The Distributed Complexity of Locally Checkable Problems on Paths is Decidable

Alkida Balliu
alkida.balliu@aalto.fi
Aalto University

Sebastian Brandt
brandts@ethz.ch
ETH Zurich

Yi-Jun Chang
cyijun@umich.edu
University of Michigan

Dennis Olivetti
dennis.olivetti@aalto.fi
Aalto University

Mikaël Rabie
mikael.rabie@irif.fr
Aalto University and
IRIF, University of Paris

Jukka Suomela
jukka.suomela@aalto.fi
Aalto University

ABSTRACT

Consider a computer network that consists of a path with n nodes. The nodes are labeled with inputs from a constant-sized set, and the task is to find output labels from a constant-sized set subject to some local constraints—more formally, we have an LCL (locally checkable labeling) problem. How many communication rounds are needed (in the standard LOCAL model of computing) to solve this problem?

It is well known that the answer is always either $O(1)$ rounds, or $\Theta(\log^* n)$ rounds, or $\Theta(n)$ rounds. In this work we show that this question is *decidable* (albeit PSPACE-hard): we present an algorithm that, given any LCL problem defined on a path, outputs the distributed computational complexity of this problem and the corresponding asymptotically optimal algorithm.

CCS CONCEPTS

• **Theory of computation → Distributed computing models**; *Complexity classes*; *Distributed algorithms.*

KEYWORDS

distributed computing, complexity, decidability

ACM Reference Format:
Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. 2019. The Distributed Complexity of Locally Checkable Problems on Paths is Decidable. In *2019 ACM Symposium on Principles of Distributed Computing (PODC '19), July 29-August 2, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3293611.3331606

1 INTRODUCTION

To what extent is it possible to *automate* the design of algorithms and the study of computational complexity? While algorithm synthesis problems are typically undecidable, there are areas of theoretical computer science in which we can make use of computational techniques in algorithm design—at least in principle, and sometimes also in practice. One such area is the theory of *distributed computing*; see [4, 6, 8, 10, 11, 17, 19, 26] for examples of recent success stories. In this work we bring yet another piece of good news:

> Consider this setting: there is a computer network that consists of a path with n nodes, the nodes are labeled with inputs from a constant-sized set, and the task is to find output labels from a constant-sized set subject to some local constraints. We show that for any given set of local constraints, *it is decidable* to tell what is the asymptotically optimal number of communication rounds needed to solve this problem (as a function of n, for the worst-case input).

Background: **LCL***s and the* **LOCAL** *Model.* We focus on what are known as LCL (*locally checkable labeling*) problems [22] in the LOCAL model of distributed computing [20, 24]. We define the setting formally in Section 2, but in essence we look at the following question:

- We are given an unknown input graph of maximum degree $\Delta = O(1)$; the nodes are labeled with *input labels* from a constant-size set Σ_{in}, and the nodes also have unique identifiers from a polynomially-sized set.
- The task is to label the nodes with *output labels* from a constant-size set Σ_{out}, subject to some *local* constraints $\mathcal{P}$; a labeling is globally feasible if it is locally feasible in all radius-r neighborhoods for some $r = O(1)$.
- Each node has to produce its own output label based on the information that it sees in its own radius-$T(n)$ neighborhoods for some function T.

Here the local constraints $\mathcal{P}$ define an LCL problem. The rule that the nodes apply to determine their output labels is called a *distributed algorithm* in the LOCAL model, and function $T(n)$ is the *running time* of the algorithm—here $T(n)$ determines how *far* a node has to see in order to choose its own part of the solution, or equivalently, how many *communication rounds* are needed for each node to gather the relevant information if we view the input graph as a communication network.

In this setting, the case of $T(n) = \Theta(n)$ is trivial, as all nodes can see the entire input. The key question is to determine which problems $\mathcal{P}$ can be solved in sublinear time—here are some examples:

PODC '19, July 29-August 2, 2019, Toronto, ON, Canada

ACM ISBN 978-1-4503-6217-7/19/07…$15.00
https://doi.org/10.1145/3293611.3331606

- Vertex coloring with $\Delta + 1$ colors: can be solved in time $O(\log^* n)$ [9, 16] and this is tight [20, 21].
- Vertex coloring with Δ colors, for $\Delta > 2$: can be solved in polylogarithmic time [23] and requires at least logarithmic time [7] for deterministic algorithms.

While the study of this setting was initiated already in the seminal work by Naor and Stockmeyer in 1995 [22], our understanding of these questions has rapidly advanced in the past three years [2, 3, 5, 7, 8, 12–15, 25]. The big surprises have been these:

- There are LCL problems with infinitely many different time complexities—for example, we can construct LCL problems with a time complexity exactly $\Theta(n^\alpha)$ for any rational number $0 < \alpha \leq 1$.
- Nevertheless, there are also wide gaps in the complexity landscape: for example, no LCL problem has a (deterministic) computational complexity that is between $\omega(\log^* n)$ and $o(\log n)$.

However, what is perhaps most relevant for us is the following observation: if we look at the case of $\Delta = 2$ (paths and cycles), then the time complexity of any LCL problem is either $O(1)$, $\Theta(\log^* n)$, or $\Theta(n)$, and the same holds for both deterministic and randomized algorithms [6, 8, 22].

Decidability of **LCL** *Time Complexities.* For a fixed Δ, any LCL problem has a trivial finite representation: simply enumerate all feasible radius-r local neighborhoods. Hence it makes sense to ask whether, given an LCL problem, it is possible to determine its time complexity. The following results are known by prior work:

- If the input graph is an *unlabeled path or cycle*, the time complexity is decidable [6, 22].
- If the input graph is a *grid or toroidal grid*, the time complexity is undecidable [22]. However, there are also some good news: in unlabeled toroidal grids, the time complexity falls in one of the classes $O(1)$, $\Theta(\log^* n)$, or $\Theta(n)$, it is trivial to tell if the time complexity is $O(1)$, and it is semi-decidable to tell if it is $\Theta(\log^* n)$ [6].
- In the case of trees, there are infinitely many different time complexities, but there is a gap between $\omega(\log n)$ and $n^{o(1)}$, and it is decidable to tell on which side of the gap a given problem lies [8].

Somewhat surprisingly, the seemingly simple case of *labeled paths or cycles* has remained open all the way since the 1995 paper by Naor and Stockmeyer [22], which defined LCLs with inputs but analyzed decidability questions only in the case of unlabeled graphs.

We initially expected that the question of paths with input labels is a mere technicality and the interesting open questions are related to much broader graph families, such as rooted trees, trees, and bounded-treewidth graphs. However, it turned out that the *main obstacle for understanding decidability in any such graph family seems to lie in the fact that the structure of the graph can be used to encode arbitrary input labels*, hence it is necessary to first understand how the input labels influence decidability—and it turns out that this makes all the difference in the case of paths.

In this work we show that the time complexity of a given LCL problem on *labeled paths or cycles* is decidable. However, we also show that decidability is far from trivial: the problem is PSPACE-hard, as LCL problems on labeled paths are expressive enough to capture linear bounded automata (Turing machines with bounded tapes).

Full Version. Due to space constraints, this version is missing some proofs and other details. The full version can be found here [1].

2 MODEL

The **LOCAL** *Model.* The model of computation we consider in this work is the LOCAL model of distributed computing [20, 24]. In the LOCAL model, each node of the input graph is considered as a computational entity that can communicate with the neighboring nodes in order to solve some given graph problem. Computation is divided into synchronous rounds, where in each round each node first sends messages of arbitrary size to its neighbors, then receives the messages sent by its neighbors, and finally performs some local computation of arbitrary complexity. Each node is equipped with a globally unique identifier (ID) which is simply a bit string of length $O(\log n)$, where n denotes the number of nodes of the input graph. In the beginning of the computation, each node is aware of its own ID, the number of nodes and the maximum degree Δ of the input graph, and potentially some additional problem-specific input. Each node has to decide at some point that it terminates, upon which it returns a local output and does not take part in any further computation; the problem is solved correctly if the local outputs of all nodes together constitute a global output that satisfies the output constraints of the given problem.

Each node executes the same algorithm; the running time of the distributed algorithm is the number of rounds until the last node terminates. It is well known that, due to the unbounded message sizes, an algorithm with runtime $T(n)$ can be equivalently described as a function from the set of all possible radius-$T(n)$ neighborhoods to the set of allowed outputs. In other words, we can assume that in a $T(n)$-round algorithm, each node first gathers the topology of and the input labels contained in its radius-$T(n)$ neighborhood, and then decides on its output based solely on the collected information.

Locally Checkable Labelings. The class of problems we consider is *locally checkable labeling* (LCL) problems [22]. LCL problems are defined on graphs of bounded degree, i.e., we will assume that $\Delta = O(1)$. Formally, an LCL problem is given by a finite input label set Σ_{in}, a finite output label set Σ_{out}, an integer r, and a finite set C of graphs where every node is labeled with a pair $(\ell_{\text{in}}, \ell_{\text{out}}) \in \Sigma_{\text{in}} \times \Sigma_{\text{out}}$ and one node is marked (as the center). Each node of the input graph is assigned an input label from Σ_{in} before the computation begins, and the global output of a distributed algorithm is correct if the radius-r neighborhood of each node v, including the input labels given to the contained nodes and the output labels returned by the contained nodes, is isomorphic to an element of C where v corresponds to the node marked as the center.

