Efficient Classification of Locally Checkable Problems in Regular Trees

Alkida Balliu
Gran Sasso Science Institute
alkida.balliu@gssi.it

Sebastian Brandt
CISPA Helmholtz Center for Information Security
brandt@cispa.de

Yi-Jun Chang
National University of Singapore
cyijun@nus.edu.sg

Dennis Olivetti
Gran Sasso Science Institute
dennis.olivetti@gssi.it

Jan Studený
Aalto University
jan.studeny@aalto.fi

Jukka Suomela
Aalto University
jukka.suomela@aalto.fi

Abstract

We give practical, efficient algorithms that automatically determine the asymptotic distributed round complexity of a given locally checkable graph problem in the $[\Theta(\log n), \Theta(n)]$ region, in two settings. We present one algorithm for unrooted regular trees and another algorithm for rooted regular trees. The algorithms take the description of a locally checkable labeling problem as input, and the running time is polynomial in the size of the problem description. The algorithms decide if the problem is solvable in $O(\log n)$ rounds. If not, it is known that the complexity has to be $\Theta(n^{1/k})$ for some $k = 1, 2, \ldots$, and in this case the algorithms also output the right value of the exponent k.

In rooted trees in the $O(\log n)$ case we can then further determine the exact complexity class by using algorithms from prior work; for unrooted trees the more fine-grained classification in the $O(\log n)$ region remains an open question.

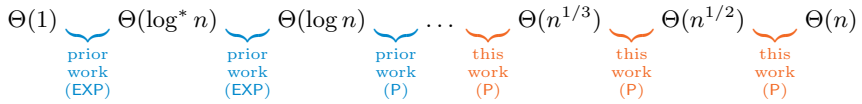
1 Introduction

We give practical, efficient algorithms that automatically determine the asymptotic distributed round complexity of a given *locally checkable* graph problem in *rooted or unrooted regular trees* in the $[\Theta(\log n), \Theta(n)]$ region, for both LOCAL and CONGEST models, see Section 3 for the precise definitions. In these cases, the distributed round complexity of any locally checkable problem is known to fall in one of the classes shown in Figure 1 [10, 11, 13, 19, 20, 21, 30]. Our algorithms are able to distinguish between all higher complexity classes from $\Theta(\log n)$ to $\Theta(n)$.

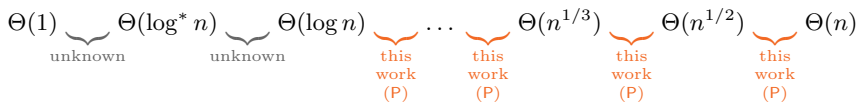
1.1 State of the art

Since 2016, there has been a large body of work studying the possible complexities of LCL problems. After an impressive sequence of works, the complexity landscape of LCL problems on bounded-degree general graphs, trees, and paths is now well-understood. For example, it is known that there

(a) Rooted regular trees in deterministic and randomized CONGEST and LOCAL:



(b) Unrooted regular trees in deterministic CONGEST and LOCAL:



(c) Unrooted regular trees in randomized CONGEST and LOCAL:

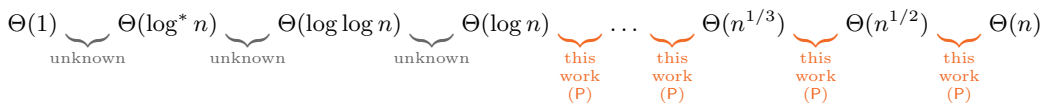


Figure 1: The most efficient algorithms for the classification of distributed round complexities. In the figure we show all possible complexity classes. Each gap between two classes corresponds to a natural decision problem: given a locally checkable problem, determine on which side of the gap its complexity is. For each gap we indicate whether a practical algorithm was provided already by prior work [8], whether it is first presented in this work, or whether the existence of such a routine is still an open question. The figure also indicates whether the algorithms are in P (polynomial time in the size of the problem description) or in EXP (exponential time in the size of the problem description).

are no LCLs with deterministic complexity between $\omega(\log^* n)$ and $o(\log n)$. The proofs of some of the complexity gaps implies that the design of asymptotically optimal distributed algorithms can be *automated* in certain settings, leading to a series of research studying the computational complexity of automated design of asymptotically optimal distributed algorithms. See Section 2 for more details.

The most recent paper [8] in this line of research presented an algorithm that takes as input the description of an LCL problem defined in *rooted regular trees* and classifies the problem into one of the four complexity classes $O(1)$, $\Theta(\log^* n)$, $\Theta(\log n)$, and $n^{\Theta(1)}$. The classification applies to both the LOCAL and CONGEST models of distributed computing, both for randomized and deterministic algorithms.

To illustrate the setting of locally checkable problems in rooted regular trees, consider, for example, the following problem, which is meaningful for rooted binary trees:

> Each node is labeled with 1 or 2. If the label of an internal node is 1, exactly one of its two children must have label 1, and if the label of an internal node is 2, both of its children must have label 1.

We can represent it in a concise manner as a problem $\mathcal{C} = \{1 : 12, 2 : 11\}$, where $a : bc$ indicates that a node of label a can have its two children labeled with b and c, in some order. We can take such a description, feed it to the algorithm from [8], and it will output that this problem requires $\Theta(\log n)$ rounds in order to be solved in a rooted tree with n nodes.

1.2 What was missing

What the prior algorithm from [8] can do is classifying a given problem into one of the four main complexity classes $O(1)$, $\Theta(\log^* n)$, $\Theta(\log n)$, and $n^{\Theta(1)}$. However, if the complexity is $n^{\Theta(1)}$, we do not learn whether its complexity is, say, $\Theta(n)$ or $\Theta(\sqrt{n})$ or maybe $\Theta(n^{1/10})$. There are locally checkable problems of complexity $\Theta(n^{1/k})$ for every $k = 1, 2, \ldots$, and there have not been any *practical* algorithm that would determine the value of the exponent k for any given problem.

Furthermore, the algorithm from [8] is only applicable in rooted regular trees, while the case of unrooted trees is perhaps even more interesting.

It has been known that the problem of distinguishing between e.g. $\Theta(n)$ and $\Theta(\sqrt{n})$ is *in principle* decidable, due to the algorithm of [19]. This algorithm is, however, best seen as a theoretical construction. To the best of our knowledge, nobody has implemented it, there are no plans of implementing it, and it seems unlikely that one could classify any nontrivial problem with it using any real-world computer, due to its doubly exponential time complexity. This is the missing piece that we provide in this work.

1.3 Contributions and motivations

We present polynomial-time algorithms that determine not only whether the round complexity of a given LCL problem is $\Theta(n^{1/k})$ for some k, but they also determine the exact value of k. We give one algorithm for the case of unrooted trees (Section 5) and one algorithm for the case of rooted trees (Section 6).

Our algorithms not only determine the asymptotic round complexity, but they also output a description of a distributed algorithm attaining this complexity. If the given LCL problem Π has optimal complexity $\Theta(n^{1/k})$, then our algorithms will output a description of a deterministic distributed algorithm that solves Π in $O(n^{1/k})$ rounds in the CONGEST model. Similarly, if the given LCL problem Π has optimal complexity $O(\log n)$, then our algorithms will output a description of a deterministic distributed algorithm that solves Π in $O(\log n)$ rounds in the CONGEST model.

We have implemented both algorithms for the case of 3-regular trees, the proof-of-concept implementations are freely available online,[1] and they work fast also in practice.

From a practical point of view, together with prior work from [8], there is now a practical algorithm that is able to *completely determine* the complexity of any LCL problem in rooted regular trees.[2] In the case of unrooted regular trees deciding between the lower complexity classes below $o(\log n)$ remains an open question.

From a theoretical point of view, this work *significantly expands* the class of LCL problems whose optimal complexity is known to be decidable in polynomial time. See Figure 1 for a summary of the current state of the art on the classification of LCL complexities for regular trees, showing where the new algorithms are applicable and where the state of the art is given by existing results.

We note that the problem of determining the optimal complexity of an LCL problem is computationally hard in general: It is undecidable in general [36], EXPTIME-hard even for bounded-degree trees [19], and PSPACE-hard even for paths and cycles with input labels [2]. Hence, in order to understand whether polynomial-time algorithms are even possible, we must restrict our consideration to restricted cases, such as LCLs with no inputs defined on regular trees. In fact, it is known that it is possible to use LCLs with no inputs defined on non-regular trees to encode LCLs with

[1] https://github.com/jendas1/poly-classifier

[2] Even though some algorithms in [8] are exponential in the size of the description of the problem, they are nevertheless very efficient in practice. In fact, the authors of [8] have implemented them for the case of binary rooted trees and they are indeed very fast in practice [40].

inputs, and hence, by allowing inputs, or constraints that depend on the degree of the nodes, we would make decidability at least PSPACE-hard.

Motivations Studying LCLs is interesting because, on the one hand, this class of problems is large enough to contain a significant fraction of problems that are commonly studied in the context of the LOCAL model (e.g., $(\Delta+1)$-coloring, $(2\Delta-1)$-edge coloring, Δ-coloring, weak 2-coloring, maximal matching, maximal independent set, sinkless orientation, many other orientation problems, edge splitting problems, locally maximal cut, defective colorings, ...), but, on the other hand, it is restricted enough so that we can prove interesting results about them, such as decidability and complexity gaps. Moreover, techniques used to prove results on LCLs have been already shown to be extremely useful outside the LCL context: for example, all recent results about lower bounds for locally checkable problems in the unbounded degree case—e.g., for MIS, maximal matching, ruling sets, and other fundamental problems—use techniques that originally were introduced in the context of LCLs [4, 5, 6, 7, 18].

In this work, we restrict our attention to the case of regular trees. The study of LCLs on trees is related with our understanding of graph problems in the general setting. Actually, for many problems of interest, unrooted regular trees are hard instances, and hence understanding the complexity of LCLs on trees could help us in understanding the complexity of problems in general unbounded-degree graphs. In fact, a relatively new and promising technique called *round elimination* has been used to prove tight lower bounds for interesting graph problems such as maximal matchings, maximal independent sets, and ruling sets, even if, for now, we are only able to apply this technique for proving lower bounds on trees [3, 4, 5, 6, 7, 14, 18, 37].

As for the more restrictive setting of *regular* trees, we would like to point out that many natural LCL problems have the same optimal complexity in both bounded-degree trees and regular trees. This includes, for example, the k-coloring problem. For any tree T whose maximum degree is at most Δ, we may consider the Δ-regular tree T^* which is the result of appending degree-1 nodes to all nodes v in T with $1 < \deg(v) < \Delta$ to increase the degree of v to Δ. We may locally simulate T^* in the network T. As any proper k-coloring of T^* restricting to T is also a proper k-coloring, this reduces the k-coloring problem on bounded-degree trees to the same problem on regular trees, showing that the k-coloring problem has the same optimal complexity in both graph classes. More generally, if an LCL problem Π has the property that removing degree-1 nodes preserves the correctness of a solution, then Π has the same optimal complexity in both bounded-degree trees and regular trees, so our results in this work also apply to these LCLs on bounded-degree trees.

2 Related work

Locally Checkable Labeling problems have been introduced by Naor and Stockmeyer [36], but the class of locally checkable problems has been studied in the distributed setting even before (e.g., in the context of self-stabilisation [1]). For many locally checkable problems, researchers have been trying to understand the exact time complexity, and while in many cases upper bounds have been known since the 80s, matching lower bound have been discovered only recently. Examples of this line of research relate to the problems of colorings, matchings, and independent sets, see e.g. [4, 6, 24, 25, 27, 28, 31, 32, 33, 38, 39].

In parallel, there have been many works that tried to understand these problems from a *complexity theory* point of view, trying to develop general techniques for classifying problems, understanding which complexities can actually exist, and developing generic algorithmic techniques to

solve whole classes of problems at once. In particular, a broad class[3] of locally checkable problems, called *Locally Checkable Labelings* (LCLs), has been studied in the LOCAL model of distributed computing, which will be formally defined later.

Paths and cycles The first graph topologies on which promising results have been proved are paths and cycles. In these graphs, we now know that there are problems with the following *three* possible time complexities:

- $O(1)$: this class contains, among others, trivial problems, e.g. problems that require every node to output the same label.
- $\Theta(\log^* n)$: this class contains, for example, the 3-coloring problem [24, 31, 35].
- $\Theta(n)$: this class contains hard problems, for example the problem of consistently orient the edges of a cycle, or the 2-coloring problem.

For LCLs in paths and cycles, we know that there are no other possible complexities, that is, there are *gaps* between the above classes. In other words, there are no LCLs with a time complexity that lies between $\omega(1)$ and $o(\log^* n)$ [36], and no LCLs with a time complexity that lies between $\omega(\log^* n)$ and $o(n)$ [20]. These results hold also for *randomized* algorithms, and they are constructive: if for example we find a way to design an $O(\log n)$-rounds randomized algorithm for a problem, then we can automatically convert it into an $O(\log^* n)$-round deterministic algorithm.

Moreover, in paths and cycles, given an LCL problem, we can *decide* its time complexity. In particular, it turns out that for problems with no inputs defined on directed cycles, deciding the complexity of an LCL is as easy as drawing a diagram and staring at it for few seconds [17]. This result has later been extended to undirected cycles with no inputs [22]. Unfortunately, as soon as we consider LCLs where the constraints of the problem may depend on the given inputs, decidability becomes much harder, and it is now known to be PSPACE-hard [2], even for paths and cycles.

Trees Another class of graphs that has been studied quite a lot is the one containing *trees*. While there are still problems with complexities $O(1)$, $\Theta(\log^* n)$, and $\Theta(n)$, there are also additional complexity classes, and sometimes here randomness can help. For example, there are problems that require $\Theta(\log n)$ rounds for both deterministic and randomized algorithms, while there are problems, like *sinkless orientation*, that require $\Theta(\log n)$ rounds for deterministic algorithms and $\Theta(\log \log n)$ rounds for randomized ones [16, 20, 29]. Moreover, there are problems with complexity $\Theta(n^{1/k})$, for any natural number $k \geq 1$ [21]. It is known that these are the only possible time complexities in trees [10, 13, 19, 20, 21, 30]. In [11], it has been shown that the same results hold also in a more restrictive model of distributed computing, called CONGEST model, and that for any given problem, its complexities in the LOCAL and in the CONGEST model, on trees, are actually the same.

Concerning decidability, the picture is not as clear as in the case of paths and cycles. As discussed in the introduction, it is decidable, *in theory*, if a problem requires $n^{\Omega(1)}$ rounds, and, in that case, it is also decidable to determine the exact exponent [19, 21], but the algorithm is very far from being practical, and in this work we address exactly this issue. Moreover, for *lower* complexities, the problem is still open. Different works tried to tackle this issue by considering restricted cases. In [3], authors showed that it is indeed possible to achieve decidability in some cases, that is, when problems are restricted to the case of unrooted regular trees, where leaves are unconstrained, and the problem uses only *two* labels. Then, promising results have been achieved

[3] For example, our definition of LCL does not allow an infinite number of labels, so it does not capture some locally checkable problems such as fractional matching.

in [8], where it has been shown that, if we consider *rooted* trees, then we can decide the complexity of LCLs even for $n^{o(1)}$ complexities. Unfortunately, it is very unclear if such techniques can be used to solve the problem in the general case. In fact, we still do not know if it is decidable whether a problem can be solved in $O(1)$ rounds or it requires $\Omega(\log^* n)$ rounds, and it is not known if it is decidable whether a problem can be solved in $O(\log^* n)$ rounds or it requires $\Omega(\log n)$ for deterministic algorithms and $\Omega(\log \log n)$ for randomized ones. These two questions are very important, and understanding them may also help in understanding problems that are not restricted to regular trees of bounded degree. This is because, as already mentioned before, for many problems it happens that unrooted regular trees are hard instances, and studying the complexity of problems in these instances may give insights for understanding problems in the general setting.

General graphs In general graphs, many more LCL complexities are possible. For example, there is a gap similar to the one between $\omega(1)$ and $o(\log^* n)$ of trees, but now it holds only up to $o(\log \log^* n)$, and we know that there are problems in the region between $\Omega(\log \log^* n)$ and $o(\log^* n)$. In fact, for any rational $\alpha \geq 1$, it is possible to construct problems with complexity $\Theta(\log^\alpha \log^* n)$ [12]. A similar statement holds for complexities between $\Omega(\log n)$ and $O(n)$ [10, 12].

There are still complexity regions in which we do not know if there are problems or not. For example, while it is known that any problem that has randomized complexity $o(\log n)$ can be sped up to $O(T_{\mathrm{LLL}})$ [21], where T_{LLL} is the distributed complexity of the constructive version of the Lovász LOCAL Lemma, the exact value of T_{LLL} is unknown, and we only known that it lies between $\Omega(\log \log n)$ and $O(\mathrm{poly} \log \log n)$ [16, 23, 26, 39]. Another problem that falls in this region is the Δ-coloring problem, for which we still do not know the exact complexity.

Another open question regards the role of randomness. In general graphs, we know that randomness can also help outside the $O(\log n)$ region [9], but we still do not know exactly when it can help and how much.

In general graphs, unfortunately, determining the complexity of a given LCL problem is undecidable. In fact, we know that this question is undecidable even on grids [36].

3 Preliminaries

Graphs Let $G = (V, E)$ be a graph. We denote with $n = |V|$ the number of nodes of G, with Δ the maximum degree of G, and with $\deg(v)$, for $v \in V$, the degree of v. If G is a directed graph, we denote with $\deg_{\mathsf{in}}(v)$ and $\deg_{\mathsf{out}}(v)$, the indegree and the outdegree of v, respectively. The radius-r neighborhood of a node v is defined to be the subgraph of G induced by the nodes at distance at most r from v.

Model of computing In the LOCAL model of distributed computing, the network is represented with a graph $G = (V, E)$, where the nodes correspond to computational entities, and the edges correspond to communication links. In this model, the computational power of the nodes is unrestricted, and nodes can send arbitrarily large messages to each other.

This model is synchronous, and computation proceeds in rounds. Nodes all start the computation at the same time, and at the beginning they know n (the total number of nodes), Δ (the maximum degree of the graph), and a unique ID in $\{1, \ldots, n^c\}$, for some constant $c \geq 1$, assigned to them. Then, the computation proceeds in rounds, and at each round nodes can send (possibly different) messages to each neighbor, receive messages, and perform some LOCAL computation.

At the end of the computation, each node must produce its own part of the solution. For example, in the case of the $(\Delta + 1)$-coloring problem, each node must output its own color, that

must be different from the ones of its neighbors. The time complexity is measured as the worst case number of rounds required to terminate, and it is typically expressed as a function of n, Δ, and c.

4 Technical overview

Our new results build on several techniques developed in previous works [8, 22] designing polynomial-time algorithms that determine the distributed complexity of LCL problems. In this section, we first give a brief overview of these techniques, then we discuss how in this paper we build upon them and obtain our new results. The aim of this section is to present the intuition behind the results. To keep the discussion at a high level, the presentation here will be a bit imprecise, see Sections 5 and 6 for the precise statements of our results.

4.1 The high-level framework

Existing algorithms for deciding the complexity of a given LCL problem are often based on the following approach.

1. Define some combinatorial property P of LCL problems.
2. Show that computing $P(\Pi)$ for a given problem Π can be done efficiently.
3. Show that Π is in a certain complexity class if and only if $P(\Pi)$ holds.

As discussed in [17, 22], any LCL Π on directed paths can be viewed as a regular language. Taking the corresponding non-deterministic automaton, we obtain a directed graph $G(\Pi)$ that represents Π on directed paths.

For example, the maximal independent set problem can be described as the automaton with states $V = \{00, 01, 10\}$ and transitions $E = \{00 \to 01, 01 \to 10, 10 \to 00, 10 \to 01\}$. Each state corresponds to a possible labeling of the two endpoints u and v of a directed edge $u \to v$. Each transition describes a valid configuration of two neighboring directed edges $u \to v$ and $v \to w$.

It has been shown [8, 15, 17, 22] that in several cases the distributed complexity of an LCL can be characterized by simple graph properties of $G(\Pi)$, even if the underlying graph class is much more complicated than directed paths. The precise definition of $G(\Pi)$ will depend on the choice of the LCL formalism.

4.2 Paths and cycles

It was shown in [17, 22] that the distributed complexity and solvability of Π on paths and cycles can be characterized by simple graph properties of $G(\Pi)$. In particular, Π on directed cycles is solvable in $O(\log^* n)$ rounds if and only if $G(\Pi)$ contains a node v that is *path-flexible*, in the sense that there exists a number K such that, in $G(\Pi)$, there is a length-k returning walk for v, for each $k \geq K$. If such a path-flexible node v exists in $G(\Pi)$, then Π on directed cycles can be solved in $O(\log^* n)$ rounds in the following manner.

1. In $O(\log^* n)$ rounds, compute an independent set I such that the distance between the nodes in I is at least K and at most $2K$.
2. Fix the labels for the nodes in I according to the path-flexible node v in $G(\Pi)$.

3. By the path-flexibility of v, this partial labeling can be completed into a correct complete labeling.

For example, in the automaton for maximal independent set described above, the state 01 is flexible, as for each $k \geq 5$, there is a length-k walk starting and ending at 01, so a maximal independent set can be found in $O(\log^* n)$ rounds on directed cycles via the above algorithm.

The above characterization can be generalized to both paths and cycles, undirected and directed, after some minor modifications, see [22] for the details. For further examples of representing LCLs as automata and how the round complexity of an LCL can be inferred from basic properties of its associated automaton, see [17, Fig. 3] and [22, Fig. 1 and 3].

4.3 The $O(\log n)$ complexity class in regular trees

Subsequently, it was shown in [8, 15] that the class of $O(\log n)$-round solvable LCL problems on rooted and unrooted regular trees can be characterized in a similar way, based on the notion of path-flexibility in the directed graph $G(\Pi)$. To keep the discussion at a high level, we do not discuss the difference between rooted and unrooted trees here. Roughly speaking, Π can be solved in $O(\log n)$ rounds on rooted or unrooted regular trees if and only if there exists a subset of labels S such that, if we restrict Π to S, then its corresponding directed graph is strongly connected and contains a path-flexible node. Such a set S of labels is also called a *certificate* for $O(\log n)$-round solvability.[4]

A key property of such a directed graph is that there exists a number K such that, for each pair of nodes (u, v), and for each integer $k \geq K$, there is a length-k walk from u to v (here we allow the possibility of $u = v$). The property can be described in the following more intuitive manner. For any path of length at least K, regardless of how we fix the labels of its two endpoints using S, it is always possible to complete the partial labeling into a correct labeling w.r.t. Π of the entire path using only labels in S.

The intuition behind such a characterization is the fact [21] that all LCLs solvable in $O(\log n)$ rounds on bounded-degree trees can be solved in a canonical way based on *rake-and-compress decompositions*. Roughly speaking, a rake-and-compress process is a procedure that decomposes a tree by iteratively removing degree-1 nodes (rake) and removing degree-2 nodes (compress). This process partitions the set of nodes into several parts:

$$V = V_1^{\mathsf{R}} \cup V_1^{\mathsf{C}} \cup V_2^{\mathsf{R}} \cup V_2^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}},$$

where V_i^{R} is the set of nodes removed by the rake operation in the ith iteration and V_i^{C} is the set of nodes removed by the compress operation in the ith iteration. It can be shown that $L = O(\log n)$ [34].

There are several variants of a rake-and-compress process. Here the considered variant is such that, in the compress operation, a degree-2 node v is removed if v belongs to a path whose length is at least ℓ, so we may assume that the connected components in the subgraph induced by V_i^{C} are paths with length at least ℓ.

Let Π be any LCL problem satisfying the combinatorial characterization for $O(\log n)$-round solvability discussed above, and let the set of labels S be a certificate for $O(\log n)$-round solvability. By setting $\ell = K$ in the property of the combinatorial characterization, we may obtain an $O(\log n)$-round algorithm solving the given LCL problem Π using only the labels in S. The high-level

[4] Although the certificate described in [8] also includes the steps in the construction of S, the set S alone suffices to certify that Π can be solved in $O(\log n)$ rounds, as the $O(\log n)$-round algorithm described in [8] uses only S.

idea is that we can label the tree in an order that is the reverse of the one of the rake-and-compress procedure: $V_L^{\mathsf{R}}, \ldots, V_2^{\mathsf{C}}, V_2^{\mathsf{R}}, V_1^{\mathsf{C}}, V_1^{\mathsf{R}}$, as we observe that the property of the combinatorial characterization discussed above ensures that any correct labeling of $V_L^{\mathsf{R}} \cup \cdots \cup V_i^{\mathsf{R}}$ can be extended to a correct labeling of $V_L^{\mathsf{R}} \cup \cdots \cup V_i^{\mathsf{R}} \cup V_{i-1}^{\mathsf{C}}$ and similarly any correct labeling of $V_L^{\mathsf{R}} \cup \cdots \cup V_i^{\mathsf{C}}$ can be extended to a correct labeling of $V_L^{\mathsf{R}} \cup \cdots \cup V_i^{\mathsf{C}} \cup V_i^{\mathsf{R}}$.

The requirement that Π is an LCL problem defined on *regular* trees is *critical* in the above approach, as this requirement ensures that for each non-leaf node, the set of constraints is the same, so we do not need to worry about the possibility for different nodes in the tree to have different sets of constraints in Π. Indeed, if we allow nodes of different degrees to have different sets of constraints, then the problem of determining the distributed complexity of an LCL in bounded-degree trees becomes EXPTIME-hard [19].

