma* for regular languages.

To the best of our knowledge, this work gives the first systematic study of the solvability of LCL problems in the LOCAL model. Some other papers [2, 3] also took solvability into consideration in their classification of LCL problems, but solvability is not a main subject in these papers. The solvability of LCL problems is sometimes implicitly ignored [16] in the study of the classification of LCL problems by either only considering LCL problems that are solvable in all instances or by putting all the unsolvable LCL problems into the highest complexity class.

1.3 LCLs as nondeterministic automata over a unary alphabet

In this work we study the solvability and the round complexity of LCL problems from an automata-theoretic perspective. Specifically, we generalize the graph-theoretic characterization for LCL problems on unlabeled directed cycles in [12] to all paths and cycles, directed and undirected, and identify a connection between such a characterization and automata theory.

This connection allows us to leverage prior work on automata theory. For example, as we will later see in this work, the co-NP-completeness of the universality problem for nondeterministic finite automata [39] allows us to deduce the NP-hardness for distinguishing between zero and infinitely many unsolvable instances for LCL problems on paths.

We would like to emphasize that there are many ways to interpret LCLs as automata—and the approach that might seem most natural does not make it possible to directly leverage prior work on automata theory. We will later see that the approach we take enables us to identify direct connections between distributed computational complexity and automata theory.

Let us first briefly describe the "obvious" encoding and show why it does not achieve what we want: A labeling of a directed path with symbols from some alphabet Σ can be interpreted

as a string. Then a locally checkable problem can be interpreted as a regular language over alphabet Σ. We can then represent an LCL problem Π as a finite automaton $\mathcal{M}$ such that $\mathcal{M}$ accepts a string $x \in \Sigma^*$ if and only if a directed path labeled with x is a feasible solution to Π.

However, such an interpretation does *not* seem to lead to a useful theory of LCL problems. To see one challenge, consider these problems on paths:

- Π_{2col}: finding a proper 2-coloring,
- Π_{3col}: finding a proper 3-coloring.

These are fundamentally different problems from the perspective of LCLs in the LOCAL model: problem Π_{2col} requires $\Theta(n)$ rounds while problem Π_{3col} is solvable in $\Theta(\log^* n)$ rounds [19, 31]. However, if we consider analogous automata $\mathcal{M}_{\mathsf{2col}}$ and $\mathcal{M}_{\mathsf{3col}}$ that recognize these solutions, it is not easy to identify a classical automata-theoretic concept that would separate these cases.

Instead of identifying the *alphabet* of the automaton with the set of labels in the LCL, it turns out to be a better idea to have a *unary* alphabet and identify the *set of states* of the automaton with the set of labels. In brief, the perspective that we take throughout this work is as follows (this is a simplified version of the idea):

> Assume Π is an LCL problem in which the set of output labels is Γ. We interpret Π as a *nondeterministic finite automaton* $\mathcal{M}_\Pi$ over the *unary alphabet* $\Sigma = \{o\}$ such that the set of states of $\mathcal{M}_\Pi$ is Γ.

At first this approach may seem counter-intuitive. But as we will see in this work, it enables us to connect classical automata-theoretic concepts to properties of LCLs this way.

To give one nontrivial example, consider the question of whether a given LCL problem Π can be solved in $O(\log^* n)$ rounds. With the above interpretation, this turns out to be directly connected to the existence of *synchronizing words* [13, 21], in the following nondeterministic sense: we say that w is a synchronizing word for an NFA $\mathcal{M}$ that takes $\mathcal{M}$ into state t if, given any starting state $s \in Q$ there is a sequence of state transitions that takes $\mathcal{M}$ to state t when it processes w. Such a sequence w is known as the *D3-directing word* introduced in [30] and further studied in [20, 24, 29, 32]. We will show that the following holds (up to some minor technicalities):

> An LCL on directed paths and cycles has a round complexity of $O(\log^* n)$ if and only if a strongly connected component of the corresponding NFA $\mathcal{M}$ over the unary alphabet has a D3-directing word.

Moreover, we will show that for the unary alphabet, the existence of such a word can be decided in *polynomial time* in the size of the NFA $\mathcal{M}$, or equivalently, in the size of the description of the LCL Π. In contrast, when the size of the alphabet is at least two, the problem of deciding the existence of a D3-directing word is known to be PSPACE-hard [32].

We would like to emphasize that this connection between LCL problems and automata theory is not inherent to unlabeled paths and cycles. For example, *tree automata* can be used to encode LCL problems on bounded-degree trees, and to encode LCL problems with input labels Σ, it suffices to consider automata over the alphabet Σ. Whether such a connection beyond unlabeled paths and cycles can lead to new results is an interesting future work direction.

1.4 Organization

In Section 2, we formally define LCL problems and their representation as automata. In Sections 3 and 4, we present our classification of LCL problems on cycles and paths. In Section 5, we show that the classification is polynomial-time computable. In Section 6, we prove the correctness of our classification. In Section 7, we extend our classification to rooted trees. In Section 8, we give some concluding remark and point to some open problems.

2 Representation of LCLs as automata

To reiterate, LCL problems [33], broadly speaking, are problems in which the task is to label nodes and/or edges with labels from a constant-size alphabet (denoted by Γ), subject to local constraints. That is, a solution is globally feasible if it looks good in all radius-r neighborhoods for some constant r. In this section we will develop a way to represent all LCL problems on paths and cycles as nondeterministic automata.

In this paper, we consider as input graphs only paths and cycles in which the edges are either undirected (undirected case) or consistently oriented (directed case). We say that a cycle or a path has *consistently oriented edges* if it does not contain a node with two incoming or two outgoing edges.

2.1 Formalizing LCLs as node-edge-checkable problems

LCL problems can be specified in many different forms, and we have to be able to capture, among others, problems of the following forms:

- The problem may ask for a labeling of nodes, a labeling of edges, a labeling of the endpoints of the edges, an orientation of the edges, or any combination of these.
- The input graph can be a path or a cycle.
- The input graph may be directed or undirected.

As discussed in the recent papers [3, 4], a rather elegant way to capture all LCL problems is the following approach: We imagine that we have split every edge into two *half-edges*, which are also called *ports*. The labeling refers only to the ports.

More formally, a *port* or a *half-edge* p is a pair (e, v) consisting of an edge e and a node $v \in e$ incident to e. Let P be the set of all ports. A *labeling* is a function $\lambda\colon P \to \Gamma$ from ports to labels from some alphabet Γ.

It is easy to see that we can represent LCL problems of different flavors in this formalism, for example:

- If the task is to label nodes, we require all ports incident to a node to be labeled by the same label, so that the label of a node is well-defined.
- If the task is to label edges, we require that both half-edges of each edge have the same label, so that the label of an edge is well-defined.
- If the task is to find an orientation, we can use e.g. symbols H (head) and T (tail) and require that for each edge exactly one half is labeled with H and the other half is labeled with T, so that the orientation of each edge is well-defined.

Moreover, the constraints for node-edge-checkable problems will be divided into *node constraints* and *edge constraints*. Node constraints consider only incident port labels of a node and edge constraints consider only incident port labels of an edge.

We will now formally define an LCL problem in the node-edge-checkable formalism. Let us first consider the case of directed cycles or paths. By assumption, a directed cycle or a directed path is consistently oriented. For each edge, one port is a *tail port* and the other port is a *head port*. Furthermore, for each degree-2 node, there is also exactly one head port and exactly one tail port incident to it.

Definition 2.1 (LCL problem)**.** An LCL problem Π in the node-edge-checkable formalism on cycles or paths is a tuple $\Pi = (\Gamma, \mathcal{C}_{\mathsf{edge}}, \mathcal{C}_{\mathsf{node}}, \mathcal{C}_{\mathsf{start}}, \mathcal{C}_{\mathsf{end}})$ consisting of

- a finite set Γ of output labels,
- an edge constraint $\mathcal{C}_{\mathsf{edge}} \subseteq \Gamma \times \Gamma$,
- a node constraint $\mathcal{C}_{\mathsf{node}} \subseteq \Gamma \times \Gamma$, and

- start and end constraints $\mathcal{C}_{\mathsf{start}} \subseteq \Gamma$ and $\mathcal{C}_{\mathsf{end}} \subseteq \Gamma$.

Definition 2.2 (Solution on directed cycles or paths)**.** Let G be a directed cycle or a directed path, and let Π be an LCL problem, and let $\lambda\colon P \to \Gamma$ be a labeling of G. We say that λ is a *solution* to Π if the following holds:

- For each edge e, if p is the tail port and q is the head port of e, then $(\lambda(p), \lambda(q)) \in \mathcal{C}_{\mathsf{edge}}$.
- For each degree-2 node v, if p is the head port and q is the tail port of v, then $(\lambda(p), \lambda(q)) \in \mathcal{C}_{\mathsf{node}}$.
- For each degree-1 node v with only one tail port p, we have $\lambda(p) \in \mathcal{C}_{\mathsf{start}}$.
- For each degree-1 node v with only one head port p, we have $\lambda(p) \in \mathcal{C}_{\mathsf{end}}$.

Informally, when we follow the labeling in the positive direction along the directed path, we will first see a label from $\mathcal{C}_{\mathsf{start}}$, then each edge is labeled with a pair from $\mathcal{C}_{\mathsf{edge}}$, each internal node is labeled with a pair from $\mathcal{C}_{\mathsf{node}}$, and the final label along the path is $\mathcal{C}_{\mathsf{end}}$.

Next we consider the case of undirected cycles or paths.

Definition 2.3 (Symmetric LCL problems)**.** We say that an LCL problem $\Pi = (\Gamma, \mathcal{C}_{\mathsf{edge}}, \mathcal{C}_{\mathsf{node}}, \mathcal{C}_{\mathsf{start}}, \mathcal{C}_{\mathsf{end}})$ is *symmetric* if $\mathcal{C}_{\mathsf{edge}}$ and $\mathcal{C}_{\mathsf{node}}$ are symmetric relations and $\mathcal{C}_{\mathsf{start}} = \mathcal{C}_{\mathsf{end}}$. Otherwise the problem is *asymmetric*.

In the undirected case we cannot consistently distinguish ports, and hence we can only solve and define solution for symmetric LCL problems.

Definition 2.4 (Solution on undirected cycles or paths)**.** Let G be an undirected cycle or an undirected path, and let Π be a symmetric LCL problem, and let $\lambda\colon P \to \Gamma$ be a labeling of G. We say that λ is a *solution* to Π if the following holds:

- For each edge e, if the ports of e are p and q, then $(\lambda(p), \lambda(q)) \in \mathcal{C}_{\mathsf{edge}}$.
- For each degree-2 node v, if the ports incident to v are p and q, then $(\lambda(p), \lambda(q)) \in \mathcal{C}_{\mathsf{node}}$.
- For each degree-1 node v, if the port incident to v is p, then $\lambda(p) \in \mathcal{C}_{\mathsf{start}} = \mathcal{C}_{\mathsf{end}}$.

Recall that in symmetric problems $\mathcal{C}_{\mathsf{edge}}$ and $\mathcal{C}_{\mathsf{node}}$ are symmetric, so the above formulation is well-defined. When we study the case of cycles, we can set $\mathcal{C}_{\mathsf{start}} = \mathcal{C}_{\mathsf{end}} = \emptyset$. For brevity, in what follows, we will usually write the pair (a, b) simply as ab.

It is usually fairly easy to encode any given LCL problem in a natural manner in this formalism—see Figure 1 for examples. In the figure, *maximal matching* serves as an example of a problem in which the natural encoding of indicating which edges are part of the matching does not work (it does not capture maximality) but with one additional label we can precisely define a problem that is equivalent to maximal matchings.