In the case of directed paths as our class of input graphs, we are interested in identifying the simplest possible form of LCL problems. For this purpose, we define *β-normalized* LCLs; these are problems for which the input is just binary, and the size of the set of output labels is β. Moreover, the solution can be checked at each node v by just inspecting the input and output of v, and, separately, the output

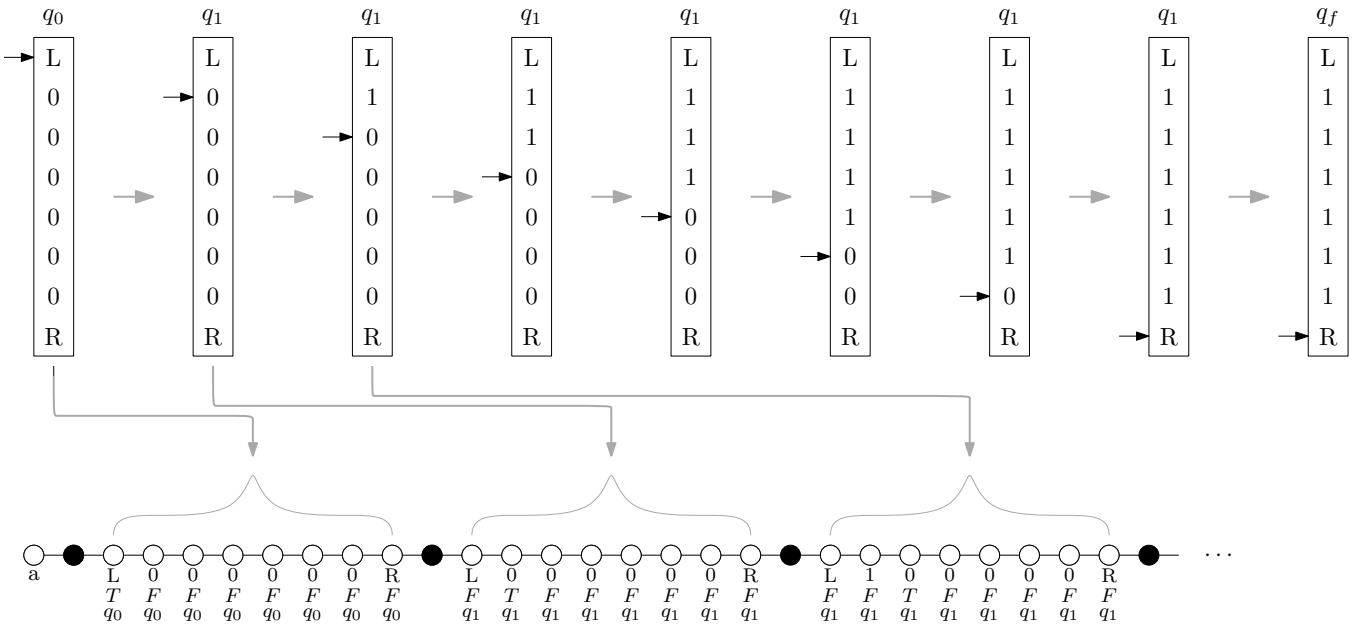

Figure 1: Illustration of a correct encoding of the execution of an LBA on a path; black nodes act as separators between the encoding of two consecutive steps of the LBA; in the example, the LBA executes a unary counter.

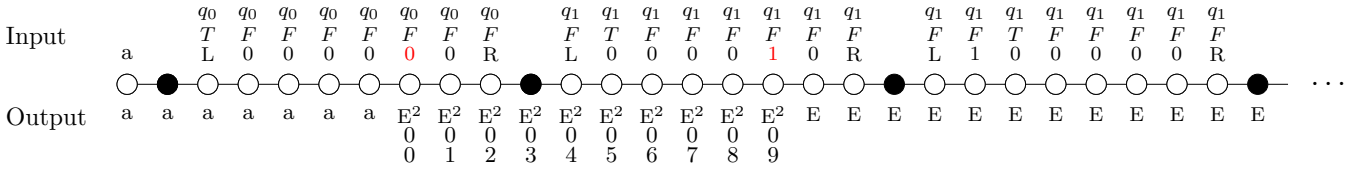

Figure 2: Illustration of an incorrect encoding of the execution of an LBA on a path; in the example, the tape of the LBA is wrongly copied (the inputs in red are different, while they should be the same). The error output E^2 encodes the distance of $B+1$ between the two nodes, and the input wrongly copied.

of v and the output of its predecessor. More formally, a *β-normalized* LCL problem is given by finite input and output label sets Σ_{in}, Σ_{out} satisfying $|\Sigma_{\text{in}}| = 2$, $|\Sigma_{\text{out}}| = \beta$, a finite set $C_{\text{in}-\text{out}}$ of pairs $(\ell_{\text{in}}, \ell_{\text{out}}) \in \Sigma_{\text{in}} \times \Sigma_{\text{out}}$ and a finite set $C_{\text{out}-\text{out}}$ of pairs $(\ell_{\text{out}}, \ell'_{\text{out}}) \in \Sigma_{\text{out}} \times \Sigma_{\text{out}}$. The global output of a distributed algorithm for the β-normalized LCL problem is correct if the following hold:

- For each node v, we have $(\text{Input}(v), \text{Output}(v)) \in C_{\text{in}-\text{out}}$, where $\text{Input}(v)$ denotes the input label of v, and $\text{Output}(v)$ the output label of v.
- For each node v that has a predecessor, we have $(\text{Output}(v), \text{Output}(u)) \in C_{\text{out}-\text{out}}$, where u is the predecessor of v, and $\text{Output}(v), \text{Output}(u)$ are the output labels of v and u, respectively.

It is straightforward to check that a β-normalized LCL problem is indeed a special case of an LCL problem where $r = 1$.

3 HARDNESS

In this section we study the hardness of determining the distributed complexity of LCLs on paths and cycles with input labels. More precisely, we start by proving the existence of a family Π of LCL problems for consistently globally oriented paths, such that, given an LCL problem in Π, it is PSPACE-hard to decide if its distributed complexity is $O(1)$ or $\Theta(n)$. The main result is summarised in the following theorem.

Theorem 1. *It is PSPACE-hard to distinguish whether a given* LCL *problem $\mathcal{P}$ with input labels can be solved in $O(1)$ time or needs $\Omega(n)$ time on globally oriented path graphs.*

The high level idea of the proof of Theorem 1 is as follows. We would like to encode the execution of Turing machines as LCLs on consistently oriented paths, and then define some LCL for which the complexity depends on the running time of the machine. This is fairly easy on oriented grids, for example, where we can use one dimension of the grid as a tape, and the other dimension as time. One may try to do the same on paths, by projecting everything on a single dimension, concatenating the tape state of each step. Unfortunately, the obtained encoding is not locally checkable, since the length of the tape may be non-constant. Hence, in order to guarantee the local checkability, we should consider Turing machines having a tape of size at most B, where B is a constant with respect to the number of nodes in the path where we want to encode its

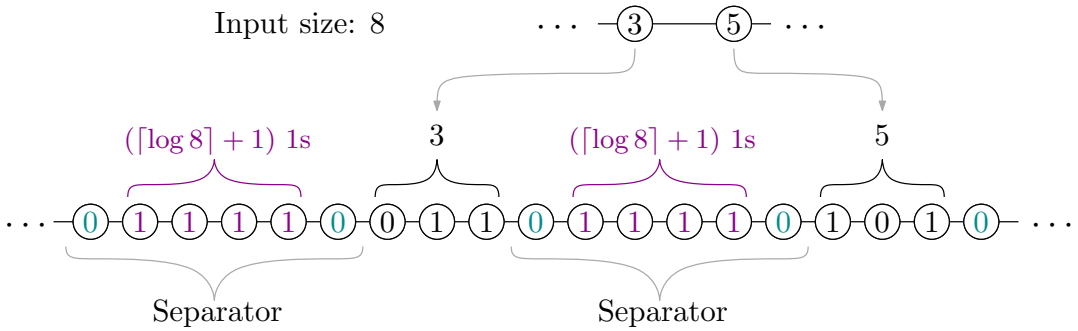

Figure 3: Illustration of the normalization of an LCL.

execution. For this purpose, we consider Linear Bounded Automata (LBA) [18, p. 225]. An LBA is a Turing machine that has a tape of size upper bounded by some B. We show that, if B is constant with respect to the number of nodes in the path, we can then encode the execution of an LBA M_B as an LCL for directed paths. Moreover, we show that by seeing this encoding as a two party game between a *prover* and a *disprover*, we can encode the execution of M_B using labels of constant size that do not depend on B, even in the case in which the LCL checkability radius is 1. If the execution of M_B is not correctly encoded in the input of the LCL, then we can disprove its correctness using output labels of size $O(B)$. Moreover, we ensure that, if the execution of M_B is correctly encoded in the input of the LCL, it is not possible to produce a correct proof of non-correctness. Then, in order to obtain an LCL with a distributed complexity that depends on the execution time of M_B, we encode some secret input at the first node of the path. We require then that all nodes involved in a correct encoding must produce the same secret as output. Figure 1 shows an example of an LBA that executes a unary counter, and its encoding as input to nodes on a path. In this instance, all nodes must produce the symbol a as output. Figure 2 shows an example of the wrong input (the tape has been copied incorrectly between two consecutive steps of the LBA). In this case, nodes are allowed to produce a chain of errors. Different types of errors will be handled using different types of error labels. In the example, all nodes that produce the error chain, output E^2, indicating an error of type 2. We will show that we need $O(B)$ symbols to handle all possible errors (including the case in which the input tape is too long, way more than B). Also, it is necessary that all error chains that we allow as outputs must be locally checkable.