4.4 The polynomial complexity region in regular trees

In this work, we will extend the above approach to cover all complexity classes in the $[\Theta(\log n), \Theta(n)]$ region. By [10, 19, 21], we know that the possible complexity classes in this region are $\Theta(\log n)$ and $\Theta(n^{1/k})$ for all positive integers k. Similar to the complexity class $O(\log n)$, any LCL problem Π solvable in $O(n^{1/k})$ rounds can be solved in a canonical way in $O(n^{1/k})$ rounds using a variant of rake-and-compress decomposition [19].

Specifically, Π is $O(n^{1/k})$-round solvable if and only if it can be solved in a canonical way using a rake-and-compress decomposition, where in each iteration, we perform $\gamma = O(n^{1/k})$ rake operations and one compress operation. Similar to the case of complexity class $O(\log n)$, in the compress operation, a degree-2 node v is removed if v belongs to a path whose length is at least ℓ, where $\ell = O(1)$ is some sufficiently large number depending only on the LCL problem Π. It can be shown [19] that by selecting $\gamma = O(n^{1/k})$ to be large enough, the number of layers L in the decomposition $V = V_1^{\mathsf{R}} \cup V_1^{\mathsf{C}} \cup V_2^{\mathsf{R}} \cup V_2^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$ is k, and such a decomposition can be computed in $O(n^{1/k})$ rounds.

To derive a certificate for $O(n^{1/k})$-round solvability based on the result of [19], we will need to take into consideration the following properties about the variant of the rake-and-compress decomposition described above.

- The number of layers $L = k$ is now a *finite* number independent of the size of the graph n. For technical reasons, this means that the certificate for $O(n^{1/k})$-round solvability cannot be based on a single set of labels S, as the certificate for $O(\log n)$-round solvability [8, 15]. We need to consider the possibility that different sets of labels are used for different layers in the design of the certificate for $O(n^{1/k})$-round solvability.
- The number of rake operations for a layer can be unbounded as n goes to infinity. That is, V_i^{R} is no longer an independent set, and each connected component in the subgraph induced by V_i^{R} can be a very large tree.

The certificate Our certificate for $O(n^{1/k})$-round solvability will be based on the notion of a *good* sequence of sets of labels. The definition of a good sequence relies on two functions on a set of labels: trim and $\mathsf{flexible\text{-}SCC}$. As we will later see, these two functions correspond to rake and compress, respectively. Given an LCL problem Π and a set of labels S, $\mathsf{trim}(S)$ and $\mathsf{flexible\text{-}SCC}$ are defined as follows.

- $\mathsf{trim}(S)$ is the subset of S resulting from removing all labels $\sigma \in S$ meeting the following conditions: There exists some number i such that if the root of the complete regular tree T of

height i is labeled by σ, then we are not able to complete the labeling of T using only labels in S such that the overall labeling is correct w.r.t. Π.

- $\mathsf{flexible\text{-}SCC}(S)$ is a collection of disjoint subsets of S defined as follows. Consider the directed graph representing the LCL problem Π restricted to S. Let $\mathsf{flexible\text{-}SCC}(S)$ be the set of strongly connected components that have a path-flexible node. The intuition behind this definition is similar to the intuition behind the certificate for $O(\log n)$-round solvability.

We briefly explain the connection between trim and rake. Suppose we want to find a correct labeling of a regular tree T using only the labels in S. If a label σ is in $\mathsf{trim}(S)$, then σ can only be used in places that are sufficiently close to a leaf. To put it another way, if we do a large number of rakes to T, then the labels in $\mathsf{trim}(S)$ can only be used to label the nodes that removed due to a rake operation.

The connection between $\mathsf{flexible\text{-}SCC}$ to compress is due to the fact that the nodes removed due to a compress operation form long paths, and we know that in order to label long paths efficiently in $O(\log^* n)$ rounds, it is necessary to use labels corresponding to path-flexible nodes, due to the existing automata-theoretic characterization [17, 22] of round complexity of LCLs on paths and cycles.

We say that a sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \dots, \Sigma_k^{\mathsf{R}})$ is *good* if it satisfies the following rules, where Σ is the set of all labels of Π.

$$\Sigma_i^{\mathsf{R}} = \begin{cases} \mathsf{trim}(\Sigma) & \text{if } i = 1, \\ \mathsf{trim}(\Sigma_{i-1}^{\mathsf{C}}) & \text{if } i > 1. \end{cases}$$
$$\Sigma_i^{\mathsf{C}} \in \mathsf{flexible\text{-}SCC}(\Sigma_i^{\mathsf{R}}).$$
$$\Sigma_k^{\mathsf{R}} \neq \emptyset.$$

The only nondeterminism in the above rules is the choice of $\Sigma_i^{\mathsf{C}} \in \mathsf{flexible\text{-}SCC}(\Sigma_i^{\mathsf{R}})$ for each i. We will show that such a sequence exists if and only if the underlying LCL problem can be solved in $O(n^{1/k})$ rounds. Intuitively, Σ_i^{R} represents the set of labels that are eligible to label the nodes in V_i^{R}, and similarly Σ_i^{C} represents the set of labels that are eligible to label the nodes in V_i^{C}.

The classification The notion of a good sequence allows us to classify the complexity classes in the region $[\Theta(\log n), \Theta(n)]$. Specifically, we define the *depth* d_Π of an LCL problem Π as the largest k such that a good sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \dots, \Sigma_k^{\mathsf{R}})$ exists. If there is no good sequence, then we set $d_\Pi = 0$. If there is a good sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \dots, \Sigma_k^{\mathsf{R}})$ for each positive integer k, then we set $d_\Pi = \infty$. We will show that d_Π characterizes the distributed complexity of Π in the following manner.

- If $d_\Pi = 0$, then Π is unsolvable in the sense that there exists a regular tree such that there is no correct solution of Π on this rooted tree. This follows from the definition of trim and the observation that $d_\Pi = 0$ if $\mathsf{trim}(\Sigma) = \emptyset$.
- If $d_\Pi = k$ is a positive integer, then the distributed complexity of Π is $\Theta(n^{1/k})$.
- If $d_\Pi = \infty$, then Π can be solved in $O(\log n)$ rounds. If we can have a good sequence that is arbitrarily long, then there must be a *fixed point* S in the sequence such that $\mathsf{trim}(S) = S$ and $\mathsf{flexible\text{-}SCC}(S) = \{S\}$, because $\Sigma_1^{\mathsf{R}} \supseteq \Sigma_1^{\mathsf{C}} \supseteq \cdots \supseteq \Sigma_k^{\mathsf{R}}$. We will show that the fixed point S qualifies to be a certificate for $O(\log n)$-round solvability.

The fixed point phenomenon explains why the notion of good sequence was not needed in [8, 15], as the existence of a fixed point for the case Π is $O(\log n)$-round solvable implies that we may

apply the same strategy according to the fixed point to label each layer of the rake-and-compress decomposition to solve Π in $O(\log n)$ rounds.

The proof ideas To show the correctness and efficiency of our characterization, we need to do the following.

Upper bound: Given a good sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_k^{\mathsf{R}})$, show that there exists an $O(n^{1/k})$-round algorithm solving Π. Therefore, $d_\Pi = k$ implies $O(n^{1/k})$-round solvability.

Lower bound: Given an $o(n^{1/k})$-round algorithm solving Π, show that a good sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_{k+1}^{\mathsf{R}})$ exists. Therefore, $d_\Pi = k$ implies $\Omega(n^{1/k})$-round solvability.

Efficiency: Design a polynomial-time algorithm that computes d_Π for any given description of an LCL problem Π.

The upper bound proof is relatively simple. Similar to the certificate $O(\log n)$-round solvability, we just need to show that Π can be solved in $O(n^{1/k})$ rounds using rake-and-compress decompositions given that a good sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_k^{\mathsf{R}})$ exists.

The lower bound proof is much more complicated. Given an algorithm $\mathcal{A}$ solving Π in $t = o(n^{1/k})$ rounds, we will consider a tree G that is a result of a hierarchical combination of complete trees and paths of length greater than t. Intuitively, G is chosen to be the fullest possible tree that can be partitioned into $V = V_1^{\mathsf{R}} \cup V_1^{\mathsf{C}} \cup V_2^{\mathsf{R}} \cup V_2^{\mathsf{C}} \cup \cdots \cup V_{k+1}^{\mathsf{R}}$ with a rake-and-compress decomposition of [19] with $L = k+1$ layers. We will prove by induction that if we take Σ_i^{R} to be the set of possible output labels of $\mathcal{A}$ for $V_i^{\mathsf{R}} \cup V_i^{\mathsf{C}} \cup \cdots \cup V_{k+1}^{\mathsf{R}}$ and take Σ_i^{C} to be the set of possible output labels of $\mathcal{A}$ for $V_i^{\mathsf{C}} \cup V_{i+1}^{\mathsf{R}} \cup \cdots \cup V_{k+1}^{\mathsf{R}}$, then $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_{k+1}^{\mathsf{R}})$ must be a good sequence. In particular, the non-emptiness of $\Sigma_{k+1}^{\mathsf{R}}$ follows from the correctness of $\mathcal{A}$.

To design a polynomial-time algorithm computing d_Π, we recall that the only nondeterminism in the rules for a good sequence is the choice of $\Sigma_i^{\mathsf{C}} \in \mathsf{flexible\text{-}SCC}(\Sigma_i^{\mathsf{R}})$, so we will just do a brute-force search for all possibilities. Although this seems very inefficient, we recall that $\mathsf{flexible\text{-}SCC}(\Sigma_i^{\mathsf{R}})$ is a collection of disjoint subsets of Σ_i^{R}, so the summation of the size of all sets of labels considered in each level is at most the total number of labels $|\Sigma|$ in Π. The number of levels we need to explore is also bounded, as $\Sigma_1^{\mathsf{R}} \supseteq \Sigma_1^{\mathsf{C}} \supseteq \cdots \supseteq \Sigma_k^{\mathsf{R}}$. If k exceeds $|\Sigma|$, then we know that there is a fixed point Σ_i^{R} such that $\Sigma_i^{\mathsf{R}} = \Sigma_i^{\mathsf{C}} = \Sigma_{i+1}^{\mathsf{R}} = \Sigma_{i+1}^{\mathsf{C}} = \cdots$, so $d_\Pi = \infty$.

The differences between rooted and unrooted trees The high-level proof strategy presented in this technical overview applies to both rooted and unrooted regular trees, showing that these two graph classes behave very similarly in the complexity region $[\Theta(\log n), \Theta(n)]$. There are still some technical differences between rooted and unrooted trees.

- The formalisms for representing LCL problems are different for rooted and unrooted trees. In the case of rooted trees, the problem can refer to orientations. For example, what is permitted for a parent can be different from what is permitted for a child. Instead of specifying node and edge configurations, we follow [8] and specify what are permitted multisets of child labels for each node label.
- For the upper bound, we need to generalize the rake-and-compress decomposition of [19] so that it is applicable in rooted trees.
- For the lower bound, the lower bound graph for unrooted trees does not work for the rooted trees. Roughly speaking, this is because the presence of edge orientation increases the symmetry breaking capability of nodes, so some indistinguishability arguments in the lower bound

proof for unrooted trees do not work for rooted trees. Therefore, we will need to consider a different approach for crafting the lower bound graph for rooted trees.

5 Unrooted trees

In this section, we give a polynomial-time-computable characterization of LCL problems for regular unrooted trees with complexity $O(\log n)$ or $\Theta(n^{1/k})$ for any positive integer k.

5.1 Locally checkable labeling for unrooted trees

A Δ-regular tree is a tree where the degree of each node is either 1 or Δ. An LCL problem for Δ-regular unrooted trees is defined as follows.

Definition 5.1 (LCL problems for regular unrooted trees)**.** *For unrooted trees, an* LCL *problem* $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ *is defined by the following components.*

- Δ *is a positive integer specifying the maximum degree.*
- Σ *is a finite set of labels.*
- $\mathcal{V}$ *is a set of size-*Δ *multisets of labels in* Σ *specifying the node constraint.*
- $\mathcal{E}$ *is a set of size-*2 *multisets of labels in* Σ *specifying the edge constraint.*

We call a size-Δ multiset C of labels in Σ a *node configuration*. A node configuration C is correct with respect to $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ if $C \in \mathcal{V}$. We call a size-2 multiset D of labels in Σ an *edge configuration*. An edge configuration D is correct with respect to $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ if $D \in \mathcal{E}$. We define the correctness criteria for a labeling of a Δ-regular tree in Definition 5.2.

Definition 5.2 (Correctness criteria)**.** *Let* $G = (V, E)$ *be a tree whose maximum degree is at most* Δ*. For each edge* $e = \{u, v\}$ *in the tree, there are two half-edges* (u, e) *and* (v, e)*. A solution of* $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ *on* G *is a labeling that assigns a label in* Σ *to each half-edge in* G*.*

- *For each node* $v \in V$ *with* $\deg(v) = \Delta$ *its node configuration* C *is the multiset of* Δ *half-edge labels of* (v, e_1)*,* (v, e_2)*, . . . ,* (v, e_Δ)*, where* $e_1, e_2, \ldots, e_\Delta$ *are the* Δ *edges incident to* v*. We say that the labeling is locally-consistent on* v *if* $C \in \mathcal{V}$*.*
- *For each edge* $e = \{u, v\} \in E$*, its edge configuration* D *is the multiset of two half-edge labels of* (u, e) *and* (v, e)*. We say that the labeling is locally-consistent on* e *if* $D \in \mathcal{E}$*.*

The labeling is a correct solution if it is locally-consistent on all $v \in V$ *with* $\deg(v) = \Delta$ *and all* $e \in E$*.*

In other words, a labeling of $G = (V, E)$ is correct if the edge configuration for each $e \in E$ is correct and the node configuration for each $v \in V$ with $\deg(v) = \Delta$ is correct. All nodes whose degree is not Δ are unconstrained.

Although $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ is defined for Δ-regular unrooted trees, Definition 5.2 applies to all trees whose maximum degree is at most Δ. We emphasize that all nodes v whose degree is not Δ are *unconstrained* in that there is no requirement about the node configuration of v. Nevertheless, we may focus on Δ-regular unrooted trees without loss of generality. The reason is that for any unrooted tree G whose maximum degree is at most Δ, we may consider the unrooted tree G^* which is the result of appending degree-1 nodes to all nodes v in G with $1 < \deg(v) < \Delta$ to increase the degree of v to Δ. This only blows up the number of nodes by at most a Δ factor. We claim that the asymptotic optimal round complexity of Π is the same in both G and G^*. Any correct solution of Π on G^* restricted to G is a correct solution of Π on G, as all nodes whose degree is not Δ

are unconstrained. Therefore, if we have an algorithm for Π in Δ-regular unrooted trees, then the same algorithm also allows us to solve Π in unrooted trees with maximum degree Δ in the same asymptotic round complexity.

Definition 5.3 (Complete trees of height i)**.** *We define the rooted trees T_i and T_i^* recursively as follows.*

- *T_0 is the trivial tree with only one node.*
- *T_i is the result of appending $\Delta - 1$ trees T_{i-1} to the root r.*
- *T_i^* is the result of appending Δ trees T_{i-1} to the root r.*

Observe that T_i^* is the unique maximum-size tree of maximum degree Δ and height i. All nodes within distance $i-1$ to the root r in T_i^* have degree Δ. All nodes whose distance to r is exactly i are degree-1 nodes. Although T_i and T_i^* are defined as rooted trees, they can also be viewed as unrooted trees.

Definition 5.4 (Trimming)**.** *Given an* LCL *problem $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ and a subset $\mathcal{S} \subseteq \mathcal{V}$ of node configurations, we define $\mathsf{trim}(\mathcal{S})$ as the set of all node configurations $C \in \mathcal{S}$ such that for each $i \geq 1$ it is possible to find a correct labeling of T_i^* such that the node configuration of the root is C and the node configurations of the remaining degree-Δ nodes are in $\mathcal{S}$.*

In the definition, note that if for some $i \geq 1$ it is not possible to find such a labeling of T_i^*, then it is also not possible for any larger i. The reason is that if such a labeling for larger i exists, then by taking subgraph, we obtain such a labeling for of T_i^*. Here we use the fact that nodes by taking subgraph, and using the fact that all nodes whose degree is not Δ are unconstrained.

Intuitively, $\mathsf{trim}(\mathcal{S})$ is the subset of $\mathcal{S}$ resulting from removing all node configurations in $\mathcal{S}$ that are not usable in a correct labeling of a sufficiently large Δ-regular tree using only node configurations in $\mathcal{S}$.

In fact, given any tree G of maximum degree Δ and a node v of degree Δ in G, after labeling the half-edges surrounding v using a node configuration in $\mathsf{trim}(\mathcal{S})$, it is always possible to extend this labeling to a complete correct labeling of G using only node configurations in $\mathsf{trim}(\mathcal{S})$. Such a labeling extension is possible due to Lemma 5.1.

Lemma 5.1 (Property of trimming)**.** *Let $\mathcal{S} \subseteq \mathcal{V}$ such that $\mathsf{trim}(\mathcal{S}) \neq \emptyset$. For each node configuration $C \in \mathsf{trim}(\mathcal{S})$ and each label $\sigma \in C$, there exist a node configuration $C' \in \mathsf{trim}(\mathcal{S})$ and a label $\sigma' \in C'$ such that the multiset $\{\sigma, \sigma'\}$ is in $\mathcal{E}$.*

Proof. Assuming that such C' and σ' do not exist, we derive a contradiction as follows. We pick s to be the smallest number such that there is no correct labeling of T_s^* where the node configuration of the root r is in $\mathcal{S} \setminus \mathsf{trim}(\mathcal{S})$ and the node configuration of each remaining degree-Δ node of T_s^* is in $\mathcal{S}$. Such a number s exists due to the definition of trim.

Now consider a correct labeling of T_{s+1}^* where the node configuration of the root r is C and the node configuration of each remaining degree-Δ node is in $\mathcal{S}$. Such a correct labeling exists due to the fact that $C \in \mathsf{trim}(\mathcal{S})$. Our assumption on the non-existence of C' and σ' implies that the node configuration $\tilde{C}$ of one child w of the root r of T_{s+1}^* must be in $\mathcal{S} \setminus \mathsf{trim}(\mathcal{S})$. However, the radius-$s$ neighborhood of w in T_{s+1}^* is isomorphic to T_s^* rooted at w. Since the node configuration of w is in $\mathcal{S} \setminus \mathsf{trim}(\mathcal{S})$, our choice of s implies that the labeling of the radius-s neighborhood of w cannot be correct, which is a contradiction. □

Path-form of an LCL problem Given an LCL problem $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ and a subset $\mathcal{S} \subseteq \mathcal{V}$ of node configurations, we define

$$\mathcal{D}_{\mathcal{S}} = \text{the set of all size-2 multisets } D \text{ such that } D \text{ is a sub-multiset of } C \text{ for some } C \in \mathcal{S}.$$

To understand the intuition behind the definition $\mathcal{D}_{\mathcal{S}}$, define the length-$k$ hairy path H_k as the result obtained by starting from a length-k path $P = (v_1, v_2, \ldots, v_{k+1})$ and then adding degree-1 nodes to make $\deg(v_i) = \Delta$ for all $1 \leq i \leq k+1$. If our task is to label hairy paths using node configurations in $\mathcal{S}$, then this task is identical to labeling paths using node configurations in $\mathcal{D}_{\mathcal{S}}$. In other words, the LCL problem $(\Delta, \Sigma, \mathcal{S}, \mathcal{E})$ on hairy paths is equivalent to the LCL problem $(2, \Sigma, \mathcal{D}_{\mathcal{S}}, \mathcal{E})$ on paths. Hence $(2, \Sigma, \mathcal{D}_{\mathcal{S}}, \mathcal{E})$ is the *path-form* of $(\Delta, \Sigma, \mathcal{S}, \mathcal{E})$.

Automaton for the path-form of an LCL problem Given a set $\mathcal{D}$ of size-2 multisets whose elements are in Σ, we define the directed graph $\mathcal{M}_{\mathcal{D}}$ as follows. The node set $V(\mathcal{M}_{\mathcal{D}})$ of $\mathcal{M}_{\mathcal{D}}$ is the set of all pairs $(a, b) \in \Sigma^2$ such that the multiset $\{a, b\}$ is in $\mathcal{D}$. The edge set $E(\mathcal{M}_{\mathcal{D}})$ of $\mathcal{M}_{\mathcal{D}}$ is defined as follows. For any two pairs $(a, b) \in V(\mathcal{M}_{\mathcal{D}})$ and $(c, d) \in V(\mathcal{M}_{\mathcal{D}})$, we add a directed edge $(a, b) \to (c, d)$ if the multiset $\{b, c\}$ is an edge configuration in $\mathcal{E}$. Note that $\mathcal{M}_{\mathcal{D}}$ could contain self-loops.

The motivation for considering $\mathcal{M}_{\mathcal{D}}$ is that it can be seen as an automaton recognizing the correct solutions for the LCL problem $(2, \Sigma, \mathcal{D}, \mathcal{E})$ on paths, as each length-k walk $(a_1, b_1) \to (a_2, b_2) \to \cdots \to (a_{k+1}, b_{k+1})$ of $\mathcal{M}_{\mathcal{D}}$ corresponds to a correct labeling of a length-k path $(v_1, v_2, \ldots, v_{k+1})$ where the labeling of half-edge $(v_i, \{v_{i-1}, v_i\})$ is a_i and the labeling of half-edge $(v_i, \{v_i, v_{i+1}\})$ is b_i.

Path-flexibility With respect to the directed graph $\mathcal{M}_{\mathcal{D}}$, we say that $(a, b) \in V(\mathcal{M}_{\mathcal{D}})$ is path-flexible if there exists an integer K such that for each integer $k \geq K$, there exist length-k walks $(a, b) \rightsquigarrow (a, b)$, $(a, b) \rightsquigarrow (b, a)$, $(b, a) \rightsquigarrow (a, b)$, and $(b, a) \rightsquigarrow (b, a)$ in $\mathcal{M}_{\mathcal{D}}$. Throughout this paper, we write $u \rightsquigarrow v$ to denote a walk starting from u and ending at v.

It is clear that (a, b) is path-flexible if and only if (b, a) is path-flexible. Hence we may extend the notion of path-flexibility from $V(\mathcal{M}_{\mathcal{D}})$ to $\mathcal{D}$. That is, we say that a size-2 multiset $\{a, b\} \in \mathcal{D}$ is path-flexible if (a, b) is path-flexible.

The following lemma is useful in lower bound proofs. For any $\{a, b\} \in \mathcal{D}$ that is not path-flexible, the following lemma shows that there are infinitely many path lengths k such that there is no length-k $s \rightsquigarrow t$ walk for some $s \in \{(a, b), (b, a)\}$ and $t \in \{(a, b), (b, a)\}$. As we will later see, this inflexibility in the possible path lengths implies lower bounds for distributed algorithms that may use the configuration $\{a, b\}$.

Lemma 5.2 (Property of path-inflexibility)**.** *Suppose that the size-2 multiset $\{a, b\} \in \mathcal{D}$ is not path-flexible. Then one of the following holds.*

- *There is no $s \rightsquigarrow t$ walk for at least one choice of $s \in \{(a, b), (b, a)\}$ and $t \in \{(a, b), (b, a)\}$.*
- *There is an integer $2 \leq x \leq |\Sigma|^2$ such that for any positive integer k that is not an integer multiple of x, there are no length-k walks $(a, b) \rightsquigarrow (a, b)$ and $(b, a) \rightsquigarrow (b, a)$ in $\mathcal{M}_{\mathcal{D}}$.*

Proof. Suppose that $\{a, b\} \in \mathcal{D}$ is not path-flexible. We assume that there are $s \rightsquigarrow t$ walks for all choices of $s \in \{(a, b), (b, a)\}$ and $t \in \{(a, b), (b, a)\}$. To prove this lemma, it suffices to show that there is an integer $2 \leq x \leq |\Sigma|^2$ such that for any positive integer k that is not an integer multiple of x, there are no length-k walks $(a, b) \rightsquigarrow (a, b)$ and $(b, a) \rightsquigarrow (b, a)$.

First of all, we claim that for any integer K there is an integer $k \geq K$ such that there is no length-k walk $(a, b) \rightsquigarrow (a, b)$. If this claim does not hold, then there is an integer K such that

there is a length-k walk $(a,b) \rightsquigarrow (a,b)$ for each $k \geq K$. Combining these walks with existing walks $(a,b) \rightsquigarrow (b,a)$ and $(b,a) \rightsquigarrow (a,b)$, we infer that there exists an integer K' such that for each integer $k \geq K'$, there exist length-k walks $(a,b) \rightsquigarrow (a,b)$, $(a,b) \rightsquigarrow (b,a)$, $(b,a) \rightsquigarrow (a,b)$, and $(b,a) \rightsquigarrow (b,a)$ in $\mathcal{M}_{\mathcal{D}}$, contradicting the assumption that $\{a,b\} \in \mathcal{D}$ is not path-flexible.

Let U be the set of integers k such that there is a length-k walk $(a,b) \rightsquigarrow (a,b)$. Note that by taking reversal, the existence of a length-k walk $(a,b) \rightsquigarrow (a,b)$ implies the existence of a length-k walk $(b,a) \rightsquigarrow (b,a)$, and vice versa. Our assumption on the existence of a walk $(a,b) \rightsquigarrow (a,b)$ implies $U \neq \emptyset$. We choose $x = \gcd(U)$ to be the greatest common divisor of U, so that for any integer k that is not an integer multiple of x, there are no length-k walks $(a,b) \rightsquigarrow (a,b)$ and $(b,a) \rightsquigarrow (b,a)$ in $\mathcal{M}_{\mathcal{D}}$. We must have $x \geq 2$ because there cannot be two co-prime numbers in U, since otherwise there exists an integer K such that U includes all integers that are at least K, contradicting the claim proved above. Specifically, if the two co-prime numbers are k_1 and k_2, then we may set $K = g(k_1,k_2)+1 = k_1k_2 - k_1 - k_2 + 1$, where $g(k_1,k_2)$ is the Frobenius number of the set $\{k_1,k_2\}$ [41]. We also have $x \leq |\Sigma|^2$, since the smallest number in U is at most the number of nodes in $\mathcal{M}_{\mathcal{D}}$, which is upper bounded by $|\Sigma|^2$. □

For the special case of $|\Sigma| = 1$ and $\mathcal{D} \neq \emptyset$, we must have $a = b$ in Lemma 5.2. Since there is no integer x satisfying $2 \leq x \leq |\Sigma|^2$ when $|\Sigma| = 1$, Lemma 5.2 implies that if $\{a,a\}$ is not path-flexible, then there is no walk $(a,a) \rightsquigarrow (a,a)$, where $\{a,a\}$ is the unique element in $\mathcal{D}$.