In general, if we have any LCL problem Π (in which the problem description can refer to radius-r neighborhoods for some constant r), we can define an equivalent problem Π' that can be represented in the node-edge-checkable formalism, modulo constant-time preprocessing and postprocessing. In brief, one label in the new problem Π' corresponds to the labeling of a subpath of length $\Theta(r)$ in Π. Now given a solution of Π, one can construct a solution of Π' in $O(r)$ rounds, and given a solution of Π', one can construct a solution of Π in zero rounds. Moreover, Π' can be specified in the node-edge-checkable formalism. We will give the details in Section 2.2. From now on, all LCL problems considered are by default problems defined using the node-edge-checkable formalism.

2.2 Universality of the node-edge-checkable formalism

In this section, we show that the node-edge-checkable formalism is universal in the following sense. Let Π be any LCL given in the standard format by listing all valid local neighborhoods of some constant radius r. We can construct an LCL problem Π' that is in the node-edge-checkable formalism satisfying the following two properties. For simplicity, we only consider the undirected case here.

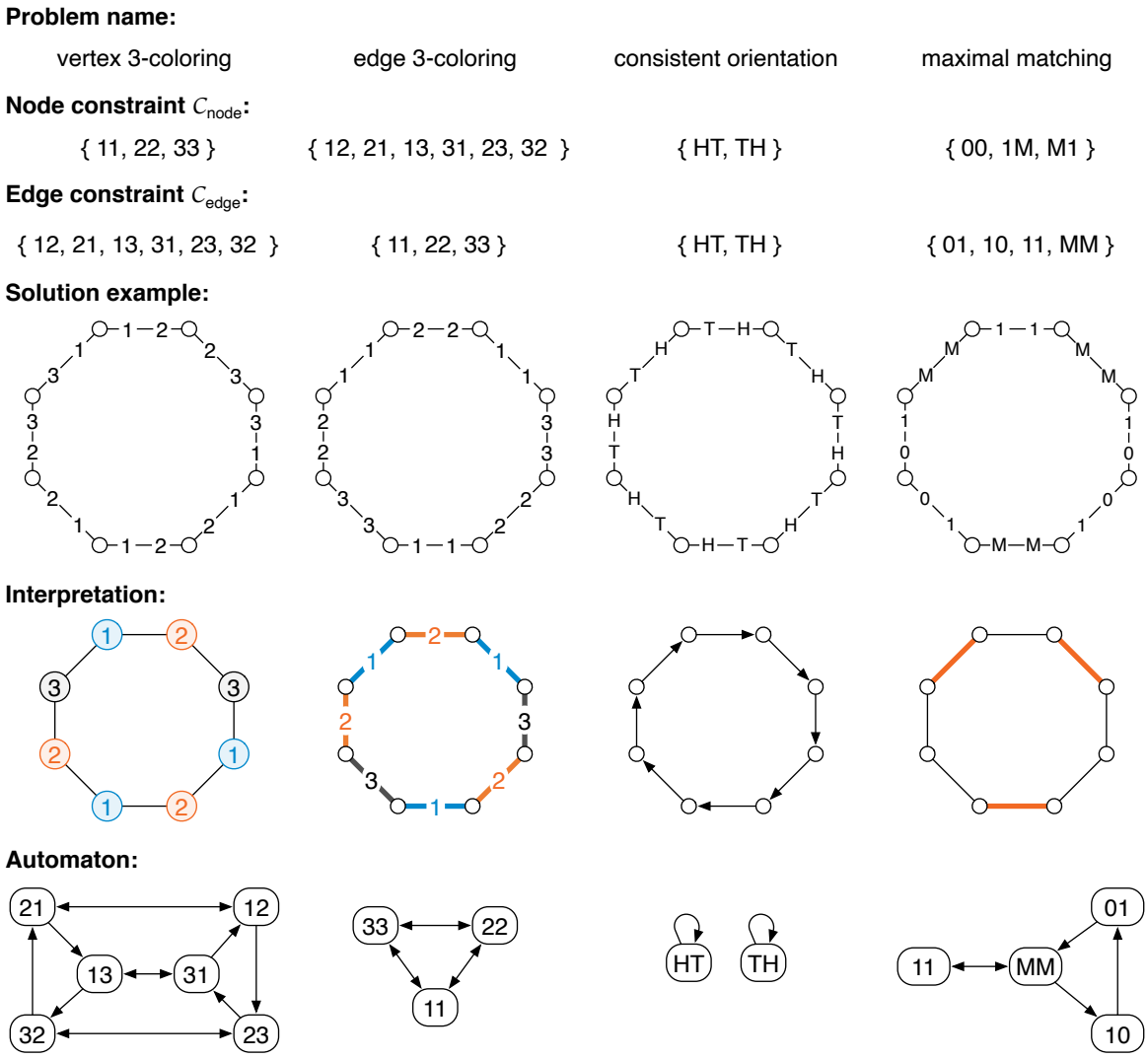

Figure 1: Examples of how to encode LCL problems in the node-edge-checkable formalism, and how to represent the problem as an automaton. Here the problems are symmetric, so they are also well-specified on undirected cycles. For maximal matching, ports incident to matched nodes are labeled with "1" and "M", ports incident to unmatched nodes are labeled with "0", and the edge constraints ensure that there are no unmatched nodes adjacent to each other.

Efficiency: Both the runtime of the construction and the description length of Π' are polynomial in the description length of Π.

Equivalence: Let the communication network G be a cycle of length at least $2r+2$ or a path. Starting from any given legal labeling λ for Π on G, in $O(1)$ rounds we can transform it into a legal labeling λ' for Π'. Similarly, starting from any given legal labeling λ' for Π' on G, in $O(1)$ rounds we can transform it into a legal labeling λ for Π.

In particular, Π and Π' must have the same distributed complexity, since it is trivial to solve any graph problem on constant-size instances in $O(1)$ rounds. Thus, if we have a black-box sequential algorithm $\mathcal{A}$ that decides the optimal distributed complexity for an LCL problem Π' given in the node-edge-checkable formalism, then the same algorithm can be applied to an LCL problem Π given in the standard format. Furthermore, if $\mathcal{A}$ also outputs a description of a distributed algorithm solving Π', then this distributed algorithm can also be applied to solve Π, modulo an $O(1)$-round post-processing step.

The number of solvable and unsolvable instances for Π and Π' are the same for the case of

paths, but they might differ by at most an additive constant for the case of cycles. Suppose we have a black-box sequential algorithm $\mathcal{A}$ that given an LCL problem Π' in the node-edge-checkable formalism, decides

$$(\#\text{solvable instances}, \#\text{unsolvable instances}) \in \{(0,\infty), (\Theta(1),\infty), (\infty,\infty), (\infty,\Theta(1)), (\infty,0)\}.$$

Then obviously the same algorithm can be applied to an LCL problem Π in the standard format for the case of paths.

For solvability on cycles, we can still apply $\mathcal{A}$ to decide the solvability of Π, but things are a little more complicated as the behavior of Π' might be different from Π for cycles of length at most $2r+1$. To deal with this issue, instead of applying $\mathcal{A}$ directly on Π', we apply $\mathcal{A}$ to a modified LCL problem Π^* such that Π^* is unsolvable on cycles of length at most $2r+1$, and its solvability on longer cycles are the same as that of Π'. When the output of $\mathcal{A}$ on Π^* is $(\Theta(1),\infty)$, (∞,∞), or $(\infty,\Theta(1))$, then the same result applies to Π. If the output is $(0,\infty)$ or $(\infty,0)$, we just need to further check in polynomial time the number of solvable and unsolvable instances for cycles of length at most $2r+1$ in order to determine the correct solvability of Π. To construct Π^* from Π', we simply let Π^* be an LCL that is required to solve Π' and another problem $\Pi^\star$ simultaneously, where the $\Pi^\star$ is an arbitrary node-edge-checkable problem that is unsolvable for cycles of length at most $2r+1$, and is solvable for all cycles of length at least $2r+2$.

LCL in standard form. Recall that an LCL problem Π may come in many different forms. It may ask for a labeling of nodes, a labeling of edges, a labeling of half-edges, an orientation of the edges, or any combination of these. The canonical way to specify an LCL with locality radius r is to list all allowed labeled radius-r subgraphs in the set $\mathcal{C}$. An output labeling λ for Π on the instance G is legal if for each node v in G, its radius-r subgraph with the output labeling λ belongs to $\mathcal{C}$.

Description length. From now on, we write $|\Pi|$ to denote the description length of the LCL problem Π. For example, if Π only asks for an edge orientation, then $|\Pi| = 2^{O(r)}$. If Π also asks for an edge labeling from the alphabet Σ_e and a node labeling from the alphabet Σ_v, then $|\Pi| = (|\Sigma_e| + |\Sigma_v|)^{O(r)}$. Note that we only consider paths and cycles, and we assume that $\mathcal{C}$ is described using a truth table mapping each labeled radius-r subgraph to yes/no.

From general labels to half-edge labels. We first observe that labels of all forms can be transformed into half-edge labels, and so from now on we can assume that Π only have half-edge labels. Specifically, if Π asks for an edge labeling from the alphabet Σ_e, a node labeling from the alphabet Σ_v, and also an edge orientation, then we can simply assume that Π asks for a half-edge labeling from the alphabet $\Sigma_e \times \Sigma_v \times \{H,T\}$. That is, each half-edge label is of the form $(a \in \Sigma_e, b \in \Sigma_v, c \in \{H,T\})$. For each edge e, it is required that the Σ_e-part of the two half-edges of e are the same, and this label represents the edge label of e. For each node v, it is required that the Σ_v-part of the two half-edges surrounding v are the same, and this label represents the node label of v. For each edge e, it is required that the $\{H,T\}$-part of the two half-edges of e are different, and this label represents the edge orientation of e. This reduction from a general labeling to a half-edge labeling increases the description length, but only polynomially.

Reducing the locality radius. We assume that Π only asks for a half-edge labeling from the alphabet Γ. We will first show a construction of Π' in the node-edge-checkable formalism satisfying all the needed requirements. In what follows, we assume that the communication network G must not be a cycle of at most $2r+1$ nodes. In particular, this ensures that any radius-r subgraph of G is a path, not a cycle.

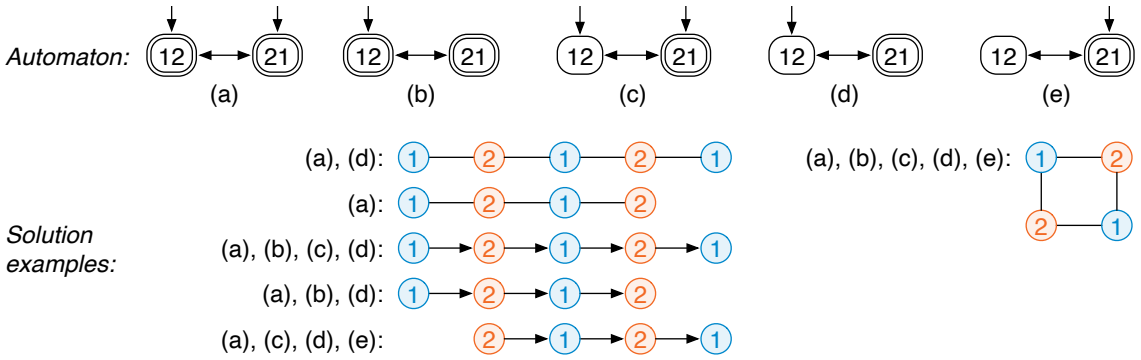

Figure 2: Five variants of the node 2-coloring problem. Starting states have an arrow towards them and accepting states have double border. Each variant has different allowed colors for the endpoints, hence also different starting and accepting states. Here (a) and (d) are the only problems that are symmetric; therefore problems (b), (c), and (e) are not meaningful on undirected paths.