Another interesting problem is to identify, for an LCL that can be distributedly solved in constant time, how big this constant can be. In particular, we first focus on identifying the simplest possible description of an LCL, and then, we provide a lower bound on the complexity of a constant time LCL, as a function of the size of the LCL description. For this purpose, we consider β-normalized LCLs, i.e., problems for which the input labeling is just binary and there are β possible output labels. Also, the verifier for these LCLs is the simplest possible: it can only check if the output of a node is correct w.r.t. its input, and separately, if the output of a node is correct w.r.t. the output of its predecessor. Therefore, we show how to convert an LCL to a β-normalized one by encoding the input in binary (Figure 3 shows an example), and obtain the following result.

Theorem 2. *There are β-normalized* LCL*s that can be solved in constant time but the distributed time complexity is $2^{\Omega(\beta)}$.*

All results we described so far apply to globally oriented paths. Nevertheless, we show that the ideas and techniques can be generalized to work on undirected paths and cycles as well, obtaining essentially the same results. Finally, we will show how to lift these results to trees *without* input labels, proving the following theorem.

Theorem 3. *It is PSPACE-hard to distinguish whether a given* LCL *problem $\mathcal{P}$ without input labels can be solved in $O(1)$ time or needs $\Omega(n)$ time on trees with degree $\Delta = 3$.*

4 DECIDABILITY

In this section, we show that the two gaps $\omega(1)$—$o(\log^* n)$ and $\omega(\log^* n)$—$o(n)$ for LCL problems *with input labels* on paths and cycles are decidable. More specifically, given a specification of an LCL problem $\mathcal{P}$, there is an algorithm that outputs a description of an asymptotically optimal deterministic LOCAL algorithm for $\mathcal{P}$, as well as its time complexity.

We will prove the statements for the case of cycles, but the analogous results for cycles and paths follows as a simple corollary, as we can encode constraints related to degree-1 nodes as constraints related to nodes adjacent to a special input label. Furthermore, having a promise that the input is a path does not change the time complexity of an LCL problem: if a problem can be solved in time $T = o(n)$ in labeled paths, the same algorithm will solve it also in time $T = o(n)$ in labeled cycles.

The proof of Theorem 4 is in Section 4.2; the proof of Theorem 5 is in Section 4.5.

Theorem 4. *For any* LCL *problem $\mathcal{P}$ on cycle graphs, its deterministic* LOCAL *complexity is either $\Omega(n)$ or $O(\log^* n)$. Moreover, there is an algorithm that decides whether $\mathcal{P}$ has complexity $\Omega(n)$ or $O(\log^* n)$ on cycle graphs; for the case the complexity is $O(\log^* n)$, the algorithm outputs a description of an $O(\log^* n)$-round deterministic* LOCAL *algorithm that solves $\mathcal{P}$.*

Theorem 5. *For any* LCL *problem $\mathcal{P}$ on cycle graphs, its deterministic* LOCAL *complexity is either $\Omega(\log^* n)$ or $O(1)$. Moreover, there is an algorithm that decides whether $\mathcal{P}$ has complexity $\Omega(\log^* n)$ or $O(1)$ on cycle graphs; for the case the complexity is $O(1)$, the algorithm outputs a description of an $O(1)$-round deterministic* LOCAL *algorithm that solves $\mathcal{P}$.*

For convenience, in this section, a directed path P with input labels is alternatively described as a string in Σ_{in}^k, where $k > 0$ is the number of nodes in P. Similarly, an output labeling $\mathcal{L}$ of P is alternatively described as a string in Σ_{out}^k. In subsequent discussion, we freely switch between the graph-theoretic notation and the string notation. Given an output labeling $\mathcal{L}$ of P, we say that $\mathcal{L}$ is *locally consistent* at v if the input and output labeling assigned to $N^r(v)$ is acceptable for v. Note that $N^r(v)$ refers to the radius-r neighborhood of v. Given two integers $a \le b$, the notation $[a, b]$ represents the set of all integers $\{a, a+1, \ldots, b\}$. Given a string w, denote w^R as the reverse of w.

4.1 Pumping Lemmas for Paths

Let $P = (s, \ldots, t)$ be a directed path, where each node has an input label from Σ_{in}. The *tripartition* of the nodes $\xi(P) = (D_1, D_2, D_3)$ is defined as follows:

$$D_1 = N^{r-1}(s) \cup N^{r-1}(t),$$
$$D_2 = \left(N^{2r-1}(s) \cup N^{2r-1}(t)\right) \setminus D_1,$$
$$D_3 = P \setminus (D_1 \cup D_2).$$

See Figure 4 for an illustration. More specifically, suppose $P = (u_1, \ldots, u_k)$, and let $i \in [1, k]$. Then we have:

- $u_i \in D_1$ if and only if $i \in [1, r] \cup [k - r + 1, k]$.
- $u_i \in D_2$ if and only if $i \in [r + 1, 2r] \cup [k - 2r + 1, k - r]$.
- $u_i \in D_3$ if and only if $i \notin [1, 2r] \cup [k - 2r + 1, k]$.

Let $\mathcal{L}\colon D_1 \cup D_2 \to \Sigma_{\text{out}}$ assign output labels to $D_1 \cup D_2$. We say that $\mathcal{L}$ is *extendible* w.r.t. P if there exists a complete labeling $\mathcal{L}_\diamond$ of P such that $\mathcal{L}_\diamond$ agrees with $\mathcal{L}$ on $D_1 \cup D_2$, and $\mathcal{L}_\diamond$ is locally consistent at all nodes in $D_2 \cup D_3$.

An Equivalence Class. We define an equivalence class $\overset{\star}{\sim}$ for the directed paths (i.e., the set of all non-empty strings in Σ_{in}^*), as follows.

> Consider two directed paths $P = (u_1, \ldots, u_x)$ and $P' = (v_1, \ldots, v_y)$, and let $\xi(P) = (D_1, D_2, D_3)$ and $\xi(P') = (D_1', D_2', D_3')$. Consider the following natural 1-to-1 correspondence $\phi\colon (D_1 \cup D_2) \to (D_1' \cup D_2')$ defined as $\phi(u_i) = v_i$ and $\phi(u_{x-i+1}) = v_{y-i+1}$ for each $i \in [1, 2r]$. The 1-to-1 correspondence is well-defined so long as (i) $x = y$ or (ii) $x \ge 4r$ and $y \ge 4r$. We have $P \overset{\star}{\sim} P'$ if and only if the following two statements are met:
>
> - **Isomorphism:** The 1-to-1 correspondence ϕ is well-defined, and for each $u_i \in D_1 \cup D_2$, the input label of u_i is identical to the input label of $\phi(u_i)$.
> - **Extendibility:** Let $\mathcal{L}$ be *any* assignment of output labels to nodes in $D_1 \cup D_2$, and let $\mathcal{L}'$ be the corresponding output labeling of $D_1' \cup D_2'$ under ϕ. Then $\mathcal{L}$ is extendible w.r.t. P if and only if $\mathcal{L}'$ is extendible w.r.t. P'.
>
> Note that for the special case of $x \le 4r$, we have $P \overset{\star}{\sim} P'$ if and only if P is identical to P'.

Define $\mathsf{Type}(P)$ as the equivalence class of P w.r.t. $\overset{\star}{\sim}$. The following lemma is analogous to [8, Theorem 3] in a specialized setting. One useful consequence of this lemma is that if we start with a path or a cycle G with a legal labeling, after replacing its subpath P with another one P' having the same type as P, then it is always possible to assign output labeling to P' to get a legal labeling without changing the already-assigned output labels of nodes outside of P'.

Lemma 6. *Let G be a path graph or a cycle graph where all nodes have input labels from Σ_{in}. Let P be a directed subpath of G, and let P' be another directed path such that $\mathsf{Type}(P') = \mathsf{Type}(P)$. Let $\mathcal{L}_\diamond$ any complete labeling of G such that $\mathcal{L}_\diamond$ is locally consistent at all nodes in P. Let $G' = \mathsf{Replace}(G, P, P')$ be the graph resulting from replacing P with P' in G. Then there exists a complete labeling $\mathcal{L}_\diamond'$ of G' such that the following two conditions are met.*

(1) For each $v \in V(G) \setminus V(P)$ and its corresponding $v' \in V(G') \setminus V(P')$, we have $\mathcal{L}_\diamond(v) = \mathcal{L}_\diamond'(v')$. Moreover, if $\mathcal{L}_\diamond$ is locally consistent at $v \in V(G) \setminus V(P)$, then $\mathcal{L}_\diamond'$ is locally consistent at v'.

(2) $\mathcal{L}_\diamond'$ is locally consistent at all nodes in P'.

The following lemma is analogous to [8, Theorem 4] in a specialized setting. We only use this lemma in Section 4.1.

Lemma 7. *Let $P = (v_1, \ldots, v_k)$, and let $P' = (v_1, \ldots, v_{k-1})$. Let the input label of v_k be α. Then $\mathsf{Type}(P)$ is a function of α and $\mathsf{Type}(P')$.*

The number of types can be upper bounded as follows.