Path-flexible strongly connected components Since each $\{a,b\} \in \mathcal{D}$ corresponds to two nodes (a,b) and (b,a) in $\mathcal{M}_{\mathcal{D}}$, we will consider a different notion of a strongly connected component. In Definition 5.5, we do not require the elements a, b, c, and d to be distinct. For example, we may have $\{a,b\} = \{c,d\}$ or $a = b$.

Definition 5.5 (Strongly connected components)**.** *Let $\mathcal{D}$ be a set of size-2 multisets of elements in Σ. For each $\{a,b\} \in \mathcal{D}$ and $\{c,d\} \in \mathcal{D}$, we write $\{a,b\} \sim \{c,d\}$ if there is a walk $s \rightsquigarrow t$ in $\mathcal{M}_{\mathcal{D}}$ for each choice of $s \in \{(a,b),(b,a)\}$ and $t \in \{(c,d),(d,c)\}$.*

Let $\mathcal{D}^{\sim}$ be the set of all $\{a,b\} \in \mathcal{D}$ such that $\{a,b\} \sim \{a,b\}$. Then we define the strongly connected components of $\mathcal{D}$ as the equivalence classes of $\sim$ over $\mathcal{D}^{\sim}$.

By taking reversal, the existence of an $(a,b) \rightsquigarrow (c,d)$ walk implies the existence of a $(d,c) \rightsquigarrow (b,a)$ walk. Therefore, if there is a walk $s \rightsquigarrow t$ in $\mathcal{M}_{\mathcal{D}}$ for each choice of $s \in \{(a,b),(b,a)\}$ and $t \in \{(c,d),(d,c)\}$, then there is also a walk $t \rightsquigarrow s$ in $\mathcal{M}_{\mathcal{D}}$ for each choice of $s \in \{(a,b),(b,a)\}$ and $t \in \{(c,d),(d,c)\}$. Hence the relation $\sim$ in Definition 5.5 is symmetric over $\mathcal{D}$. It is clear from the definition of $\sim$ in Definition 5.5 that it is transitive over $\mathcal{D}$ and it is reflexive over $\mathcal{D}^{\sim}$, so $\sim$ is indeed an equivalence relation over $\mathcal{D}^{\sim}$.

For any strongly connected component $\mathcal{D}'$ of $\mathcal{D}$, it is clear that either all $\{a,b\} \in \mathcal{D}'$ are path-flexible or all $\{a,b\} \in \mathcal{D}'$ are not path-flexible. We say that a strongly connected component $\mathcal{D}'$ is path-flexible if all $\{a,b\} \in \mathcal{D}'$ are path-flexible. We define $\mathsf{flexibility}(\mathcal{D}')$ as the minimum number K such that for each integer $k \geq K$ there is an $(a,b) \rightsquigarrow (c,d)$ walk of length k for all choices of a, b, c, and d such that $\{a,b\} \in \mathcal{D}'$ and $\{c,d\} \in \mathcal{D}'$. It is clear that such a number K exists given that $\mathcal{D}'$ is a path-flexible strongly connected component. We define

$$\mathsf{flexible\text{-}SCC}(\mathcal{D}) = \begin{array}{l}\text{the set of all subsets of } \mathcal{D} \text{ that are a path-flexible} \\ \text{strongly connected component of } \mathcal{D}.\end{array}$$

Clearly, elements in $\mathsf{flexible\text{-}SCC}(\mathcal{D})$ are disjoint subsets of $\mathcal{D}$. It is possible that $\mathsf{flexible\text{-}SCC}(\mathcal{D})$ is an empty set, and this happens when all nodes in the directed graph $\mathcal{M}_{\mathcal{D}}$ are not path-flexible.

Restriction of a set of node configurations Given an LCL problem $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$, a subset $\mathcal{S} \subseteq \mathcal{V}$ of node configurations, and a set $\mathcal{D}$ of size-2 multisets whose elements are in Σ, we define the restriction of $\mathcal{S}$ to $\mathcal{D}$ as follows.

$$\mathcal{S} \restriction_{\mathcal{D}} = \{C \in \mathcal{S} \mid \text{all size-2 sub-multisets of } C \text{ are in } \mathcal{D}\}.$$

Lemma 5.3 shows that if we label the two endpoints of a sufficiently long path using node configurations in $\mathcal{S} \restriction_{\mathcal{D}^*}$, where $\mathcal{D}^* \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{S}})$, then it is always possible to complete the labeling of the path using only node configurations in $\mathcal{S}$ in such a way that the entire labeling is correct. Specifically, consider a path $P = (v_1, v_2, \ldots, v_{d+1})$ of length $d \geq \mathsf{flexibility}(\mathcal{D}^*)$. Assume that the node configuration of v_1 is already fixed to be $C \in \mathcal{S} \restriction_{\mathcal{D}^*}$ where the half-edge $(v_1, \{v_1, v_2\})$ is labeled by $\beta \in C$ and the node configuration of v_{d+1} is already fixed to be $C' \in \mathcal{S} \restriction_{\mathcal{D}^*}$ where the half-edge $(v_{d+1}, \{v_d, v_{d+1}\})$ is labeled by $\alpha' \in C'$. Lemma 5.3 shows that it is possible to complete the labeling of P using only node configurations in $\mathcal{S}$, as we may label v_i using the node configuration C_i where the two half-edges $(v_i, \{v_{i-1}, v_i\})$ and $(v_i, \{v_i, v_{i+1}\})$ are labeled by α_i and β_i, for each $2 \leq i \leq d$.

Lemma 5.3 (Property of path-flexible strongly connected components)**.** *Let $\mathcal{S} \subseteq \mathcal{V}$ be a set of node configurations, and let $\mathcal{D}^* \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{S}})$. For any choices of $C \in \mathcal{S} \restriction_{\mathcal{D}^*}$, $C' \in \mathcal{S} \restriction_{\mathcal{D}^*}$, size-2 sub-multisets $\{\alpha, \beta\} \subseteq C$, $\{\alpha', \beta'\} \subseteq C'$, and a number $d \geq \mathsf{flexibility}(\mathcal{D}^*)$, there exists a sequence*

$$\alpha_1, C_1, \beta_1, \alpha_2, C_2, \beta_2, \ldots, \alpha_{d+1}, C_{d+1}, \beta_{d+1}$$

satisfying the following conditions.

- *First endpoint: $\alpha_1 = \alpha$, $\beta_1 = \beta$, and $C_1 = C$.*
- *Last endpoint: $\alpha_{d+1} = \alpha'$, $\beta_{d+1} = \beta'$, and $C_{d+1} = C'$.*
- *Node configurations: for $1 \leq i \leq d+1$, $\{\alpha_i, \beta_i\}$ is a size-2 sub-multiset of C_i, and $C_i \in \mathcal{S}$.*
- *Edge configurations: for $1 \leq i \leq d$, $\{\beta_i, \alpha_{i+1}\} \in \mathcal{E}$.*

Proof. By the path-flexibility of $\mathcal{D}^*$, there exists a length-d walk $(\alpha, \beta) \rightsquigarrow (\alpha', \beta')$ in $\mathcal{M}_{\mathcal{D}_{\mathcal{S}}}$. We fix

$$(\alpha_1, \beta_1) \to (\alpha_2, \beta_2) \to \cdots \to (\alpha_{d+1}, \beta_{d+1})$$

to be any such walk. This implies that $\{\beta_i, \alpha_{i+1}\} \in \mathcal{E}$ for each $1 \leq i \leq d$. Since $\{\alpha_i, \beta_i\}$ is a size-2 multiset of $\mathcal{D}_{\mathcal{S}}$, there exists a choice of $C_i \in \mathcal{S}$ for each $2 \leq i \leq d$ such that $\{\alpha_i, \beta_i\}$ is a sub-multiset of C_i. □

Good sequences Given an LCL problem $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ on Δ-regular trees, we say that a sequence

$$(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_k)$$

is *good* if it satisfies the following requirements.

- $\mathcal{V}_1 = \mathsf{trim}(\mathcal{V})$. That is, we start the sequence from the result of trimming the set $\mathcal{V}$ of all node configurations in the given LCL problem $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$.
- For each $1 \leq i \leq k-1$, $\mathcal{D}_i \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{V}_i})$. That is, $\mathcal{D}_i$ is a path-flexible strongly connected component of the automaton associated with the path-form of the LCL problem $(\Delta, \Sigma, \mathcal{V}_i, \mathcal{E})$, which is Π restricted to the set of node configurations $\mathcal{V}_i$.
- For each $2 \leq i \leq k$, $\mathcal{V}_i = \mathsf{trim}(\mathcal{V}_{i-1} \restriction_{\mathcal{D}_{i-1}})$. That is, $\mathcal{V}_i$ is the result of taking the restriction of the set of node configurations $\mathcal{V}_{i-1}$ to $\mathcal{D}_{i-1}$ and then performing a trimming.

- $\mathcal{V}_k \neq \emptyset$. That is, we require that the last set of node configurations is non-empty.

It is straightforward to see that $\mathcal{V}_1 \supseteq \mathcal{V}_2 \supseteq \cdots \supseteq \mathcal{V}_k$ since $\mathcal{V}_i = \mathsf{trim}(\mathcal{V}_{i-1} \restriction_{\mathcal{D}_{i-1}})$ is always a subset of $\mathcal{V}_{i-1}$. Similarly, we also have $\mathcal{D}_1 \supseteq \mathcal{D}_2 \supseteq \cdots \supseteq \mathcal{D}_{k-1}$, as $\mathcal{D}_i \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{V}_i})$ is a subset of $\mathcal{D}_{\mathcal{V}_i}$ and $\mathcal{D}_{\mathcal{V}_i}$ is a subset of $\mathcal{D}_{i-1}$ due to the definition $\mathcal{V}_i = \mathsf{trim}(\mathcal{V}_{i-1} \restriction_{\mathcal{D}_{i-1}})$.

Depth of an LCL problem We define the depth d_Π of an LCL problem $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ on Δ-regular trees as follows. If there is no good sequence, then we set $d_\Pi = 0$. If there is a good sequence $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_k)$ for each positive integer k, then we set $d_\Pi = \infty$. Otherwise, we set d_Π as the largest integer k such that there is a good sequence $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_k)$. We prove the following results.

Theorem 5.1 (Characterization of complexity classes)**.** *Let $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ be an LCL problem on Δ-regular trees. We have the following.*

- *If $d_\Pi = 0$, then Π is unsolvable in the sense that there exists a tree of maximum degree Δ such that there is no correct solution of Π on this tree.*
- *If $d_\Pi = k$ is a positive integer, then the optimal round complexity of Π is $\Theta(n^{1/k})$.*
- *If $d_\Pi = \infty$, then Π can be solved in $O(\log n)$ rounds.*

In Theorem 5.1, all the upper bounds hold in the CONGEST model, and all the lower bounds hold in the LOCAL model. For example, if $d_\Pi = 5$, then Π can be solved in $O(n^{1/5})$ rounds in the CONGEST model, and there is a matching lower bound $\Omega(n^{1/5})$ in the LOCAL model.

We note that there are several natural definitions of unsolvability of an LCL w.r.t. a given graph class [22] that are different from the one in Theorem 5.1.

Theorem 5.2 (Complexity of the characterization)**.** *There is a polynomial-time algorithm $\mathcal{A}$ that computes d_Π for any given LCL problem $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ on Δ-regular trees. If $d_\Pi = k$ is a positive integer, then $\mathcal{A}$ also outputs a description of an $O(n^{1/k})$-round algorithm for Π. If $d_\Pi = \infty$, then $\mathcal{A}$ also outputs a description of an $O(\log n)$-round algorithm for Π.*

The distributed algorithms returned by the polynomial-time algorithm $\mathcal{A}$ in Theorem 5.2 are also in the CONGEST model.

5.2 Upper bounds

In this section, we prove the upper bound part of Theorem 5.1. If a good sequence $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_k)$ exists for some positive integer k, we show that the LCL problem $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ can be solved in $O(n^{1/k})$ rounds. If a good sequence $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_k)$ exists for all positive integers k, then we show that $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ can be solved in $O(\log n)$ rounds. All these algorithms do not require sending large messages and can be implemented in the CONGEST model.

Rake-and-compress decompositions Roughly speaking, a rake-and-compress process is a procedure that decomposes a tree by iteratively removing degree-1 nodes (rake) and removing degree-2 nodes (compress). There are several tree decompositions resulting from variants of a rake-and-compress process. Here we use a variant of decomposition considered in [19] that is parameterized by three positive integers γ, ℓ, and L. A (γ, ℓ, L) decomposition of a tree $G = (V, E)$ is a partition of the node set

$$V = V_1^{\mathsf{R}} \cup V_1^{\mathsf{C}} \cup V_2^{\mathsf{R}} \cup V_2^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$$

satisfying the following requirements.

Requirements for V_i^{R} For each connected component S of the subgraph of G induced by V_i^{R}, it is required that there is a root $z \in S$ meeting the following conditions.

- z has at most one neighbor in $V_i^{\mathsf{C}} \cup V_{i+1}^{\mathsf{R}} \cup \cdots \cup V_L^{\mathsf{R}}$.
- All nodes in $S \setminus \{z\}$ have no neighbor in $V_i^{\mathsf{C}} \cup V_{i+1}^{\mathsf{R}} \cup \cdots \cup V_L^{\mathsf{R}}$.
- All nodes in $S \setminus \{z\}$ are within distance $\gamma - 1$ to z.

Intuitively, each connected component S of the subgraph of G induced by V_i^{R} is a rooted tree of height at most $\gamma - 1$. The root can have at most one neighbor residing in the higher layers of the decomposition. The remaining nodes in S cannot have neighbors in the higher layers of the decomposition. For the special case of $\gamma = 1$, the set V_i^{R} is an independent set.

Requirements for V_i^{C} For each connected component S of the subgraph of G induced by V_i^{C}, S is a path $(v_1, v_2, \ldots, v_s)$ of $s \in [\ell, 2\ell]$ nodes meeting the following conditions.

- There exist two nodes u and w in $V_{i+1}^{\mathsf{R}} \cup V_{i+1}^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$ such that u is adjacent to v_1 and w is adjacent to v_s.
- For each $1 \leq j \leq s$, v_j has no neighbor in $\left(V_{i+1}^{\mathsf{R}} \cup V_{i+1}^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}\right) \setminus \{u, w\}$.

Intuitively, each connected component S of the subgraph of G induced by V_i^{C} is a path. Only the endpoints of the paths can have neighbors residing in the higher layers of the decomposition. The remaining nodes in S cannot have neighbors in the higher layers of the decomposition.

Existing algorithms for rake-and-compress decompositions For any $k = O(1)$ and $\ell = O(1)$, there is an $O(n^{1/k})$-round algorithm computing a (γ, ℓ, L) decomposition of a tree $G = (V, E)$ with $\gamma = O(n^{1/k})$ and $L = k$ [19]. For any $\ell = O(1)$, there is an $O(\log n)$-round algorithm computing a (γ, ℓ, L) decomposition of a tree $G = (V, E)$ with $\gamma = 1$ and $L = O(\log n)$ [21]. These algorithms are deterministic and can be implemented in the CONGEST model. We will employ these algorithms as subroutines to prove the upper bound part of Theorem 5.1.

Lemma 5.4 (Solving Π using rake-and-compress decompositions)**.** *Suppose we are given an* LCL *problem $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ on Δ-regular trees that admits a good sequence*

$$(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_k).$$

Suppose we are given a (γ, ℓ, L) decomposition of an n-node tree $G = (V, E)$ of maximum degree at most Δ

$$V = V_1^{\mathsf{R}} \cup V_1^{\mathsf{C}} \cup V_2^{\mathsf{R}} \cup V_2^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$$

with $L = k$ and $\ell = \max\{1, \mathsf{flexibility}(\mathcal{D}_1) - 1, \ldots, \mathsf{flexibility}(\mathcal{D}_{k-1}) - 1\}$. Then a correct solution of Π on G can be computed in $O((\gamma + \ell)L)$ rounds in the CONGEST *model.*

Proof. We present an $O((\gamma + \ell)L)$-round CONGEST algorithm finding a correct solution of Π on G. The algorithm labels the half-edges surrounding the nodes of the graph in the order $V_L^{\mathsf{R}}, V_{L-1}^{\mathsf{C}}, \ldots, V_1^{\mathsf{R}}$. Our algorithm has the property that it only uses the node configurations in $\mathcal{V}_i$ to label the half-edges surrounding the nodes in V_i^{R} and V_i^{C}.

Not all nodes v in G have $\deg(v) = \Delta$. In general, for each node v in G, we say that the node configuration of v is in $\mathcal{V}_i$ if the multiset of the $\deg(v)$ half-edge labels surrounding v is a sub-multiset of some $C \in \mathcal{V}_i$.

Labeling V_i^{R} By induction hypothesis, assume the algorithm has finished labeling the half-edges surrounding the nodes in $V_i^{\mathsf{C}} \cup V_{i+1}^{\mathsf{R}} \cup \cdots \cup V_L^{\mathsf{R}}$ in such a way that their node configurations are in $\mathcal{V}_i$, as we recall that $\mathcal{V}_i \supseteq \mathcal{V}_{i+1} \supseteq \cdots \supseteq \mathcal{V}_k$. The algorithm then labels each connected component S of the subgraph of G induced by V_i^{R}, in parallel and using $O(\gamma)$ rounds in the CONGEST model, as follows.

The set S has the property that there is at most one node $z \in S$ that may have a neighbor in $V_i^{\mathsf{C}} \cup V_{i+1}^{\mathsf{R}} \cup \cdots \cup V_L^{\mathsf{R}}$, and the number of neighbors of z in $V_i^{\mathsf{C}} \cup V_{i+1}^{\mathsf{R}} \cup \cdots \cup V_L^{\mathsf{R}}$ is at most one. We claim that it is always possible to complete the labeling of half-edges surrounding the nodes in S using only node configurations in $\mathcal{V}_i$. To see that this is possible, it suffices to consider the following situation. Given that the existing half-edge labels surrounding a node v form a node configuration in $\mathcal{V}_i$, consider a neighbor u of v, and we want to label the half-edges surrounding u in such a way that the edge configuration of $e = \{u, v\}$ is in $\mathcal{E}$ and the node configuration of u is in $\mathcal{V}_i$. This is always doable due to Lemma 5.1, as we recall from the definition of $\mathcal{V}_i$ that $\mathcal{V}_i = \mathsf{trim}(\mathcal{S})$ for some set $\mathcal{S} \subseteq \mathcal{V}$. The round complexity of labeling S is $O(\gamma)$ because S is a tree rooted at z of depth at most $\gamma - 1$.

Labeling V_i^{C} Similarly, by induction hypothesis, assume the algorithm has already finished labeling the half-edges surrounding the nodes in $V_{i+1}^{\mathsf{R}} \cup V_{i+1}^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$ in such a way that their node configurations are in $\mathcal{V}_{i+1}$, as we recall that $\mathcal{V}_{i+1} \supseteq \mathcal{V}_{i+2} \supseteq \cdots \supseteq \mathcal{V}_k$. The algorithm then labels each connected component S of the subgraph of G induced by V_i^{C}, in parallel and using $O(\ell)$ rounds in the CONGEST model, as follows.

The set S has the property that there are exactly two nodes u and w in $V_{i+1}^{\mathsf{R}} \cup V_{i+1}^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$ adjacent to S, and the subgraph induced by $S \cup \{u, w\}$ is a path $(u, v_1, v_2, \ldots, v_s, w)$, with $s \in [\ell, 2\ell]$. Hence the length of this path is $s + 1 \geq \ell + 1 \geq \mathsf{flexibility}(\mathcal{D}_i)$ by our choice of ℓ.

Recall that $\mathcal{D}_i \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{V}_i})$ and the node configurations of u and w are in $\mathcal{V}_{i+1} = \mathsf{trim}(\mathcal{V}_i \restriction_{\mathcal{D}_i}) \subseteq \mathcal{V}_i \restriction_{\mathcal{D}_i}$, so Lemma 5.3 ensures that we can label the half-edges surrounding the nodes $v_1, v_2, \ldots, v_s$ using only node configurations in $\mathcal{V}_i$ in such a way that the edge configurations of all edges in the path $(u, v_1, v_2, \ldots, v_s, w)$ are in $\mathcal{E}$. The round complexity of labeling S is $O(\ell)$ because S is a path of at most 2ℓ nodes.

Summary The number of rounds spent on labeling each part V_i^{R} is $O(\gamma)$, and the number of rounds spent on labeling each part V_i^{C} is $O(\ell)$, so the overall round complexity for solving Π given a (γ, ℓ, L) decomposition is $O((\gamma + \ell)L)$ rounds in the CONGEST model. □

Combining Lemma 5.4 with existing algorithms for computing (γ, ℓ, L) decompositions, we obtain the following results.

Lemma 5.5 (Upper bound for the case $d_\Pi = k$)**.** *If $d_\Pi = k$ for some positive integer k, then Π can be solved in $O(n^{1/k})$ rounds in the* CONGEST *model.*

Proof. In this case, a good sequence $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_k)$ exists. As a (γ, ℓ, L) decomposition with $\gamma = O(n^{1/k})$, $\ell = O(1)$, and $L = k$ can be computed in $O(n^{1/k})$ rounds [19], Π can be solved in $O(n^{1/k}) + O((\gamma + \ell)L) = O(n^{1/k})$ rounds using the algorithm of Lemma 5.4. Here both k and ℓ are $O(1)$, as they are independent of the number of nodes n. □

Lemma 5.6 (Upper bound for the case $d_\Pi = \infty$)**.** *If $d_\Pi = \infty$, then Π can be solved in $O(\log n)$ rounds in the* CONGEST *model.*

Proof. In this case, a good sequence $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_k)$ exists for all positive integers k. As a (γ, ℓ, L) decomposition with $\gamma = 1$, $\ell = O(1)$, and $L = O(\log n)$ can be computed in $O(\log n)$ rounds [21], by choosing a good sequence $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_k)$ with $k = L$, Π can be solved in $O(\log n) + O((\gamma + \ell)L) = O(\log n)$ rounds using the algorithm of Lemma 5.4. Similarly, here $\ell = O(1)$, as it is independent of the number of nodes n. □

5.3 Lower bounds

In this section, we prove the lower bound part of Theorem 5.1. In our lower bound proofs, we pick γ to be the smallest integer satisfying the following requirements. For each subset $\mathcal{S} \subseteq \mathcal{V}$ and each $C \in \mathcal{S} \setminus \mathsf{trim}(\mathcal{S})$, there exists no correct labeling of T_γ^* where the node configuration of the root r is C and the node configurations of the remaining degree-Δ nodes are in $\mathcal{S}$. Such a number γ exists due to the definition of trim.

Lemma 5.7 (Unsolvability for the case $d_\Pi = 0$)**.** *If $d_\Pi = 0$, then Π is unsolvable in the sense that there exists a tree G of maximum degree Δ such that there is no correct solution of Π on G.*

Proof. We take $G = T_\gamma^*$. Since $d_\Pi = 0$, we have $\mathsf{trim}(\mathcal{V}) = \emptyset$. Our choice of γ implies that there is no correct solution of Π on $G = T_\gamma^*$. □

For the rest of this section, we focus on the case that $d_\Pi = k$ is a positive integer. We will prove that solving Π requires $\Omega(n^{1/k})$ rounds in the LOCAL model. The exact choice of $s = \Theta(t)$ in Definition 5.6 is to be determined later. In our lower bound proof, we will assume that there exists an algorithm $\mathcal{A}$ violating the lower bound the parameter, and then we will derive a contradiction. As we will later see, the parameter t in Definition 5.6 will corresponds to the time complexity of $\mathcal{A}$.

Definition 5.6 (Lower bound graphs)**.** *Let t be any positive integer and choose $s = \Theta(t)$ to be a sufficiently large integer.*

- *$G_{\mathsf{R},1}$ is the rooted tree T_γ, and $G_{\mathsf{R},1}^*$ is the rooted tree T_γ^*. All nodes in $G_{\mathsf{R},1}$ and $G_{\mathsf{R},1}^*$ are said to be in layer $(\mathsf{R}, 1)$.*
- *For each integer $i \geq 1$, $G_{\mathsf{C},i}$ is the result of the following construction. Start with an s-node path $(v_1, v_2, \ldots, v_s)$ and let v_1 be the root. For each $1 \leq i < s$, append $\Delta - 2$ copies of $G_{\mathsf{R},i}$ to v_i. For $i = s$, append $\Delta - 1$ copies of $G_{\mathsf{R},i}$ to v_s. The nodes $v_1, v_2, \ldots, v_s$ are said to be in layer (C, i).*
- *For each integer $i \geq 2$, $G_{\mathsf{R},i}$ is the result of the following construction. Start with a rooted tree T_γ. Append $\Delta - 1$ copies of $G_{\mathsf{C},i-1}$ to each leaf in T_γ. All nodes in T_γ are said to be in layer (R, i). The rooted tree $G_{\mathsf{R},i}^*$ is defined analogously by replacing T_γ with T_γ^* in the construction.*

Although the trees $G_{\mathsf{R},i}$, $G_{\mathsf{R},i}^*$, and $G_{\mathsf{C},i}$ are rooted in their definitions, we may also treat them as unrooted trees. Throughout our lower bound proof in Section 5.3, we fix our main lower bound graph $G = (V, E)$ to be the tree $G_{\mathsf{R},k+1}^*$, see Figure 2 for an example.