Each radius-r subpath $P = (u_a, \ldots, u_2, u_1, v, w_1, w_2, \ldots, w_b)$ centered at v with half-edge labels from Γ can be represented as a string $S \in (\Gamma \cup \{\bot\})^{4r}$, as follows. The string S is of the form $S = S_1 \circ S_2 \circ S_3 \circ S_4$, where

$$\begin{aligned}
S_1 &= \bot^{2(r-a)},\\
S_2 &= L_{u_a,\{u_a,u_{a-1}\}} L_{u_{a-1},\{u_a,u_{a-1}\}} L_{u_{a-1},\{u_{a-1},u_{a-2}\}} L_{u_{a-2},\{u_{a-1},u_{a-2}\}} \cdots L_{v,\{u_1,v\}},\\
S_3 &= L_{v,\{v,w_1\}} \cdots L_{w_{a-2},\{w_{a-2},w_{a-1}\}} L_{w_{a-1},\{w_{a-2},w_{a-1}\}} L_{w_{a-1},\{w_{a-1},w_a\}} L_{w_a,\{w_{a-1},u_a\}},\\
S_4 &= \bot^{2(r-b)}.
\end{aligned}$$

Here $L_{z,e}$ represents the half-edge label of the edge e at the node z. Note that S_2 represents the half-edge labels within $(u_a, \ldots, u_2, u_1, v)$, and S_3 represents the half-edge labels within $(v, w_1, w_2, \ldots, w_b)$. This string notation is sensitive to the direction of P. If P is reversed, then the resulting string S is also reversed.

If we do not care about cycles of length at most $2r + 1$, then we can simply assume that the set of allowed configurations $\mathcal{C}$ is specified by a set of strings $S \in (\Gamma \cup \{\bot\})^{4r}$ in the above form. Note that $S \in \mathcal{C}$ implies that its reverse S^R is also in $\mathcal{C}$, as we only consider the undirected case here.

Now we are ready to describe the new LCL problem Π'. In this new LCL problem, each half-edge label is a string $S \in \mathcal{C}$. We have the following constraints.

Node constraint: For each node v, let S and S' be the two half-edge labels surrounding v, then S' is the reverse of S. Furthermore, if v is of degree-1, then the length-$2r$ prefix of S must be $\bot^{2r}$.

Edge constraint: For each edge e, let S and S' be the two half-edge labels of e, then the length-$2(r-1)$ suffix of S is the reverse of the length-$2(r-1)$ suffix of S'.

Intuitively, a half-edge label S on the v-side of the edge $e = \{u, v\}$ is intended to represent the radius-r subgraph P centered at v, where the direction of P is chosen as $v \to u$. The above constraints ensure that the half-edge labels must be consistent.

The transformation from a legal labeling λ for Π on G to a legal labeling λ' for Π' on G is straightforward, and it only costs $O(1)$ rounds. The reverse transformation is also straightforward. The description length of Π' is $|\Gamma|^{O(r)}$, which is polynomial in $|\Pi|$.

2.3 Turning node-edge-checkable problems into automata

Now consider an LCL problem Π that is specified in the node-edge-checkable formalism. Construct a nondeterministic finite automaton $\mathcal{M}_\Pi$ as follows; see Figures 1 and 2 for examples.

- The set of states is $\mathcal{C}_{\mathsf{edge}}$.
- There is a transition from (a, b) to (c, d) whenever $(b, c) \in \mathcal{C}_{\mathsf{node}}$.
- $(a, b) \in \mathcal{C}_{\mathsf{edge}}$ is a starting state whenever $a \in \mathcal{C}_{\mathsf{start}}$.
- $(a, b) \in \mathcal{C}_{\mathsf{edge}}$ is an accepting state whenever $b \in \mathcal{C}_{\mathsf{end}}$.

We will interpret $\mathcal{M}_\Pi$ as an NFA over the unary alphabet $\Sigma = \{o\}$. Note that there can be multiple starting states; the automaton can choose the starting state nondeterministically. We remark that in case of cycles, the sets $\mathcal{C}_{\mathsf{start}}$ and $\mathcal{C}_{\mathsf{end}}$ are empty which transforms an NFA into a nondeterministic semiautomaton (i.e., an automation having no starting or accepting states). In the following part we will see how to view the constructed automata.

We define the following concepts:

Definition 2.5 (generating paths and cycles)**.** Automaton $\mathcal{M}$ can *generate* the cycle $(x_1, x_2, \ldots, x_m)$ if each x_i is a state of $\mathcal{M}$, there is a state transition from x_i to x_{i+1} for each $i < m$, and there is a state transition from x_m to x_1.

Automaton $\mathcal{M}$ can *generate* the path $(x_1, x_2, \ldots, x_m)$ if each x_i is a state of $\mathcal{M}$, x_1 is a starting state, x_m is an accepting state, and there is a state transition from x_i to x_{i+1} for each $i < m$.

Note that $\mathcal{M}$ can generate cycles even if there are no starting states or accepting states. We allow $m = 1$ in the above definition.

Example 2.6. Consider the state machines in Figure 1. The state machine for consistent orientation can generate the following cycles:

(HT), (TH), (HT, HT), (TH, TH), (HT, HT, HT), (TH, TH, TH), ...

The state machine for maximal matching can generate the following cycles:

(11, MM), (MM, 11), (10, 01, MM), (01, MM, 10), (MM, 10, 01),
(11, MM, 11, MM), (MM, 11, MM, 11), ...

Remark. If we start with a symmetric problem, the automaton will be *mirror-symmetric* in the following sense: there is a state transition $(a, b) \to (c, d)$ if and only if there is a state transition $(d, c) \to (b, a)$, and the automaton can generate $(x_1 y_1, \ldots, x_m y_m)$ if and only if it can generate $(y_m x_m, \ldots, y_1 x_1)$. All automata in Figure 1 have this property, while in Figure 2 only automata (a) and (d) are mirror-symmetric.

Automata capture node-edge-checkable problems. These observations follow directly from the definitions:

- Let Π be a symmetric or asymmetric problem. Automaton $\mathcal{M}_\Pi$ can generate a cycle $(x_1, x_2, \ldots, x_m)$ if and only if the following is a feasible solution for problem Π: Take a *directed* cycle with m nodes and m edges and walk along the cycle in the positive direction, starting at an arbitrary edge. Label the ports of the first edge with x_1, the ports of the second edge with x_2, etc.
- Let Π be a symmetric problem. Automaton $\mathcal{M}_\Pi$ can generate a cycle $(x_1, x_2, \ldots, x_m)$ if and only if the following is a feasible solution for problem Π: Take an *undirected* cycle with m nodes and m edges and walk the cycle in some consistent direction, starting at an arbitrary edge. Label the ports of the first edge with x_1, the ports of the second edge with x_2, etc.

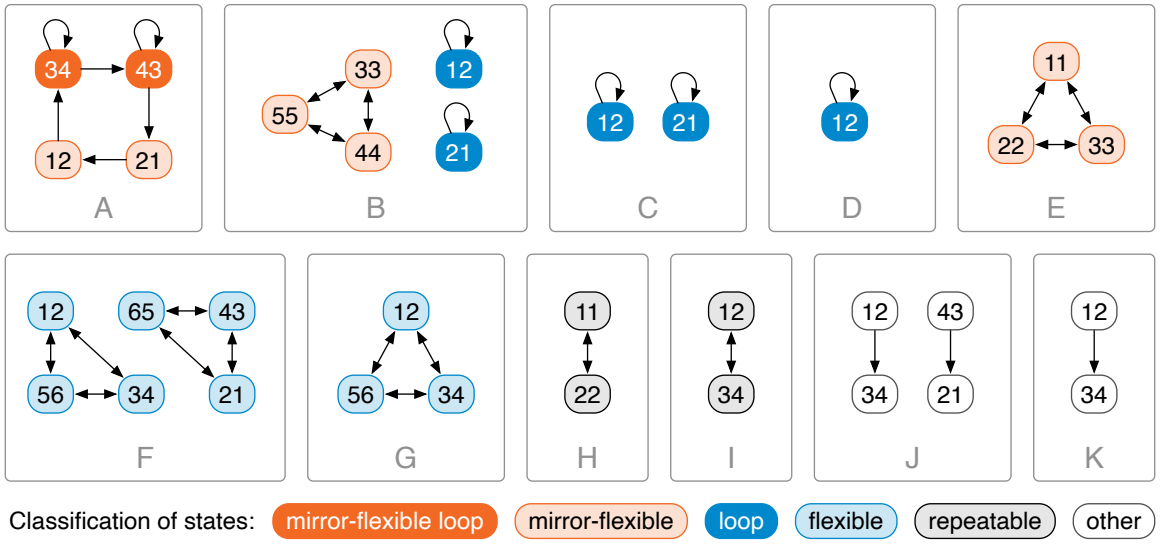

Figure 3: Examples of LCL problems of each type (types A–K in Table 1) represented as automata, together with a classification of their states using Definitions 3.1–3.6. The states are colored only by the most restrictive property. Here is a brief description of each sample problem: **A:** orient the edges so that each consistently oriented fragment consists of at least two edges, one with the label pair 12 and at least one with the label pair 34. **B:** either find a consistent orientation (encoded with labels 1–2) or find a proper 3-coloring of the edges (encoded with labels 3–5). **C:** consistent orientation. **D:** orientation in the positive direction. **E:** edge 3-coloring. **F:** consistent orientation together with an edge 3-coloring. **G:** orientation in the positive direction together with an edge 3-coloring. **H:** edge 2-coloring. **I:** orientation in the positive direction together with an edge 2-coloring. **J–K:** problems only solvable on paths of length at most 2 (assuming appropriate starting and accepting states).

- Let Π be a symmetric or asymmetric problem. Automaton $\mathcal{M}_\Pi$ can generate a path $(x_1, x_2, \ldots, x_m)$ if and only if the following is a feasible solution for problem Π: Take a *directed* path with $m+1$ nodes and m edges and walk along the path in the positive direction, starting with the first edge. Label the ports of the first edge with x_1, the ports of the second edge with x_2, etc.

- Let Π be a symmetric problem. Automaton $\mathcal{M}_\Pi$ can generate a path $(x_1, x_2, \ldots, x_m)$ if and only if the following is a feasible solution for problem Π: Take an *undirected* path with $m+1$ nodes and m edges and walk along the path in some consistent direction, starting with the first edge. Label the ports of the first edge with x_1, the ports of the second edge with x_2, etc.

Hence, for example, the question of whether a given problem Π is solvable in a path of length m is equivalent to the question of whether $\mathcal{M}_\Pi$ accepts the string o^m. Similarly, the question of whether Π is solvable in a cycle of length m is equivalent to the question of whether there is a state q such that $\mathcal{M}_\Pi$ can return to state q after processing o^m.

However, the key question is what can be said about the complexity of solving Π in a distributed setting. As we will see, this is also captured in the structural properties of $\mathcal{M}_\Pi$.

3 Classification of all LCL problems on cycles

We will now discuss our classification of LCL problems on cycles. Consider a problem Π and its corresponding automation $\mathcal{M}_\Pi$. In what follows, if we have two states ab and cd of $\mathcal{M}_\Pi$, a *walk* from ab to cd (denoted by $ab \rightsquigarrow cd$) is a sequence of state transitions starting at state ab and ending at state cd. We introduce the following definitions; see Figure 3 for examples.

Definition 3.1 (repeatable state)**.** State $ab \in \mathcal{C}_{\mathsf{edge}}$ is repeatable if there is a walk $ab \rightsquigarrow ab$ in $\mathcal{M}_\Pi$.

Definition 3.2 (flexible state [12])**.** State $ab \in \mathcal{C}_{\mathsf{edge}}$ is flexible with flexibility K if for all $k \geq K$ there is a walk $ab \rightsquigarrow ab$ of length exactly k in $\mathcal{M}_\Pi$.

Definition 3.3 (loop)**.** State $ab \in \mathcal{C}_{\mathsf{edge}}$ is a loop if there is a state transition $ab \to ab$ in $\mathcal{M}_\Pi$.