Lemma 8. *The number of equivalence classes of $\overset{\star}{\sim}$ (i.e., types) is at most $|\Sigma_{\text{in}}|^{4r} 2^{|\Sigma_{\text{out}}|^{4r}}$.*

Proof. Let P be a directed path, and let $\xi(P) = (D_1, D_2, D_3)$. Then $\mathsf{Type}(P)$ is determined by the following information.

- The input labels in $D_1 \cup D_2$. Note that there are at most $|\Sigma_{\text{in}}|^{4r}$ possible input labeling of $D_1 \cup D_2$.
- A length-x binary string indicating the extendibility of each possible output labeling of $D_1 \cup D_2$, where $x = |\Sigma_{\text{out}}|^{4r}$.

Therefore, there are at most $|\Sigma_{\text{in}}|^{4r} 2^{|\Sigma_{\text{out}}|^{4r}}$ equivalence classes of $\overset{\star}{\sim}$. □

Define ℓ_{pump} as the total number of types. Observe that Lemma 7 implies that $\mathsf{Type}(P)$ can be computed by a finite automaton whose number of states is the total number of types, which is a constant independent of P. Thus, we have the following two *pumping lemmas* which allow us to extend the length of a given directed path P while preserving the type of P. The following two lemmas follow from the standard pumping lemma for regular language.

Lemma 9. *Let $P \in \Sigma_{\text{in}}^k$ with $k \ge \ell_{\text{pump}}$. Then P can be decomposed into three substrings $P = x \circ y \circ z$ such that (i) $|xy| \le \ell_{\text{pump}}$, (ii) $|y| \ge 1$, and (iii) for each non-negative integer i, $\mathsf{Type}(x \circ y^i \circ z) = \mathsf{Type}(P)$.*

Lemma 10. *For each $w \in \Sigma_{\text{in}}^{>0}$, there exist two positive integers a and b such that $a + b \le \ell_{\text{pump}}$, and $\mathsf{Type}(w^{ai+b})$ is invariant for each non-negative integer i.*

4.2 The $\omega(\log^* n)$—$o(n)$ Gap

In this section we show that the $\omega(\log^* n)$—$o(n)$ gap is decidable. More specifically, we show that an LCL problem $\mathcal{P}$ can be solved

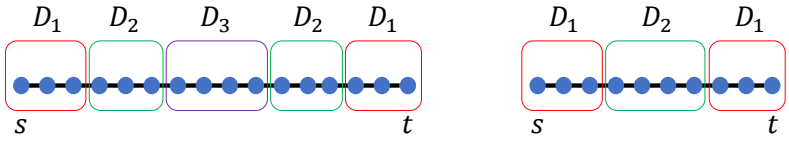

Figure 4: Illustration of the tripartition $\xi(P) = (D_1, D_2, D_3)$ with $r = 3$.

in $O(\log^* n)$ rounds if and only if there exists a *feasible function*, which is defined as follows.

Input: A directed path $P = w_1 \circ S \circ w_2$, where $|w_1| \in [\ell_{\text{pump}}, \ell_{\text{pump}}+1]$, $|w_2| \in [\ell_{\text{pump}}, \ell_{\text{pump}}+1]$, and $|S| = 2r$. The decomposition $P = w_1 \circ S \circ w_2$ is considered part of the input.

Output: A string $\mathcal{L} \in \Sigma_{\text{out}}^{2r}$ that represents the output labeling of S.

Requirement: Any such function f is said to be *feasible* if the following requirement is met for any paths S_1, S_2 and w_a, w_b, w_c, w_d such that $\{|w_a|, |w_b|, |w_c|, |w_d|\} \subseteq [\ell_{\text{pump}}, \ell_{\text{pump}}+1]$ and $|S_1| = |S_2| = 2r$. Let $P = w_a \circ S_1 \circ w_b \circ w_c \circ S_2 \circ w_d$, and consider the following assignment of output labels to $S_1 \cup S_2$.

- Either label S_1 by $f(w_a \circ S_1 \circ w_b)$ or label S_1^R by $f(w_b^R \circ S_1^R \circ w_a^R)$.
- Either label S_2 by $f(w_c \circ S_2 \circ w_d)$ or label S_2^R by $f(w_d^R \circ S_2^R \circ w_c^R)$.

It is required that given such a partial labeling of P, the middle part $w_b \circ w_c$ can be assigned output labels in such a way that the labeling of (i) the last r nodes of S_1, (ii) all nodes in $w_b \circ w_c$, and (iii) the first r nodes of S_2 are locally consistent.

The following lemma is a straightforward consequence of the well-known $O(\log^* n)$-round MIS algorithm on cycles.

Lemma 11. *Let G be a cycle graph of n nodes, and let $s \le k$ be two constant integers such that $n \ge (s+k)^2$. Then in $O(\log^* n)$ rounds we can compute a decomposition $V = A \cup B$ such that each connected component of A has size s, and each connected component of B has size within $[k, k+1]$.*

Lemma 12. *If a feasible function f exists, then there is an $O(\log^* n)$-round deterministic* LOCAL *algorithm for $\mathcal{P}$ on cycles.*

Proof. Given that the number of nodes n is at least some large enough constant, in $O(\log^* n)$ rounds we can compute a decomposition $V = A \cup B$ such that each connected component of A has size $2r$, and each connected component of B has size within $[2\ell_{\text{pump}}, 2\ell_{\text{pump}}+1]$. This can be done using Lemma 11 with $s = 2r$ and $k = 2\ell_{\text{pump}}$. We further decompose each connected component P of B into two paths $P = P_1 \circ P_2$ in such a way that the size of both P_1 and P_2 are within the range $[\ell_{\text{pump}}, \ell_{\text{pump}}+1]$. We write $\mathscr{P}$ to denote the set of all these paths.

Let S be a connected component of A, and let w_1 and w_2 be its two neighboring paths in $\mathscr{P}$ so that $(w_1 \circ S \circ w_2)$ is a subpath of the underlying graph G. The output labels of S are assigned either by labeling S with $f(w_1 \circ S \circ w_2)$ or by labeling S^R with $f(w_2^R \circ S^R \circ w_1^R)$. At this moment, all components of A have been assigned output labels using f. By the feasibility of f, each connected component of B is able to label itself output labels in such a way that the labeling of all nodes are locally consistent. □

Lemma 13. *If there is an $o(n)$-round deterministic* LOCAL *algorithm for $\mathcal{P}$ on cycles, then a feasible function f exists.*

Proof. Fix s to be some sufficiently large number, and fix $n = 8(s + \ell_{\text{pump}}) + 2(2r)$. We select s to be large enough so that the runtime of $\mathcal{A}$ is smaller than $0.1s$. For any given directed path w with $|w| \in [\ell_{\text{pump}}, \ell_{\text{pump}}+1]$, we fix w^+ as the result of applying the pumping lemma (Lemma 9) on w so that the following two conditions are met: (i) $|w^+| \in [s, s + \ell_{\text{pump}}]$ and (ii) $\text{Type}(w) = \text{Type}(w^+)$.

Constructing a Feasible Function f by Simulating $\mathcal{A}$. The function $f(w_1 \circ S \circ w_2)$ is constructed by simulating a given $o(n)$-round deterministic LOCAL algorithm for $\mathcal{P}$. The output labeling given by $f(w_1 \circ S \circ w_2)$ is exactly the result of simulating $\mathcal{A}$ on the path $P = w_1^+ \circ S \circ w_2^+$ while assuming the number of nodes of the underlying graph is n. Remember that the round complexity of $\mathcal{A}$ is $o(n)$ on n-node graphs. By setting s to be large enough, the runtime of $\mathcal{A}$ can be made smaller than $0.1s$. Thus, the calculation of $f(w_1 \circ S \circ w_2)$ only depends on the IDs and the input labels of (i) the last $0.1s$ nodes in w_1^+, (ii) all nodes in S, and (iii) the first $0.1s$ nodes in w_2^+. In the calculation of $f(w_1 \circ S \circ w_2)$, the IDs of the nodes that participate in the simulation of $\mathcal{A}$ are chosen arbitrarily so long as they are distinct.

Feasibility of f. Now we verify that the function f constructed above is feasible. Consider any choices of paths S_1, S_2 and w_a, w_b, w_c, w_d such that $\{|w_a|, |w_b|, |w_c|, |w_d|\} \subseteq [\ell_{\text{pump}}, \ell_{\text{pump}}+1]$ and $|S_1| = |S_2| = 2r$. Define $P = w_a \circ S_1 \circ w_b \circ w_c \circ S_2 \circ w_d$, and let G be the cycle graph formed by connecting the two ends of the path P. To show that f is feasible, we need to consider the following four ways of assigning output labels to $S_1 \cup S_2$.

(1) Label S_1 by $f(w_a \circ S_1 \circ w_b)$; label S_2 by $f(w_c \circ S_2 \circ w_d)$.
(2) Label S_1 by $f(w_a \circ S_1 \circ w_b)$; label S_2^R by $f(w_d^R \circ S_2^R \circ w_c^R)$.
(3) Label S_1^R by $f(w_b^R \circ S_1^R \circ w_a^R)$; label S_2 by $f(w_c \circ S_2 \circ w_d)$.
(4) Label S_1^R by $f(w_b^R \circ S_1^R \circ w_a^R)$; label S_2^R by $f(w_d^R \circ S_2^R \circ w_c^R)$.