Definition 5.7 (Main lower bound graph)**.** *Define $G = (V, E)$ as the tree $G_{\mathsf{R},k+1}^*$.*

In the subsequent discussion, we focus on the tree $G = G_{\mathsf{R},k+1}^*$. It is clear from its construction that G has $n = O(t^k)$ nodes, if we treat γ as a constant independent of t. Indeed, γ only depends on the underlying LCL problem $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$. More generally, the number of nodes in $G_{\mathsf{R},i}$ or $G_{\mathsf{R},i}^*$ is $O(t^{i-1})$, and the number of nodes in $G_{\mathsf{C},i}$ is $O(t^i)$. To show that Π requires $\Omega(n^{1/k})$ rounds to solve, it suffices to show that Π requires t rounds to solve on G, for any positive integer t.

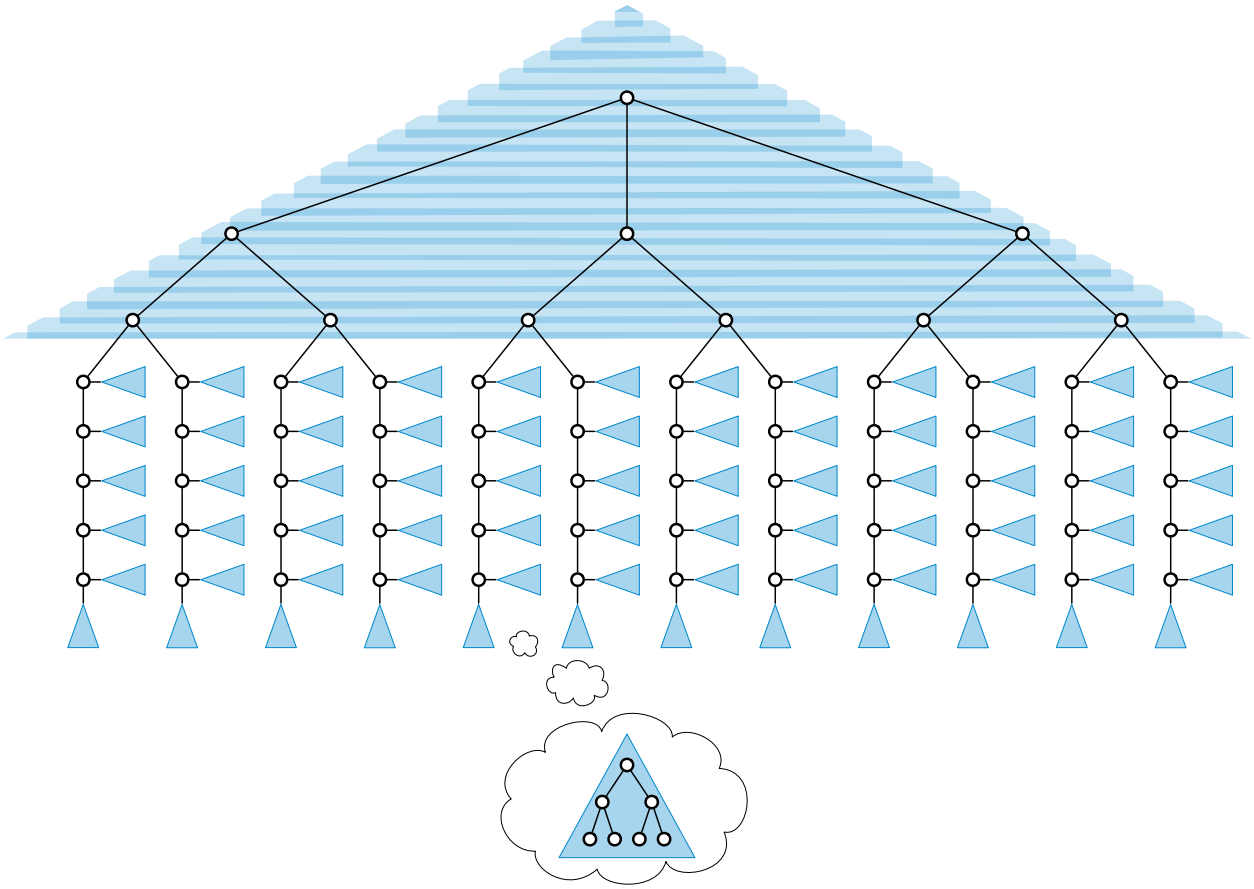


Figure 2: The graph $G^*_{\mathrm{R},2}$ with $\Delta = 3$, $s = 5$, and $\gamma = 2$.

The nodes in $G = G^*_{\mathsf{R},k+1}$ are partitioned into layers $(\mathsf{R},1),(\mathsf{C},1),(\mathsf{R},2),(\mathsf{C},2),\ldots,(\mathsf{R},k+1)$ according to the rules in the above recursive construction. In the subsequent discussion, we order the layers by $(\mathsf{R},1) \prec (\mathsf{C},1) \prec (\mathsf{R},2) \prec (\mathsf{C},2) \prec \cdots \prec (\mathsf{R},k+1)$. For example, when we say layer (R,i) or higher, we mean the set of all layers $(\mathsf{R},i),(\mathsf{C},i),\ldots,(\mathsf{R},k+1)$.

Intuitively, layers (R,i) and (C,i) resemble the parts V_i^{R} and V_i^{C} in a rake-and-compress decomposition. Except for some leaf nodes in layer $(\mathsf{R},1)$, all nodes in the graph have degree Δ. Each connected component of the subgraph of G induced by layer (C,i) nodes is a path of s nodes. Each connected component of the subgraph of G induced by layer (R,i) nodes is a rooted tree T_γ (if $1 \le i \le k$) or a rooted tree T^*_γ (if $i = k+1$). We further classify the nodes in layer (C,i) as follows.

Definition 5.8 (Classification of nodes in layer (C,i))**.** *The nodes in the s-node path $(v_1,v_2,\ldots,v_s)$ in the construction of $G_{\mathsf{C},i}$ are classified as follows.*

- *We say that v_j is a front node if $1 \le j \le t$.*
- *We say that v_j is a central node if $t+1 \le j \le s-t$.*
- *We say that v_j is a rear node if $s-t+1 \le j \le s$.*

Based on Definition 5.8, we define the following subsets of nodes in $G = G^*_{\mathsf{R},k+1}$. We assume that $s = \Theta(t)$ is chosen to be sufficiently large so that central nodes exist.

Definition 5.9 (Subsets of nodes in G)**.** *We define the following subsets of nodes in $G = G^*_{\mathsf{R},k+1}$.*

- *Define $S_{\mathsf{R},1}$ as the set of nodes v in G such that the radius-γ neighborhood of v is isomorphic to T^*_γ.*
- *For $2 \le i \le k+1$, define $S_{\mathsf{R},i}$ as the set of nodes v in G such that the radius-γ neighborhood of v is isomorphic to T^*_γ and contains only nodes in $S_{\mathsf{C},i-1}$.*
- *For $1 \le i \le k$, define $S_{\mathsf{C},i}$ as the set of nodes v in G that are in layer $(\mathsf{R},i+1)$ or above or are central or front nodes in layer (C,i).*

We prove some basic properties of the sets in Definition 5.9.

Lemma 5.8 (Subset containment)**.** *We have $S_{\mathsf{R},1} \supseteq S_{\mathsf{C},1} \supseteq \cdots \supseteq S_{\mathsf{R},k+1} \neq \emptyset$.*

Proof. We have $S_{\mathsf{C},i} \supseteq S_{\mathsf{R},i+1}$ since it follows from the definition of $S_{\mathsf{R},i+1}$ that $v \in S_{\mathsf{C},i}$ is a necessary condition for $v \in S_{\mathsf{R},i+1}$. The claim that $S_{\mathsf{R},i} \supseteq S_{\mathsf{C},i}$ follows from the fact that each $v \in S_{\mathsf{C},i}$ is in layer (C,i) or above: The radius-γ neighborhood of any such node v is isomorphic to T^*_γ and contains only nodes in layer (R,i) or above, and we know that all nodes in layer (R,i) or above are in $S_{\mathsf{C},i-1}$. Hence $v \in S_{\mathsf{C},i}$ implies $v \in S_{\mathsf{R},i}$.

To see that $S_{\mathsf{R},k+1} \neq \emptyset$, consider the root r of $G = G^*_{\mathsf{R},k+1}$. The radius-γ neighborhood of r is isomorphic to T^*_γ and contains only nodes in layer $(\mathsf{R},k+1)$. We know that all nodes in layer $(\mathsf{R},k+1)$ are in $S_{\mathsf{C},k}$, so $r \in S_{\mathsf{R},k+1}$. □

Lemma 5.9 (Property of $S_{\mathsf{C},i}$)**.** *For each node $v \in S_{\mathsf{C},i}$, one of the following holds.*

- *v is a central node in layer (C,i).*
- *For each neighbor u of v such that $u \in S_{\mathsf{C},i}$, there exists a path $P = (v,u,\ldots,w)$ such that w is a central node in layer (C,i) and all nodes in P are in $S_{\mathsf{C},i}$.*

Proof. We assume that $v \in S_{\mathsf{C},i}$ is not a central node in layer (C,i). Consider any neighbor u of v such that $u \in S_{\mathsf{C},i}$. To prove the lemma, its suffices to find a path $P = (v,u,\ldots,w)$ such that w is a central node in layer (C,i) and all nodes in P are in $S_{\mathsf{C},i}$. The existence of such a path P

follows from the simple observation that $S_{\mathsf{C},i}$ induces a connected subtree where all the leaf nodes are central nodes in layer (C, i).

More specifically, such a path P can be constructed as follows. If u itself is a central node in layer (C, i), then we can simply take $P = (v, u = w)$.

We first consider the case where v is the parent of u in the rooted tree $G = G^*_{\mathsf{R},k+1}$. In this case, there must exist a descendant w of u such that w is a central node in layer (C, i). This gives us a desired path $P = (v, u, \ldots, w)$.

Next, we consider the case where v is a child of u in the rooted tree $G = G^*_{\mathsf{R},k+1}$. In this case, it might be possible that all descendants w of u such that w is a central node in layer (C, i) are also descendants of v. Hence we will consider a different approach. Starting from (v, u), we first follow the parent pointers to the root r of $G = G^*_{\mathsf{R},k+1}$. There are Δ children of r, and it is clear that each child of r has a descendant w that is a central node in layer (C, i). Hence we can extend the current path $(v, u, \ldots, r)$ to a desired $P = (v, u, \ldots, r, \ldots, w)$ such that w that is a central node in layer (C, i). □

Assumptions We are given an LCL problem $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ such that $d_\Pi = k$. Hence there does not exist a good sequence

$$(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_{k+1}).$$

Recall that the rules for a good sequence are as follows:

$$\mathcal{V}_i = \begin{cases} \mathsf{trim}(\mathcal{V}) & \text{if } i = 1, \\ \mathsf{trim}(\mathcal{V}_{i-1} \restriction_{\mathcal{D}_{i-1}}) & \text{if } i > 1, \end{cases}$$
$$\mathcal{D}_i \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{V}_i}).$$

The only nondeterminism in the above rules is the choice of $\mathcal{D}_i \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{V}_i})$ for each i. The fact that $d_\Pi = k$ implies that for all possible choices of $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$, we always end up with $\mathcal{V}_{k+1} = \emptyset$.

We assume that there is an algorithm $\mathcal{A}$ that solves Π in t rounds on $G = G^*_{\mathsf{R},k+1}$. As the number n of nodes in G satisfies $t = \Omega(n^{1/k})$, to prove the desired $\Omega(n^{1/k})$ lower bound, it suffices to derive a contradiction. Specifically, we will prove that the existence of such an algorithm $\mathcal{A}$ forces the existence of a good sequence $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_{k+1})$, contradicting the fact that $d_\Pi = k$.

Induction hypothesis Our proof proceeds by an induction on the subsets $S_{\mathsf{R},1}$, $S_{\mathsf{C},1}$, $S_{\mathsf{R},2}$, $S_{\mathsf{C},2}$, …, $S_{\mathsf{R},k+1}$. For each $1 \le i \le k$, the choice of $\mathcal{D}_i \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{V}_i})$ is fixed in the induction hypothesis for $S_{\mathsf{C},i}$. The choice of $\mathcal{V}_i$ is uniquely determined once $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{i-1}$ have been fixed.

Before defining our induction hypothesis, we recall that the output labels of the half-edges surrounding a node v are determined by the subgraph induced by the radius-t neighborhood U of v, together with the distinct IDs of the nodes in U. For each node v in G, we define $\mathcal{V}^{\mathcal{A}}_v$ as the set of all possible node configurations of v that can possibly appear when we run $\mathcal{A}$ on G. In other words, $C \in \mathcal{V}^{\mathcal{A}}_v$ implies that there exists an assignment of distinct IDs to nodes in the radius-t neighborhood of v such that the output labels of the half-edges surrounding v form the node configuration C.

Similarly, for any two edges e_1 and e_2 incident to a node v, we define $\mathcal{D}^{\mathcal{A}}_{v,e_1,e_2}$ to be the set of all size-2 multisets $\{a, b\}$ of labels such that $\{a, b\}$ is a possible outcome of labeling the two half-edges (v, e_1) and (v, e_2) when we run the algorithm $\mathcal{A}$ on G.

Definition 5.10 (Induction hypothesis for layer $S_{\mathsf{R},i}$)**.** *For each $1 \le i \le k+1$, the induction hypothesis for $S_{\mathsf{R},i}$ specifies that each $v \in S_{\mathsf{R},i}$ satisfies $\mathcal{V}^{\mathcal{A}}_v \subseteq \mathcal{V}_i$.*

Definition 5.11 (Induction hypothesis for layer $S_{\mathsf{C},i}$)**.** *For each $1 \le i \le k$, the induction hypothesis for $S_{\mathsf{C},i}$ specifies that for each $v \in S_{\mathsf{C},i}$ and any two incident edges $e_1 = \{v, u\}$ and $e_2 = \{v, w\}$ such that u and w are in layer (C, i) or higher, we have $\mathcal{D}^{\mathcal{A}}_{v,e_1,e_2} \subseteq \mathcal{D}_i$.*

Next, we prove that the induction hypotheses stated in Definitions 5.10 and 5.11 hold.

Lemma 5.10 (Base case: $S_{\mathsf{R},1}$)**.** *The induction hypothesis for $S_{\mathsf{R},1}$ holds.*

Proof. Recall that the number γ satisfies the following property. For each subset $\mathcal{S} \subseteq \mathcal{V}$ and each $C \in \mathcal{S} \setminus \mathsf{trim}(\mathcal{S})$, there exists no correct labeling of T^*_γ where the node configuration of the root r is C and the node configuration of remaining degree-Δ nodes is in $\mathcal{S}$.

To prove the induction hypothesis for $S_{\mathsf{R},1}$, consider any node $v \in S_{\mathsf{R},1}$. By the definition of $S_{\mathsf{R},1}$, the radius-γ neighborhood of v is isomorphic to T^*_γ rooted at v, and, by setting $\mathcal{S} = \mathcal{V}$, we infer that there is no correct labeling of G such that the node configuration of v is in $\mathcal{V} \setminus \mathsf{trim}(\mathcal{V}) = \mathcal{V} \setminus \mathcal{V}_1$, so we must have $\mathcal{V}^{\mathcal{A}}_v \subseteq \mathcal{V}_1$, as $\mathcal{A}$ is correct. □

Lemma 5.11 (Inductive step: $S_{\mathsf{R},i}$)**.** *Let $2 \le i \le k$. If the induction hypothesis for $S_{\mathsf{R},i-1}$ and $S_{\mathsf{C},i-1}$ holds, then the induction hypothesis for $S_{\mathsf{R},i}$ holds.*

Proof. To prove the induction hypothesis for layer $S_{\mathsf{R},i}$, consider any node $v \in S_{\mathsf{R},i}$. By the definition of $S_{\mathsf{R},i}$, the radius-γ neighborhood of v is isomorphic to T^*_γ rooted at v and contains only nodes from $S_{\mathsf{C},i-1}$. The goal is to prove that $\mathcal{V}^{\mathcal{A}}_v \subseteq \mathcal{V}_i = \mathsf{trim}(\mathcal{V}_{i-1} \restriction_{\mathcal{D}_{i-1}})$.

The set of degree-Δ nodes in this T^*_γ is precisely the set of nodes in the radius-$(\gamma - 1)$ neighborhood of v in G. Consider any node u in the radius-$(\gamma - 1)$ neighborhood of v. As u is in $S_{\mathsf{C},i-1} \subseteq S_{\mathsf{R},i-1}$, the induction hypothesis for $S_{\mathsf{R},i-1}$ implies that

$$\mathcal{V}^{\mathcal{A}}_u \subseteq \mathcal{V}_{i-1}.$$

Since all neighbors of u are in $S_{\mathsf{C},i-1}$, from the induction hypothesis for $S_{\mathsf{C},i-1}$, we have

$$\mathcal{D}^{\mathcal{A}}_{u,e_1,e_2} \subseteq \mathcal{D}_{i-1}$$

for any two edges e_1 and e_2 incident to u. Combining these two facts, we infer that

$$\mathcal{V}^{\mathcal{A}}_u \subseteq \mathcal{V}_{i-1} \restriction_{\mathcal{D}_{i-1}}.$$

Given that all degree-Δ nodes u in this T^*_γ satisfy that $\mathcal{V}^{\mathcal{A}}_u \subseteq \mathcal{V}_{i-1} \restriction_{\mathcal{D}_{i-1}}$, the same argument as in the proof of Lemma 5.10 shows that $\mathcal{V}^{\mathcal{A}}_v \subseteq \mathsf{trim}(\mathcal{V}_{i-1} \restriction_{\mathcal{D}_{i-1}}) = \mathcal{V}_i$, as required. □

Lemma 5.12 (Inductive step: $S_{\mathsf{C},i}$)**.** *Let $1 \le i \le k$. If the induction hypothesis for $S_{\mathsf{R},i}$ holds, then the induction hypothesis for $S_{\mathsf{C},i}$ holds.*

Proof. To prove the induction hypothesis for $S_{\mathsf{C},i}$, we show that there exists a choice $\mathcal{D}_i \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{V}_i})$ such that the following statement holds. For any node $v \in S_{\mathsf{C},i}$ and any two incident edges $e_1 = \{v, u\}$ and $e_2 = \{v, w\}$ such that u and w are in layer (C, i) or higher, we have $\mathcal{D}^{\mathcal{A}}_{v,e_1,e_2} \subseteq \mathcal{D}_i$.

We first make an observation about central nodes in layer (C, i). Let v be a central node in layer (C, i), and let u and w be the two neighbors of v in layer (C, i). Let $e_1 = \{v, u\}$ and $e_2 = \{v, w\}$. Then it is clear that $\mathcal{D}^{\mathcal{A}}_{v,e_1,e_2}$ is the same for each choice of v that is a central node in layer (C, i), as the radius-t neighborhoods of central nodes in the same layer are isomorphic, due to the definition of central nodes and the construction in Definition 5.6. For notational convenience, we write $\tilde{\mathcal{D}} = \mathcal{D}^{\mathcal{A}}_{v,e_1,e_2}$ to denote this set.

Plan of the proof Consider any node $v \in S_{\mathsf{C},i}$ and its two incident edges $e_1 = \{v, u\}$ and $e_2 = \{v, w\}$ such that u and w are in layer (C, i) or higher. Due to the induction hypothesis for $S_{\mathsf{R},i}$, we already have $\mathcal{V}_v^{\mathcal{A}} \subseteq \mathcal{V}_i$ as $v \in S_{\mathsf{C},i} \subseteq S_{\mathsf{R},i}$, and so $\mathcal{D}_{v,e_1,e_2}^{\mathcal{A}} \subseteq \mathcal{D}_{\mathcal{V}_i}$.

To prove that there exists a choice $\mathcal{D}_i = \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{V}_i})$ such that $\mathcal{D}_{v,e_1,e_2}^{\mathcal{A}} \subseteq \mathcal{D}_i$ for all such v, e_1, and e_2, we will first show that $\tilde{\mathcal{D}}$ must be a subset of a path-flexible strongly connected component of $\mathcal{D}_{\mathcal{V}_i}$, and then we fix $\mathcal{D}_i \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{V}_i})$ to be this path-flexible strongly connected component.

Next, we will argue that for each $\{a, b\} \in \mathcal{D}_{v,e_1,e_2}^{\mathcal{A}}$, there exist a walk in $\mathcal{M}_{\mathcal{D}_{\mathcal{V}_i}}$ that starts from (a, b) and ends in $\tilde{\mathcal{D}}$ and a walk in $\mathcal{M}_{\mathcal{D}_{\mathcal{V}_i}}$ that starts from $\tilde{\mathcal{D}}$ and ends in (a, b). This shows that $\{a, b\}$ is in the same strongly connected component as the members in $\tilde{\mathcal{D}}$, so we conclude that $\mathcal{D}_{v,e_1,e_2}^{\mathcal{A}} \subseteq \mathcal{D}_i$.

Part 1: $\tilde{\mathcal{D}}$ is a subset of some $\mathcal{D}_i \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{V}_i})$ Consider any path $(u_1, u_2, \ldots, u_s)$ of s nodes in layer (C, i) of $G = G_{\mathsf{R},k+1}^*$. Note that any such a path must be the s-node path $(v_1, v_2, \ldots, v_s)$ in the construction of some $G_{\mathsf{C},i}$ in Definition 5.6. We choose $s = \Theta(t)$ to be sufficiently large to ensure that for each integer $0 \leq d \leq |\Sigma|^2$, there exist two nodes u_j and u_l in the path meeting the following conditions.

- $t + 1 \leq j < l \leq s - t$, so u_j and u_l are central nodes.
- The distance $l - j$ between u_j and u_l equals $4t + 3 + d$.

The choice of the number $4t + 3$ is to ensure that the union of the radius-t neighborhoods of the endpoints of any edge in G does not node-intersect the radius-t neighborhood of both u_j and u_l. This implies that after arbitrarily fixing distinct IDs in the radius-t neighborhood of u_j and u_l, it is possible to complete the ID assignment of the entire graph G in such a way that the union of the radius-t neighborhoods of the endpoints of each edge in G does not contain repeated IDs. If we run $\mathcal{A}$ with such an ID assignment, it is guaranteed that the output is correct.

Consider the directed graph $\mathcal{M}_{\mathcal{D}_{\mathcal{V}_i}}$ and any two of its nodes (a, b) and (c, d) such that $\{a, b\} \in \tilde{D}$ and $\{c, d\} \in \tilde{D}$. Our choice of $\tilde{D}$ implies that there exists an assignment of distinct IDs to the radius-t neighborhood of both u_j and u_l such that the output labels of $(u_j, \{u_j, u_{j-1}\})$, $(u_j, \{u_j, u_{j+1}\})$, $(u_l, \{u_l, u_{l-1}\})$, and $(u_l, \{u_l, u_{l+1}\})$ are a, b, c, and d, respectively. We complete the ID assignment of the entire graph G in such a way that the radius-t neighborhood of each edge in G does not contain repeated IDs. For each node u_y with $j < y < l$, the pair of the two output labels of $(u_y\{u_y, u_{y-1}\})$ and $(u_y, \{u_y, u_{y+1}\})$ resulting from $\mathcal{A}$ is also a node in $\mathcal{M}_{\mathcal{D}_{\mathcal{V}_i}}$. Hence there exists a walk $(a, b) \rightsquigarrow (c, d)$ of length $4t + 3 + d$, for any $0 \leq d \leq |\Sigma|^2$. Since this holds for all choices of (a, b) and (c, d) such that $\{a, b\} \in \tilde{D}$ and $\{c, d\} \in \tilde{D}$, all members in $\tilde{D}$ are in the same strongly connected component. Furthermore, Lemma 5.2 implies that this strongly connected component is path-flexible, as the length of the walk can be any integer in between $4t + 3$ and $4t + 3 + |\Sigma|^2$.

Part 2: $\mathcal{D}_{v,e_1,e_2}^{\mathcal{A}} \subseteq \mathcal{D}_i$ For this part, we use Lemma 5.9, which shows that in the graph $G = G_{\mathsf{R},k+1}^*$ there are a path P_1 from v to a central node in layer (C, i) through e_1 and a path P_2 from v to a central node in layer (C, i) through e_2, and these paths use only nodes in $S_{\mathsf{C},i}$.

Consider the output labels resulting from running $\mathcal{A}$. Let a be the label of the half-edge (v, e_1) and let b be the label of the half-edge (v, e_2). We have $\{a, b\} \in \mathcal{D}_{v,e_1,e_2}^{\mathcal{A}} \subseteq \mathcal{D}_{\mathcal{V}_i}$. By taking the output labels in P_1 and P_2 resulting from running $\mathcal{A}$, we obtain two walks in the directed graph $\mathcal{M}_{\mathcal{D}_{\mathcal{V}_i}}$: $(a, b) \rightsquigarrow (c, d)$ and $(b, a) \rightsquigarrow (e, f)$, where both (c, d) and (e, f) are nodes in $V(\mathcal{M}_{\mathcal{D}_{\mathcal{V}_i}})$ such that $\{a, b\} \in \tilde{D}$ and $\{c, d\} \in \tilde{D}$. By taking the opposite directions, we also obtain two walks:

$(d,c) \rightsquigarrow (b,a)$ and $(f,e) \rightsquigarrow (a,b)$. Hence $\{a,b\}$ is in the same strongly connected component of $\mathcal{D}_{\mathcal{V}_i}$ as the members in $\tilde{D}$.