Observe that each defined property of a state is a proper strengthening of the previous property (i.e. each loop is a flexible state and each flexible state is a repeatable state).

For a symmetric problem Π we also define:

Definition 3.4 (mirror-flexible state)**.** State $ab \in \mathcal{C}_{\mathsf{edge}}$ is mirror-flexible with flexibility K if for all $k \geq K$ there are walks $ab \rightsquigarrow ab$, $ab \rightsquigarrow ba$, $ba \rightsquigarrow ab$, and $ba \rightsquigarrow ba$ of length exactly k in $\mathcal{M}_\Pi$.

Example 3.5. In Figure 3A, states 12 and 21 are mirror-flexible with flexibility $K = 5$. To see this, note that these are examples of walks of length exactly $k = 5$:

$$12 \to 34 \to 34 \to 43 \to 21 \to 12,$$
$$12 \to 34 \to 34 \to 34 \to 43 \to 21,$$
$$21 \to 12 \to 34 \to 43 \to 21 \to 12,$$
$$21 \to 12 \to 34 \to 34 \to 43 \to 21.$$

By using the self-loop $34 \to 34$ repeatedly, we can also construct walks of lengths exactly $k = 6, 7, \ldots$.

Definition 3.6 (mirror-flexible loop)**.** State $ab \in \mathcal{C}_{\mathsf{edge}}$ is a mirror-flexible loop with flexibility K if ab is a mirror-flexible state with flexibility K and ab is also a loop.

Note that if ab is mirror-flexible loop, then so is ba, as the problem is symmetric.

3.1 Flexibility and synchronizing words

Flexibility is a key concept that we will use in our characterization of LCL problems. We will now connect it to the automata-theoretic concept of synchronizing words.

First, let us make a simple observation that allows us to study automata by their strongly connected components:

Lemma 3.7. *Let $\mathcal{M}'$ be a strongly connected component of automaton $\mathcal{M}_\Pi$, and let q be a state in $\mathcal{M}'$. Then q is flexible in $\mathcal{M}_\Pi$ if and only if q is flexible in $\mathcal{M}'$.*

Proof. A walk from q back to q in $\mathcal{M}_\Pi$ cannot leave $\mathcal{M}'$. □

Recall that a word w is called *D3-directing word* [30] for NFA $\mathcal{M}$ if there is a state t such that starting with any state s of $\mathcal{M}$ there is a sequence of state transitions that takes $\mathcal{M}$ to state t when it processes w. We show that this specific notion of a nondeterministic synchronizing word is, in essence, equivalent to the concept of flexibility:

Lemma 3.8. *Consider a strongly connected component $\mathcal{M}'$ of some automaton $\mathcal{M}_\Pi$. The following statements are equivalent:*

1. *There is a flexible state in $\mathcal{M}'$.*
2. *All states of $\mathcal{M}'$ are flexible.*
3. *There is a D3-directing word for $\mathcal{M}'$.*

Proof. (1) $\Longrightarrow$ (2): Assume that state q has flexibility K. Let x be another state in $\mathcal{M}'$. As it is in the same connected component, there is some r such that we can walk from x to q and back in r steps. Therefore for any $k \geq K$ we can walk from x back to x in $k + r$ steps by following the route $x \rightsquigarrow q \rightsquigarrow q \rightsquigarrow x$. Hence x is a flexible state with flexibility at most $K + r$.

(2) $\Longrightarrow$ (3): Assume that state q has flexibility K, and there is a walk of length at most r from any state x to state q. Then we can walk from any state x to q in exactly $r + K$ steps: first in $r' \leq r$ steps we can reach q and then in $K + r - r' \geq K$ steps we can walk from q back to itself. Hence $w = o^{K+r}$ is a D3-directing word for automaton $\mathcal{M}'$ that takes it from any state to state q.

(3) $\Longrightarrow$ (1): Assume that there is some D3-directing word $w = o^K$ that can take one from any state of $\mathcal{M}'$ to state q in exactly K steps. Then we can also walk from q to itself in k steps for any $k \geq K$: first take $k - K$ steps arbitrarily inside $\mathcal{M}'$, and then walk back to q in exactly K steps. □

Hence, in what follows, we can freely use any of the above perspectives when reasoning about the distributed complexity of LCL problems. Mirror-flexibility can be then seen as a mirror-symmetric extension of D3-directing words.

There is also a natural connection between flexibility and *Markov chains*. Automaton $\mathcal{M}_\Pi$ over the unary alphabet can be viewed as the diagram of a Markov chain for unknown probabilities of the transitions. If we assume that every edge will have a non-zero probability, then a strongly connected component of the automaton is an *irreducible* Markov chain, and in such a component the notion of flexibility coincides with the notion of *aperiodicity*.

3.2 Results

Our main result is the classification presented in Table 1; see also Figure 3 for some examples of problems in each class. What was already well-known by prior work [1, 16] is that there are only three possible complexities: $O(1)$, $\Theta(\log^* n)$, and $\Theta(n)$. However, our work gives the first concise classification of exactly which problems belong to which complexity class. In Section 6 we show that our classification of locally checkable problems on cycles or paths into types A–K, defined by properties of the automaton, is correct and complete.

The entire classification *can be computed efficiently*. In particular, for all of the defined properties (repeatable states, flexible states, loops, mirror-flexible states and mirror-flexible loops) a polynomial-time algorithm can determine if an automaton contains a state with such a property. The non-trivial cases here are flexibility and mirror-flexibility; we present the proofs in Section 5.

3.3 Key building blocks

The role of mirror-flexibility. Consider the following problem that we call *distance-k anchoring*; here the selected edges are called *anchors*:

Definition 3.9. A distance-k anchoring is a maximal subset of edges that splits the cycle in fragments of length at least $k - 1$.

This problem can be solved in $O(\log^* n)$ rounds (e.g. by applying maximal independent set algorithms in the kth power of the line graph of the input graph). Now consider an LCL problem Π that has a flexible state q with flexibility k. It is known by prior work [12] that we can now solve Π on directed cycles in $O(\log^* n)$ rounds, as follows: Solve distance-k anchoring and label the anchor edges with the label pair of state q. As state q is flexible, we can walk along the cycle from one anchor to another, and find a way to fill in the fragment between two anchors with a feasible label sequence.

Mirror-flexibility plays a similar role for undirected cycles: the key difference is that the anchor edges cannot be consistently oriented, and hence we need to be able to also fill a gap between state $q = ab$ and its mirror $q' = ba$, in any order. It is easy to see that mirror-flexibility then implies $O(\log^* n)$-round solvability—what is more surprising is that the converse also holds: $O(\log^* n)$-round solvability necessarily implies the existence of a mirror-flexible state.

A new canonical problem for constant-time solvability. One of the new conceptual contributions of this work is related to the following problem, which we call *distance-k orientation*:

Definition 3.10. A distance-k orientation is an orientation in which each consistently oriented fragment has a length at least k.

The problem is trivial to solve in directed cycles in 0 rounds, but the case of undirected cycles is not equally simple. However, with some thought, one can see that the problem can be solved in $O(1)$ rounds also on undirected cycles [16]. This shows that there are infinite families of nontrivial $O(1)$-time solvable problems, and hence it seems at first challenging to concisely and efficiently characterize all such problems. However, as we will see in Section 6, distance-k orientation can be seen as the *canonical $O(1)$-time solvable problem* on undirected cycles. We show that any problem Π that is $O(1)$-time solvable on undirected cycles has to be of type A (see Table 1), and any such problem can be solved in two steps: first find a distance-k orientation for some constant k that only depends on the structure of $\mathcal{M}_\Pi$, and then map the distance-k orientation to a feasible solution of Π.

We summarize the key new observations related to undirected cycles as follows:

$$\Theta(1) \text{ rounds} \iff \text{mirror-flexible loop} \iff \text{solvable with distance-}k \text{ orientation}$$
$$\Theta(\log^* n) \text{ rounds} \iff \text{mirror-flexible state} \iff \text{solvable with distance-}k \text{ anchoring}$$

4 Classification of all LCL problems on paths

So far we have discussed LCL problems on cycles; let us now have a look at the case of paths. We have already presented the classification for both cases in Table 1. In what follows, we discuss the key new aspects that arise in paths in comparison with the case of cycles.

What is similar: distributed complexity. Broadly speaking, *efficient distributed solvability* on paths is not that different from efficient solvability on cycles. Consider an LCL problem Π and the state machine $\mathcal{M}_\Pi$. As a first, preprocessing step, we have to *remove all states that are not reachable from a starting state, and all states from which there is no path to an accepting state*—such states can never appear in any feasible labeling of a path. The removal of irrelevant states can be done in polynomial time, and hence throughout this work we assume that such states have already been eliminated and, to avoid trivialities, the resulting automaton is nonempty.

Now consider, for example, the case of directed paths. If there is a loop q in $\mathcal{M}_\Pi$, we can solve Π in constant time. By assumption q can be reached from some starting state s and we can reach some accepting state t from q. Hence near the endpoints of a path, we can label according to the walks $s \rightsquigarrow q$ and $q \rightsquigarrow t$, and fill in everything in between with q; the round complexity is simply the maximum of the lengths of the (shortest) walks $s \rightsquigarrow q$ and $q \rightsquigarrow t$. Similarly, if q is not a loop but a flexible state with flexibility k, we can find a distance-k anchoring for the internal part of the path, use q at the anchor points, and fill the gaps just like in the case of a cycle. The case of undirected paths and mirror-flexibility is analogous.

Furthermore, negative results on cycles imply negative results on paths. To see this, consider a hypothetical algorithm $\mathcal{A}$ that solves Π efficiently in directed paths. Then we could also apply

$\mathcal{A}$ to each local neighborhood of a long directed cycle, and hence $\mathcal{A}$ would also solve Π efficiently in directed cycles. If Π cannot be solved in $o(n)$ rounds in directed cycles, it cannot be solved in $o(n)$ rounds in directed paths, either. The same holds for the undirected case. Hence the classification of distributed complexities in Table 1 generalizes to paths almost verbatim.

What is new: solvability. In directed cycles, global problems (i.e., problems of round complexity $\Theta(n)$, types H and I) came in only one possible flavor: there are infinitely many solvable instances and infinitely many unsolvable instances. A simple example is the problem of finding a proper 2-coloring: even cycles are solvable and odd cycles are unsolvable. Our classification for cycles implies that it is not possible to have an LCL problem of complexity $\Theta(n)$ in directed cycles that is always solvable.

This is clearly different in directed paths. As a simple example, 2-coloring a path is a global problem on directed paths that is always solvable. Figure 2 shows both examples of LCLs that are solvable in all paths (e.g. 2-coloring), and examples of LCLs that are solvable in infinitely many paths and unsolvable in infinitely many paths (e.g. 2-coloring in which all endpoints must have color 1). It is also easy to construct problems that are solvable in all but finitely many instances and problems that are solvable only in finitely many instances. However, can we efficiently tell the difference between these cases if we are given a description of an LCL problem?

This is a question in which the automata-theoretic perspective gives direct answers. In essence, the question is rephrased as follows: for which values of k a nondeterministic finite automaton $\mathcal{M}$ accepts the unary string o^k; whether $\mathcal{M}$ accepts all such strings is the classical *universality problem* [28] for unary languages. Prior work directly implies the following (see Section 4.1 for the details):

- **0 vs. $\Theta(1)$ unsolvable instances:** Consider the following decision problem: given an automaton $\mathcal{M}$, answer "yes" if $\mathcal{M}$ accepts all strings, "no" if $\mathcal{M}$ rejects at least one but finitely many strings, and answer "yes" or "no" otherwise. This problem can be solved in polynomial time, as a consequence of Chrobak's theorem [18, 40].
- **0 vs. ∞ unsolvable instances:** Consider the following decision problem: given an automaton $\mathcal{M}$, answer "yes" if $\mathcal{M}$ accepts all strings, "no" if $\mathcal{M}$ rejects infinitely many strings, and answer "yes" or "no" otherwise. This is a well-known co-NP-complete problem [39].