For each of the above four partial labelings of P, we need to show that the middle part $w_b \circ w_c$ can still be assigned output labels in such a way that the labeling of (i) the last r nodes of S_1, (ii) all nodes in $w_b \circ w_c$, and (iii) the first r nodes of S_2 are locally consistent.

Proof of the First Case. In what follows, we focus on the first case, i.e., the partial labeling is given by labeling S_1 by $f(w_a \circ S_1 \circ w_b)$ and labeling S_2 by $f(w_c \circ S_2 \circ w_d)$; the proof for the other three cases are analogous. In this case, we define $P' = w_a^+ \circ S_1 \circ w_b^+ \circ w_c^+ \circ S_2 \circ w_d^+$, and

let G' be the cycle graph formed by connecting the two ends of P'. Note that the number of nodes in G' is at most $8(s+\ell_{\text{pump}})+2(2r) = n$. All we need to do is to find an output labeling $\mathcal{L}$ of G such that the following conditions are satisfied.

(a) The output labels of S_1 is given by $f(w_a \circ S_1 \circ w_b)$.
(b) The output labels of S_2 is given by $f(w_c \circ S_2 \circ w_d)$.
(c) The labeling of (i) the last r nodes of S_1, (ii) all nodes in $w_b \circ w_c$, and (iii) the first r nodes of S_2 are locally consistent.

We first generate an output labeling $\mathcal{L}'$ of G' by executing $\mathcal{A}$ on G' under the following ID assignment. The IDs of (i) the last $0.1s$ nodes in w_a^+, (ii) all nodes in S_1, and (iii) the first $0.1s$ nodes in w_b^+ are chosen as the ones used in the definition of $f(w_a \circ S_1 \circ w_b)$. Similarly, the IDs of (i) the last $0.1s$ nodes in w_c^+, (ii) all nodes in S_2, and (iii) the first $0.1s$ nodes in w_d^+ are chosen as the ones used in the definition of $f(w_c \circ S_2 \circ w_d)$. The IDs of the rest of the nodes are chosen arbitrarily so long as when we run $\mathcal{A}$ on G', no node sees two nodes with the same ID. Due to the way we define f, the output labeling $\mathcal{L}'$ of the subpath S_1 is exactly given by $f(w_a \circ S_1 \circ w_b)$, and the output labeling $\mathcal{L}'$ of S_2 is exactly $f(w_c \circ S_2 \circ w_d)$. Due to the correctness of $\mathcal{A}$, $\mathcal{L}'$ is a legal labeling.

We transform the output labeling $\mathcal{L}'$ of G' to a desired output labeling $\mathcal{L}$ of G. Remember that G is the result of replacing the four subpaths w^+ of G' by w, and we have $\text{Type}(w^+) = \text{Type}(w)$. In view of Lemma 6, there is a legal labeling $\mathcal{L}$ of G such that all nodes in S_1 and S_2 are labeled the same as in G'. Therefore, the labeling $\mathcal{L}$ satisfies the above three conditions (a), (b), and (c).

The Other Cases. We briefly discuss how we modify the proof to deal with the other three cases. For example, consider the second case, where the partial labeling is given by labeling S_1 by $f(w_a \circ S_1 \circ w_b)$ and labeling S_2^R by $f(w_d^R \circ S_2^R \circ w_c^R)$. In this case, the path P' is defined as

$$P' = w_a^+ \circ S_1 \circ w_b^+ \circ \left((w_c^R)^+\right)^R \circ S_2^R \circ \left((w_d^R)^+\right)^R.$$

During the ID assignment of G', the IDs of (i) the last $0.1s$ nodes in w_c^+, (ii) all nodes in S_2, and (iii) the first $0.1s$ nodes in w_d^+ are now chosen as the ones used in the definition of $f(w_d^R \circ S_2^R \circ w_c^R)$. Using such an ID assignment, the output labeling $\mathcal{L}'$ of S_2^R as the result of executing $\mathcal{A}$ on G' will be exactly the same as the output labeling given by $f(w_d^R \circ S_2^R \circ w_c^R)$. The rest of the proof is the same. □

Theorem 4 follows from the above two lemmas. The decidability result is due to the simple observation that whether a feasible function exists is decidable.

4.3 Partitioning a Cycle

In the following sections, we prove the decidability result associated with the $\omega(1)$—$o(\log^* n)$ gap. In this proof, we also define a feasible function, prove its decidability, and show the existence given an $o(\log^* n)$-time algorithm. The main challenge here is that an MIS cannot be computed in $O(1)$ time. To solve this issue, we decompose a cycle into paths with unrepetitive patterns and paths with repetitive patterns. For paths with unrepetitive patterns, we are able to compute a sufficiently well-spaced MIS in $O(1)$ time by making use of the irregularity of the input patterns.

Section 4.3 considers an $O(1)$-round algorithm that partitions a cycle into some short paths and some paths that have a repeated input pattern. Section 4.4 defines a *feasible function* whose existence characterizes the $O(1)$-round solvable LCL problems. In Section 4.5, we prove Theorem 5.

Partitioning an Undirected Cycle into Directed Paths. Let G be a cycle graph. An orientation of a node v is an assignment to one of its neighbor, this can be specified using port-numbering. An orientation of the nodes in G is called *ℓ-orientation* if the following condition is met. If $|V(G)| \le \ell$, then all nodes in G are oriented to the same direction. If $|V(G)| > \ell$, then each node $v \in V(G)$ belongs to a path P such that (i) all nodes in P are oriented to the same direction, and (ii) the number of nodes in P is at least ℓ. In $O(1)$ rounds we can compute an ℓ-orientation of G for any constant ℓ.

Lemma 14 ([8])**.** *Let G be a cycle graph. Let ℓ be a constant. There is a deterministic* LOCAL *algorithm that computes an ℓ-orientation of G in $O(1)$ rounds.*

In this section, we will use a generalization of an ℓ-orientation that satisfies an additional requirement that the input labels of each directed path P in the decomposition with $|V(P)| > 2\ell_{\text{width}}$ (where $2\ell_{\text{width}}$ is a threshold) must form a periodic string (whose period length is at most ℓ_{pattern}).

A string $w \in \Sigma_{\text{in}}^*$ is called *primitive* if w cannot be written as x^i for some $x \in \Sigma_{\text{in}}^*$ and $i \ge 2$. Let G be a cycle graph or a path graph where each node $v \in V(G)$ has an input label from Σ_{in}. We define an $(\ell_{\text{width}}, \ell_{\text{count}}, \ell_{\text{pattern}})$-*partition* as a partition of G into a set of connected subgraphs $\mathscr{P}$ meeting the following criteria. We assume $|V(G)| > 2\ell_{\text{width}}$ and $\ell_{\text{pattern}} \ge \ell_{\text{width}}$.

Direction and Minimum Length: For each $P \in \mathscr{P}$, the nodes in P are oriented to the same direction, and $|V(P)| \ge \ell_{\text{width}}$.
Short Paths: Define $\mathscr{P}_{\text{short}}$ as the subset of $\mathscr{P}$ that contains paths having at most $2\ell_{\text{width}}$ nodes. For each $P = (v_1, \ldots, v_k) \in \mathscr{P}_{\text{short}}$, each node v_i in P knows its rank i.
Long Paths: Define $\mathscr{P}_{\text{long}} = \mathscr{P} \setminus \mathscr{P}_{\text{short}}$. Then the input labeling of the nodes in P is of the form w^k for some primitive string $w \in \Sigma_{\text{in}}^*$ such that $|w| \le \ell_{\text{pattern}}$ and $k \ge \ell_{\text{count}}$. Moreover, each node v in P knows the string w.

Note that $\mathscr{P}$ may contain a cycle. This is possible only when G is a cycle where the input labeling is a repetition (at least ℓ_{count} times) of a primitive string $w \in \Sigma_{\text{in}}^*$ of length at most ℓ_{pattern}. In this case, we must have $\mathscr{P} = \mathscr{P}_{\text{long}} = \{G\}$. Otherwise, $\mathscr{P}$ contains only paths.

The goal of this section is to show that an $(\ell_{\text{width}}, \ell_{\text{count}}, \ell_{\text{pattern}})$-partition can be found in $O(1)$ rounds.

We first show that an $(\ell_{\text{width}}, \ell_{\text{count}}, \ell_{\text{pattern}})$-partition can be found in $O(1)$ rounds for the case G is *directed*. That is, all nodes in G are initially oriented to the same direction, and we are allowed to re-orient the nodes.

Lemma 15. *Let G be a directed cycle or a directed path where each node $v \in V(G)$ has an input label from Σ_{in}, and $|V(G)| > 2\ell_{\text{width}}$. Let $\ell_{\text{width}}, \ell_{\text{count}}, \ell_{\text{pattern}}$ be three constants such that $\ell_{\text{pattern}} \ge \ell_{\text{width}}$. There is a deterministic* LOCAL *algorithm that computes an $(\ell_{\text{width}}, \ell_{\text{count}}, \ell_{\text{pattern}})$-partition in $O(1)$ rounds*

Combining Lemma 15 and Lemma 14, we are able to construct an $(\ell_{\text{width}}, \ell_{\text{count}}, \ell_{\text{pattern}})$-partition in $O(1)$ rounds for undirected graphs.

