Local problems in trees across a wide range of distributed models

Anubhav Dhar ✉
Aalto University
Indian Institute of Technology Kharagpur

Eli Kujawa ✉
Aalto University
University of Illinois Urbana-Champaign

Henrik Lievonen ✉ ⓘ
Aalto University

Augusto Modanese ✉ ⓘ
Aalto University

Mikail Muftuoglu ✉
Aalto University

Jan Studený ✉ ⓘ
Aalto University

Jukka Suomela ✉ ⓘ
Aalto University

──── **Abstract** ────

The *randomized online-LOCAL* model captures a number of models of computing; it is at least as strong as all of these models:

- the classical LOCAL model of distributed graph algorithms,
- the quantum version of the LOCAL model,
- finitely dependent distributions [e.g. Holroyd 2016],
- any model that does not violate physical causality [Gavoille, Kosowski, Markiewicz, DICS 2009],
- the SLOCAL model [Ghaffari, Kuhn, Maus, STOC 2017], and
- the dynamic-LOCAL and online-LOCAL models [Akbari et al., ICALP 2023].

In general, the online-LOCAL model can be much stronger than the LOCAL model. For example, there are *locally checkable labeling problems* (LCLs) that can be solved with logarithmic locality in the online-LOCAL model but that require polynomial locality in the LOCAL model.

However, in this work we show that in *trees*, many classes of LCL problems have the same locality in deterministic LOCAL and randomized online-LOCAL (and as a corollary across all the above-mentioned models). In particular, these classes of problems do not admit any distributed quantum advantage.
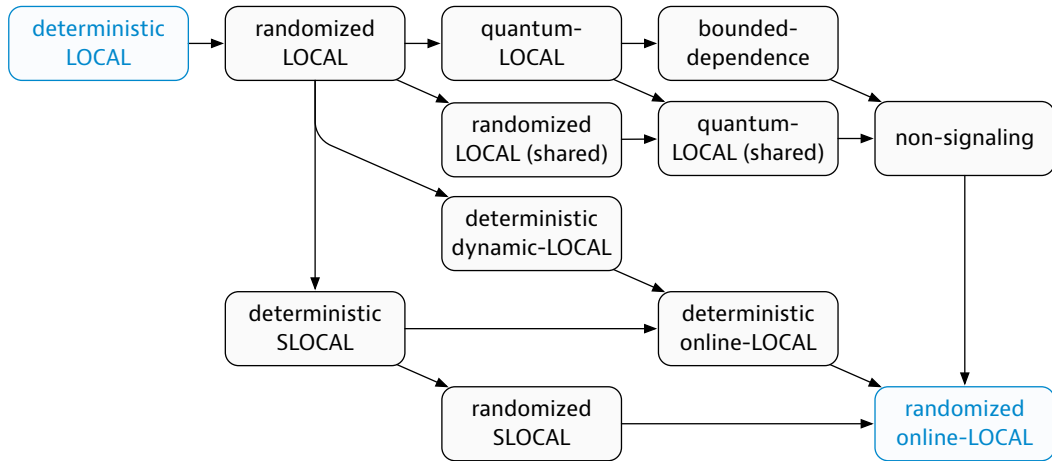
We present a near-complete classification for the case of *rooted regular trees*. We also fully classify the super-logarithmic region in *unrooted regular trees*. Finally, we show that in general trees (rooted or unrooted, possibly irregular, possibly with input labels) problems that are global in deterministic LOCAL remain global also in the randomized online-LOCAL model.

1 Introduction

The *randomized online-*LOCAL *model* was recently introduced in [1]; this is a model of computing that is at least as strong as many other models that have been widely studied in

■ **Figure 1** Landscape of models, based on [1]. In this work we show that for many families of LCL problems, the two extreme models—*deterministic* LOCAL and *randomized online*-LOCAL—are equally strong, and hence the same holds for *all* intermediate models in this diagram.