4.1 Complexity of deciding solvability in paths

We show in Theorem 4.1 (see below) that the unary NFA universality problem becomes polynomial-time solvable once we have a promise that $\mathcal{M}$ rejects only finitely many strings. The theorem implies that distinguishing between 0 unsolvable instances and $\Theta(1)$ unsolvable instances is in polynomial time, for both LCLs on paths and on cycles. Although the automaton $\mathcal{M}$ used in the node-edge-checkable formalism has a different acceptance condition than that of the standard NFA, it is straightforward to transform $\mathcal{M}$ into an equivalent NFA with the standard NFA acceptance condition (i.e., there is one starting state $q_0 \in Q$, and a set of accepting states F).

Theorem 4.1. *There is a polynomial-time algorithm $\mathcal{A}$ that achieves the following for any given unary NFA $\mathcal{M}$. If $\mathcal{M}$ does not reject any string, then the output of $\mathcal{A}$ is Yes. If $\mathcal{M}$ rejects at least one but only finitely many strings, then the output of $\mathcal{A}$ is No. If $\mathcal{M}$ rejects infinitely many strings, the output of $\mathcal{A}$ can be either No or Yes.*

Proof. This is an immediate consequence of Chrobak's theorem [18, 40], which shows that any unary NFA $\mathcal{M}$ is equivalent to some NFA $\mathcal{M}'$ in the Chrobak normal form, and the number of states in $\mathcal{M}'$ is at most $|Q|^2$. An NFA $\mathcal{M}'$ is in Chrobak normal form if it can be constructed

as follows. Start with a directed path $P = (q_0 \to q_1 \to \cdots \to q_m)$ and k directed cycles $C_i = (r_{0,i} \to r_{1,i} \to \cdots \to r_{\ell_i,i} \to r_{0,i})$, for each $i \in \{1, \ldots, k\}$, where ℓ_i is the length of C_i. Add a transition from q_m to $r_{0,i}$ for each $i \in \{1, \ldots, k\}$. The starting state is q_0. The set of accepting states F can be arbitrary.

The algorithm $\mathcal{A}$ works as follows. It tests whether $\mathcal{M}$ accepts all strings of length at most $|Q|^2$. If so, then the output is Yes; otherwise, the output is No. To see the correctness, we only need to show that whenever $\mathcal{M}$ rejects at least one but only finitely many strings, then the output of $\mathcal{A}$ is No. To show this, it suffices to prove that if there is a string w of length higher than $|Q|^2$ that is rejected by $\mathcal{M}$, then there must be infinitely many strings rejected by $\mathcal{M}$.

Let L be the length of w. Now consider some NFA $\mathcal{M}'$ that is in the Chrobak normal form and is equivalent to $\mathcal{M}$. We can assume that the number of states in $\mathcal{M}'$ is at most $|Q|^2 < L$. Define S to be the set of states that is reachable from q_0 in $\mathcal{M}'$ via a walk of length exactly L. Since the number of states in $\mathcal{M}'$ is smaller than the length L of w, the set S contains exactly one state from each cycle C_i. Since w is rejected, all states in S are not accepting states. It is clear that for any non-negative integer k, the set of states that is reachable from q_0 in $\mathcal{M}'$ via a walk of length exactly $L + k \prod_{1 \le i \le k} \ell_i$ is also S. Hence $\mathcal{M}'$ (and also $\mathcal{M}$) rejects infinitely many strings. □

Theorem 4.2 (see below) is a well-known result of co-NP-completeness of testing universality of a unary NFA. To see that the same hardness result applies to the analogous question of solvability of LCLs on paths, given any NFA $\mathcal{M}$, we construct a finite state machine $\mathcal{M}^*$ representing an LCL in the node-edge-checkable formalism such that $\mathcal{M}$ is equivalent to $\mathcal{M}^*$. For each transition $a \to b$ in $\mathcal{M}$, add the state (a, b) to $\mathcal{M}^*$. Add a transition $(a, b) \to (c, d)$ in $\mathcal{M}^*$ if $b = c$. Each state (a, b) with $a = q_0$, where q_0 is the starting state of $\mathcal{M}$, is designated as a starting state of $\mathcal{M}^*$. Each state (a, b) with $b \in F$, where F is the set of accepting states of $\mathcal{M}$, is designated as an accepting state of $\mathcal{M}^*$. Now the new finite state machine $\mathcal{M}^*$ represents an LCL on paths. Note that this reduction only works for LCL on paths—the same solvability problem on cycles can be solved in polynomial time.

Theorem 4.2 (Stockmeyer and Meyer [39])**.** *Given a unary NFA $\mathcal{M}$, the following problem is* NP*-hard. If $\mathcal{M}$ rejects zero strings, then the output is required to be No. If $\mathcal{M}$ rejects at least one but only finitely many strings, then the output can be either No or Yes. If $\mathcal{M}$ rejects infinitely many strings, the output is required to be Yes.*

5 Efficient computation of the classification of LCL problems

In view of Table 1, the task to classify for an LCL problem Π to which class it belongs to can be reduced to testing certain graph properties of $\mathcal{M}_\Pi$. In this section, we show that checking whether a state q is flexible or mirror-flexible can be done in polynomial time, and so deciding the optimal distributed complexity of an LCL problem Π is also in polynomial time.

Definition 5.1. Let Q be the set of states of $\mathcal{M}$. For each $q \in Q$ we define:

- L_q is the set of values ℓ such that there is a walk $q \rightsquigarrow q$ of length ℓ in $\mathcal{M}$.
- $L'_q = \{\ell \in L_q : \ell \le 2|Q| - 1\}$ is the restriction of L_q to walks of length at most $2|Q| - 1$.

Lemma 5.2. *For any automaton $\mathcal{M}$ and for any state u, we have* $\gcd(L_u) = \gcd(L'_u)$.

Proof. We show that for each $\ell \in L_u \setminus L'_u$, we can find $\ell_1, \ell_2, \ell_3 \in L_u$ such that $\ell_1, \ell_2, \ell_3 < \ell$ and $\ell = x\ell_1 + y\ell_2 + z\ell_3$ for some integers x, y, z. By applying this argument recursively to each ℓ_i, we can eventually write any $\ell \in L_u$ as a linear combination of sufficiently small numbers $\ell' \in L'_u$. Hence if all values in L'_q are multiples of some d, all values in L_q have to be also multiples of d.

Therefore it suffices to show that for each walk w of the form $u \rightsquigarrow u$ of length $\ell > 2|Q| - 1$, it is possible to find shorter returning walks w_1, w_2, w_3 of the form $u \rightsquigarrow u$ of lengths $\ell_1, \ell_2, \ell_3 < \ell$ such that $\ell = x\ell_1 + y\ell_2 + z\ell_3$ for some integers x, y, z.

We write $w = (v_1, v_2, \ldots, v_\ell, v_{\ell+1})$, where $v_1 = v_{\ell+1} = u$. Since this vector has $\ell + 1 \geq 2|Q| + 1$ elements, by the pigeonhole principle, there exists a state v that appears at least three times. Therefore, w can be decomposed into four walks: $p_1 = (v_1, \ldots, v_i)$, $p_2 = (v_i, \ldots, v_j)$, $p_3 = (v_j, \ldots, v_k)$, and $p_4 = (v_k, \ldots, v_{\ell+1})$, where $v_i = v_j = v_k$ and $1 \leq i < j < k \leq \ell + 1$. We write L_i to denote the length of p_i.

Now define $w_1 = p_1 \circ p_4$, $w_2 = p_1 \circ p_2 \circ p_4$, and $w_3 = p_1 \circ p_3 \circ p_4$; the lengths of these paths are $\ell_1 = L_1 + L_4$, $\ell_2 = L_1 + L_2 + L_4$, and $\ell_3 = L_1 + L_3 + L_4$. Now the length ℓ of w can be expressed as $\ell = -w_1 + w_2 + w_3$. Since $L_2 = j - i \geq 1$ and $L_3 = k - j \geq 1$, the three lengths ℓ_1, ℓ_2, ℓ_3 are all smaller than ℓ, as required. □

Lemma 5.3. *A state q is flexible if and only if* $\gcd(L_q) = 1$.

Proof. If $\gcd(L_q) = x > 1$, then $kx + 1 \notin L_q$ and hence there is no walk $q \rightsquigarrow q$ of length $kx + 1$ for any k, and q cannot be flexible.

For the other direction, given a set of positive integers S with $\gcd(S) = 1$, the *Frobenius number* $g(S)$ of the set S is the largest number x such that x cannot be expressed as a linear combination of S, where each coefficient is a non-negative integer. It is known that $g(S) < \max(S)^2$ [38].

By Lemma 5.2, $\gcd(L_q) = \gcd(L'_q)$ and $\max(L'_q) \leq 2|Q| - 1$. Hence $\gcd(L_q) = 1$ implies that for all $k \geq (2|Q| - 1)^2$, it is possible to find a length-k walk $q \rightsquigarrow q$ by combining some returning walks of length at most $2|Q| - 1$, and so q is flexible. □

We remark that the problem of calculating the Frobenius number when the input numbers can be encoded in binary is NP-hard [36]. However, the flexibility of a given automaton can be nevertheless found efficiently.

Lemma 5.4. *Testing whether a state $q \in Q$ is flexible and finding its flexibility number is solvable in polynomial time.*

Proof. By Lemma 5.3, it is sufficient to test if $\gcd(L_u) = 1$, and by Lemma 5.2, it suffices to find the set L'_u and compute its $\gcd(L'_u)$, which can be done in polynomial time. □

Lemma 5.5. *Testing whether a state $q \in Q$ is mirror-flexible and finding its mirror-flexibility number is solvable in polynomial time.*

Proof. Follows from Lemma 5.4: $q \in Q$ is mirror-flexible if and only if q is flexible and is reachable to its mirror q' and q can be reached back from q'. Reachability between two states can be tested in polynomial time. □

Theorem 5.6. *Given an* LCL *problem* Π*, classifying its type can be computed in polynomial time.*

Proof. The non-trivial cases are captured in Lemmas 5.4 and 5.5. □

An immediate corollary of Lemma 5.4 is that the existence of a D3-directing word in an NFA over the unary alphabet can be decided in polynomial time, since a unary NFA $\mathcal{M}$ has a D3-directing word if and only if there exists a strongly connected component S such that all states in $\mathcal{M}$ are reachable to S and the subgraph induced by S admits a flexible state. For comparison, when the alphabet size is at least two, the same problem is known to be PSPACE-hard [32].

Corollary 5.7. *Given an NFA $\mathcal{M}$ over the unary alphabet, the existence of a D3-directing word can be decided in polynomial time.*

Type:		**A**	**B**	**C/D**	**E**	**F/G**	**H/I**	**J/K**
Number of instances:								
· solvable cycles	Theorem:	6.13	6.13	6.13	6.13	6.13	6.13	6.16
· solvable paths	Theorem:	6.13	6.13	6.13	6.13	6.13	6.13	6.18
· unsolvable cycles	Theorem:	6.15	6.15	6.15	6.14	6.14	6.17	6.18
· unsolvable paths	Theorem:	6.14	6.14	6.14	6.14	6.14	—	6.18
Round complexity for directed graphs:								
· lower bound	Theorem:	triv.	triv.	triv.	6.1	6.1	6.5	triv.
· upper bound	Theorem:	6.9	6.9	6.9	6.11	6.11	triv.	6.8
Round complexity for undirected graphs:								
· lower bound	Theorem:	triv.	6.6	6.7	6.2	6.7	6.5	triv.
· upper bound	Theorem:	6.10	6.12	triv.	6.12	triv.	triv.	6.8

Table 2: Connection from problem types to proofs establishing their correctness—cf. Table 1.