Lemma 16. *Let G be a cycle or a path where each node $v \in V(G)$ has an input label from Σ_{in}, and $|V(G)| > 2\ell_{\text{width}}$. Let ℓ_{width}, ℓ_{count}, and ℓ_{pattern} be three constants such that $\ell_{\text{pattern}} \geq \ell_{\text{width}}$. There is a deterministic* LOCAL *algorithm that computes an $(\ell_{\text{width}}, \ell_{\text{count}}, \ell_{\text{pattern}})$-partition in $O(1)$ rounds*

Proof. The algorithm is as follows. Compute an ℓ-orientation of G by Lemma 14 in $O(1)$ rounds with $\ell = 2\ell_{\text{width}} + 1$. For each maximal-length connected subgraph P where each constituent node is oriented to the same direction, find an $(\ell_{\text{width}}, \ell_{\text{count}}, \ell_{\text{pattern}})$-partition of P in $O(1)$ rounds by Lemma 15. □

4.4 Feasible Function

The goal of this section is to define a *feasible function* whose existence characterizes the $O(1)$-round solvable LCL problems. With respect to an LCL problem $\mathcal{P}$ and a function f which takes a string $w \in \Sigma_{\text{in}}^k$ with $1 \leq k \leq \ell_{\text{pump}}$ as input, and returns a string $f(w) \in \Sigma_{\text{out}}^k$, we define some partially or completely labeled path graphs which are used in the definition of a feasible function.

Completely Labeled Graph $\mathcal{G}_{w,z}$: Let $w \in \Sigma_{\text{in}}^*$ be any string of length at least 1 and at most ℓ_{pump}. Let z be any non-negative integer. Define $\mathcal{G}_{w,z} = (G_{w,z}, \mathcal{L})$ as follows. The graph $G_{w,z}$ is a path of the form $w^r \circ w^z \circ w^r$. The labeling $\mathcal{L}$ is a complete labeling of the form $f(w)^{z+2r}$. Define $\mathsf{Mid}(G_{w,z})$ as the middle subpath w^z of $G_{w,z}$.

Partially Labeled Graph $\mathcal{G}_{w_1,w_2,S}$: Let $w_1, w_2 \in \Sigma_{\text{in}}^*$ be any two strings of length at least 1 and at most ℓ_{pump}. Let $S \in \Sigma_{\text{in}}^*$ be any string (can be empty). Define $\mathcal{G}_{w_1,w_2,S} = (G_{w_1,w_2,S}, \mathcal{L})$ as follows. The path graph $G_{w_1,w_2,S}$ is of the form $w_1^{\ell_{\text{pump}}+2r} \circ S \circ w_2^{\ell_{\text{pump}}+2r}$. The labeling $\mathcal{L}$ is a partial labeling of $G_{w_1,w_2,S}$ which fixes the output labels of the first $2r|w_1|$ and the last $2r|w_2|$ nodes by $f(w_1)^{2r}$ and $f(w_2)^{2r}$, respectively. Define $\mathsf{Mid}(G_{w_1,w_2,S})$ as the middle subpath $w_1^{\ell_{\text{pump}}+r} \circ S \circ w_2^{\ell_{\text{pump}}+r}$ of $G_{w_1,w_2,S}$.

Feasible Function: We call f a *feasible function* if the following conditions are met: (i) For each $\mathcal{G}_{w,z} = (G_{w,z}, \mathcal{L})$, the complete labeling $\mathcal{L}$ is locally consistent at all nodes in $\mathsf{Mid}(G_{w,z})$. (ii) Each partially labeled graph $\mathcal{G}_{w_1,w_2,S}$ admits a complete labeling $\mathcal{L}_\diamond$ that is locally consistent at all nodes in $\mathsf{Mid}(\mathcal{G}_{w_1,w_2,S})$.

Lemma 17. *Given an* LCL *problem $\mathcal{P}$ on cycle graphs. It is decidable whether there is a feasible function.*

Proof. Note that it is not immediate from its definition as to whether a feasible function exists is decidable, since there appears to be *infinitely* many graphs $\mathcal{G}_{w,z}$ and $\mathcal{G}_{w_1,w_2,S}$ needed to be examined. However, the following simple observations show that it suffices to check only a constant number of these graphs.

- If the complete labeling $\mathcal{L}$ of $\mathcal{G}_{w,1} = (G_{w,1}, \mathcal{L})$ is locally consistent at all nodes in $\mathsf{Mid}(G_{w,1})$, then for all $z \geq 1$, the complete labeling $\mathcal{L}$ of $\mathcal{G}_{w,z} = (G_{w,z}, \mathcal{L})$ is also locally consistent at all nodes in $\mathsf{Mid}(G_{w,z})$.
- If $\mathcal{G}_{w_1,w_2,S}$ admits a complete labeling $\mathcal{L}_\diamond$ that is locally consistent at all nodes in $\mathsf{Mid}(\mathcal{G}_{w_1,w_2,S})$, then for each S' such that $\mathsf{Type}(S) = \mathsf{Type}(S')$, the partially labeled graph $\mathcal{G}_{w_1,w_2,S'}$ also admits a complete labeling $\mathcal{L}_\diamond$ that is locally consistent at all nodes in $\mathsf{Mid}(\mathcal{G}_{w_1,w_2,S'})$. This is due to Lemma 6.

Therefore, to decide whether a function f is feasible, we only need to check all possible $\mathcal{G}_{w,z}$ and $\mathcal{G}_{w_1,w_2,S}$. For each w we only need to consider the graph $\mathcal{G}_{w,z}$ with $z = 1$. For each w_1 and w_2, we do not need to go over all S; we only need to consider (i) the empty string $S = \emptyset$, and (ii) for each type τ, a string $S \in \Sigma_{\text{in}}^*$ such that $\mathsf{Type}(S) = \tau$. By Lemma 9, for each type τ, there exists $P \in \Sigma_{\text{in}}^x$ with $x \leq \ell_{\text{pump}}$ such that $\mathsf{Type}(P) = \tau$. Therefore, a string S with $\mathsf{Type}(S) = \tau$ can be found in bounded amount of time; also note that the number of types is bounded; see Lemma 8. □

For the rest of this section, we show that as long as the deterministic LOCAL complexity of $\mathcal{P}$ is $o(\log^* n)$ on cycle graphs, there exists a feasible function f. In Lemma 18 we show how to extract a function f from a given $o(\log^* n)$-round deterministic LOCAL algorithm $\mathcal{A}$, and then in Lemma 19 we prove that such a function f is feasible. Intuitively, Lemma 18 shows that there exists an ID-assignment such that when we run $\mathcal{A}$ on a subpath whose input labeling is a repetition of a length-k pattern w, the output labeling is also a repetition of a length-k pattern w'. The function f will be defined as $f(w) = w'$.

Lemma 18. *Let $\mathcal{A}$ be any deterministic* LOCAL *algorithm that solves $\mathcal{P}$ in $t(n) = o(\log^* n)$ rounds. Then there is a number n' and function f which takes a string $w \in \Sigma_{\text{in}}^k$ with $1 \leq k \leq \ell_{\text{pump}}$ as input, and returns a string $f(w) \in \Sigma_{\text{out}}^k$ meeting the following condition. For any $P = w^i \circ w^{2r+1} \circ w^i$ such that $|w^i| \geq t(n')$ and $1 \leq |w| \leq \ell_{\text{pump}}$, there is an assignment of distinct $\Theta(\log n')$-bit IDs to the nodes in P such that the following is true. Simulating $\mathcal{A}$ on P while assuming that the total number of nodes in the underlying graph is n' yields the output labeling $f(w)^{2r+1}$ for the middle subpath w^{2r+1}.*

Proof. In this proof we assume that there is no such a number n'. Then we claim that using $\mathcal{A}$ it is possible to obtain a deterministic LOCAL algorithm for MIS on an n-node directed cycle G *without input labeling*, in $O(t(n)) + O(1) = o(\log^* n)$ rounds. This contradicts the well-known $\Omega(\log^* n)$ lower bound for MIS [20].

Let G be an n-node directed cycle without input labeling. The MIS algorithm on G is described as follows. Let $w \in \Sigma_{\text{in}}^k$ with $1 \leq k \leq \ell_{\text{pump}}$ be chosen such that for any function f, the string $f(w) \in \Sigma_{\text{out}}^k$ does not satisfy the conditions stated in the lemma for the number $n' = nk$. Define G' as the graph resulting from replacing each node $v \in V(G)$ with a path w. We can simulate the imaginary graph G' in the communication network G by letting each node $v \in V(G)$ simulate a path w.

We execute the algorithm $\mathcal{A}$ on G' while assuming that the total number of nodes is n'. The execution takes $t(n') = O(t(n))$ rounds. For each node $v \in V(G)$, define the color of v as the sequence of the output labels of the path w^{2r} simulated by the node v and the $2r - 1$ nodes following v in the directed cycle G. This gives us a proper $O(1)$-coloring, since otherwise there must exist a subpath $P = w^{2r+1}$ of G' such that the output labeling of P is of the form

y^{2r+1} for some y, contradicting our choice of w. Using the standard procedure of computing an MIS from a coloring, with extra $O(1)$ rounds, an MIS of G can be obtained.