the theory of distributed computing, as well as a number of emerging models; see Figure 1. In particular, different variants of the *quantum*-LOCAL and SLOCAL models are sandwiched between the classical *deterministic* LOCAL model and the randomized online-LOCAL model.
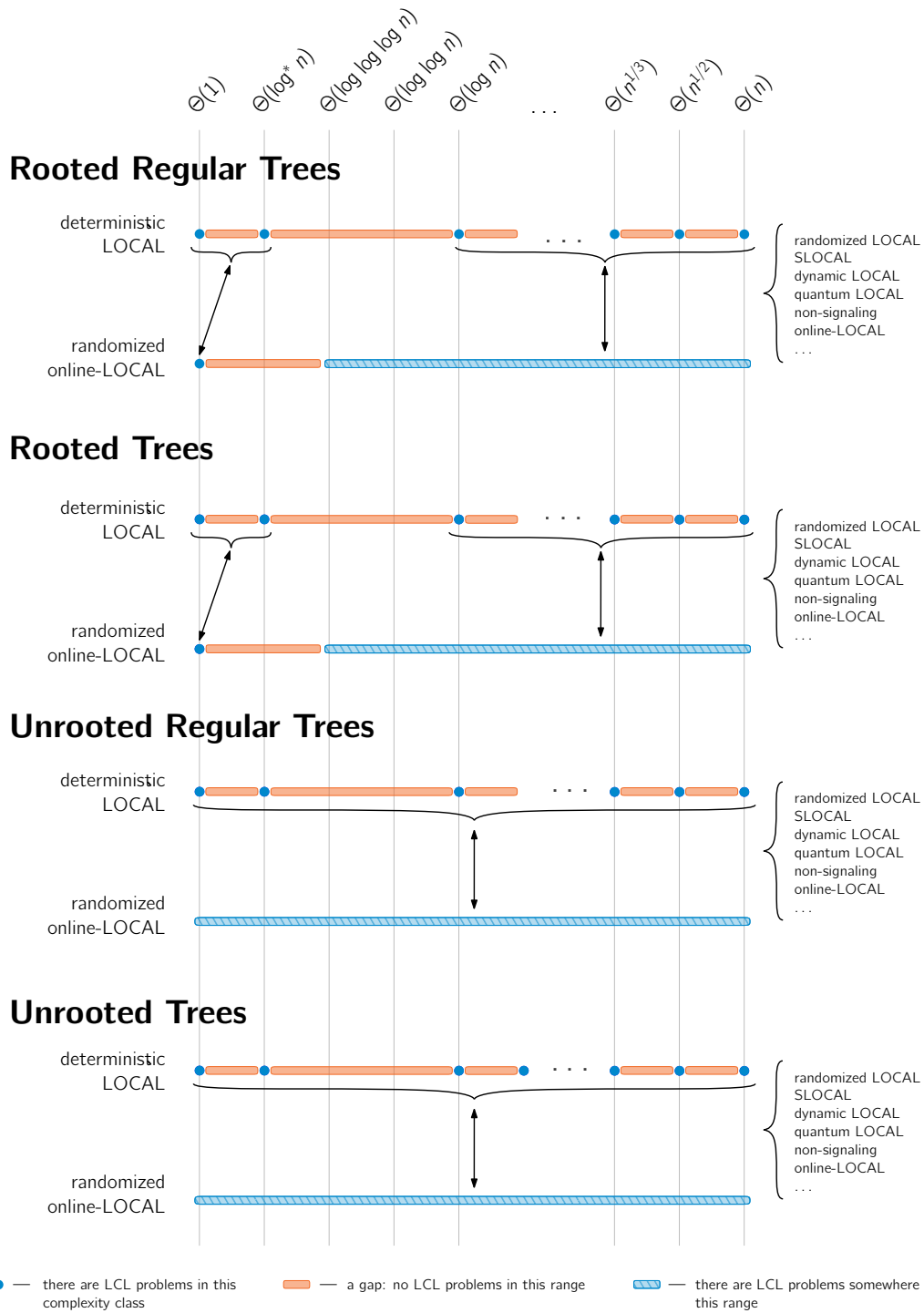
While the randomized online-LOCAL model is in general much stronger than the deterministic LOCAL model, in this work we show that in *trees*, for many families of *local* problems these two models (and hence all models in between) are asymptotically equally strong. Figure 3 summarizes the relations that we have thanks to this work; for comparison, Figure 2 shows the state of the art before this work.
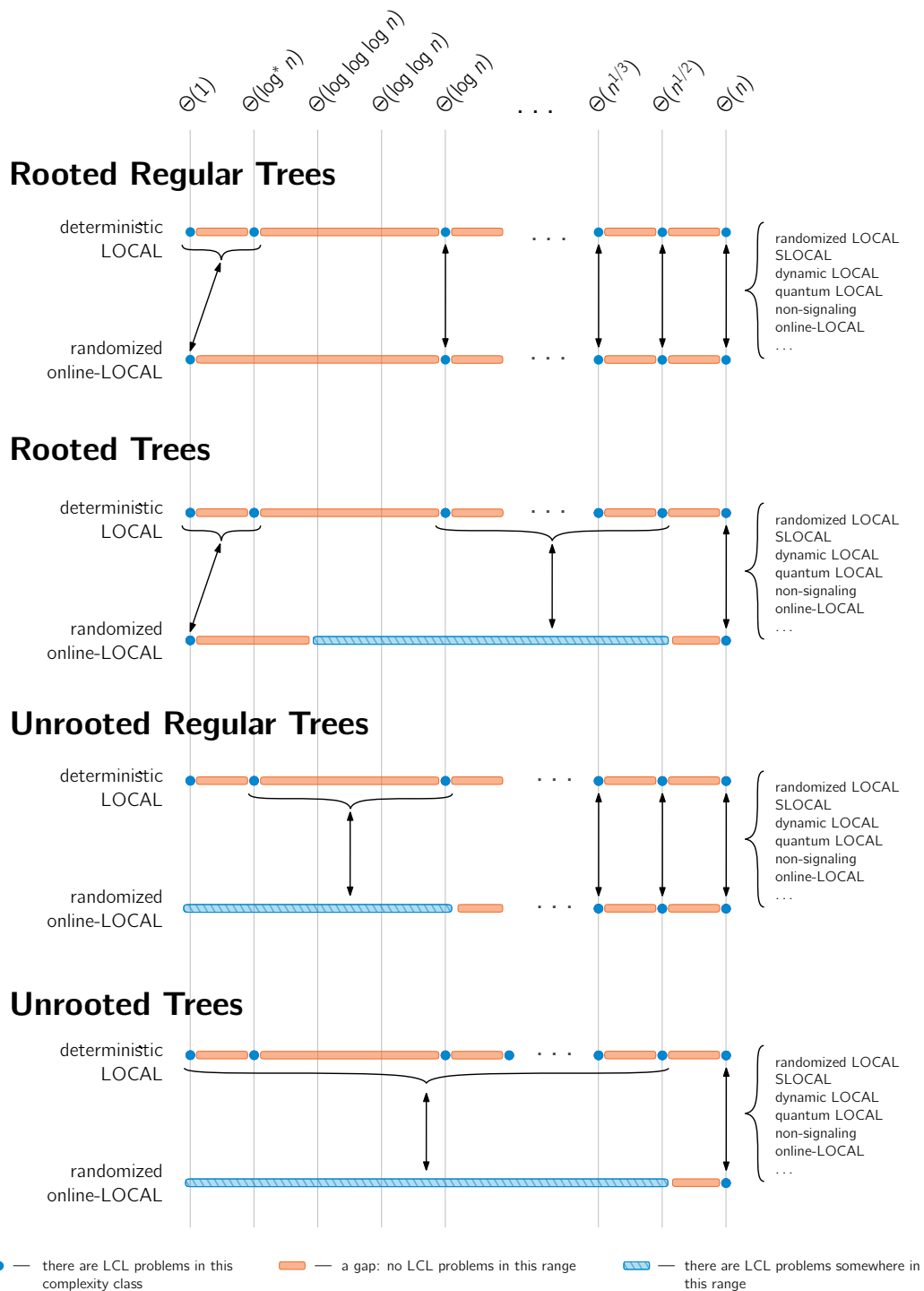
1.1 Models