6 Correctness of the classification of LCL problems

In this section, we show that the classification of LCL problems in Table 1 is correct and complete. We first prove the round complexity of each type and then the solvability. The connection between the proofs and the results they establish is depicted in Table 2.

6.1 Round complexity lower bounds

In all proofs in this section, we need a technical assumption that $\mathcal{M}_\Pi$ contains a repeatable state. This ensures that for every number N, we can find an n-node solvable instance G for some $n \geq N$. This assumption is necessary: If $\mathcal{M}_\Pi$ does not contain a repeatable state, then we can find a number N such that for all $n \geq N$ the problem Π has no solution on a cycle or path of n nodes, and so the round complexity of Π is trivially $O(1)$ in all solvable instances.

Theorem 6.1. *Let Π be an LCL problem on directed cycles or paths. Suppose that the automaton $\mathcal{M}_\Pi$ contains a repeatable state, but it does not contain a loop. Then the round complexity Π is $\Omega(\log^* n)$.*

Proof. We show how to turn any legal labeling λ of Π into an edge 3-coloring in a constant number of rounds. As 3-coloring of edges requires $\Omega(\log^* n)$ rounds [31], so does Π.

Let Q be the set of states of $\mathcal{M}_\Pi$, and consider a valid solution λ of Π. Such a labeling can be easily turned into an edge $|Q|$-coloring f: an edge that was labeled with the pair (a, b) in λ will be colored with the color (a, b) in f. As there are no loops in $\mathcal{M}_\Pi$, adjacent edges must have different label pairs and hence different colors. Finally, we can reduce the number of colors from $|Q|$ to 3 in a constant number of rounds (w.r.t. to n) with the trivial algorithm [27] that eliminates colors one at a time. □

Theorem 6.2. *Let Π be an LCL problem on undirected cycles or paths. Suppose that the automaton $\mathcal{M}_\Pi$ contains a repeatable state, but it does not contain a loop. Then the round complexity Π is $\Omega(\log^* n)$.*

Proof. We use an idea similar to Theorem 6.1, with one extra ingredient. Assume that λ is a feasible solution of Π. First construct a labeling of the edges with (at most) $|Q|$ colors as follows: an edge that was labeled with the pair (a, b) in λ will be colored with the color $\{a, b\}$ in f (note that the colors are now unordered pairs).

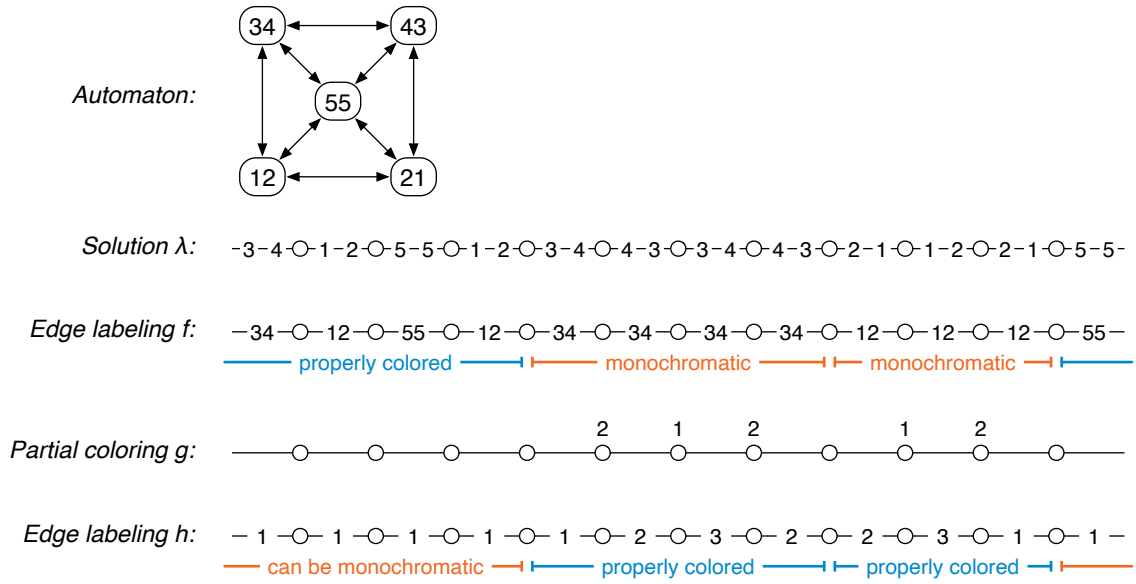

Figure 4: An illustration of the proof of Theorem 6.2. Pairs $(f(e), h(e))$ form a proper edge coloring.

Now such a labeling f is not necessarily a proper coloring. There may be an arbitrarily long sequence of edges that have the same label $\{a, b\}$, for some $a < b$; such a path is called *monochromatic*. However, this would arise only if λ contains a sequence of the form $ab, ba, ab, ba, \ldots$. Within such a path, we can find a partial labeling of the nodes g as follows: nodes that have both ports labeled with a are colored with 1, and nodes that have both ports labeled with b are colored with 2; all other nodes are left uncolored. See Figure 4 for an illustration.

Now we have two ingredients: a not-necessarily-proper edge coloring f with $|Q|$ colors, and a partial node coloring g with 2 colors. These complement each other: all internal nodes in monochromatic paths of f are properly 2-colored in g. Hence we can use g to find a proper edge 3-coloring h of each monochromatic path, e.g. as follows: Nodes of color 1 are active and send proposals to adjacent nodes of color 2 (proposals are sent in the order of unique identifiers), nodes of color 2 accept the first proposal that they get (breaking ties with unique identifiers), and this way we can find a maximal matching within each monochromatic path. Each such matching forms one color class in h; we delete the edges that are colored and repeat. After three such iterations all internal edges of monochromatic paths are properly colored in h; then h is easy to extend so that also the edges near the endpoints of monochromatic paths have colors different from their monochromatic neighbors (monochromatic paths of length two are also easy to 3-color). Now the pairs $(f(e), h(e))$ form a proper edge coloring with $3|Q|$ colors, and we can finally reduce the number of colors down to 3. □

In both of the following lemmas to be applicable also to the case of a path, we always assume that the "witness" of any specific behavior happens somewhere in the middle of a cycle or a path and not next to the endpoints.

Lemma 6.3. *Let Π be an* LCL *problem that is solvable in cycles or paths of length n for infinitely many values of n. Assume that $\mathcal{A}$ solves Π for all solvable instances, and assume that for arbitrarily large values of n, we can find a cycle or a path of length n such that there are two edges e_1 and e_2 with the following properties:*

- *The distance between e_1 and e_2, and the distance between each e_i and the nearest degree-1 node (if any) is more than $n/10$.*
- *Algorithm $\mathcal{A}$ labels both e_1 and e_2 with the same state q that is not flexible.*

Then the round complexity of $\mathcal{A}$ has to be $\Omega(n)$.

Proof. We give the proof for the case of a path; the case of a cycle is similar. To reach a contradiction, assume the complexity of $\mathcal{A}$ is sublinear. Pick a sufficiently large n such that the algorithm runs in $r \ll n/20$ rounds and paths of length n are solvable. Decompose the path G in fragments

$$G = (P_0, N_1, P_1, N_2, x, P_2),$$

where N_i is the radius-r neighborhood of e_i, each P_i is a path of nodes, and x is one node. Now we can move one node to construct another path

$$G' = (P_0, N_1, P_1, x, N_2, P_2).$$

Path G' has the same length as G, and hence G' is also a solvable instance and $\mathcal{A}$ has to be able to find a feasible solution. As the radius-r neighborhoods of e_1 and e_2 are the same in G and G', algorithm $\mathcal{A}$ will label them with q in both G and G'. But as q is not flexible, we can this way eventually construct an instance in which the distance between the two edges with label q is k such that $\mathcal{M}_\Pi$ does not have a walk of length k from q back to itself, and hence $\mathcal{A}$ cannot produce a valid solution. □

Lemma 6.4. *Let Π be a symmetric* LCL *problem that is solvable in undirected cycles or paths of length n for infinitely many values of n. Assume that $\mathcal{A}$ solves Π for all solvable instances, and assume that for arbitrarily large values of n, we can find a cycle or a path of length n such that there is an edge e_1 with the following properties:*

- *The distance between e_1 and the nearest degree-1 node (if any) is more than $n/10$.*
- *Algorithm $\mathcal{A}$ labels e_1 with a state q_1 that is not mirror-flexible.*

Then the round complexity of $\mathcal{A}$ has to be $\Omega(n)$.

Proof. We give the proof for the case of a path; the case of a cycle is similar. To reach a contradiction, assume the complexity of $\mathcal{A}$ is sublinear. Pick a sufficiently large n such that the algorithm runs in $r \ll n/20$ rounds and paths of length n are solvable. For the purposes of this proof, orient the path so that the distance between e_1 and the end of the path is at least $n/2$. Let e_2 be an edge between e_1 and the end of the path such that the distance between e_1 and e_2, and the distance between e_2 and the endpoint is at least $n/10$. Decompose the path G in fragments

$$G = (P_0, N_1, P_1, N_2, P_2),$$

where N_i is the radius-r neighborhood of e_i, and each P_i is a path of nodes. Let $\bar{N}_1$ be the mirror image of path X, i.e., the same nodes in the opposite direction; then $\mathcal{A}$ will label the midpoint of $\bar{N}_1$ with q_1', the mirrored version of state q_1. Construct the following paths:

$$G_1 = (P_0, N_2, P_1, N_1, P_2),$$
$$G_2 = (P_0, \bar{N}_1, P_1, N_2, P_2),$$
$$G_3 = (P_0, N_2, P_1, \bar{N}_1, P_2).$$

Now all such paths have length n, and hence they are also solvable and $\mathcal{A}$ is expected to produce a feasible solution. Such a solution in G gives a walk $q_1 \rightsquigarrow q_2$ in $\mathcal{M}_\Pi$, G_1 gives a walk $q_2 \rightsquigarrow q_1$, G_2 gives a walk $q_1' \rightsquigarrow q_2$, and G_3 gives a walk $q_2 \rightsquigarrow q_1'$. Putting these together, we can construct walks $q_1 \rightsquigarrow q_1$, $q_1 \rightsquigarrow q_1'$, $q_1' \rightsquigarrow q_1$, and $q_1' \rightsquigarrow q_1'$.

Finally, we can move nodes one by one from P_2 to P_1 in each of G, G_1, G_2, G_3 to construct such walks of any sufficiently large length. It follows that q_1 is mirror-flexible, which is a contradiction. □

Theorem 6.5. *Let Π be an* LCL *problem. Suppose that the automaton $\mathcal{M}_\Pi$ contains a repeatable state, but it does not contain a flexible state. Then the round complexity Π is $\Omega(n)$.*

Proof. We can apply Lemma 6.3: the algorithm can only use non-flexible states, and it has to use some non-flexible state repeatedly. □

Theorem 6.6. *Let* Π *be a symmetric* LCL *problem on undirected cycles or paths. Suppose that* $\mathcal{M}_\Pi$ *contains a repeatable state, but it does not contain a mirror-flexible loop. Then the round complexity of* Π *is* $\Omega(\log^* n)$*.*

Proof. Consider an algorithm $\mathcal{A}$ that solves Π, and look at the behavior of $\mathcal{A}$ in sufficiently large instances, far away from the endpoints of the paths (if any). There are two cases:

1. Algorithm $\mathcal{A}$ sometimes outputs a loop state (which by assumption cannot be mirror-flexible). Then by Lemma 6.4 we obtain a lower bound of $\Omega(n)$.