The same argument can be applied to all $\{a,b\} \in \mathcal{D}^{\mathcal{A}}_{v,e_1,e_2}$. The reason is that for each $\{a,b\} \in \mathcal{D}^{\mathcal{A}}_{v,e_1,e_2}$ there is an assignment of distinct IDs such that $\{a,b\}$ is the multiset of the two labels of (v,e_1) and (v,e_2). Hence we conclude that all members in $\mathcal{D}^{\mathcal{A}}_{v,e_1,e_2}$ are within the same strongly connected component as that of members in $\tilde{\mathcal{D}}$, so $\mathcal{D}^{\mathcal{A}}_{v,e_1,e_2} \subseteq \mathcal{D}_i$. □

Applying Lemmas 5.10 to 5.12 from $S_{\mathsf{R},1}$ all the way up to the last subset $S_{\mathsf{R},k+1}$, we obtain the following result.

Lemma 5.13 (Lower bound for the case $d_\Pi = k$)**.** *If $d_\Pi = k$ for a finite integer k, then Π requires $\Omega(n^{1/k})$ rounds to solve on trees of maximum degree Δ.*

Proof. Assume that there is a t-round algorithm solving Π on G. By Lemmas 5.10 to 5.12, we infer that the induction hypothesis for the last subset $S_{\mathsf{R},k+1}$ holds. By Lemma 5.8, $S_{\mathsf{R},k+1} \neq \emptyset$, so there is a node v in G such that $\mathcal{V}^{\mathcal{A}}_v \subseteq \mathcal{V}_{k+1}$. Therefore, the correctness of $\mathcal{A}$ implies that $\mathcal{V}_{k+1} \neq \emptyset$, which implies that $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_{k+1})$ chosen in the induction hypothesis is a good sequence, contradicting the assumption that $d_\Pi = k$. Hence such a t-round algorithm $\mathcal{A}$ that solves Π does not exist. As this argument holds for all integers t and $t = \Omega(n^{1/k})$, where n is the number of nodes in G, we conclude the proof. □

Now we are ready to prove Theorem 5.1.

Proof of Theorem 5.1. The upper bound part of the theorem follows from Lemmas 5.5 and 5.6. The lower bound part of the theorem follows from Lemmas 5.7 and 5.13. □

5.4 Complexity of the characterization

In this section, we prove Theorem 5.2. We are given a description of an LCL problem $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ on Δ-regular trees. We assume that the description is given in the form of listing the multisets in $\mathcal{V}$ and $\mathcal{E}$. Therefore, the description length of Π is $\ell = O(\log|\Sigma|) \cdot (|\mathcal{E}| \cdot 2 + |\mathcal{V}| \cdot \Delta)$. Here we allow Δ to be a non-constant, as a function of ℓ. We will design an algorithm that computes all possible good sequences $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_k)$ in time polynomial in ℓ, and this allows us to compute d_Π. As, the main objective of this section is to show that the problem is polynomial-time solvable, we do not aim to optimize the time complexity of our algorithm.

For the case of $d_\Pi = \infty$, there are good sequences that are arbitrarily long. Recall that we have $\mathcal{V}_1 \supseteq \mathcal{V}_2 \supseteq \cdots \supseteq \mathcal{V}_k$. Hence if $k > |\mathcal{V}|$, there must exist some index $1 \leq i < k$ such that $\mathcal{V}_i = \mathcal{V}_{i+1}$. This immediately implies that $\mathcal{V}_i = \mathcal{V}_{i+1} = \mathcal{V}_{i+2} = \cdots$, due to the following reasoning. The fact that $\mathcal{V}_i = \mathcal{V}_{i+1}$ implies that $\mathcal{V}_{i+1} = \mathsf{trim}(\mathcal{V}_i \restriction_{\mathcal{D}_i}) = \mathcal{V}_i \restriction_{\mathcal{D}_i} = \mathcal{V}_i$, so $\mathcal{D}_i = \mathcal{D}_{\mathcal{V}_i}$. This means that $\mathcal{D}_i = \mathcal{D}_{\mathcal{V}_i}$ itself is the only element of $\mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{V}_i})$. Therefore, starting from $\mathcal{V}_i$, the multisets $\mathcal{D}_i = \mathcal{D}_{\mathcal{V}_i}$ and $\mathcal{V}_{i+1} = \mathcal{V}_i$ are uniquely determined. Similarly, we have $\mathcal{D}_j = \mathcal{D}_i$ and $\mathcal{V}_j = \mathcal{V}_i$ for all $j \geq i$. We conclude that any good sequence with $k > |V|$ must stabilize at some point $i \leq |\mathcal{V}|$, in the sense that $\mathcal{D}_j = \mathcal{D}_i$ and $\mathcal{V}_j = \mathcal{V}_i$ for all $j \geq i$.

High-level plan Recall that the rules for a good sequence are as follows:

$$\mathcal{V}_i = \begin{cases} \mathsf{trim}(\mathcal{V}) & \text{if } i = 1, \\ \mathsf{trim}(\mathcal{V}_{i-1} \restriction_{\mathcal{D}_{i-1}}) & \text{if } i > 1, \end{cases}$$
$$\mathcal{D}_i \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{V}_i}).$$

To compute all good sequences $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_k)$, we go through all choices of $\mathcal{D}_i \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{V}_i})$ and apply the rules recursively until we cannot proceed any further. The process stops when $\mathcal{V}_i = \mathcal{V}_{i-1}$ (the sequence stabilizes) or $\mathcal{V}_i = \emptyset$ (the sequence ends).

In time $O(|\Sigma|^2)$, we build a look-up table that allows us to check whether $\{\alpha, \beta\} \in \mathcal{E}$ in $O(1)$ time, for any given size-2 multiset $\{\alpha, \beta\}$. In the subsequent discussion, we assume that such a look-up table is available.

We start with describing the algorithm for computing $\mathsf{trim}(\mathcal{S})$ for a given $\mathcal{S} \subseteq \mathcal{V}$.

Lemma 5.14 (Algorithm for trim)**.** *The set $\mathsf{trim}(\mathcal{S})$ can be computed in $O(\Delta|\mathcal{S}||\Sigma|^2)$ time, for any given $\mathcal{S} \subseteq \mathcal{V}$.*

Proof. We write T_i' to denote the tree resulting from adding one extra edge e^* incident to the root r of T_i. We write Σ_i to denote set of all possible $\sigma \in \mathcal{S}$ such that there is a correct labeling of T_i' where the node configuration of each degree-Δ node is in $\mathcal{S}$ and the half-edge label of (r, e^*) is σ. The set Σ_i can be computed recursively as follows.

- For the base case, Σ_1 is the set of all labels appearing in $\mathcal{S}$.
- For the inductive step, each $\sigma \in \Sigma_{i-1}$ is added to Σ_i if there exists $C \in \mathcal{S}$ such that $\sigma \in C$ and each of the $\Delta - 1$ labels α in $C \setminus \{\sigma\}$ satisfies that $\{\alpha, \beta\} \in \mathcal{E}$ for some $\beta \in \Sigma_{i-1}$.

Using the above look-up table, given that Σ_{i-1} has been computed, the computation of Σ_i costs $O(\Delta|\mathcal{S}||\Sigma_{i-1}|) = O(\Delta|\mathcal{S}||\Sigma|)$ time. Clearly, we have $\Sigma_1 \supseteq \Sigma_2 \supseteq \cdots$, and whenever $\Sigma_i = \Sigma_{i+1}$, the sequence stabilizes: $\Sigma_i = \Sigma_{i+1} = \Sigma_{i+2} = \cdots$. We write Σ^* to denote the fix point Σ_i such that $\Sigma_i = \Sigma_{i+1} = \Sigma_{i+2} = \cdots$. It is clear that $i \leq |\Sigma|$, so the fixed point Σ^* can be computed in $O(\Delta|\mathcal{S}||\Sigma|^2)$ time.

Given the fix point Σ^*, the set $\mathsf{trim}(\mathcal{S})$ can be computed as follows. Observe that the tree T_i^* is simply the result of merging Δ trees T_{i-1}' by merging the Δ degree-1 endpoints of e^* into one node r. Therefore, $C \in \mathsf{trim}(\mathcal{S})$ if and only if each of the Δ labels $\alpha \in C$ satisfies that $\{\alpha, \beta\} \in \mathcal{E}$ for some $\beta \in \Sigma^*$. Using this characterization, given the fix point Σ^*, the set $\mathsf{trim}(\mathcal{S})$ can be similarly computed in $O(\Delta|\mathcal{S}||\Sigma|)$ time. We conclude the following result. □

Next, we give an algorithm that computes all path-flexible strongly connected components $\mathcal{D}' \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{S}})$ and their corresponding restriction $\mathcal{S} \restriction_{\mathcal{D}'}$, for any given $\mathcal{S} \subseteq \mathcal{V}$.

Lemma 5.15 (Algorithm for $\mathsf{flexible\text{-}SCC}$)**.** *The set of all $\mathcal{D}' \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_{\mathcal{S}})$ and their corresponding restrictions $\mathcal{S} \restriction_{\mathcal{D}'}$ can be computed in $O(\Delta^8|\mathcal{S}|^4)$ time, for any given $\mathcal{S} \subseteq \mathcal{V}$.*

Proof. Observe that $|\mathcal{D}_{\mathcal{S}}| = O(\Delta^2|\mathcal{S}|)$, so the directed graph $\mathcal{M}_{\mathcal{D}_{\mathcal{S}}}$ has $O(\Delta^2|\mathcal{S}|)$ nodes and $O(\Delta^4|\mathcal{S}|^2)$ edges.

Using the definition of Definition 5.5, testing whether $\{a, b\} \sim \{c, d\}$ for any $\{a, b\} \in \mathcal{D}_{\mathcal{S}}$ and $\{c, d\} \in \mathcal{D}_{\mathcal{S}}$ costs $O(|E(\mathcal{M}_{\mathcal{D}_{\mathcal{S}}})|) = O(\Delta^4|\mathcal{S}|^2)$ time by doing four s-t reachability computation. By going over all $\{a, b\} \in \mathcal{D}_{\mathcal{S}}$ and $\{c, d\} \in \mathcal{D}_{\mathcal{S}}$, the set of all strongly connected components of $\mathcal{D}_{\mathcal{S}}$ can be computed in time $O(|E(\mathcal{M}_{\mathcal{D}_{\mathcal{S}}})| \cdot |V(\mathcal{M}_{\mathcal{D}_{\mathcal{S}}})|^2) = O(\Delta^8|\mathcal{S}|^4)$.

For each strongly connected component $\mathcal{D}'$ of $\mathcal{D}_{\mathcal{S}}$, to decide whether $\mathcal{D}'$ is path-flexible, it suffices to pick one element $\{a, b\} \in \mathcal{D}'$ and check if $\{a, b\}$ is path-flexible. Recall that $\{a, b\}$ is path-flexible if there exists an integer K such that for each integer $k \geq K$, there exist length-s walks $(a, b) \rightsquigarrow (a, b)$, $(a, b) \rightsquigarrow (b, a)$, $(b, a) \rightsquigarrow (a, b)$, and $(b, a) \rightsquigarrow (b, a)$ in $\mathcal{M}_{\mathcal{D}}$.

The fact that $(a, b) \in \mathcal{D}'$ and $\mathcal{D}'$ is a strongly connected component of $\mathcal{D}_{\mathcal{S}}$ implies the existence of walks $(a, b) \rightsquigarrow (a, b)$, $(a, b) \rightsquigarrow (b, a)$, $(b, a) \rightsquigarrow (a, b)$, and $(b, a) \rightsquigarrow (b, a)$ in $\mathcal{M}_{\mathcal{D}}$. Therefore, the task for deciding whether $\mathcal{D}'$ is path-flexible is reduced to the following task. Given a node $s = (a, b)$ in the directed graph $\mathcal{M}_{\mathcal{D}_{\mathcal{S}}}$, let L be the set of possible lengths of an $s \rightsquigarrow s$ walk, check

if there exists an integer K such that L contains all integers that are at least K. Such an integer K exists if and only if the greatest common divisor $\gcd(L)$ of L is not one. Define L' as the set of numbers in L that are at most $2|V(\mathcal{M}_{\mathcal{D}_\mathcal{S}})| - 1$. As shown in [22], we have $\gcd(L) = \gcd(L')$.

The computation of $\gcd(L')$ can be done in $O(|V(\mathcal{M}_{\mathcal{D}_\mathcal{S}})|^3)$ time, as follows. From $i = 0$ up to $i = 2|V(\mathcal{M}_{\mathcal{D}_\mathcal{S}})| - 1$, we compute a list of nodes $U_i \subseteq V(\mathcal{M}_{\mathcal{D}_\mathcal{S}})$ such that $v \in U_i$ if there is a walk $s \rightsquigarrow v$ of length i. Given U_{i-1}, it takes $O(|V(\mathcal{M}_{\mathcal{D}_\mathcal{S}})|^2)$ time to compute U_i, as we just need to go over all $O(|V(\mathcal{M}_{\mathcal{D}_\mathcal{S}})|^2)$ edges between the nodes in $V(\mathcal{M}_{\mathcal{D}_\mathcal{S}})$. The summation of the time complexity $O(|V(\mathcal{M}_{\mathcal{D}_\mathcal{S}})|^3)$, over all strongly connected component of $V(\mathcal{M}_{\mathcal{D}_\mathcal{S}})$, is $O(|V(\mathcal{M}_{\mathcal{D}_\mathcal{S}})|^4) = O(\Delta^8|\mathcal{S}|^4)$.

To summarize, the computation of $\mathsf{flexible\text{-}SCC}(\mathcal{D}_\mathcal{S})$ costs $O(\Delta^8|\mathcal{S}|^4)$ time. Given that we have computed all path-flexible strongly connected components $\mathcal{D}' \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_\mathcal{S})$, the computation of the restriction $\mathcal{S} \restriction_{\mathcal{D}'}$ for all path-flexible strongly connected components $\mathcal{D}' \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_\mathcal{S})$ costs $O(|\mathcal{D}_\mathcal{S}| + \Delta^2|\mathcal{S}|) = O(\Delta^2|\mathcal{S}|)$ time, as we just need to check for each $C \in \mathcal{S}$ whether there is $\mathcal{D}' \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_\mathcal{S})$ such that all Δ^2 size-2 sub-multisets of C belong to the $\mathcal{D}'$. □

Combining Lemmas 5.14 and 5.15, we obtain the following result.

Lemma 5.16 (Computing all good sequences)**.** *The set of all good sequences can be computed in* $O(\Delta|\mathcal{V}|^2|\Sigma|^2 + \Delta^8|\mathcal{V}|^5)$ *time, for any given* LCL *problem* $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$.

Proof. Combining Lemmas 5.14 and 5.15, we infer that given $\mathcal{V}_i \subseteq \mathcal{V}$, the cost of computing all possible $(\mathcal{D}_i, \mathcal{V}_{i+1})$ is $O(\Delta|\mathcal{V}_i||\Sigma|^2 + \Delta^8|\mathcal{V}_i|^4)$ time, as the set of $\mathcal{V}_i \restriction_{\mathcal{D}'}$ over all $\mathcal{D}' \in \mathsf{flexible\text{-}SCC}(\mathcal{D}_i)$ are disjoint subsets of $\mathcal{V}_i$.

Since all the sets $\mathcal{V}_i$ in the depth i of the recursion are disjoint subsets of $\mathcal{V}$, the total cost for the depth i of the recursion is $O(\Delta|\mathcal{V}||\Sigma|^2 + \Delta^8|\mathcal{V}|^4)$.

The recursion stops when $\mathcal{V}_i = \mathcal{V}_{i-1}$ or $\mathcal{V}_i = \emptyset$, so the depth of the recursion is at most $|\mathcal{V}|$. The reason is that we must have $|\mathcal{V}_i| < |\mathcal{V}_{i-1}|$ if $\mathcal{V}_i \neq \mathcal{V}_{i-1}$ and $\mathcal{V}_i \neq \emptyset$. Therefore, the total cost of computing all good sequences is $O(\Delta|\mathcal{V}|^2|\Sigma|^2 + \Delta^8|\mathcal{V}|^5)$. □

We are ready to prove Theorem 5.2.

Proof of Theorem 5.2. By Lemma 5.16, the set of all good sequences can be computed in polynomial time, and we can compute d_Π given the set of all good sequences. If $d_\Pi = k$ is a positive integer, then from the discussion in Section 5.2 we know how to turn a good sequence $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_k)$ into a description of an $O(n^{1/k})$-round algorithm for Π. If $d_\Pi = \infty$, then similarly a good sequence $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{V}_2, \mathcal{D}_2, \ldots, \mathcal{V}_{O(\log n)})$ leads to a description of an $O(\log n)$-round algorithm for Π. □

6 Rooted trees

In this section, we give a polynomial-time-computable characterization of LCL problems for regular rooted trees with complexity $O(\log n)$ or $\Theta(n^{1/k})$ for any positive integer k.

6.1 Locally checkable labeling for rooted trees

A rooted tree is a tree where each edge is oriented in such a way that the outdegree of each node is at most 1. A δ-regular rooted tree is a rooted tree where the indegree of each node is either 0 or δ. The root of a rooted tree is the unique node v with $\deg_{\mathsf{out}}(v) = 0$. Each node v with $\deg_{\mathsf{in}}(v) = 0$ is called a leaf. For each directed edge $u \to v$, we say that u is a child of v and v is the parent of u. An LCL problem for δ-regular rooted trees is defined as follows.

Definition 6.1 (LCL problems for regular rooted trees)**.** *For rooted trees, an* LCL *problem* $\Pi = (\delta, \Sigma, \mathcal{C})$ *is defined by the following components.*

- δ *is a positive integer specifying the maximum indegree.*
- Σ *is a finite set of labels.*
- $\mathcal{C}$ *is a set of pairs* (σ, S) *such that* $\sigma \in \Sigma$ *and* S *is a size-*δ *multiset of labels in* Σ*.*

For notational simplicity, we also write $(\sigma : a_1 a_2 \cdots a_\delta)$ to denote (σ, S), where $\sigma \in \Sigma$ and $S = \{a_1, a_2, \ldots, a_\delta\}$ is a size-δ multiset of elements in Σ. We call any such (σ, S) a node configuration. A node configuration (σ, S) is correct if $(\sigma, S) \in \mathcal{C}$. In this sense, the set $\mathcal{C}$ specifies the node constraint. We define the correctness criteria for a labeling in Definition 6.2.

Definition 6.2 (Correctness criteria)**.** *Let* $G = (V, E)$ *be a rooted tree whose maximum indegree is at most* δ*. A solution of* $\Pi = (\delta, \Sigma, \mathcal{C})$ *on* G *is a labeling that assigns a label in* Σ *to each node in* G*.*

- *For each node* $v \in V$ *with* $\deg_{\mathsf{in}}(v) = \delta$*, we define its node configuration* $C = (\sigma : a_1 a_2 \cdots a_\delta)$ *by setting* σ *as the label of* v *and setting* $\{a_1, a_2, \ldots, a_\delta\}$ *as the multiset of the labels of the* δ *children of* v*. We say that the labeling is locally-consistent on* v *if* $C \in \mathcal{C}$*.*

The labeling is a correct solution if it is locally-consistent on all $v \in V$ *with* $\deg_{\mathsf{in}}(v) = \delta$*.*

Similarly, although $\Pi = (\delta, \Sigma, \mathcal{C})$ is defined for δ-regular rooted trees, Definition 6.2 applies to all rooted trees whose maximum indegree is at most δ. We may focus on δ-regular rooted trees without loss of generality, as for any rooted tree G whose maximum indegree is at most δ, we may consider the rooted tree G^* which is the result of appending leaf nodes to all nodes v in G with $1 < \deg_{\mathsf{in}}(v) < \delta$ to increase the indegree of v to δ. This only blows up the number of nodes by at most a δ factor. Any correct solution of Π on G^* restricted to G is a correct solution of Π on G.

Definition 6.3 is the same as Definition 5.3 except that we change $\Delta - 1$ to δ.

Definition 6.3 (Complete trees of height i)**.** *We define the rooted trees* T_i *recursively as follows.*

- T_0 *is the trivial tree with only one node.*
- T_i *is the result of appending* δ *trees* T_{i-1} *to the root* r*.*

Observe that T_i is the unique maximum-size rooted tree of maximum indegree δ and radius i. All nodes within distance $i - 1$ to the root r in T_i have indegree δ. All nodes whose distance to r is exactly i are leaf nodes.

Definition 6.4 (Trimming)**.** *Given a subset* $\tilde{\Sigma} \subseteq \Sigma$ *of labels, we define* $\mathsf{trim}(\tilde{\Sigma})$ *as a set of all labels* $\sigma \in \tilde{\Sigma}$ *such that for each* $i \geq 1$ *it is possible to find a correct labeling of* T_i *such that the label of the root is* σ *and the label of the remaining nodes are in* $\tilde{\Sigma}$*.*

Given any rooted tree G of maximum indegree δ, after labeling the a node v with $\deg_{\mathsf{in}}(v) = \delta$ with a label $\sigma \in \mathsf{trim}(\tilde{\Sigma})$, it is always possible to extend this labeling to a complete correct labeling of the subtree rooted at v using only node configurations in $\mathsf{trim}(\tilde{\Sigma})$. Such a labeling extension is possible due to Lemma 6.1.

Lemma 6.1 (Property of trimming)**.** *Let* $\Pi = (\delta, \Sigma, \mathcal{C})$ *be an* LCL *problem. Let* $\tilde{\Sigma} \subseteq \Sigma$ *such that* $\mathsf{trim}(\tilde{\Sigma}) \neq \emptyset$*. For each label* $\sigma \in \mathsf{trim}(\tilde{\Sigma})$*, there exists a node configuration* $(\sigma : a_1 a_2 \cdots a_\delta) \in \mathcal{C}$ *such that* $a_i \in \mathsf{trim}(\tilde{\Sigma})$ *for all* $1 \leq i \leq \delta$*.*

Proof. Assuming that such a node configuration $(\sigma : a_1 a_2 \cdots a_\delta) \in \mathcal{C}$ do not exist, we derive a contradiction as follows. We pick s to be the smallest number such that there is no correct labeling

of T_s where the label of the root r is in $\tilde{\Sigma} \setminus \mathsf{trim}(\tilde{\Sigma})$ and the label of each remaining node of T_s is in $\tilde{\Sigma}$. Such a number s exists due to the definition of trim.

Now consider a correct labeling of T^*_{s+1} where the label of the root r is σ and the label of each remaining node is in $\tilde{\Sigma}$. Such a correct labeling exists due to the fact that $\sigma \in \mathsf{trim}(\tilde{\Sigma})$. Our assumption on the non-existence of $(\sigma \ : \ a_1 a_2 \cdots a_\delta) \in \mathcal{C}$ such that $a_i \in \mathsf{trim}(\tilde{\Sigma})$ for all $1 \leq i \leq \delta$ implies that the label b_i of one child u_i of the root r of T_{s+1} must be in $\tilde{\Sigma} \setminus \mathsf{trim}(\tilde{\Sigma})$. However, the subtree of T_{s+1} rooted at u_i is isomorphic to the rooted tree T_s. Since the label b_i of u_i is in $\tilde{\Sigma} \setminus \mathsf{trim}(\tilde{\Sigma})$, our choice of s implies that the labeling of the subtree of T_{s+1} rooted at u_i cannot be correct, which is a contradiction. □

Restriction of an LCL problem Given a subset $\tilde{\Sigma} \subseteq \Sigma$ of labels, we define the restriction of Π to $\tilde{\Sigma}$ as follows.

$$\Pi \restriction_{\tilde{\Sigma}} = (\tilde{\Sigma}, \tilde{\mathcal{C}}),$$
$$\text{where } \tilde{\mathcal{C}} = \begin{array}{l}\text{the set of all } (\sigma \ : \ a_1 a_2 \cdots a_\delta) \in \mathcal{C} \text{ such that } \sigma \in \tilde{\Sigma} \\ \text{and } a_i \in \tilde{\Sigma} \text{ for all } 1 \leq i \leq \delta.\end{array}$$

That is, $\Pi \restriction_{\tilde{\Sigma}}$ is simply the result of removing all labels and node configurations in Π that involve labels not in $\tilde{\Sigma}$.

Path-form of an LCL problem Given an LCL problem $\Pi = (\delta, \Sigma, \mathcal{C})$, we define its path-form $\Pi^{\mathrm{path}} = (1, \Sigma, \mathcal{C}^{\mathrm{path}})$ as follows.

$$\mathcal{C}^{\mathrm{path}} = \begin{array}{l}\text{the set of all } (\sigma \ : \ a) \text{ such that there exists } (\sigma \ : \ a_1 a_2 \cdots a_\delta) \in \mathcal{C} \text{ such} \\ \text{that } a \in \{a_1, a_2, \ldots, a_\delta\}.\end{array}$$

Automaton for the path-form of an LCL problem The path-form $\Pi^{\mathrm{path}} = (1, \Sigma, \mathcal{C}^{\mathrm{path}})$ of an LCL problem $\Pi = (\delta, \Sigma, \mathcal{C})$ can be interpreted as a directed graph, where the node set is Σ and the edge set is $\mathcal{C}^{\mathrm{path}}$, by viewing each $(\sigma \ : \ a) \in \mathcal{C}^{\mathrm{path}}$ as a directed edge $a \to \sigma$.