We will define all relevant models formally in Section 3, but for now the following brief definitions suffice:

- In the **deterministic LOCAL** model, an algorithm A with locality T works as follows: The adversary chooses a graph G and an assignment of polynomially-sized unique identifiers. Algorithm A is applied to all nodes *simultaneously in parallel*. When we apply A to node v, algorithm A gets to see the radius-T neighborhood of v and, using this information, it has to choose the output label of node v.
- In the **randomized online-LOCAL** model, an algorithm A with locality T works as follows: The adversary chooses a graph G and a processing order σ. Then the adversary presents nodes *sequentially* following the order σ. Whenever a node v is presented, algorithm A gets to see the radius-T neighborhood of v and, using this information *as well as* all information it has seen previously and a global source of random bits, it has to choose the output label of node v.

This means that randomized online-LOCAL is stronger than deterministic LOCAL in at least three different ways: (1) we have access to shared randomness, (2) the sequential processing order can be used to break symmetry, and (3) there is global memory thanks to which we can remember everything we have seen so far. A reader familiar with the SLOCAL model can interpret it as randomized SLOCAL augmented with global memory. Note that the adversary is oblivious; it cannot adapt G and σ based on the actions of A.

Rooted Regular Trees

Rooted Trees

Unrooted Regular Trees

Unrooted Trees

Figure 2 Landscape of LCL problems in trees before this work—compare with Figure 3 to see the impact of our new contributions.

Figure 3 Landscape of LCL problems in trees after this work—compare with Figure 2 that shows the state of the art before this work.