2. Otherwise $\mathcal{A}$ essentially solves the restriction of Π where loop states are not allowed (except near the endpoints of the path), and we can use Theorem 6.2 to obtain a lower bound of $\Omega(\log^* n)$. □

Theorem 6.7. *Let* Π *be a symmetric* LCL *problem on undirected cycles or paths. Suppose that* $\mathcal{M}_\Pi$ *contains a repeatable state, but it does not have a mirror-flexible state. Then the round complexity of* Π *is* $\Omega(n)$*.*

Proof. Again consider an algorithm $\mathcal{A}$ that solves Π, and look at the behavior of $\mathcal{A}$ in sufficiently large instances, far away from the endpoints of the paths (if any). There are two cases:

1. Algorithm $\mathcal{A}$ sometimes outputs a flexible state (which by assumption cannot be mirror-flexible). Then by Lemma 6.4 we obtain a lower bound of $\Omega(n)$.

2. Otherwise $\mathcal{A}$ essentially solves the restriction of Π where flexible states are not allowed (except near the endpoints of the path), therefore it is using some non-flexible state repeatedly far from endpoints, and Lemma 6.3 applies. □

6.2 Round complexity upper bounds

Let us first consider the trivial case of automata without repeating states.

Theorem 6.8. *Let* Π *be an* LCL *problem. Suppose* $\mathcal{M}_\Pi$ *does not have repeatable state. Then* Π *can be solved in constant time in solvable instances.*

Proof. Let Q be a set of states of $\mathcal{M}_\Pi$. As $\mathcal{M}_\Pi$ does not have a repeatable state, it is not solvable in any cycle, and it is only solvable in some paths of length at most $|Q|$. Hence Π can be solved in constant time by brute force (and also in constant time all nodes can detect if the given instance is solvable. □

In the rest of this section, we design efficient algorithms for solving problems with flexible or mirror-flexible states. We present the algorithms first for the case of a cycle. The case of a path is then easy to solve: we can first label the path as if it was a cycle, remove the labels near the endpoints (up to distance k, where k is bounded by the (mirror-)flexibility of a chosen (mirror-)flexible state plus the number of states in $\mathcal{M}_\Pi$), and fill constant-length path fragments near the endpoints by brute force. We refer to this process as *fixing the ends*.

Theorem 6.9. *Let* Π *be an* LCL *problem on directed cycles or paths. Suppose* $\mathcal{M}_\Pi$ *has a loop. Then the round complexity* Π *is* $O(1)$*.*

Proof. All edges can be labeled by a loop state. In a path we will then fix the ends. □

Theorem 6.10. *Let* Π *be an* LCL *problem. Suppose* $\mathcal{M}_\Pi$ *has a mirror-flexible loop. Then the round complexity* Π *is* $O(1)$*.*

Proof. Let q be a mirror-flexible loop state of mirror-flexibility k. Let $K \geq k+2$ be an even constant. The first step is to construct a distance-K orientation (Definition 3.10); this can be done in $O(1)$ rounds.

We say that an edge e is a *boundary edge* if there is another edge e' with a different orientation within distance less than $K/2$ from e; otherwise e is an *internal edge*. Note that each consistently oriented fragment contains at least one internal edge.

The internal edges are labeled as follows: each edge with orientation "$\rightarrow$" is assigned label q, and each edge with orientation "$\leftarrow$" is assigned label q', i.e., the mirror of q.

We are left with gaps of length $K-2 \geq k$ between the labeled edges. As q is mirror-flexible, we can find paths $q \rightsquigarrow q'$ and $q' \rightsquigarrow q$ of length $K-2$ to fill in such gaps. Finally, in a path we will fix the ends. □

Theorem 6.11. *Let Π be an* LCL *problem on directed cycles or paths. Suppose $\mathcal{M}_\Pi$ has a flexible state. Round complexity of such Π is $O(\log^* n)$.*

Proof. Let q be a flexible state of flexibility k. This time we first construct a distance-k anchoring (Definition 3.9); this can be done in $O(\log^* n)$ rounds. Let the set of anchors be I. If an edge is in I, we label its ports by q. We are left with the gaps, which can be of size between $k-1$ and $2k$ (anchoring is maximal). As q is flexible, for each gap of size $g \geq k-1$ we can find a returning walk of length exactly $g+1 \geq k$ and fill it by the states along such walk. Finally, in a path we will fix the ends. □

Theorem 6.12. *Let Π be an* LCL *problem on undirected cycles or paths. Suppose $\mathcal{M}_\Pi$ has a mirror-flexible state. Round complexity of such Π is $O(\log^* n)$.*

Proof. The proof is very similar to a previous proof, only with some minor changes as now we are in the undirected setting.

Let q be a mirror-flexible state of flexibility k. First, we construct a distance-k anchoring (Definition 3.9); this can be done in $O(\log^* n)$ rounds. Let the set of anchors be I. If an edge is in I, we label its ports by either q or its mirror q' arbitrarily (breaking symmetry with unique identifiers). We are left with the gaps, which can be of size between $k-1$ and $2k$ (anchoring is maximal). As q is mirror-flexible, for each gap of size $g \geq k-1$ we can find a returning walk of length exactly $g+1 \geq k$ and fill the gap no matter the combinations of anchors ($q \rightsquigarrow q'$, $q \rightsquigarrow q$, $q \rightsquigarrow q'$ or $q' \rightsquigarrow q'$). Finally, in a path we will fix the ends. □

6.3 Solvability

In this part, we consider the *solvability* of an LCL problem. That is, for a given graph class $\mathcal{G}$ (the set of all cycles of every length or the set of paths of every length), how many graphs $G \in \mathcal{G}$ are solvable instances (instances that admit a legal labeling) with respect to the given LCL problem Π.

Theorem 6.13. *Let Π be an* LCL *problem. If $\mathcal{M}_\Pi$ has a repeatable state, then the number of solvable instances is ∞.*

Proof. Let q be a repeatable state, i.e., there is a walk $q \rightsquigarrow q$ of some length ℓ. Now for every $k \in \mathbb{N}$, cycles of length $k\ell$ are solvable, as we can generate cycles of the form $q \rightsquigarrow q \rightsquigarrow \cdots$.

In paths, by assumption q is reachable from some starting state s and we can reach some accepting state t from q; let h be the length of a walk $s \rightsquigarrow q \rightsquigarrow t$. Now for every $k \in \mathbb{N}$, paths of length $h + k\ell$ are solvable, as we can generate paths of the form $s \rightsquigarrow q \rightsquigarrow q \rightsquigarrow \cdots \rightsquigarrow q \rightsquigarrow t$. □

Theorem 6.14. *Let Π be an* LCL *problem. If $\mathcal{M}_\Pi$ has a flexible state, number of unsolvable instances is at most C, where C is a constant.*

Proof. Let q be a flexible state with flexibility k. All cycles of length $n \geq k$ are now trivially solvable, as we have a walk $q \rightsquigarrow q$ of length n.

In paths, by assumption q is reachable from some starting state s and we can reach some accepting state t from q; let h be the length of a walk $s \rightsquigarrow q \rightsquigarrow t$. Now all paths of length $n \geq h + k$ are solvable, as we have a walk $s \rightsquigarrow q \rightsquigarrow q \rightsquigarrow t$ of length n. □

Theorem 6.15. *Let Π be an* LCL *problem on cycles. If $\mathcal{M}_\Pi$ has a loop the number of unsolvable instances is zero.*

Proof. As $\mathcal{M}_\Pi$ has a loop, returning walks of all lengths exists and all cycles can be labeled. □

Theorem 6.16. *Let Π be an* LCL *problem on cycles. If $\mathcal{M}_\Pi$ has does not have a repeatable state the number of solvable instances is zero.*

Proof. Any legal labeling on cycles has to contain a repeatable state. □

Theorem 6.17. *Let Π be an* LCL *problem. Assume $\mathcal{M}_\Pi$ does not have any flexible state. Then there are infinitely many unsolvable instances on cycles.*

Proof. Let Q be the set of states of $\mathcal{M}_\Pi$. Since no state is flexible in $\mathcal{M}_\Pi$, by Lemma 5.3 we have $\gcd(L_q) > 1$ for all $q \in Q$. Pick

$$b = \prod_{q \in Q} \gcd(L_q).$$

Now $kb + 1 \notin L_q$ for any $q \in Q$ and any natural number k. Therefore it is not possible to use any state q in a cycle of length $kb + 1$, as a feasible solution in such a cycle would form a walk $q \rightsquigarrow q$ of length $kb + 1$. Hence there are infinitely many unsolvable instances. □

Theorem 6.18. *Let Π be an* LCL *problem. Suppose $\mathcal{M}_\Pi$ does not have repeatable state. Then there are at most constantly many solvable instances.*

Proof. Let Q be a set of states of $\mathcal{M}_\Pi$. As $\mathcal{M}_\Pi$ does not have a repeatable state, all walks that can form legal labeling have to have to have length at most $|Q|$. So all paths of lengths $n > |Q|$ are unsolvable instances. □

7 Extension to rooted trees

In the previous sections, we have presented a complete classification of all LCL problems on paths and cycles. Now we will demonstrate how to leverage these results (in particular, the classification of LCLs on directed paths) to also classify a family of LCL problems on *rooted trees*.

There is one obstacle to keep in mind: if the LCL problem can refer to e.g. the degrees of the nodes, then we can use the structure of the tree to encode input labels; and then the setting is at least as general as LCL problems on labeled paths, which implies that questions on locality are at least PSPACE-hard [1]. Hence to have efficient classification algorithms for rooted trees we need to choose a restricted family of LCL problems. We will here use *edge-checkable* problems as an example—as we will see, it is a broad enough family of problems to capture many interesting problems but restricted enough that we can still classify all such problems.

Edge-checkable LCL problems. An *edge-checkable* LCL problem Π on rooted trees consists of a finite set Γ of output labels and a set of constraints $\mathcal{C}_{\mathsf{edge}} \subseteq \Gamma \times \Gamma$ specifying the set of allowed ordered pairs of labels $(\lambda(u), \lambda(v))$ on the two endpoints u and v of each edge, where u is the parent of v. For example, the vertex k-coloring problem is a (symmetric) edge-checkable LCL problem with $\Gamma = \{1, 2, \ldots, k\}$ and $\mathcal{C}_{\mathsf{edge}} = \{(a, b) \in \Gamma \times \Gamma \mid a \neq b\}$.

Any edge-checkable LCL problem Π can be alternatively described in our formalism: $\Pi = (\Gamma, \mathcal{C}_{\mathsf{edge}}, \mathcal{C}_{\mathsf{node}}, \mathcal{C}_{\mathsf{start}}, \mathcal{C}_{\mathsf{end}})$, where $\mathcal{C}_{\mathsf{node}} = \{(a, a) \mid a \in \Gamma\}$ and $\mathcal{C}_{\mathsf{start}} = \mathcal{C}_{\mathsf{end}} = \Gamma$. Hence Π can also be seen as an LCL problem on directed paths—in essence, Π describes what are feasible label sequences when one follows any path from a leaf to the root. We claim that Π has the same asymptotic round complexity on both rooted trees and directed paths, and hence our classification of LCL problems on directed paths also applies to edge-checkable LCL problems on rooted trees:

Theorem 7.1. *Let Π be any edge-checkable* LCL *problem on rooted trees. Then Π has the same asymptotic round complexity on both rooted trees and directed paths.*

Proof. A directed path is a special case of a rooted tree, and hence lower bounds on directed paths automatically apply to rooted trees. The non-trivial part of the proof is to transform any given algorithm $\mathcal{A}$ for Π on directed paths to an algorithm $\mathcal{A}'$ for Π on rooted trees with the same asymptotic round complexity.

Let the node with only one tail port be the first node in a directed path. We can assume that $\mathcal{A}$ is *one-sided* in the sense that the output label of v only depends on v and the nodes that precedes v in the directed path. To achieve that, we just need to shift the output labels by T nodes, where T is the runtime of $\mathcal{A}$, and then assign the output labels of the first T nodes of the directed path locally.