Path-flexibility With respect to $\Pi^{\mathrm{path}} = (1, \Sigma, \mathcal{C}^{\mathrm{path}})$, we say that $\sigma \in \Sigma$ is path-flexible if there exists an integer K such that for each integer $s \geq K$, there exist a length-s walk $\sigma \rightsquigarrow \sigma$ in the directed graph of $\Pi^{\mathrm{path}} = (1, \Sigma, \mathcal{C}^{\mathrm{path}})$. Lemma 6.2 is useful in lower bound proofs.

Lemma 6.2 (Property of path-inflexibility)**.** *Suppose that $\sigma \in \Sigma$ is not path-flexible with respect to $\Pi^{\mathrm{path}} = (1, \Sigma, \mathcal{C}^{\mathrm{path}})$. Then one of the following holds.*

- *There is no walk $\sigma \rightsquigarrow \sigma$ in the directed graph of $\Pi^{\mathrm{path}} = (1, \Sigma, \mathcal{C}^{\mathrm{path}})$.*
- *There exists an integer $2 \leq x \leq |\Sigma|$ such that for any positive integer k that is not an integer multiple of x, there is no length-k walk $\sigma \rightsquigarrow \sigma$ in the directed graph of $\Pi^{\mathrm{path}} = (1, \Sigma, \mathcal{C}^{\mathrm{path}})$.*

Proof. Since σ is not path-flexible, for any integer K there is an integer $k \geq K$ such that there is no length-k walk $s \rightsquigarrow t$. Let U be the set of integers k such that there is a length-k walk $\sigma \rightsquigarrow \sigma$ in the directed graph of $\Pi^{\mathrm{path}} = (1, \Sigma, \mathcal{C}^{\mathrm{path}})$. If $U = \emptyset$, then there is no walk $\sigma \rightsquigarrow \sigma$, so the lemma statement holds. For the rest of the proof, we assume $U \neq \emptyset$. We choose $x = \gcd(U)$ to be the greatest common divisor of U. For any integer k that is not an integer multiple of x, there is no length-k walk $\sigma \rightsquigarrow \sigma$ in the directed graph of $\Pi^{\mathrm{path}} = (1, \Sigma, \mathcal{C}^{\mathrm{path}})$. We must have $x \geq 2$ because there cannot be two co-prime numbers in U, since otherwise there exists an integer K such that U includes all integers that are at least K, implying that σ is path-flexible. We also have $x \leq |\Sigma|$, since the smallest number of U is at most the number of nodes in the directed graph of $\Pi^{\mathrm{path}} = (1, \Sigma, \mathcal{C}^{\mathrm{path}})$, which is $|\Sigma|$. □

Path-flexible strongly connected components For any strongly connected component $U \subseteq \Sigma$ of the directed graph of $\Pi^{\mathrm{path}} = (1, \Sigma, \mathcal{C}^{\mathrm{path}})$, it is clear that either all $\sigma \in U$ are path-flexible or all $\sigma \in U$ are not path-flexible. We say that a strongly connected component U is path-flexible if all $\sigma \in U$ are path-flexible. We define $\mathsf{flexibility}(U)$ as the minimum number K such that for each integer $k \geq K$ there is an $a \rightsquigarrow b$ walk of length k for all choices of source $a \in U$ and destination $b \in U$. It is clear that such a number K exists given that U is a path-flexible strongly connected component.

Given any LCL problem $\Pi = (\delta, \Sigma, \mathcal{C})$ and any $\tilde{\Sigma} \subseteq \Sigma$, we define $\mathsf{flexible\text{-}SCC}(\tilde{\Sigma})$ as the set of all subsets $U \subseteq \tilde{\Sigma}$ that is a path-flexible strongly connected component of the directed graph of the path-form of $\Pi \restriction_{\tilde{\Sigma}}$. It is possible that $\mathsf{flexible\text{-}SCC}(\tilde{\Sigma})$ is an empty set, and this happens when all nodes in the directed graph of the path-form of $\Pi \restriction_{\tilde{\Sigma}}$ are not path-flexible.

Lemma 6.3 shows that if we label the two endpoints v_1 and v_{d+1} of a sufficiently long directed path $v_1 \leftarrow v_2 \leftarrow \cdots \leftarrow v_{d+1}$ using only labels in U, where $U \in \mathsf{flexible\text{-}SCC}(\tilde{\Sigma})$, then it is always possible to complete the labeling of the path using only labels in U in such a way that the entire labeling is correct with respect to $\Pi \restriction_{\tilde{\Sigma}}$. Specifically, suppose the label of v_1 is α and the label of v_{d+1} is β. Then Lemma 6.3 shows that it is possible to complete the labeling of $v_1 \leftarrow v_2 \leftarrow \cdots \leftarrow v_{d+1}$ by labeling v_i with $\sigma_i \in U$.

Lemma 6.3 (Property of path-flexible strongly connected components)**.** *Consider any* LCL *problem* $\Pi = (\delta, \Sigma, \mathcal{C})$ *and any* $\tilde{\Sigma} \subseteq \Sigma$*. Let* $U \in \mathsf{flexible\text{-}SCC}(\tilde{\Sigma})$*. For any choices of* $\alpha \in U$*,* $\beta \in U$*, and a number* $d \geq \mathsf{flexibility}(U)$*, there exists a sequence*

$$\sigma_1, \sigma_2, \ldots, \sigma_{d+1}$$

of labels in U *satisfying the following conditions.*

- *First endpoint:* $\sigma_1 = \alpha$*.*
- *Last endpoint:* $\sigma_{d+1} = \beta$*.*
- *Node configurations: for* $1 \leq i \leq d$*, there is a node configuration* $(\sigma \ : \ a_1 a_2 \cdots a_\delta) \in \mathcal{C}$ *meeting the following conditions.*
  - $\sigma_i = \sigma$*.*
  - *There is an index* j *such that* $\sigma_{i+1} = a_j$*.*
  - $a_l \in \tilde{\Sigma}$ *for all* $1 \leq l \leq \delta$*.*

Proof. By the path-flexibility of U, there exists a length-d walk $\beta \rightsquigarrow \alpha$ in the directed graph of the path-form of $\Pi \restriction_{\tilde{\Sigma}}$. We fix

$$\sigma_1 \leftarrow \sigma_2 \leftarrow \cdots \leftarrow \sigma_{d+1}$$

to be any such walk. For $1 \leq i \leq d$, $\sigma_i \leftarrow \sigma_{i+1}$ is a directed edge in the path-flexible strongly connected component U, meaning that $(\sigma_i \ : \ \sigma_{i+1})$ is a node configuration in the path-form of $\Pi \restriction_{\tilde{\Sigma}}$, so there exists a node configuration $(\sigma_i \ : \ a_1 a_2 \cdots a_\delta)$ in $\Pi \restriction_{\tilde{\Sigma}}$ such that $\sigma_{i+1} \in \{a_1, a_2, \ldots, a_\delta\}$. □

Good sequences Given an LCL problem $\Pi = (\delta, \Sigma, \mathcal{C})$ on δ-regular rooted trees, we say that a sequence

$$(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_k^{\mathsf{R}})$$

is *good* if it satisfies the following requirements.

- $\Sigma_1^{\mathsf{R}} = \mathsf{trim}(\Sigma)$. That is, we start the sequence from the result of trimming the set Σ of all labels in the given LCL problem $\Pi = (\delta, \Sigma, \mathcal{C})$.

- For each $1 \le i \le k-1$, $\Sigma_i^{\mathsf{C}} \in \mathsf{flexible\text{-}SCC}(\Sigma_i^{\mathsf{R}})$. That is, Σ_i^{C} is a path-flexible connected component of the automaton associated with the path-form of the LCL problem $\Pi|_{\Sigma_i^{\mathsf{R}}}$.
- For each $2 \le i \le k$, $\Sigma_i^{\mathsf{R}} = \mathsf{trim}(\Sigma_{i-1}^{\mathsf{C}})$. That is, Σ_i^{R} is the result of trimming the set $\Sigma_{i-1}^{\mathsf{C}}$.
- $\Sigma_k^{\mathsf{R}} \neq \emptyset$. That is, we require that the last set of labels is non-empty.

It is straightforward to see that $\Sigma \supseteq \Sigma_1^{\mathsf{R}} \supseteq \Sigma_1^{\mathsf{C}} \supseteq \Sigma_2^{\mathsf{R}} \supseteq \Sigma_2^{\mathsf{C}} \supseteq \cdots \supseteq \Sigma_k^{\mathsf{R}} \neq \emptyset$.

Depth of an LCL problem We define the depth d_Π of an LCL problem $\Pi = (\delta, \Sigma, \mathcal{C})$ on δ-regular rooted trees as follows. If there is no good sequence, then we set $d_\Pi = 0$. If there is a good sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_k^{\mathsf{R}})$ for each positive integer k, then we set $d_\Pi = \infty$. Otherwise, we set d_Π as the largest integer k such that there is a good sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_k^{\mathsf{R}})$. We prove the following results.

Theorem 6.1 (Characterization of complexity classes)**.** *Let $\Pi = (\delta, \Sigma, \mathcal{C})$ be an LCL problem on δ-regular rooted trees. We have the following.*

- *If $d_\Pi = 0$, then Π is unsolvable in the sense that there exists a rooted tree of maximum indegree δ such that there is no correct solution of Π on this rooted tree.*
- *If $d_\Pi = k$ is a positive integer, then the optimal round complexity of Π is $\Theta(n^{1/k})$.*
- *If $d_\Pi = \infty$, then Π can be solved in $O(\log n)$ rounds.*

In Theorem 6.1, all the upper bounds hold in the CONGEST model, and all the lower bounds hold in the LOCAL model. For example, if $d_\Pi = 5$, then Π can be solved in $O(n^{1/5})$ rounds in the CONGEST model, and there is a matching lower bound $\Omega(n^{1/5})$ in the LOCAL model.

Theorem 6.2 (Complexity of the characterization)**.** *There is a polynomial-time algorithm $\mathcal{A}$ that computes d_Π for any given LCL problem $\Pi = (\delta, \Sigma, \mathcal{C})$ on δ-regular rooted trees. If $d_\Pi = k$ is a positive integer, then $\mathcal{A}$ also outputs a description of an $O(n^{1/k})$-round algorithm for Π. If $d_\Pi = \infty$, then $\mathcal{A}$ also outputs a description of an $O(\log n)$-round algorithm for Π.*

The distributed algorithms returned by the polynomial-time algorithm $\mathcal{A}$ in Theorem 6.2 are also in the CONGEST model.

6.2 Upper bounds

In this section, we prove the upper bound part of Theorem 6.1. If a good sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_k^{\mathsf{R}})$ exists for some positive integer k, we show that the LCL problem $\Pi = (\delta, \Sigma, \mathcal{C})$ can be solved in $O(n^{1/k})$ rounds. If a good sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_k^{\mathsf{R}})$ exists for all positive integers k, we show that $\Pi = (\delta, \Sigma, \mathcal{C})$ can be solved in $O(\log n)$ rounds. All these algorithms do not require sending large messages and can be implemented in the CONGEST model.

Rake-and-compress decompositions Similar to the case of unrooted trees, we will use a variant of rake-and-compress decomposition for rooted trees. Our rake-and-compress decomposition for rooted trees is also parameterized by three positive integers γ, ℓ, and L. A (γ, ℓ, L) decomposition of a rooted tree $G = (V, E)$ is a partition of the node set

$$V = V_1^{\mathsf{R}} \cup V_1^{\mathsf{C}} \cup V_2^{\mathsf{R}} \cup V_2^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$$

satisfying the following requirements. The requirements will be different from the ones in Section 5.2 due to the difference between rooted trees and unrooted trees.

Requirements for V_i^{R} For each connected component S of the subgraph of G induced by V_i^{R}, it is required that S is a rooted tree meeting the following conditions.

- All nodes in S have no in-neighbor in $V_i^{\mathsf{C}} \cup V_{i+1}^{\mathsf{R}} \cup \cdots \cup V_L^{\mathsf{R}}$.
- All nodes in S are within distance $\gamma - 1$ to z.

Here we only require that the nodes in S do not have in-neighbors from the higher layers of the decomposition. Only the root z of S can possibly have an out-neighbor outside of S, and we do not restrict anything about this out-neighbor.

Requirements for V_i^{C} For each connected component S of the subgraph of G induced by V_i^{C}, S is a directed path $v_1 \leftarrow v_2 \leftarrow \cdots \leftarrow v_s$ of $s \in [\ell, 2\ell]$ nodes meeting the following conditions.

- There is a node $u \in V_{i+1}^{\mathsf{R}} \cup \cdots \cup V_L^{\mathsf{R}}$ such that $u \leftarrow v_1$.
- There is a node $w \in V_{i+1}^{\mathsf{R}} \cup \cdots \cup V_L^{\mathsf{R}}$ such that $v_s \leftarrow w$.
- Other than u and w, all the remaining neighbors of S are not in $V_{i+1}^{\mathsf{R}} \cup V_{i+1}^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$.

The rest of the section is organized as follows. In Section 6.2.1, we will design distributed algorithms that efficiently compute a (γ, ℓ, L) decomposition, for certain choices of parameters. In Section 6.2.2, we use our (γ, ℓ, L) decomposition algorithms to prove the upper bound part of Theorem 6.1.

6.2.1 Algorithms for rake-and-compress decomposition

Given a rooted tree $G = (V, E)$, we define the two operations rake and compress as follows, where the operation compress depends on a parameter ℓ, which is a positive integer.

- The operation rake: Remove all nodes $v \in V$ with $\deg_{\mathsf{in}}(v) = 0$.
- The operation compress: Remove all nodes $v \in V$ such that v belongs to an ℓ-node directed path $P = v_1 \leftarrow v_2 \leftarrow \cdots \leftarrow v_\ell$ such that $\deg_{\mathsf{in}}(v_i) = \deg_{\mathsf{out}}(v_i) = 1$ for each $1 \le i \le s$.

Intuitively, the rake operation removes the set of all leaf nodes, and the compress operation removes the set of all nodes that belong to an s-node directed path consisting of only degree-2 nodes.

The decomposition algorithm Recall that our goal is to find a (γ, ℓ, L) decomposition of a rooted tree $G = (V, E)$, which is a partition $V = V_1^{\mathsf{R}} \cup V_1^{\mathsf{C}} \cup V_2^{\mathsf{R}} \cup V_2^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$ meeting all the requirements. We will first describe our algorithm, and then we analyze for which combinations of parameters (γ, ℓ, L) our algorithm works.

Our algorithm for finding such a decomposition is as follows. For $i = 1, 2, \ldots$, perform γ rake operations and then perform one compress operation. We initially set V_i^{R} to be the set of nodes removed during a rake operation in the ith iteration. Similarly, we initially set V_i^{C} to be the set of nodes removed during the compress operation in the ith iteration.

It is clear that these sets V_i^{R} and V_i^{C} already satisfy all the specified requirements, except that a connected component of the subgraph induced by V_i^{C} may be a path whose number of nodes exceeds 2ℓ. Similar to existing rake-and-compress decomposition algorithms [8, 19, 21], to fix this, we will do a post-processing step which promotes some nodes from V_i^{C} to V_i^{R} to break long paths of V_i^{C} into small paths, for each i. We need the following definition.

Definition 6.5 (Ruling set)**.** *Let P be a path. A subset $I \subset V(P)$ is called an (α, β)-independent set if the following conditions are met: (i) I is an independent set that does not contain either*

endpoint of P, and (ii) each connected component of the subgraph induced by $V(P) \setminus I$ has at least α node and at most β node, unless $|V(P)| < \alpha$, in which case $I = \emptyset$.

It is a well-known [2, 21, 24] that an $(\ell, 2\ell)$-independent set of a path P can be computed in $O(\log^* n)$ rounds deterministically in the CONGEST model when $\ell = O(1)$. In our post-processing step, we simply compute an $(\ell, 2\ell)$-independent set I in each connected component induced by V_i^{C}, in parallel for all i. Then we promote the nodes in I from layer V_i^{C} to layer V_i^{R}. After this promotion, it is clear that the decomposition

$$V = V_1^{\mathsf{R}} \cup V_1^{\mathsf{C}} \cup V_2^{\mathsf{R}} \cup V_2^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$$

is a (γ, ℓ, L) decomposition meeting all the requirements, where L can be any number such that no node remains after the γ rake operations in the Lth iteration.

Assuming that $\ell = O(1)$, the round complexity of computing the decomposition is clearly $O(\gamma L) + O(\log^* n)$ in the CONGEST model, where $O(\gamma L)$ is the round complexity for executing L iterations of the rake-and-compress process, and $O(\log^* n)$ is the cost for the post-processing step.

Number of layers Next, we consider the following question. Given two positive integers γ and ℓ, what is the smallest number L such that no node remains after the γ rake operations in the Lth iteration, for any given input n-node rooted tree $G = (V, E)$?

To answer this question, we consider the following notation. For each node $v \in V$, we write $U_{\mathsf{R},i}^v$ to denote the set of nodes in the subtree rooted at v right after we finish the γ rake operation in the ith iteration. Similarly, we write $U_{\mathsf{C},i}^v$ to denote the set of nodes in the subtree rooted at v right after we finish the compress operation in the ith iteration.

In these definitions, the notion of subtree is with respect to the subgraph induced by the nodes that are not yet removed, not with respect to the original rooted tree. In particular, if v is already removed before the ith iteration, then we have $U_{\mathsf{R},i}^v = U_{\mathsf{C},i}^v = \emptyset$. For notational simplicity, we write $U_{\mathsf{C},0}^v$ to denote the set of nodes in the subtree rooted at v in the original rooted tree G.

Lemma 6.4 (Shrinkage rate)**.** *For each $v \in V$ and $i \geq 1$, we have $|U_{\mathsf{C},i}^v| < |U_{\mathsf{C},i-1}^v| \cdot \frac{2\ell}{\gamma+2\ell}$.*

Proof. We consider the ith iteration of the rake-and-compress process and focus on the set of nodes $U_{\mathsf{R},i}^v$. We partition $U_{\mathsf{R},i}^v = A \cup B \cup C \cup D$ into four parts as follows.

- A is the set of nodes u in $U_{\mathsf{R},i}^v$ such that u belongs to an ℓ-node directed path in $U_{\mathsf{R},i}^v$ consisting of only nodes whose indegree in $U_{\mathsf{R},i}^v$ is exactly one.
- B is the set of nodes u in $U_{\mathsf{R},i}^v$ such that the indegree of u in $U_{\mathsf{R},i}^v$ is exactly one and $u \notin A$.
- C the set of nodes u in $U_{\mathsf{R},i}^v$ whose indegree in $U_{\mathsf{R},i}^v$ is greater than one.
- D the set of nodes u in $U_{\mathsf{R},i}^v$ whose indegree in $U_{\mathsf{R},i}^v$ is zero.

We prove the following inequalities.

- We have $|A| \leq |U_{\mathsf{R},i}^v| - |U_{\mathsf{C},i}^v|$, since A is precisely the set of nodes in $U_{\mathsf{R},i}^v$ that will subsequently be removed during the compress operation in the ith iteration. The reason that we have an inequality rather than an equality is that all the descendants of A in $U_{\mathsf{R},i}^v$ are also not included in $U_{\mathsf{C},i}^v$.
- We have $|C| + 1 \leq |D|$, since the number of leaf nodes in a rooted tree is at least one plus the number of nodes with more than one child.
- We have $|B| \leq (\ell - 1)(|C| + |D|)$, since the number of connected components induced by indegree-1 nodes in a rooted tree is at most the number of nodes whose indegree is not one, and B is the union of all these connected components of size at most $\ell - 1$.

- We have $\gamma|D| \le |U^v_{\mathsf{C},i-1}| - |U^v_{\mathsf{R},i}|$, since the fact that each leaf node of $U^v_{\mathsf{R},i}$ is not removed during the γ rake operations in the ith iteration implies that it has at least γ descendants removed during these γ rake operations. That is, the number $|U^v_{\mathsf{C},i-1}|-|U^v_{\mathsf{R},i}|$ of nodes in $U^v_{\mathsf{C},i-1}$ removed during the γ rake operations in the ith iteration is at least γ times the number $|D|$ of leaf nodes of $U^v_{\mathsf{R},i}$.

Combining these four inequalities, we have

$$\begin{aligned}
|U^v_{\mathsf{C},i}| &\le |U^v_{\mathsf{R},i}| - |A| \\
&= |B| + |C| + |D| \\
&\le \ell(|C| + |D|) \\
&< 2\ell|D| \\
&\le \frac{2\ell}{\gamma}(|U^v_{\mathsf{C},i-1}| - |U^v_{\mathsf{R},i}|) \\
&\le \frac{2\ell}{\gamma}(|U^v_{\mathsf{C},i-1}| - |U^v_{\mathsf{C},i}|).
\end{aligned}$$

Hence $|U^v_{\mathsf{C},i}| < |U^v_{\mathsf{C},i-1}| \cdot \frac{2\ell}{\gamma+2\ell}$. □

Lemma 6.5 (Number of layers)**.** *If the inequality* $n \cdot \left(\frac{2\ell}{\gamma+2\ell}\right)^{L-1} \le \gamma$ *holds, then we have* $V = V^{\mathsf{R}}_1 \cup V^{\mathsf{C}}_1 \cup V^{\mathsf{R}}_2 \cup V^{\mathsf{C}}_2 \cup \cdots \cup V^{\mathsf{R}}_L$*. In particular, we have the following.*

- *If* $\ell = O(1)$ *and* $\gamma = 1$*, then we may set* $L = O(\log n)$ *to satisfy the inequality.*
- *If* $\ell = O(1)$ *and* $L = k$*, then we may set* $\gamma = O(n^{1/k})$ *to satisfy the inequality.*

Proof. For any $v \in V$, we have $|U^v_{\mathsf{C},0}| \le n$, as $U^v_{\mathsf{C},0}$ is the set of nodes in the subtree rooted at v in the original rooted tree G. By Lemma 6.4, we have $|U^v_{\mathsf{C},L-1}| < n \cdot \left(\frac{2\ell}{\gamma+2\ell}\right)^{L-1} \le \gamma$, which implies that v must be removed during the γ rake operations in the Lth iteration, if v has not been removed by the time the Lth begins. Hence $V = V^{\mathsf{R}}_1 \cup V^{\mathsf{C}}_1 \cup V^{\mathsf{R}}_2 \cup V^{\mathsf{C}}_2 \cup \cdots \cup V^{\mathsf{R}}_L$. □

We are ready to prove the main results of Section 6.2.1.

Lemma 6.6 ($O(\log n)$-round rake-and-compress algorithm)**.** *For any positive integer* $\ell = O(1)$*, a* (γ, ℓ, L) *decomposition of an* n*-node rooted tree with* $\gamma = 1$ *and* $L = O(\log n)$ *can be computed in* $O(\log n)$ *rounds in the* CONGEST *model.*

Proof. By Lemma 6.5, we may set $\gamma = 1$ and $L = O(\log n)$ in such a way that we always have $V = V^{\mathsf{R}}_1 \cup V^{\mathsf{C}}_1 \cup V^{\mathsf{R}}_2 \cup V^{\mathsf{C}}_2 \cup \cdots \cup V^{\mathsf{R}}_L$. The round complexity for computing the decomposition is $O(\gamma L) + O(\log^* n) = O(\log n)$. □

Lemma 6.7 ($O(n^{1/k})$-round rake-and-compress algorithm)**.** *For any positive integers* $\ell = O(1)$ *and* $k = O(1)$*, a* (γ, ℓ, L) *decomposition of an* n*-node rooted tree with* $\gamma = O(n^{1/k})$ *and* $L = k$ *can be computed in* $O(\log n)$ *rounds in the* CONGEST *model.*

Proof. By Lemma 6.5, we may set $\gamma = O(n^{1/k})$ and $L = k$ in such a way that we always have $V = V^{\mathsf{R}}_1 \cup V^{\mathsf{C}}_1 \cup V^{\mathsf{R}}_2 \cup V^{\mathsf{C}}_2 \cup \cdots \cup V^{\mathsf{R}}_L$. The round complexity for computing the decomposition is $O(\gamma L) + O(\log^* n) = O(n^{1/k})$. □

6.2.2 Distributed algorithms via rake-and-compress decompositions

In this section, we use Lemmas 6.6 and 6.7 to design distributed algorithms solving a given LCL problem $\Pi = (\delta, \Sigma, \mathcal{C})$ on δ-regular rooted trees.

Lemma 6.8 (Solving Π using rake-and-compress decompositions)**.** *Suppose we are given an* LCL *problem $\Pi = (\delta, \Sigma, \mathcal{C})$ that admits a good sequence*

$$(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \dots, \Sigma_k^{\mathsf{R}}).$$

Suppose we are given a (γ, ℓ, L) decomposition of an n-node rooted tree $G = (V, E)$ of maximum indegree at most δ

$$V = V_1^{\mathsf{R}} \cup V_1^{\mathsf{C}} \cup V_2^{\mathsf{R}} \cup V_2^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$$

with $L = k$ and $\ell = \max\{1, \mathsf{flexibility}(\Sigma_1^{\mathsf{C}}), \dots, \mathsf{flexibility}(\Sigma_{k-1}^{\mathsf{C}})\}$. Then a correct solution of Π on G can be computed in $O((\gamma + \ell)L)$ rounds in the CONGEST *model.*

Proof. We present an $O((\gamma+\ell)L)$-round CONGEST algorithm finding a correct solution of Π on G. In this proof, whenever we say the algorithm labels a node v, we mean picking a node configuration $(\sigma \ : \ a_1, a_2, \dots, a_\delta) \in \mathcal{C}$ and fixing the labels of v and its δ children according to the chosen node configuration.