In general, online-LOCAL (with or without randomness) is much stronger than the classical LOCAL model (with or without randomness). For example, leader election is trivial in online-LOCAL, even with locality $T = 0$ (the first node that the adversary presents is marked as the leader). However, what is much more interesting is whether online-LOCAL has advantage over LOCAL for problems defined using local constraints, such as graph coloring.

1.2 Prior Work on LCL Problems

We will study here locally checkable labeling problems (LCLs) [19]; these are problems that can be specified by giving a finite set of valid local neighborhoods. By prior work, we know a number of LCL problems that can separate the models in Figure 1, for example:

- Sinkless orientation has locality $\Theta(\log n)$ in deterministic LOCAL, locality $\Theta(\log \log n)$ in randomized LOCAL, and locality $\Theta(\log \log \log n)$ in randomized SLOCAL [7,8,10,12,14].
- One can construct an (artificial) LCL problem that shows that having access to shared randomness helps exponentially in comparison with private randomness, and this also gives a separation between e.g. randomized LOCAL and randomized online-LOCAL [6].
- In 2-dimensional grids, 3-coloring has polynomial locality in e.g. randomized SLOCAL and non-signaling models, while it can be solved with logarithmic locality in deterministic online-LOCAL [2].
- In paths, 3-coloring requires $\Theta(\log^* n)$ locality in randomized LOCAL [17,18], while it can be solved with $O(1)$ locality in the bounded-dependence model [16].

However, we do not have any such separations in rooted trees outside the $O(\log^* n)$ region. Moreover, in general unrooted trees, all separations are in the sub-logarithmic region. In this work we give a justification for this phenomenon.

1.3 Contributions

We study in this work LCL problems in the following three main settings, all familiar from prior work:
1. LCL problems in trees in general,
2. LCL problems in unrooted regular trees (there are no inputs and we only care about nodes with exactly d neighbors),
3. LCL problems in rooted regular trees (there are no inputs, edges are oriented, and we only care about nodes with exactly 1 successor and d predecessors).

In all these settings, it is known that any LCL problem falls in one of the following complexity classes in the deterministic LOCAL model: the locality is $O(1)$, $\Theta(\log^* n)$, $\Theta(\log n)$, or $\Theta(n^{1/k})$ for some $k = 1, 2, 3, \ldots$ [3,4,9,11,15]. Furthermore, in the case of rooted regular trees, we can (relatively efficiently) also decide in which of these classes any given LCL problem belongs to [3].

In this work we show that in many of these complexity classes, **deterministic LOCAL and randomized online-LOCAL are asymptotically equally strong**. Our main contributions are:
1. In general trees, the localities in randomized online-LOCAL and deterministic LOCAL are asymptotically equal in the region $\omega(\sqrt{n})$.
2. In unrooted regular trees, the localities in randomized online-LOCAL and deterministic LOCAL are asymptotically equal in the region $\omega(\log n)$.
3. In rooted regular trees, the localities in randomized online-LOCAL and deterministic LOCAL are asymptotically equal in the region $\omega(\log^* n)$.

By prior work, the relation between deterministic LOCAL and randomized online-LOCAL was well-understood in the $o(\log \log \log n)$ region for rooted (not necessarily regular) trees [1]; see Figure 2 in the appendix for the state of the art before this work. Putting together prior results and new results, the landscape shown in Figure 3 emerges.

1.4 Roadmap

We give an overview of key ideas in Section 2. We formally define LCL problems and the models of computing in Section 3, and then we are ready to analyze the sub-logarithmic region in rooted regular trees in Section 4. Then in Section 5 we introduce machinery related to the notion of *depth* from prior work, and equipped with that we can analyze the super-logarithmic region in rooted and unrooted regular trees in Sections 6 and 7. Finally, we analyze the case of general trees in Section 8.

2 Key Ideas, Technical Overview, and Comparison with Prior Work

We keep the discussion at a high level but invite the interested reader to consult the formal definitions on Section 3 as needed.

Our results heavily build on prior work that has studied LCL problems in deterministic and randomized LOCAL models. In essence, our goal is to extend its scope across the entire landscape of models in Figure 1.

For example, by prior work we know that any LCL problem Π in trees has locality either $O(\sqrt{n})$ or $\Omega(n)$ in the deterministic LOCAL model [5]. Results of this type are known as *gap results*. For our purposes, it will be helpful to interpret such a gap result as a *speedup theorem* that allows us to speed up deterministic LOCAL algorithms:

> If one can solve Π with locality $o(n)$ in **deterministic LOCAL**, then one can solve the same problem Π with locality $O(\sqrt{n})$ in **deterministic LOCAL**.