Note that there is a subtle issue about how we set the IDs of nodes in $V(G')$. The following method is guaranteed to output distinct IDs. Let $v \in V(G)$, and let $u_1, \ldots, u_k$ be the nodes in $V(G')$ simulated by v. Then we may use $\mathrm{ID}(u_i) = k \cdot \mathrm{ID}(v) + i$. □

Lemma 19. *Suppose that the deterministic* LOCAL *complexity of $\mathcal{P}$ is $o(\log^* n)$ on cycle graphs. Then there exists a feasible function f.*

Proof. Let $\mathcal{A}$ be any deterministic LOCAL algorithm that solves $\mathcal{P}$ in $t(n) = o(\log^* n)$ rounds. Let n' and f be chosen to meet the conditions in Lemma 18 for $\mathcal{A}$. The goal of the proof is to show that f is a feasible function. According to the conditions specified in Lemma 18 for the function f, we already know that the complete labeling $\mathcal{L}$ of each $\mathcal{G}_{w,z} = (G_{w,z}, \mathcal{L})$ is locally consistent at all nodes in $\mathsf{Mid}(G_{w,z})$. Therefore, all we need to do is the following. For each partially labeled graph $\mathcal{G}_{w_1,w_2,S}$, find a complete labeling $\mathcal{L}_\diamond$ that is locally consistent at all nodes in $\mathsf{Mid}(\mathcal{G}_{w_1,w_2,S})$.

Given the three parameters w_1, w_2, and S, define G as the cycle resulting from linking the two ends of the path $w_1^{\ell_{\text{pump}}} \circ w_1^{2r+1} \circ w_1^{\ell_{\text{pump}}} \circ S \circ w_2^{\ell_{\text{pump}}} \circ w_2^{2r+1} \circ w_2^{\ell_{\text{pump}}}$. Define $\mathcal{L}$ as the partial labeling of G which fixes the output labeling of the two subpaths w_1^{2r+1} and w_2^{2r+1} by $f(w_1)^{2r+1}$ and $f(w_2)^{2r+1}$, respectively. We write P_1^{mid} and P_2^{mid} to denote the two subpaths w_1^{2r+1} and w_2^{2r+1}, respectively.

In what follows, we show that the partially labeled graph $\mathcal{G} = (G, \mathcal{L})$ admits a legal labeling $\mathcal{L}_\diamond$. Since $\mathcal{G}_{w_1,w_2,S}$ is a subgraph of $\mathcal{G} = (G, \mathcal{L})$, such a legal labeling $\mathcal{L}_\diamond$ is also a complete labeling of $\mathcal{G}_{w_1,w_2,S}$ that is locally consistent at all nodes in $\mathsf{Mid}(\mathcal{G}_{w_1,w_2,S})$.

For the rest of the proof, we show the existence of $\mathcal{L}_\diamond$. This will be established by applying a pumping lemma. Define the graph G' as the result of the following operations on G.

- Replace the two subpaths $w_1^{\ell_{\text{pump}}}$ by w_1^x, where the number x is chosen such that $x|w_1| \geq 2t(n') + r$, and $\mathsf{Type}(w_1^{\ell_{\text{pump}}}) = \mathsf{Type}(w_1^x)$.
- Replace the two subpaths $w_2^{\ell_{\text{pump}}}$ by w_2^y, where the number y is chosen such that $y|w_2| \geq 2t(n') + r$, and $\mathsf{Type}(w_2^{\ell_{\text{pump}}}) = \mathsf{Type}(w_2^y)$.

The existence of the numbers x and y above is guaranteed by Lemma 10. The IDs of nodes in G' are assigned as follows. For $i = 1, 2$, select the IDs of the nodes in $\bigcup_{v \in P_i^{\text{mid}}} N^{t(n')}(v)$ in such a way that the output labeling of P_i^{mid} resulting from executing $\mathcal{A}$ on G' while assuming that the total number of nodes is n' is $f(w_i)^{2r+1}$. The existence of such an ID assignment is guaranteed by Lemma 18. For all remaining nodes in G', select their IDs in such a way that all nodes in $N^{r+t(n')}(v)$ receive distinct IDs, for each $v \in V(G')$. This ensures that the outcome of executing $\mathcal{A}$ on G' while assuming that the total number of nodes is n' is a legal labeling.

Let $\mathcal{L}'_\diamond$ be the legal labeling of G' resulting from executing $\mathcal{A}$ with the above IDs while pretending that the total number of nodes is n'. Note that $\mathcal{L}'_\diamond$ must label P_1^{mid} and P_2^{mid} by $f(w_1)^{2r+1}$ and $f(w_2)^{2r+1}$, respectively. A desired legal labeling $\mathcal{L}_\diamond$ of $\mathcal{G}$ can be obtained from the legal labeling $\mathcal{L}'_\diamond$ of G' by applying Lemma 6, as we have $\mathsf{Type}(w_1^{\ell_{\text{pump}}}) = \mathsf{Type}(w_1^x)$ and $\mathsf{Type}(w_2^{\ell_{\text{pump}}}) = \mathsf{Type}(w_2^y)$. □

4.5 The $\omega(1)$—$o(\log^* n)$ Gap

In this section we prove that it is decidable whether a given LCL problem $\mathcal{P}$ has complexity $\Omega(\log^* n)$ or $O(1)$ on cycle graphs.

Lemma 20. *Let f be any feasible function. Let G be any cycle graph. Let $\mathscr{P}$ be any set of disjoint subgraphs in G such that the input labeling of each $P \in \mathscr{P}$ is of the form w^x such that $x \geq 2\ell_{\text{pump}} + 2r$, and $w \in \Sigma_{\text{in}}^k$ is a string with $1 \leq k \leq \ell_{\text{pump}}$. For each $P \in \mathscr{P}$, define the subgraph P' as follows. If P is a cycle, define $P' = P$. If P is a path, write $P = w^{\ell_{\text{pump}}} \circ w^i \circ w^{\ell_{\text{pump}}}$, and define P' as the middle subpath w^i. Let $\mathcal{L}$ be a partial labeling of G defined as follows. For each $P = w^x \in \mathscr{P}$, fix the output labels of each subpath w of P' by $f(w)$. Then $\mathcal{G} = (G, \mathcal{L})$ admits a legal labeling $\mathcal{L}_\diamond$.*

Proof. Define V_1 as the set of all nodes such that $v \in V_1$ if v belongs to the middle subpath w^j of some path $P = w^{\ell_{\text{pump}}} \circ w^r \circ w^j \circ w^r \circ w^{\ell_{\text{pump}}} \in \mathscr{P}$. By the definition of feasible function, $\mathcal{L}$ is already locally consistent at all nodes in V_1. Thus, all we need to do is to construct a complete labeling $\mathcal{L}_\diamond$ of $\mathcal{G} = (G, \mathcal{L})$, and argue that $\mathcal{L}_\diamond$ is locally consistent at all nodes in $V_2 = V(G) \setminus V_1$.

There are two easy special cases. If $\mathscr{P} = \emptyset$, then no output label of any node in G is fixed, and so $\mathcal{G}$ trivially admits a legal labeling. If $\mathscr{P}$ contains a cycle, then $\mathscr{P} = \{G\}$, and hence $\mathcal{L}$ is already a legal labeling as $V_1 = V(G)$.

In subsequent discussion, we restrict ourselves to the case $\mathscr{P}$ is non-empty and contains only paths. The output labeling $\mathcal{L}_\diamond$ is constructed as follows. Define $\mathscr{P}_{\text{unlabeled}}$ as the maximal-length subpaths of G that are not assigned any output labels by $\mathcal{L}$. A path $P \in \mathscr{P}_{\text{unlabeled}}$ must be of the form $w_1^{\ell_{\text{pump}}} \circ S \circ w_2^{\ell_{\text{pump}}}$, where $w_1, w_2 \in \Sigma_{\text{in}}^*$ are two strings of length at least 1 and at most ℓ_{pump}, and $S \in \Sigma_{\text{in}}^*$ can be any string (including the empty string). Given $P \in \mathscr{P}_{\text{unlabeled}}$, we make the following definitions.

- Define P^+ as the subpath of G that includes P and the $r|w_1|$ nodes preceding P, and the $r|w_2|$ nodes following P in the graph G. Note that the set V_2 is exactly the union of nodes in P^+ for all $P \in \mathscr{P}_{\text{unlabeled}}$.
- Define P^{++} as the subpath of G that includes P and the $2r|w_1|$ nodes preceding P, and the $2r|w_2|$ nodes following P in the graph G. The path P^{++} must be of the form $w_1^{\ell_{\text{pump}}+2r} \circ S \circ w_2^{\ell_{\text{pump}}+2r}$, and the labeling $\mathcal{L}$ already fixes the output labels of the first $2r|w_1|$ and the last $2r|w_2|$ nodes of P^{++} by $f(w_1)^{2r}$ and $f(w_2)^{2r}$, respectively.