The algorithm processes the nodes of the graph in the order $V_L^{\mathsf{R}}, V_{L-1}^{\mathsf{C}}, \dots, V_1^{\mathsf{R}}$. We require that when the algorithm processes V_i^{R}, the algorithm only uses node configurations $(\sigma \ : \ a_1, a_2, \dots, a_\delta) \in \mathcal{C}$ such that σ and all of $a_1, a_2, \dots, a_\delta$ are in Σ_i^{R}. We also require that when the algorithm processes V_i^{C}, the algorithm uses node configurations $(\sigma \ : \ a_1, a_2, \dots, a_\delta) \in \mathcal{C}$ such that $\sigma \in \Sigma_i^{\mathsf{C}}$ and all of $a_1, a_2, \dots, a_\delta$ are in Σ_i^{R}.

Labeling V_i^{R} Suppose the algorithm has finished labeling the nodes in $V_i^{\mathsf{C}} \cup V_{i+1}^{\mathsf{R}} \cup \cdots \cup V_L^{\mathsf{R}}$. The algorithm then labels each connected component S of the subgraph of G induced by V_i^{R}, in parallel and using $O(\gamma)$ rounds in the CONGEST model, as follows.

Recall that The set S induces a rooted tree of height at most $\gamma - 1$ such that no node in S has an in-neighbor in $V_i^{\mathsf{C}} \cup V_{i+1}^{\mathsf{R}} \cup \cdots \cup V_L^{\mathsf{R}}$. The root z of S may have an out-neighbor u.

If $u \notin V_i^{\mathsf{C}} \cup V_{i+1}^{\mathsf{R}} \cup \cdots \cup V_L^{\mathsf{R}}$ of u does not exist, then the label of z is not fixed yet. In this case, we choose any node configuration $(\sigma \ : \ a_1, a_2, \dots, a_\delta) \in \mathcal{C}$ such that σ and all of $a_1, a_2, \dots, a_\delta$ are in Σ_i^{R} to label z and its children. Such a node configuration exists because of Lemma 6.1, as we recall that $\Sigma_i^{\mathsf{R}} = \mathsf{trim}(\Sigma_{i-1}^{\mathsf{C}})$ (if $i > 1$) and $\Sigma_1^{\mathsf{R}} = \mathsf{trim}(\Sigma)$ (if $i = 1$).

If $u \in V_i^{\mathsf{C}} \cup V_{i+1}^{\mathsf{R}} \cup \cdots \cup V_L^{\mathsf{R}}$, then the label of u is fixed to be some label $\sigma \in \Sigma_i^{\mathsf{R}}$, due to the above requirement of our algorithm for labeling $V_i^{\mathsf{C}} \cup V_{i+1}^{\mathsf{R}} \cup \cdots \cup V_L^{\mathsf{R}}$, as we recall that $\Sigma_i^{\mathsf{R}} \supseteq \Sigma_i^{\mathsf{C}} \supseteq \cdots \supseteq \Sigma_k^{\mathsf{R}}$. In this case, we choose any node configuration $(\sigma \ : \ a_1, a_2, \dots, a_\delta) \in \mathcal{C}$ such that all of $a_1, a_2, \dots, a_\delta$ are in Σ_i^{R} to label z and its children. Similarly, the existence of such a node configuration is due to Lemma 6.1 and the fact that $\sigma \in \Sigma_i^{\mathsf{R}}$.

The node configuration for the remaining nodes in S can be fixed similarly. We start processing a node $v \in S$ once the node configuration of its parent u is fixed. Our requirement for labeling V_i^{R} ensures that the label of v is fixed to be some $\sigma \in \Sigma_i^{\mathsf{R}}$, so we can choose any node configuration $(\sigma \ : \ a_1, a_2, \dots, a_\delta) \in \mathcal{C}$ where all of $a_1, a_2, \dots, a_\delta$ are in Σ_i^{R} and use this node configuration for v to label its children. The round complexity of labeling S is $O(\gamma)$ because S is a rooted tree of depth at most $\gamma - 1$.

Labeling V_i^{C} Suppose the algorithm has finished labeling the nodes in $V_{i+1}^{\mathsf{R}} \cup V_{i+1}^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$. The algorithm then labels each connected component S of the subgraph of G induced by V_i^{C}, in parallel and using $O(\ell)$ rounds in the CONGEST model, as follows.

The set S has the property that there are two nodes u and v in $V_{i+1}^{\mathsf{R}} \cup V_{i+1}^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$ adjacent to S such that the subgraph induced by $S \cup \{u, v\}$ is a directed path $u \leftarrow v_1 \leftarrow v_2 \leftarrow \cdots, v_s \leftarrow v$, with $s \in [\ell, 2\ell]$.

Similarly, the requirement of the choice of node configurations for $V_{i+1}^{\mathsf{R}} \cup V_{i+1}^{\mathsf{C}} \cup \cdots \cup V_L^{\mathsf{R}}$ ensures that the labels of u, s_1, and v have been fixed to be some labels in Σ_i^{C}, as we recall that $\Sigma_i^{\mathsf{C}} \supseteq \Sigma_{i+1}^{\mathsf{R}} \supseteq \cdots \supseteq \Sigma_k^{\mathsf{R}}$.

Now, our task is to assign node configurations to $v_1, v_2, \ldots, v_s$ in such a way that the labels used to label $v_1, v_2, \ldots, v_s$ are in Σ_i^{C} and the labels used to label their children are in Σ_i^{R}.

To find such a labeling, we use Lemma 6.3. Specifically, recall that the length of the path $v_1 \leftarrow v_2 \leftarrow \cdots, v_s \leftarrow v$ is $s \geq \ell \geq \mathsf{flexibility}(\Sigma_i^{\mathsf{C}})$ by our choice of ℓ. We let α be the existing label of v_1 and let β be the existing label of v. Recall that $\Sigma_i^{\mathsf{C}} \in \mathsf{flexible\text{-}SCC}(\Sigma_i^{\mathsf{R}})$, so we may apply Lemma 6.3 with $U = \Sigma_i^{\mathsf{C}}$, $\tilde{\Sigma} = \Sigma_i^{\mathsf{R}}$, $d = s$, and our choices of $\alpha \in U$ and $\beta \in U$.

Lemma 6.3 returns a sequence of labels $\alpha = \sigma_1, \sigma_2, \ldots, \sigma_{s+1} = \beta$. For each $1 \leq j \leq s$, we use σ_j to label v_j. Moreover, for each $1 \leq j \leq s$, Lemma 6.3 guarantees that there is a node configuration $(\sigma_j \ : \ \sigma_{j,1}\sigma_{j,2}\cdots\sigma_{j,\delta}) \in \mathcal{C}$ such that all of $\sigma_{j,1}\sigma_{j,2}\cdots\sigma_{j,\delta}$ are in $\tilde{\Sigma} = \Sigma_i^{\mathsf{R}}$ and there exists an index l such that $\sigma_{j,l} = \sigma_{j+1}$. Therefore, we may use the labels in this size-$(\delta-1)$ multiset $\{\sigma_{j,1}\sigma_{j,2}\ldots\sigma_{j,\delta}\} \setminus \{\sigma_{j,l}\}$ to label the remaining $\delta - 1$ children of v_j, so that the node configuration of v_j is $(\sigma_j \ : \ \sigma_{j,1}\sigma_{j,2}\cdots\sigma_{j,\delta}) \in \mathcal{C}$. The round complexity of labeling S is $O(\ell)$ because S is a path of at most 2ℓ nodes.

Summary The number rounds spent on labeling each part V_i^{R} is $O(\gamma)$, and the number rounds spent on labeling each part V_i^{C} is $O(\ell)$, so the overall round complexity for solving Π given a (γ, ℓ, L) decomposition is $O((\gamma + \ell)L)$ rounds in the CONGEST model. □

Combining Lemma 5.4 with existing algorithms for computing (γ, ℓ, L) decompositions, we obtain the following results.

Lemma 6.9 (Upper bound for the case $d_\Pi = k$)**.** *If $d_\Pi = k$ for some positive integer k, then Π can be solved in $O(n^{1/k})$ rounds in the* CONGEST *model.*

Proof. In this case, a good sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_k^{\mathsf{R}})$ exists. By Lemma 6.7, a (γ, ℓ, L) decomposition with $\gamma = O(n^{1/k})$ and $L = k$ can be computed in $O(n^{1/k})$ rounds, Π can be solved in $O(n^{1/k}) + O((\gamma + \ell)L) = O(n^{1/k})$ rounds using the algorithm of Lemma 6.8. Here both k and ℓ are $O(1)$, as they are independent of the number of nodes n. □

Lemma 6.10 (Upper bound for the case $d_\Pi = \infty$)**.** *If $d_\Pi = \infty$, then Π can be solved in $O(\log n)$ rounds in the* CONGEST *model.*

Proof. In this case, a good sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_k^{\mathsf{R}})$ exists for all positive integers k. By Lemma 6.6, a (γ, ℓ, L) decomposition with $\gamma = 1$ and $L = O(\log n)$ can be computed in $O(\log n)$ rounds. By choosing a good sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_k^{\mathsf{R}})$ with $k = L$, Π can be solved in $O(\log n) + O((\gamma + \ell)L) = O(\log n)$ rounds using the algorithm of Lemma 6.8. Similarly, here $\ell = O(1)$, as it is independent of the number of nodes n. □

6.3 Lower bounds

In this section, we prove the lower bound part of Theorem 6.1. Similar to the case of unrooted trees, in our lower bound proofs, we pick γ to be the smallest integer satisfying the following requirements. For each subset $\tilde{\Sigma} \subseteq \Sigma$ and each $\sigma \in \tilde{\Sigma} \setminus \mathsf{trim}(\tilde{\Sigma})$, there exists no correct labeling of T_γ where the label of the root r is σ and the label of remaining nodes is in $\tilde{\Sigma}$. Such a number γ exists due to the definition of trim. Recall that T_γ is defined in Definition 6.3.

Lemma 6.11 (Unsolvability for the case $d_\Pi = 0$)**.** *If $d_\Pi = 0$, then Π is unsolvable in the sense that there exists a rooted tree G of maximum indegree δ such that there is no correct solution of Π on G.*

Proof. We take $G = T_\gamma$. Since $d_\Pi = 0$, we have $\mathsf{trim}(\Sigma) = \emptyset$. Our choice of γ implies that there is no correct solution of Π on $G = T_\gamma$. □

For the rest of this section, we focus on the case $d_\Pi = k$ is a positive integer. We will prove that Π needs $\Omega(n^{1/k})$ rounds to solve in the LOCAL model. Definition 6.6 is similar to Definition 5.6. The exact choice of $s = \Theta(t)$ in Definition 6.6 is to be determined.

Definition 6.6 (Lower bound graphs)**.** *We let t be any positive integer, and we let $s = \Theta(t)$.*

- *Define $G_{\mathsf{R},1}$ as the rooted tree T_γ. All nodes in $G_{\mathsf{R},1}$ are said to be in layer $(\mathsf{R}, 1)$.*
- *For each integer $i \geq 1$, define $G_{\mathsf{C},i}$ as the result of the following construction. Start with an s-node directed path $v_1 \leftarrow v_2 \leftarrow \cdots \leftarrow v_s$. For each $1 \leq i \leq s-1$, append $\delta - 1$ copies of $G_{\mathsf{R},i-1}$ to v_i. Append δ copies of $G_{\mathsf{R},i-1}$ to v_s. The nodes $v_1, v_2, \ldots, v_s$ are said to be in layer (C, i). We call $v_1 \leftarrow v_2 \leftarrow \cdots \leftarrow v_s$ the core path of $G_{\mathsf{C},i}$.*
- *For each integer $i \geq 1$, define $G^\circ_{\mathsf{C},i}$ as the result of the construction of $G_{\mathsf{C},i}$, with a difference that we append $\delta - 1$ copies of $G_{\mathsf{R},i-1}$ to v_s.*
- *For each integer $i \geq 2$, define $G_{\mathsf{R},i}$ as follows. Start with a rooted tree T_γ. Append δ copies of $G_{\mathsf{C},i-1}$ to each leaf in T_γ. All nodes in T_γ are said to be in layer (R, i).*

The only difference between Definition 6.6 and Definition 5.6 is that here we define a new rooted tree $G^\circ_{\mathsf{C},i}$ where we append only $\delta - 1$ copies of $G_{\mathsf{R},i-1}$ to the last node v_s of the core path, so $\deg_{\mathsf{in}}(v_s) = \delta - 1$. The purpose of this modification is to allow us to concatenate the rooted trees together without violating the maximum indegree bound δ. Specifically, for our lower bound proof in Section 6.3, we define our main lower bound graph $G = (V, E)$ as follows.

Definition 6.7 (Main lower bound graph)**.** *Define the rooted tree $G = (V, E)$ as the result of the following construction.*

- *The construction starts with the rooted trees $G^\circ_{\mathsf{C},1}, G^\circ_{\mathsf{C},2}, \ldots, G^\circ_{\mathsf{C},k}$ and $G_{\mathsf{R},k+1}$.*
- *Let $P_i = v_1^i \leftarrow v_2^i \leftarrow \cdots \leftarrow v_s^i$ be the core path of $G^\circ_{\mathsf{C},i}$ and let r be the root of $G_{\mathsf{R},k+1}$.*
- *Add the directed edges $v_s^1 \leftarrow v_1^2, v_s^2 \leftarrow v_1^3, \ldots, v_s^{k-1} \leftarrow v_1^k$, and $v_s^k \leftarrow r$.*

It is clear that all nodes in the rooted tree G have indegree either 0 or δ, and the total number of nodes in G is $O(t^k)$, if we treat γ as a constant independent of t. Therefore, to show that Π requires $\Omega(n^{1/k})$ rounds to solve, it suffices to show that there is no algorithm that solves Π within t rounds on G.

Similar to Section 5.3, the nodes in G are partitioned into layers $(\mathsf{R}, 1)$, $(\mathsf{C}, 1)$, $(\mathsf{R}, 2)$, $(\mathsf{C}, 2)$, …, $(\mathsf{R}, k+1)$ according to the rules in the above recursive construction, and we order the layers by $(\mathsf{R}, 1) \prec (\mathsf{C}, 1) \prec (\mathsf{R}, 2) \prec (\mathsf{C}, 2) \prec \cdots \prec (\mathsf{R}, k+1)$. Recall that in the graph G, the nodes in

layer (C, i) form directed paths of s nodes. We consider the following classification of nodes in layer (C, i). Again, we will choose $s = \Theta(t)$ to be sufficiently large to ensure that central nodes exist.

Definition 6.8 (Classification of nodes in layer (C, i))**.** *The nodes in the s-node directed path $v_1 \leftarrow v_2 \leftarrow \cdots \leftarrow v_s$ in the construction of $G_{\mathsf{C},i}$ and $G^{\circ}_{\mathsf{C},i}$ are classified as follows.*

- *We say that v_j is a front node if $1 \le j \le t$.*
- *We say that v_j is a central node if $t+1 \le j \le s-t$.*
- *We say that v_j is a rear node if $s-t+1 \le j \le s$.*

Based on Definition 6.8, we define the following subsets of nodes in G. In Definition 6.9, recall that P_i is the core path of the rooted tree $G^{\circ}_{\mathsf{C},i}$ in the construction of G in Definition 6.7.

Definition 6.9 (Subsets of nodes in G)**.** *We define the following subsets of nodes in G.*

- *Define $S_{\mathsf{R},1}$ as the set of nodes v in G such that the subgraph induced by v and its descendants within radius-γ neighborhood of v is isomorphic to T_γ.*
- *For $2 \le i \le k+1$, define $S_{\mathsf{R},i}$ as the set of nodes v in G such that the subgraph induced by v and its descendants within radius-γ neighborhood of v is isomorphic to T_γ and contains only nodes in $S_{\mathsf{C},i-1}$.*
- *For $1 \le i \le k$, define $S_{\mathsf{C},i}$ as the set of nodes v in G meeting one of the following conditions.*
  - *v is in layer $(\mathsf{R}, i+1)$ or above.*
  - *$v \in P_i$ is a central or rear node in layer (C, i).*
  - *$v \notin P_i$ is a central or front node in layer (C, i).*

We prove some basic properties of the sets in Definition 6.9.

Lemma 6.12 (Subset containment)**.** *We have $S_{\mathsf{R},1} \supseteq S_{\mathsf{C},1} \supseteq \cdots \supseteq S_{\mathsf{R},k+1} \neq \emptyset$.*

Proof. The claim that $S_{\mathsf{C},i} \supseteq S_{\mathsf{R},i+1}$ follows from the definition of $S_{\mathsf{R},i+1}$ that $v \in S_{\mathsf{C},i}$ is a necessary condition for $v \in S_{\mathsf{R},i+1}$. To prove claim that $S_{\mathsf{R},i} \supseteq S_{\mathsf{C},i}$, we recall that $v \in S_{\mathsf{C},i}$ implies that v is in layer (C, i) or above. By the construction of G, the subgraph induced by v and its descendants within the radius-γ neighborhood of v is isomorphic to T_γ and contains only nodes in layer (R, i) or above. Since all nodes in layer (R, i) or above are in $S_{\mathsf{C},i-1}$. we infer that $v \in S_{\mathsf{C},i}$ implies $v \in S_{\mathsf{R},i}$.

To see that $S_{\mathsf{R},k+1} \neq \emptyset$, consider the node r in the construction of G. Since r is the root of $G_{\mathsf{R},k+1}$, the subgraph induced by r and its descendants within radius-γ neighborhood of r is isomorphic to T_γ and contains only nodes in layer $(\mathsf{R}, k+1)$. We know that all nodes in layer $(\mathsf{R}, k+1)$ are in $S_{\mathsf{C},k}$, so $r \in S_{\mathsf{R},k+1}$. □

Lemma 6.13 (Property of $S_{\mathsf{C},i}$)**.** *For each node $v \in S_{\mathsf{C},i}$, either one of the following holds.*

- *v is a central node in layer (C, i).*
- *For each child u of v such that $u \in S_{\mathsf{C},i}$, there exists a directed path $P = w_1 \leftarrow \cdots \leftarrow v \leftarrow u \leftarrow \cdots, w_2$ such that $w_1 \in P_i$ and $w_2 \notin P_i$ are central nodes in layer (C, i) and all nodes in P are in $S_{\mathsf{C},i}$.*

Proof. We assume that $v \in S_{\mathsf{C},i}$ is not a central node in layer (C, i). Consider any child u of v such that $u \in S_{\mathsf{C},i}$. The goal of the proof is to find a path $P = w_1 \leftarrow \cdots \leftarrow v \leftarrow u \leftarrow \cdots \leftarrow w_2$ such that all nodes of P are in $S_{\mathsf{C},i}$, and $w_1 \in P_i$ and $w_2 \notin P_i$ are central nodes in layer (C, i). The existence of such a directed path P follows from a simple observation that $S_{\mathsf{C},i}$ induces a connected subtree where all the leaf nodes are central nodes in layer (C, i) that are not in P_i and the root node is a central node in layer (C, i) that is in P_i.

Specifically, the directed path P can be constructed as follows. To construct the part $w_1 \leftarrow \cdots \leftarrow v$, we simply start from v and follow the parent pointers until we reach a node w_1 that is a central node in P_i. The correctness of the construction of this part follows from the definition of G and the fact that either v is a rear node in P_i or $v \notin P_i$ is in layer (C, i) or above.

To construct the remaining part $v \leftarrow u \leftarrow \cdots \leftarrow w_2$, we simply observe that either u itself is a central node in layer (C, i) or there is a descendant w_2 of u such that w_2 is a central node in layer (C, i). Hence we can always extend $v \leftarrow u$ to a desired path $v \leftarrow u \leftarrow \cdots \leftarrow w_2$. □

We note that the reason for attaching the rooted trees $G^{\circ}_{\mathsf{C},1}, G^{\circ}_{\mathsf{C},2}, \ldots, G^{\circ}_{\mathsf{C},k}$ to $G_{\mathsf{R},k+1}$ in the definition of G is precisely that we want to define $S_{\mathsf{C},i}$ in such a way that allows us to have Lemma 6.13. That is, the design objective is to ensure that for each node $v \in S_{\mathsf{C},i}$ that is not a central node in layer (C, i), there is a directed path in $S_{\mathsf{C},i}$ passing through v and starting and ending at central nodes in layer (C, i).

Assumptions We are given an LCL problem $\Pi = (\delta, \Sigma, \mathcal{C})$ such that $d_\Pi = k$. Hence there does not exist a good sequence

$$(\Sigma^{\mathsf{R}}_1, \Sigma^{\mathsf{C}}_1, \Sigma^{\mathsf{R}}_2, \Sigma^{\mathsf{C}}_2, \ldots, \Sigma^{\mathsf{R}}_k).$$

Recall that the rules for a good sequence are as follows:

$$\Sigma^{\mathsf{R}}_i = \begin{cases} \mathsf{trim}(\Sigma) & \text{if } i = 1, \\ \mathsf{trim}(\Sigma^{\mathsf{C}}_{i-1}) & \text{if } i > 1, \end{cases}$$
$$\Sigma^{\mathsf{C}}_i \in \mathsf{flexible\text{-}SCC}(\Sigma^{\mathsf{R}}_i).$$

The only nondeterminism in the above rules is the choice of $\Sigma^{\mathsf{C}}_i \in \mathsf{flexible\text{-}SCC}(\Sigma^{\mathsf{R}}_i)$ for each i. The fact that $d_\Pi = k$ implies that for all possible choices of $\Sigma^{\mathsf{C}}_1, \Sigma^{\mathsf{C}}_2, \ldots, \Sigma^{\mathsf{C}}_k$, we always end up with $\Sigma^{\mathsf{R}}_{k+1} = \emptyset$.

We assume that there is an algorithm $\mathcal{A}$ that solves Π in $t = O(n^{1/k})$ rounds on $G = G^*_{\mathsf{R},k+1}$, where n is the number of nodes in G. To prove the desired $\Omega(n^{1/k})$ lower bound, it suffices to derive a contradiction. Specifically, we will prove that the existence of such an algorithm $\mathcal{A}$ forces the existence of a good sequence $(\Sigma^{\mathsf{R}}_1, \Sigma^{\mathsf{C}}_1, \Sigma^{\mathsf{R}}_2, \Sigma^{\mathsf{C}}_2, \ldots, \Sigma^{\mathsf{R}}_k)$, contradicting the fact that $d_\Pi = k$.

Induction hypothesis Our proof proceeds by an induction on the subsets $S_{\mathsf{R},1}$, $S_{\mathsf{C},1}$, $S_{\mathsf{R},2}$, $S_{\mathsf{C},2}$, …, $S_{\mathsf{R},k+1}$. For each $1 \le i \le k$, the choice of $\Sigma^{\mathsf{C}}_i \in \mathsf{flexible\text{-}SCC}(\Sigma^{\mathsf{R}}_i)$ is fixed in the induction hypothesis for $S_{\mathsf{C},i}$. The choice of Σ^{R}_i is uniquely determined once $\Sigma^{\mathsf{C}}_1, \Sigma^{\mathsf{C}}_2, \ldots, \Sigma^{\mathsf{C}}_{i-1}$ have been fixed.

Similar to Section 5.3, before defining our induction hypothesis, we recall that the output label of a node v is determined by the subgraph induced by the radius-t neighborhood U of v, together with the distinct IDs of the nodes in U. For each node v in G, we define $\Sigma^{\mathcal{A}}_v$ as the set of all possible output labels of v that can possibly appear when we run $\mathcal{A}$ on G. In other words, $\sigma \in \Sigma^{\mathcal{A}}_v$ implies that there exists an assignment of distinct IDs to nodes in the radius-t neighborhood of v such that the output label of v is σ.

Definition 6.10 (Induction hypothesis for layer $S_{\mathsf{R},i}$)**.** *For each $1 \le i \le k+1$, the induction hypothesis for $S_{\mathsf{R},i}$ specifies that each $v \in S_{\mathsf{R},i}$ satisfies $\Sigma^{\mathcal{A}}_v \subseteq \Sigma^{\mathsf{R}}_i$.*

Definition 6.11 (Induction hypothesis for layer $S_{\mathsf{C},i}$)**.** *For each $1 \le i \le k$, the induction hypothesis for $S_{\mathsf{C},i}$ specifies that there exists a choice $\Sigma^{\mathsf{C}}_i \in \mathsf{flexible\text{-}SCC}(\Sigma^{\mathsf{R}}_i)$ such that each $v \in S_{\mathsf{C},i}$ satisfies $\Sigma^{\mathcal{A}}_v \subseteq \Sigma^{\mathsf{C}}_i$.*

Next, we prove that the induction hypotheses stated in Definitions 6.10 and 6.11 hold.

Lemma 6.14 (Base case: $S_{\mathsf{R},1}$)**.** *The induction hypothesis for $S_{\mathsf{R},1}$ holds.*

Proof. Recall that the number γ satisfies the following property. For each subset $\tilde{\Sigma} \subseteq \Sigma$ and each $\sigma \in \tilde{\Sigma} \setminus \mathsf{trim}(\tilde{\Sigma})$, there exists no correct labeling of T_γ where the label of the root r is σ and the label of remaining nodes is in $\tilde{\Sigma}$.