Given that $\mathcal{A}$ is one-sided, the algorithm $\mathcal{A}'$ for Π on rooted trees applies $\mathcal{A}$ to each root-to-leaf path simultaneously. Because $\mathcal{A}$ is one-sided, the output label of each node v only depends on v and its ancestors, and hence such a simultaneous execution is possible. The correctness of $\mathcal{A}'$ follows from the correctness of $\mathcal{A}$. □

Canonical algorithms for rooted trees. Recall that any $O(\log^* n)$-round LCL problem Π on directed paths can be solved in a canonical way as follows. Let q be any flexible state with flexibility k. We find a distance-k anchoring for the directed path, use q at the anchor points, and fill the output labels in the gaps.

The proof of Theorem 7.1 implicitly implies that any $O(\log^* n)$-round edge-checkable LCL problem Π on rooted trees can be solved in a canonical way analogously based on the following variant of distance-k anchoring.

Definition 7.2. A distance-k anchoring for a rooted tree T is a maximal subset of edges that splits the rooted tree into subtrees $T_1, T_2, \ldots, T_s$ satisfying the following conditions.

- The height of T_i is at least $k - 1$.
- If v is a leaf in T_i and a non-leaf in T, then the distance between the root of T_i and v equals the height of T_i.

Such a distance-k anchoring for a rooted tree T can be computed in $O(\log^* n)$ rounds for $k = O(1)$ as follows. Apply any one-sided $O(\log^* n)$-round algorithm for computing a distance-k anchoring for directed paths on each root-to-leaf path of T, shift down the output labels by $O(k)$ nodes, and finally split the large (possibly unbalanced) subtree near the root as appropriate.

8 Discussion

We have seen that questions about the solvability of LCLs in paths are closely related to classical automata-theoretic questions, as we can directly interpret a path as a string. Our work on LCLs

in cycles can be then seen as an extension of classical questions to *cyclic words*. In particular, we see that an automaton "accepts" all but finitely many cyclic words if and only if there is a flexible state in the automaton, or equivalently if a D3-directing word exists for a strongly connected component of the automaton. Our work shows that all such questions on cyclic words can be decided in polynomial time, even if their classical non-cyclic analogs are in some cases co-NP-complete.

As we saw in Section 7, our approach can be extended to the study of LCLs beyond unlabeled paths and cycles. There are two main open questions after our work:

1. What is the largest family of LCL problems in rooted trees for which round complexity can be decided in polynomial time? We now know that edge-checkable LCL problems can be characterized efficiently, while the general case is PSPACE-hard.

2. Is the round complexity of all LCL problems on rooted trees decidable? What about unrooted trees? For LCL problems on bounded-degree trees, we know that it is decidable to distinguish between the complexity pairs $O(\log n)$ – $n^{\Omega(1)}$ and $O(n^{1/(k+1)})$ – $\Omega(n^{1/k})$ for any constant $k \geq 1$ [6, 14, 16], but the general question for deciding the round complexity of LCL problems on trees is still widely open.

Recent follow-up work. Subsequent to this work, the round complexity of LCL problems in the case of rooted and unrooted *regular* trees was studied in [2, 5, 10], and we now know that the complexity of LCL problems in regular rooted trees is decidable. The above two questions still remain open in general. In particular, the decidability in unrooted regular trees is still an open question.

Acknowledgments

This work is an extended and revised version of a preliminary conference report [17] that appeared in the 28th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2021).

We would like to thank Alkida Balliu, Sebastian Brandt, Laurent Feuilloley, Juho Hirvonen, Yannic Maus, Dennis Olivetti, Aleksandr Tereshchenko, Jara Uitto, and all participants of the Helsinki February Workshop 2018 on Theory of Distributed Computing for discussions related to the decidability of LCLs on trees. We would also like to thank the anonymous reviewers of previous versions of this works for their helpful comments and feedback.

Yi-Jun Chang was supported by Dr. Max Rössler, by the Walter Haefner Foundation, and by the ETH Zürich Foundation.

References

[1] Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. The distributed complexity of locally checkable problems on paths is decidable. In *Proc. 38th ACM Symposium on Principles of Distributed Computing (PODC 2019)*, pages 262–271. ACM Press, 2019. doi:10.1145/3293611.3331606.

[2] Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Jan Studený, and Jukka Suomela. Efficient classification of locally checkable problems in regular trees. In *Proc. 36th International Symposium on Distributed Computing (DISC 2022)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 22:1–22:19. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.DISC.2022.22.

[3] Alkida Balliu, Sebastian Brandt, Yuval Efron, Juho Hirvonen, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Classification of distributed binary labeling problems. In *Proc. 34th International Symposium on Distributed Computing (DISC 2020)*, 2020. doi:10.4230/LIPIcs.DISC.2020.17.

[4] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2019)*, pages 481–497. IEEE, 2019. doi:10.1109/FOCS.2019.00037.

[5] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, Jan Studený, Jukka Suomela, and Aleksandr Tereshchenko. Locally checkable problems in rooted trees. In *Proc. 40th ACM Symposium on Principles of Distributed Computing (PODC 2021)*, pages 263–272. ACM Press, 2021. doi:10.1145/3465084.3467934.

[6] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. Almost global problems in the LOCAL model. In *Proc. 32nd International Symposium on Distributed Computing (DISC 2018)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 9:1–9:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.DISC.2018.9.

[7] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. How much does randomness help with locally checkable problems? In *Proc. 39th Symposium on Principles of Distributed Computing (PODC 2020)*, pages 299–308, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3382734.3405715.

[8] Alkida Balliu, Juho Hirvonen, Janne H Korhonen, Tuomo Lempiäinen, Dennis Olivetti, and Jukka Suomela. New classes of distributed time complexity. In *Proc. 50th ACM Symposium on Theory of Computing (STOC 2018)*, pages 1307–1318. ACM Press, 2018. doi:10.1145/3188745.3188860.

[9] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *Journal of ACM*, 63(3), 2016. Article 20.

[10] Sebastian Brandt, Yi-Jun Chang, Jan Grebík, Christoph Grunau, Václav Rozhoň, and Zoltán Vidnyánszky. Local problems on trees from the perspectives of distributed algorithms, finitary factors, and descriptive combinatorics. In *Proc. 13th Innovations in Theoretical Computer Science Conference, (ITCS 2022)*, volume 215 of *LIPIcs*, pages 29:1–29:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.29.

[11] Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A lower bound for the distributed Lovász local lemma. In *Proc. 48th ACM Symposium on Theory of Computing (STOC 2016)*, pages 479–488. ACM Press, 2016. doi:10.1145/2897518.2897570.

[12] Sebastian Brandt, Juho Hirvonen, Janne H Korhonen, Tuomo Lempiäinen, Patric R J Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela, and Przemysław Uznański. LCL problems on grids. In *Proc. 36th ACM Symposium on Principles of Distributed Computing (PODC 2017)*, pages 101–110. ACM Press, 2017. doi:10.1145/3087801.3087833.

[13] Ján Černý. Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fyzikálny časopis*, 14(3):208–216, 1964. URL: http://dml.cz/dmlcz/126647.

[14] Yi-Jun Chang. The complexity landscape of distributed locally checkable problems on trees. In *Proc. 34th International Symposium on Distributed Computing (DISC 2020)*, volume 179, pages 18:1–18:17, 2020. doi:10.4230/LIPIcs.DISC.2020.18.

[15] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the local model. *SIAM Journal on Computing*, 48(1):122–143, 2019. doi:10.1137/17M1117537.

[16] Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the LOCAL model. *SIAM Journal on Computing*, 48(1):33–69, 2019. doi:10.1137/17M1157957.

[17] Yi-Jun Chang, Jan Studenỳ, and Jukka Suomela. Distributed graph problems through an automata-theoretic lens. In *Proc. 28th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2021)*, pages 31–49. Springer, 2021.

[18] Marek Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47:149–158, 1986. doi:10.1016/0304-3975(86)90142-8.

[19] Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986. doi:10.1016/S0019-9958(86)80023-7.

[20] Henk Don and Hans Zantema. Synchronizing non-deterministic finite automata. *Journal of Automata, Languages and Combinatorics*, 23(4):307–328, 2018.

[21] David Eppstein. Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19(3):500–510, 1990. doi:10.1137/0219033.

[22] Faith E. Fich and Vijaya Ramachandran. Lower bounds for parallel computation on linked structures. In *Proc. 2nd Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 1990)*, pages 109–116, New York, NY, USA, 1990. ACM Press. doi:10.1145/97444.97676.

[23] Manuela Fischer and Mohsen Ghaffari. Sublogarithmic distributed algorithms for Lovász local lemma, and the complexity hierarchy. In *Proc. 31st International Symposium on Distributed Computing (DISC 2017)*, pages 18:1–18:16, 2017. doi:10.4230/LIPIcs.DISC.2017.18.

[24] Zsolt Gazdag, Szabolcs Iván, and Judit Nagy-György. Improved upper bounds on synchronizing nondeterministic automata. *Information Processing Letters*, 109(17):986–990, 2009. doi:10.1016/j.ipl.2009.05.007.

[25] Mohsen Ghaffari, David G Harris, and Fabian Kuhn. On derandomizing local distributed algorithms. In *Proc. 59th IEEE Symposium on Foundations of Computer Science (FOCS 2018)*, pages 662–673, 2018. doi:10.1109/FOCS.2018.00069.

[26] Mohsen Ghaffari and Hsin-Hao Su. Distributed degree splitting, edge coloring, and orientations. In *Proc. 28th ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 2505–2523. Society for Industrial and Applied Mathematics, 2017. doi:10.1137/1.9781611974782.166.

[27] Andrew V Goldberg, Serge A Plotkin, and Gregory E Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434, 1988.

[28] Markus Holzer and Martin Kutrib. Descriptional and computational complexity of finite automata—A survey. *Information and Computation*, 209(3):456–470, 2011. doi:10.1016/j.ic.2010.11.013.

[29] Balázs Imreh and Masami Ito. On regular languages determined by nondeterministic directable automata. *Acta Cybernetica*, 17(1):1–10, 2005.

[30] Balázs Imreh and Magnus Steinby. Directable nondeterministic automata. *Acta Cybernetica*, 14(1):105–115, 1999.

[31] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

[32] Pavel Martyugin. Computational complexity of certain problems related to carefully synchronizing words for partial automata and directing words for nondeterministic automata. *Theory of Computing Systems*, 54(2):293–304, 2014. doi:10.1007/s00224-013-9516-6.

[33] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

[34] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000. doi:10.1137/1.9780898719772.

[35] Seth Pettie and Hsin-Hao Su. Distributed algorithms for coloring triangle-free graphs. *Information and Computation*, 243:263–280, 2015.

[36] J. L. Ramírez-Alfonsín. Complexity of the Frobenius problem. *Combinatorica*, 16(1):143–147, 1996. doi:10.1007/BF01300131.

[37] Václav Rozhoň and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proc. 52nd Annual ACM Symposium on Theory of Computing (STOC 2020)*, 2020. doi:10.1145/3357713.3384298.

[38] Jeffrey Shallit. The Frobenius problem and its generalizations. In *Proc. 12th International Conference on Developments in Language Theory (DLT 2008)*, volume 5257 of *LNCS*, pages 72–83, Berlin, Heidelberg, 2008. Springer. doi:10.1007/978-3-540-85780-8_5.

[39] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proc. 5th Annual ACM Symposium on Theory of Computing (STOC 1973)*, pages 1–9, New York, New York, USA, 1973. ACM Press. doi:10.1145/800125.804029.

[40] Anthony Widjaja To. Unary finite automata vs. arithmetic progressions. *Information Processing Letters*, 109(17):1010–1014, 2009. doi:10.1016/j.ipl.2009.06.005.