Observe that the path $P^{++} = w_1^{\ell_{\text{pump}}+2r} \circ S \circ w_2^{\ell_{\text{pump}}+2r}$ together with the labeling $\mathcal{L}$ is exactly the partially labeled graph $\mathcal{G}_{w_1,w_2,S}$. We assign the output labels to the nodes in P by the labeling $\mathcal{L}_\diamond$ guaranteed in the definition of feasible function. It is ensured that the labeling of all nodes within P^+ are locally consistent. By doing so for each $P \in \mathscr{P}_{\text{unlabeled}}$, we obtain a desired complete labeling that is locally consistent at all nodes in V_2. □

Lemma 21. *Suppose that there is a feasible function f for the* LCL *problem $\mathcal{P}$. Then there is an $O(1)$-round deterministic* LOCAL *algorithm $\mathcal{A}$ on cycle graphs.*

Proof. The first step of the algorithm $\mathcal{A}$ is to compute an $(\ell_{\text{width}}, \ell_{\text{count}}, \ell_{\text{pattern}})$-partition in $O(1)$ rounds by Lemma 16. We set $\ell_{\text{count}} = 2\ell_{\text{pump}} + 2r$ and $\ell_{\text{width}} = \ell_{\text{pattern}} = \ell_{\text{pump}}$. We assume $|V(G)| > 2\ell_{\text{width}}$. Recall that an $(\ell_{\text{width}}, \ell_{\text{count}}, \ell_{\text{pattern}})$-partition decomposes the cycle G into two sets of disjoint subgraphs $\mathscr{P}_{\text{short}}$ and $\mathscr{P}_{\text{long}}$.

Define G' as the graph resulting from applying the following operations on G. For each $P \in \mathscr{P}_{\text{short}}$, replace the path P by the path $P^* = x \circ y^i \circ z$ such that $i = \ell_{\text{count}}$, $1 \le |y| \le \ell_{\text{pattern}}$, and the type of P^* is the same as the type of P. The path P^* is obtained via Lemma 9. Note that each path $P \in \mathscr{P}_{\text{short}}$ has at least $\ell_{\text{width}} = \ell_{\text{pump}}$ nodes and at most $2\ell_{\text{width}} = 2\ell_{\text{pump}}$ nodes. Define $\mathscr{P}^*$ as the set of all P^* such that $P \in \mathscr{P}_{\text{short}}$. The graph G' is simulated in the communication graph G by electing a leader for each path $P \in \mathscr{P}_{\text{short}}$ to simulate P^*.

Calculate a partial labeling $\mathcal{L}'$ of G' using the feasible function f as follows. Recall $\ell_{\text{count}} = 2\ell_{\text{pump}} + 2r$. For each $P^* = x \circ y^{\ell_{\text{pump}}} \circ y^{2r} \circ y^{\ell_{\text{pump}}} \circ z \in \mathscr{P}^*$, label the middle subpath y^{2r} by the function f. For each $P = w^{\ell_{\text{pump}}} \circ w^i \circ w^{\ell_{\text{pump}}} \in \mathscr{P}_{\text{long}}$, label the middle subpath w^i by $f(w)^i$. Even though a path $P \in \mathscr{P}_{\text{long}}$ can have $\omega(1)$ nodes, this step can be done locally in $O(1)$ rounds due to the following property of $(\ell_{\text{width}}, \ell_{\text{count}}, \ell_{\text{pattern}})$-partition. All nodes in a path $P \in \mathscr{P}_{\text{long}}$ agree with the same direction and know the primitive string w.

By Lemma 20, the remaining unlabeled nodes in G' can be labeled to yield a legal labeling of G'. This can be done in $O(1)$ rounds since the connected components formed by unlabeled nodes have at most $O(1)$ nodes. Given any valid labeling of G', a legal labeling of G can be obtained by applying Lemma 6 in $O(1)$ rounds. Remember that $\text{Type}(P) = \text{Type}(P^*)$ for each $P \in \mathscr{P}_{\text{short}}$, and G' is exactly the result of replacing each $P \in \mathscr{P}_{\text{short}}$ by P^*. □

Combining Lemma 17, Lemma 19, and Lemma 21, we have proved Theorem 5. That is, for any LCL problem $\mathcal{P}$ on cycle graphs, its deterministic LOCAL complexity is either $\Omega(\log^* n)$ or $O(1)$. Moreover, there is an algorithm that decides whether $\mathcal{P}$ has complexity $\Omega(\log^* n)$ or $O(1)$ on cycle graphs; for the case the complexity is $O(1)$, the algorithm outputs a description of an $O(1)$-round deterministic LOCAL algorithm that solves $\mathcal{P}$.

ACKNOWLEDGMENTS

Many thanks to Laurent Feuilloley, Juho Hirvonen, Janne H. Korhonen, Christoph Lenzen, Yannic Maus, and Seth Pettie for discussions, and to anonymous reviewers for their helpful comments on previous versions of this work. This work was supported in part by the Academy of Finland, Grant 285721, and NSF grants CCF-1514383, CCF-1637546, and CCF-1815316.

REFERENCES

[1] Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. 2018. The distributed complexity of locally checkable problems on paths is decidable. *arXiv preprint arXiv:1811.01672* (2018).

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

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

[4] Roderick Bloem, Nicolas Braud-Santoni, and Swen Jacobs. 2016. Synthesis of Self-Stabilising and Byzantine-Resilient Distributed Systems. In *Proc. International Conference on Computer Aided Verification (CAV 2016)*. Springer, 157–176. https://doi.org/10.1007/978-3-319-41528-4_9

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

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

[7] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. 2016. An Exponential Separation between Randomized and Deterministic Complexity in the LOCAL Model. In *Proc. 57th IEEE Symposium on Foundations of Computer Science (FOCS 2016)*. IEEE, 615–624. https://doi.org/10.1109/FOCS.2016.72

[8] Yi-Jun Chang and Seth Pettie. 2017. A Time Hierarchy Theorem for the LOCAL Model. In *Proc. 58th IEEE Symposium on Foundations of Computer Science (FOCS 2017)*. IEEE, 156–167. https://doi.org/10.1109/FOCS.2017.23

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

[10] Danny Dolev, Keijo Heljanko, Matti Järvisalo, Janne H Korhonen, Christoph Lenzen, Joel Rybicki, Jukka Suomela, and Siert Wieringa. 2016. Synchronous counting and computational algorithm design. *J. Comput. System Sci.* 82, 2 (2016), 310–332. https://doi.org/10.1016/j.jcss.2015.09.002

[11] Fathiyeh Faghih and Borzoo Bonakdarpour. 2015. SMT-Based Synthesis of Distributed Self-Stabilizing Systems. *ACM Transactions on Autonomous and Adaptive Systems* 10, 3 (2015), 1–26. https://doi.org/10.1145/2767133

[12] Manuela Fischer and Mohsen Ghaffari. 2017. Sublogarithmic Distributed Algorithms for Lovász Local Lemma, and the Complexity Hierarchy. In *Proc. 31st International Symposium on Distributed Computing (DISC 2017)*. 18:1–18:16. https://doi.org/10.4230/LIPIcs.DISC.2017.18

[13] Mohsen Ghaffari, David G Harris, and Fabian Kuhn. 2018. On Derandomizing Local Distributed Algorithms. In *Proc. 59th IEEE Symposium on Foundations of Computer Science (FOCS 2018)*. https://doi.org/10.1109/FOCS.2018.00069 arXiv:1711.02194

[14] Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, and Yannic Maus. 2018. Improved Distributed Δ-Coloring. In *Proc. 37th ACM Symposium on Principles of Distributed Computing (PODC 2018)*. ACM, 427–436. https://doi.org/10.1145/3212734.3212764

[15] Mohsen Ghaffari and Hsin-Hao Su. 2017. Distributed Degree Splitting, Edge Coloring, and Orientations. In *Proc. 28th ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*. Society for Industrial and Applied Mathematics, 2505–2523. https://doi.org/10.1137/1.9781611974782.166

[16] Andrew V. Goldberg, Serge A. Plotkin, and Gregory E. Shannon. 1988. Parallel Symmetry-Breaking in Sparse Graphs. *SIAM Journal on Discrete Mathematics* 1, 4 (1988), 434–446. https://doi.org/10.1137/0401044

[17] Juho Hirvonen, Joel Rybicki, Stefan Schmid, and Jukka Suomela. 2017. Large cuts with local algorithms on triangle-free graphs. *Electronic Journal of Combinatorics* 24, 4 (2017). http://www.combinatorics.org/ojs/index.php/eljc/article/view/v24i4p21

[18] John E Hopcroft and Jeffrey D Ullman. 1979. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley.

[19] Alex Klinkhamer. 2016. *On the Limits and Practice of Automatically Designing Self-Stabilization.* Doctoral thesis. Michigan Technological University. https://digitalcommons.mtu.edu/etdr/90

[20] Nathan Linial. 1992. Locality in Distributed Graph Algorithms. *SIAM J. Comput.* 21, 1 (1992), 193–201. https://doi.org/10.1137/0221015

[21] Moni Naor. 1991. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM Journal on Discrete Mathematics* 4, 3 (1991), 409–412. https://doi.org/10.1137/0404036

[22] Moni Naor and Larry Stockmeyer. 1995. What Can be Computed Locally? *SIAM J. Comput.* 24, 6 (1995), 1259–1277. https://doi.org/10.1137/S0097539793254571

[23] Alessandro Panconesi and Aravind Srinivasan. 1995. The local nature of Δ-coloring and its algorithmic applications. *Combinatorica* 15, 2 (1995), 255–280. https://doi.org/10.1007/BF01200759

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

[25] Seth Pettie. 2018. Automatically Speeding Up LOCAL Graph Algorithms. In *7th Workshop on Advances in Distributed Graph Algorithms (ADGA 2018)*. http://adga.hiit.fi/2018/Seth.pdf

[26] Joel Rybicki and Jukka Suomela. 2015. Exact bounds for distributed graph colouring. In *Proc. 22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO 2015) (Lecture Notes in Computer Science)*, Vol. 9439. Springer, 46–60. https://doi.org/10.1007/978-3-319-25258-2_4