To prove the induction hypothesis for $S_{\mathsf{R},1}$, consider any node $v \in S_{\mathsf{R},1}$. By the definition of $S_{\mathsf{R},1}$, the subgraph induced by v and its descendants within radius-γ of v is isomorphic to T_γ rooted at v. By setting $\tilde{\Sigma} = \Sigma$, we infer that there is no correct labeling of G such that the label of v is in $\Sigma \setminus \mathsf{trim}(\Sigma) = \Sigma \setminus \Sigma_1^{\mathsf{R}}$, so we must have $\mathcal{V}_v^{\mathcal{A}} \subseteq \Sigma_1^{\mathsf{R}}$, as $\mathcal{A}$ is correct. □

Lemma 6.15 (Inductive step: $S_{\mathsf{R},i}$)**.** *Let $2 \leq i \leq k$. If the induction hypothesis for $S_{\mathsf{C},i-1}$ holds, then the induction hypothesis for $S_{\mathsf{R},i}$ holds.*

Proof. To prove the induction hypothesis for $S_{\mathsf{R},i}$, consider any node $v \in S_{\mathsf{R},i}$. By the definition of $S_{\mathsf{R},i}$, the subgraph S induced by v and its descendants within radius-γ of v is isomorphic to T_γ rooted at v and contains only nodes in $S_{\mathsf{C},i-1}$. Our goal is to prove that $\mathcal{V}_v^{\mathcal{A}} \subseteq \Sigma_i^{\mathsf{R}} = \mathsf{trim}(\Sigma_{i-1}^{\mathsf{C}})$.

Consider any node u in the subgraph S induced by v and its descendants within radius-γ of v. As $u \in S_{\mathsf{C},i-1}$, the induction hypothesis for $S_{\mathsf{C},i-1}$ implies that

$$\mathcal{V}_u^{\mathcal{A}} \subseteq \Sigma_{i-1}^{\mathsf{C}}.$$

Hence the same argument in the proof of Lemma 6.14 shows that $\mathcal{V}_v^{\mathcal{A}} \subseteq \mathsf{trim}(\Sigma_{i-1}^{\mathsf{C}}) = \Sigma_i^{\mathsf{R}}$, as required. □

Lemma 6.16 (Inductive step: $S_{\mathsf{C},i}$)**.** *Let $1 \leq i \leq k$. If the induction hypothesis for layer $S_{\mathsf{R},i}$ holds, then the induction hypothesis for layer $S_{\mathsf{C},i}$ holds.*

Proof. To prove the induction hypothesis for $S_{\mathsf{C},i}$, we show that there exists a choice $\Sigma_i^{\mathsf{C}} \in \mathsf{flexible\text{-}SCC}(\Sigma_i^{\mathsf{R}})$ such that each $v \in S_{\mathsf{C},i}$ satisfies $\Sigma_v^{\mathcal{A}} \subseteq \Sigma_i^{\mathsf{C}}$.

We first consider the case that v is a central node in layer (C, i). Then it is clear that $\Sigma_v^{\mathcal{A}}$ is the same for all v that is a central node in layer (C, i), as the radius-t neighborhood of these nodes v are isomorphic, due to the definition of central nodes and the construction in Definition 6.6. For notational convenience, we write $\tilde{\Sigma}$ to denote the set $\Sigma_v^{\mathcal{A}}$ for any central node v in layer (C, i).

Plan of the proof Consider any node $v \in S_{\mathsf{C},i}$, we claim that the node configuration $(\sigma : a_1 a_2 \cdots a_\delta)$ of v resulting from running $\mathcal{A}$ uses only labels in Σ_i^{R}. To see this, observe that v and all δ children of v are in $S_{\mathsf{R},i}$, so the induction hypothesis for $S_{\mathsf{R},i}$ implies that their output labels must be in Σ_i^{R}. Hence all labels in $(\sigma : a_1 a_2 \cdots a_\delta)$ are in Σ_i^{R}.

We write $\mathcal{M}$ to denote the directed graph representing the automaton associated with the path-form of the LCL problem $\Pi \restriction_{\Sigma_i^{\mathsf{R}}}$. The above claim implies the following. Consider any directed edge $u \leftarrow v$ such that both u and v are in $S_{\mathsf{C},i}$. Let a be the output label of u and let b be the output label of v. Then $a \leftarrow b$ must be a directed edge in $\mathcal{M}$.

To prove that there exists a choice $\Sigma_i^{\mathsf{C}} \in \mathsf{flexible\text{-}SCC}(\Sigma_i^{\mathsf{R}})$ such that $\Sigma_v^{\mathcal{A}} \subseteq \Sigma_i^{\mathsf{C}}$ for all $v \in S_{\mathsf{C},i}$. We will first show that $\tilde{\Sigma}$ must be a subset of a path-flexible strongly connected component of Σ_i^{R}, and then we fix $\Sigma_i^{\mathsf{C}} \in \mathsf{flexible\text{-}SCC}(\Sigma_i^{\mathsf{R}})$ to be this path-flexible strongly connected component.

Next, we will argue that for each $\sigma \in \Sigma_v^{\mathcal{A}}$, there exist a walk in $\mathcal{M}$ that starts from σ and ends in $\tilde{\Sigma}$ and a walk in $\mathcal{M}$ that starts from $\tilde{\Sigma}$ and ends in σ. This shows that σ is in the same strongly connected component as the members in $\tilde{\Sigma}$, so we conclude that $\Sigma_v^{\mathcal{A}} \subseteq \Sigma_i^{\mathsf{C}}$.

Part 1: $\tilde{\Sigma}$ is a subset of some $\Sigma_i^{\mathsf{C}} \in \mathsf{flexible\text{-}SCC}(\Sigma_i^{\mathsf{R}})$ Consider a path $u_1 \leftarrow u_2 \leftarrow \cdots \leftarrow u_s$ of s nodes in layer (C, i) of G. We choose $s = \Theta(t)$ to be sufficiently large to ensure that for each integer $0 \le d \le |\Sigma|$, there exist two nodes u_j and u_l in the path meeting the following conditions.

- $t+1 \le j < l \le s-t$, so u_j and u_l are central nodes.
- The distance $l-j$ between u_j and u_l equals $4t+3+d$.

Similar to the proof of Lemma 5.12, the choice of the number $4t+3$ is to ensure that the union of the radius-t neighborhood of any node v and the radius-t neighborhood of children of v in G does not node-intersect the radius-t neighborhood of both u_j and u_l. This implies that after arbitrarily fixing distinct IDs in the radius-t neighborhood of u_j and u_l, it is possible to complete the ID assignment of the entire graph G in such a way that the union of the radius-t neighborhood of any node v and the radius-t neighborhood of children of v in G does not contain repeated IDs. If we run $\mathcal{A}$ with such an ID assignment, it is guaranteed that the output is correct.

Consider the directed graph $\mathcal{M}$ and any of its two nodes a and b such that $a \in \tilde{\Sigma}$ and $b \in \tilde{\Sigma}$. Our choice of $\tilde{\Sigma}$ implies that there exists an assignment of distinct IDs to the radius-t neighborhood of both u_j and u_l such that the output labels of u_j and u_l are a and b, respectively. We complete the ID assignment of the entire graph G in such a way that the union of the radius-t neighborhood of any node v and the radius-t neighborhood of children of v in G does not contain repeated IDs.

We write σ_y to denote the output label of u_y resulting from running $\mathcal{A}$. Hence $a = \sigma_j \leftarrow \sigma_{j+1} \leftarrow \cdots \leftarrow \sigma_l = b$ is a walk $b \rightsquigarrow a$ in $\mathcal{M}$ of length $4t+3+d$. Therefore, for all choices of $a \in \tilde{\Sigma}$ and $b \in \tilde{\Sigma}$ and any $0 \le d \le |\Sigma|$, we may find a walk $b \rightsquigarrow a$ in $\mathcal{M}$ of length $4t+3+d$. This implies that all members in $\tilde{\Sigma}$ are in the same strongly connected component of $\mathcal{M}$. Furthermore, Lemma 6.2 implies that this strongly connected component is path-flexible, as the length of the walk can be any integer in between $4t+3$ and $4t+3+|\Sigma|$.

Part 2: $\Sigma_v^{\mathcal{A}} \subseteq \Sigma_i^{\mathsf{C}}$ for each $v \in S_{\mathsf{C},i}$ For this part, we use Lemma 6.13, which shows that for each $v \in S_{\mathsf{C},i}$ in the graph G, there is a directed path $P = w_1 \leftarrow \cdots \leftarrow v \leftarrow \cdots \leftarrow w_2$ such that $w_1 \in P_i$ and $w_2 \notin P_i$ are central nodes in layer (C, i) and all nodes in P are in $S_{\mathsf{C},i}$.

Consider the output labels of the nodes in P resulting from running $\mathcal{A}$. We write σ_v to denote the output label of v. Then $\sigma_{w_1} \leftarrow \cdots \leftarrow \sigma_v$ is a walk in $\mathcal{M}$ from σ_v to a node in $\tilde{\Sigma}$ and $\sigma_v \leftarrow \cdots \leftarrow \sigma_{w_2}$ is a walk in $\mathcal{M}$ from a node in $\tilde{\Sigma}$ to σ_v. This shows that σ_v is in the same strongly connected component of $\mathcal{M}$ as the members in $\tilde{\Sigma}$.

The same argument can be applied to all $\sigma_v \in \Sigma_v^{\mathcal{A}}$. The reason is that for each $\sigma_v \in \Sigma_v^{\mathcal{A}}$ there is an assignment of distinct IDs such that σ_v is the output label of v. Hence we conclude that all members in $\Sigma_v^{\mathcal{A}}$ are in the same strongly connected component of $\mathcal{M}$ as the members in $\tilde{\Sigma}$, so $\Sigma_v^{\mathcal{A}} \subseteq \Sigma_i^{\mathsf{C}}$. □

Applying Lemmas 6.14 to 6.16 from $S_{\mathsf{R},1}$ all the way up to the last subset $S_{\mathsf{R},k+1}$, we obtain the following result.

Lemma 6.17 (Lower bound for the case $d_\Pi = k$)**.** *If $d_\Pi = k$ for a finite integer k, then Π requires $\Omega(n^{1/k})$ rounds to solve on rooted trees of maximum indegree δ.*

Proof. Assume that there is a t-round algorithm solving Π on G. By Lemmas 6.14 to 6.16, we infer that the induction hypothesis for the last subset $S_{\mathsf{R},k+1}$ holds. By Lemma 6.12, $S_{\mathsf{R},k+1} \neq \emptyset$, so there is a node v in G such that $\Sigma_v^{\mathcal{A}} \subseteq \Sigma_{k+1}^{\mathsf{R}}$. Therefore, the correctness of $\mathcal{A}$ implies that $\Sigma_{k+1}^{\mathsf{R}} \neq \emptyset$, which implies that $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_{k+1}^{\mathsf{R}})$ chosen in the induction hypothesis is a good sequence, contradicting the assumption that $d_\Pi = k$. Hence such a t-round algorithm $\mathcal{A}$ that

solves Π does not exist. As t can be any positive integer and $t = \Omega(n^{1/k})$, where n is the number of nodes in G, we conclude the proof. □

Now we are ready to prove Theorem 6.1.

Proof of Theorem 6.1. The upper bound part of the theorem follows from Lemmas 6.9 and 6.10. The lower bound part of the theorem follows from Lemmas 6.11 and 6.17. □

6.4 Complexity of the characterization

In this section, we prove Theorem 6.2. We are given a description of an LCL problem $\Pi = (\delta, \Sigma, \mathcal{C})$ on δ-regular rooted trees. We assume that the description is given in the form of listing all the node configurations in $\mathcal{C}$. Therefore, the description length of Π is $\ell = O(|\mathcal{C}|\delta \log |\Sigma|)$. We allow δ to be a non-constant as a function of ℓ. We will design an algorithm that computes all possible good sequences $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_k^{\mathsf{R}})$ in time polynomial in ℓ.

For the case of $d_\Pi = \infty$, there are good sequences that are arbitrarily long. Recall that we have $\Sigma_1^{\mathsf{R}} \supseteq \Sigma_1^{\mathsf{C}} \supseteq \cdots \supseteq \Sigma_k^{\mathsf{R}}$. Hence if $k > |\Sigma|$, there must exist some index $1 \le i < k$ such that $\Sigma_i^{\mathsf{R}} = \Sigma_i^{\mathsf{C}} = \Sigma_{i+1}^{\mathsf{R}}$. This immediately implies that $\Sigma_i^{\mathsf{R}} = \Sigma_i^{\mathsf{C}} = \Sigma_{i+1}^{\mathsf{R}} = \Sigma_{i+1}^{\mathsf{C}} = \cdots$. The reason is that $\Sigma_i^{\mathsf{R}} = \Sigma_i^{\mathsf{C}}$ implies that Σ_i^{R} itself is the only element of $\mathsf{flexible\text{-}SCC}(\Sigma_i^{\mathsf{R}})$. We conclude that any good sequence with $k > |\Sigma|$ must stabilizes at some point $i \le |\Sigma|$, in the sense that $\Sigma_i^{\mathsf{R}} = \Sigma_i^{\mathsf{C}} = \Sigma_{i+1}^{\mathsf{R}} = \Sigma_{i+1}^{\mathsf{C}} = \cdots$.

High-level plan Recall that the rules for a good sequence are as follows.

$$\Sigma_i^{\mathsf{R}} = \begin{cases} \mathsf{trim}(\Sigma) & \text{if } i = 1, \\ \mathsf{trim}(\Sigma_{i-1}^{\mathsf{C}}) & \text{if } i > 1, \end{cases}$$
$$\Sigma_i^{\mathsf{C}} \in \mathsf{flexible\text{-}SCC}(\Sigma_i^{\mathsf{R}}).$$

To compute all good sequences $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_k^{\mathsf{R}})$, we go through all choices of $\Sigma_i^{\mathsf{C}} \in \mathsf{flexible\text{-}SCC}(\Sigma_i^{\mathsf{R}})$ and apply the rules recursively until we cannot proceed any further. The process stops when $\Sigma_i^{\mathsf{C}} = \Sigma_i^{\mathsf{R}}$ (the sequence stabilizes) or $\Sigma_i^{\mathsf{C}} = \emptyset$ or $\Sigma_i^{\mathsf{R}} = \emptyset$ (the sequence ends).

We start with describing the algorithm for computing $\mathsf{trim}(\tilde{\Sigma})$ for a given $\tilde{\Sigma} \subseteq \Sigma$.

Lemma 6.18 (Algorithm for trim)**.** *The set* $\mathsf{trim}(\tilde{\Sigma})$ *can be computed in* $O(|\mathcal{C}|\delta|\tilde{\Sigma}| + |\tilde{\Sigma}|^2)$ *time, for any given* $\tilde{\Sigma} \subseteq \Sigma$.

Proof. We write Σ_i to denote set of all possible $\sigma \in \tilde{\Sigma}$ such that there is a correct labeling of the rooted tree T_i where the label of each node is in $\tilde{\Sigma}$ and the label of the root r is σ. The set Σ_i can be computed recursively as follows.

- For the base case, Σ_0 is the set of all labels appearing in $\tilde{\Sigma}$.
- For the inductive step, each $\sigma \in \Sigma_{i-1}$ is added to Σ_i if there exists a node configuration $(\sigma : a_1 a_2 \cdots a_\delta) \in \mathcal{C}$ such that $a_j \in \Sigma_{i-1}$ for all $1 \le j \le \delta$.

The above recursive computation implies that given Σ_{i-1} has been computed, the computation of Σ_i costs $O(|\mathcal{C}|\delta + |\tilde{\Sigma}|)$ time. The algorithm simply goes over each $(\sigma : a_1 a_2 \cdots a_\delta) \in \mathcal{C}$ and checks whether $a_j \in \Sigma_{i-1}$ for all $1 \le j \le \delta$. If we store Σ_{i-1} as binary string of length $|\tilde{\Sigma}|$, testing whether $a_j \in \Sigma_{i-1}$ costs $O(1)$ time. Therefore, the process of going through all $(\sigma : a_1 a_2 \cdots a_\delta) \in \mathcal{C}$ costs $O(|\mathcal{C}|\delta)$ time. After that, using $O(|\mathcal{C}| + |\tilde{\Sigma}|)$ time, we may calculate Σ_i and store it as a binary string of length $|\tilde{\Sigma}|$.

Clearly, we have $\Sigma_1 \supseteq \Sigma_2 \supseteq \cdots$. Once $\Sigma_i = \Sigma_{i+1}$, the sequence stabilizes: $\Sigma_i = \Sigma_{i+1} = \Sigma_{i+2} = \cdots$. It is clear that the sequence stabilizes at some $i \le |\tilde{\Sigma}|$. We write Σ^* to denote the fix point Σ_i such that $\Sigma_i = \Sigma_{i+1} = \Sigma_{i+2} = \cdots$. It is clear that $\mathsf{trim}(\tilde{\Sigma}) = \Sigma^*$, and it can be computed in $O(|\mathcal{C}|\delta + |\tilde{\Sigma}|) \cdot |\tilde{\Sigma}| = O(|\mathcal{C}|\delta|\tilde{\Sigma}| + |\tilde{\Sigma}|^2)$ time. □

Next, we give an algorithm that computes all path-flexible strongly connected components $\Sigma' \in \mathsf{flexible\text{-}SCC}(\tilde{\Sigma})$, for any given $\tilde{\Sigma} \subseteq \Sigma$.

Lemma 6.19 (Algorithm for $\mathsf{flexible\text{-}SCC}$)**.** *The set of all $\Sigma' \in \mathsf{flexible\text{-}SCC}(\tilde{\Sigma})$ can be computed in $O(|\tilde{\Sigma}|^3 + |\mathcal{C}|\delta)$ time, for any given $\tilde{\Sigma} \subseteq \Sigma$.*

Proof. Let $\mathcal{M}$ be the directed graph representing the automaton associated with the path-form of the LCL problem $\Pi\restriction_{\tilde{\Sigma}}$. The directed graph $\mathcal{M}$ has $|\tilde{\Sigma}|$ nodes and $O(|\tilde{\Sigma}|^2)$ directed edges, and it can be constructed in $O(|\mathcal{C}|\delta)$ time by going through all node configurations in $\mathcal{C}$. The strongly connected components of $\mathcal{M}$ can be computed in time linear in the number of nodes and edges of $\mathcal{M}$, which is $O(|\tilde{\Sigma}|^2)$.

By to the proof of Lemma 5.15, for each strongly connected component U of $\mathcal{M}$, deciding whether U is path-flexible costs $O(|U|^3)$ time. The summation of the time complexity $O(|U|^3)$, over all strongly connected component U of $\mathcal{M}$, is $O(|\tilde{\Sigma}|^3)$. To summarize, all $\Sigma' \in \mathsf{flexible\text{-}SCC}(\tilde{\Sigma})$ can be computed in $O(|\tilde{\Sigma}|^3 + |\mathcal{C}|\delta)$ time. □

Combining Lemmas 6.18 and 6.19, we obtain the following result.

Lemma 6.20 (Computing all good sequences)**.** *The set of all good sequences can be computed in $O(|\Sigma|^4 + |\mathcal{C}|\delta|\Sigma|^2)$ time, for any given LCL problem $\Pi = (\delta, \Sigma, \mathcal{C})$.*

Proof. By Lemma 6.18, given $\Sigma_i^{\mathsf{C}} \subseteq \Sigma$, the cost of computing $\Sigma_{i+1}^{\mathsf{R}}$ is $O(|\mathcal{C}|\delta|\Sigma_i^{\mathsf{C}}| + |\Sigma_i^{\mathsf{C}}|^2)$ time. Since all sets Σ_i^{C} in the depth i of the recursion are disjoint, the total cost for this step of the recursion is $O(|\mathcal{C}|\delta|\Sigma| + |\Sigma|^2)$.

By Lemma 6.19, given $\Sigma_i^{\mathsf{R}} \subseteq \Sigma$, the cost of computing all possible Σ_i^{C} is $O(|\Sigma_i^{\mathsf{R}}|^3 + |\mathcal{C}|\delta)$ time. Since all sets Σ_i^{R} in the depth i of the recursion are disjoint, the total cost for this step of the recursion is $O(|\Sigma|^3 + |\mathcal{C}|\delta|\Sigma|)$.

The depth of the recursion is at most $|\Sigma|$, so the total cost of computing all good sequences is $O(|\Sigma|^4 + |\mathcal{C}|\delta|\Sigma|^2)$. □

We are ready to prove Theorem 6.2.

Proof of Theorem 6.2. By Lemma 6.20, the set of all good sequences can be computed in polynomial time, and we can compute d_Π given the set of all good sequences. If $d_\Pi = k$ is a positive integer, then from the discussion in Section 6.2 we know how to turn a good sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_k^{\mathsf{R}})$ into a description of an $O(n^{1/k})$-round algorithm for Π. If $d_\Pi = \infty$, then similarly a good sequence $(\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \Sigma_2^{\mathsf{R}}, \Sigma_2^{\mathsf{C}}, \ldots, \Sigma_{O(\log n)}^{\mathsf{R}})$ leads to a description of an $O(\log n)$-round algorithm for Π. □

References

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

[2] 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. arXiv:1811.01672, doi:10.1145/3293611.3331606.

[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)*, volume 179 of *LIPIcs*, pages 17:1–17:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. arXiv:1911.13294, 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. arXiv:1901.02441, doi:10.1109/FOCS.2019.00037.

[5] Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Improved distributed lower bounds for MIS and bounded (out-)degree dominating sets in trees. In *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 283–293. ACM, 2021. doi:10.1145/3465084.3467901.

[6] Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Deterministic δ-coloring plays hide-and-seek. In *Proc. 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)*. ACM, 2022.

[7] Alkida Balliu, Sebastian Brandt, and Dennis Olivetti. Distributed lower bounds for ruling sets. In *Proc. 61st IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 365–376, 2020. doi:10.1109/FOCS46700.2020.00042.

[8] 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.

[9] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. How much does randomness help with locally checkable problems? In *Proc. 39th ACM Symposium on Principles of Distributed Computing (PODC 2020)*, pages 299–308. ACM Press, 2020. arXiv:1902.06803, doi:10.1145/3382734.3405715.

[10] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. Almost global problems in the LOCAL model. *Distributed Computing*, 34:259–281, 2021. doi:10.1007/s00446-020-00375-2.

[11] Alkida Balliu, Keren Censor-Hillel, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Locally checkable labelings with small messages. In *35th International Symposium on Distributed Computing, DISC 2021*, volume 209 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.DISC.2021.8.

[12] 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. arXiv:1711.01871, doi:10.1145/3188745.3188860.

[13] Alkida Balliu, Juho Hirvonen, Dennis Olivetti, and Jukka Suomela. Hardness of minimal symmetry breaking in distributed computing. In *Proc. 38th ACM Symposium on Principles of Distributed Computing (PODC 2019)*, pages 369–378. ACM Press, 2019. arXiv:1811.01643, doi:10.1145/3293611.3331605.

[14] Sebastian Brandt. An automatic speedup theorem for distributed problems. In *Proc. 38th ACM Symposium on Principles of Distributed Computing (PODC 2019)*, pages 379–388. ACM, 2019. doi:10.1145/3293611.3331611.

[15] 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 *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.

[16] 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. arXiv:1511.00900, doi:10.1145/2897518.2897570.

[17] 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. arXiv:1702.05456, doi:10.1145/3087801.3087833.

[18] Sebastian Brandt and Dennis Olivetti. Truly tight-in-Δ bounds for bipartite maximal matching and variants. In *Proc. 39th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 69–78, 2020. doi:10.1145/3382734.3405745.

[19] 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 of *LIPIcs*, pages 18:1–18:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.DISC.2020.18.

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

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

[22] Yi-Jun Chang, Jan Studený, and Jukka Suomela. Distributed graph problems through an automata-theoretic lens. In *Proc. 28th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2021)*, LNCS. Springer, 2021. arXiv:2002.07659.

[23] Kai-Min Chung, Seth Pettie, and Hsin-Hao Su. Distributed algorithms for the lovász local lemma and graph coloring. *Distributed Comput.*, 30(4):261–280, 2017. doi:10.1007/s00446-016-0287-6.

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

[25] Manuela Fischer. Improved deterministic distributed matching via rounding. In *Proceedings of the 31st International Symposium on Distributed Computing (DISC 2017)*, pages 17:1–17:15, 2017. doi:10.4230/LIPIcs.DISC.2017.17.

[26] 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)*, volume 91 of *LIPIcs*, pages 18:1–18:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.DISC.2017.18.

[27] Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *Proc. 57th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 625–634, 2016. doi:10.1109/FOCS.2016.73.

[28] Mohsen Ghaffari and Fabian Kuhn. Deterministic distributed vertex coloring: Simpler, faster, and without network decomposition. In *Proc. 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2021)*, 2021.

[29] 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.

[30] Christoph Grunau, Václav Rozhoň, and Sebastian Brandt. The landscape of distributed complexities on trees, 2021. arXiv:2202.04724.

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

[32] Michael Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986. doi:10.1137/0215074.

[33] Yannic Maus and Tigran Tonoyan. Local conflict coloring revisited: Linial for lists. In *34th International Symposium on Distributed Computing, DISC 2020*, volume 179 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.DISC.2020.16.

[34] Gary L. Miller and John H. Reif. Parallel tree contraction and its application. In *Proc. 26th Annual Symposium on Foundations of Computer Science (FOCS 1985)*, pages 478–489. IEEE, 1985. doi:10.1109/SFCS.1985.43.

[35] Moni Naor. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM J. Discret. Math.*, 4(3):409–412, 1991. doi:10.1137/0404036.

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

[37] Dennis Olivetti. Round Eliminator: a tool for automatic speedup simulation, 2020. URL: https://github.com/olidennis/round-eliminator.

[38] Alessandro Panconesi and Romeo Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14(2):97–100, 2001. doi:10.1007/PL00008932.

[39] Václav Rozhoň and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proc. 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2020)*, pages 350–363. ACM, 2020. doi:10.1145/3357713.3384298.

[40] Jan Studený and Aleksandr Tereshchenko. Rooted tree classifier, 2021. URL: https://github.com/jendas1/rooted-tree-classifier.

[41] J. J. Sylvester. On subvariants, i.e. semi-invariants to binary quantics of an unlimited order. *American Journal of Mathematics*, 5(1):79–136, 1882. URL: http://www.jstor.org/stable/2369536.