In essence our objective is to strengthen the statement as follows:

> If one can solve Π with locality $o(n)$ in **randomized online-LOCAL**, then one can solve the same problem Π with locality $O(\sqrt{n})$ in **deterministic LOCAL**.

The critical implication would be that a faster randomized online-LOCAL algorithm results not only in a faster randomized online-LOCAL algorithm, but also in a faster deterministic LOCAL algorithm—we could not only reduce locality "for free" but even switch to a weaker model. As a consequence, the complexity class $\Theta(n)$ would contain the same problems across all models—since $\Omega(n)$ in randomized online-LOCAL trivially implies $\Omega(n)$ in deterministic LOCAL (which is a weaker model), the above would give us that $o(n)$ in randomized online-LOCAL implies $o(n)$ in deterministic LOCAL.

If we could prove a similar statement for *every* gap result for LCLs in trees, we would then have a similar implication for each complexity class: the complexities across all models in Figure 1 would be the same for every LCL problem in trees. Alas, such a statement cannot be true in full generality (as we discussed above, there are some known separations between the models), but in this work we take major steps in many cases where that seems to be the case. To understand how we achieve this, it is useful to make a distinction between two flavors of prior work: classification-style and speedup-style arguments; we will make use of both in this work.

2.1 Classification Arguments for Regular Trees

First, we have prior work that is based on the idea of classification of LCLs, see e.g. [3]. The high-level strategy is as follows:

1. Define some property P so that, for any given LCL Π, we can decide whether Π has property P.
2. Show that, if Π can be solved with locality $o(n)$ in **deterministic LOCAL**, then this implies that Π must have property P.
3. Show that any problem with property P can be solved with locality $O(\sqrt{n})$ in **deterministic LOCAL**.

Such a strategy not only shows that there is a gap in the complexity landscape, but it also usually gives an efficient strategy for determining what is the complexity of a given LCL (as we will need to check some set of properties $P_1, P_2, \ldots$ and see which of them holds for a given Π in order to determine where we are in the complexity landscape). For such results our key idea is to modify the second step as follows:

2'. Show that, if Π can be solved with locality $o(n)$ in **randomized online-LOCAL**, then this implies that Π must have property P.

Note that we do not need to change the third step; as soon as we establish property P, the preexisting deterministic LOCAL algorithm kicks in. This is, in essence what we do in Sections 4, 6, and 7: we take the prior classification from [3] for rooted and unrooted regular trees and show that it can be extended to cover the entire range of models all the way from deterministic LOCAL to randomized online-LOCAL.

Log-star certificates. One key property that we make use of is so-called *certificates of $O(\log^* n)$ solvability*, introduced in [4]. In Section 4 we show that the existence of an $o(\log n)$-locality randomized online-LOCAL algorithm implies that the LCL problem has a certificate of $O(\log^* n)$ solvability, which (as the name suggests) implies that the problem can be solved with $O(\log^* n)$ locality in both the **CONGEST** and **LOCAL** models. This then extends to an $O(1)$ upper bound in **SLOCAL** model [13] and in deterministic online-LOCAL [2], which then directly implies the same upper bound also for randomized online-LOCAL model. This way we can establish the following result:

▶ **Theorem 2.1.** *Let Π be an LCL problem on rooted regular trees. If Π can be solved with $o(\log n)$ locality in randomized online-LOCAL, then it can be solved in LOCAL with $O(\log^* n)$ locality. Consequently, Π can be solved with $O(1)$ locality in SLOCAL (and thus also with the same locality in online-LOCAL, even deterministically).*

Depth. Another key property that we make use of is the *depth* of an LCL (see Section 5): to every LCL problem Π on *regular* trees there is an associated quantity d_Π called its depth that depends only on the description of Π and which allows us to classify its complexity in the deterministic LOCAL model [3]. We show that depth also captures the complexity in randomized online-LOCAL. In Section 6 we first analyze the case of rooted regular trees and show the following:

▶ **Theorem 2.2.** *Let Π be an LCL problem on rooted regular trees with finite depth $k = d_\Pi > 0$. Then any algorithm $\mathcal{A}$ solving Π in randomized online-LOCAL must have locality $\Omega(n^{1/k})$.*

The high-level strategy is as follows: assuming we have an algorithm with locality $o(n^{1/k})$, if it does not fail, then its existence would imply $d_\Pi > k$, which is a contradiction. In Section 7 we prove an analogous statement for unrooted regular trees:

▶ **Theorem 2.3.** *Let Π be an LCL problem on unrooted regular trees with finite depth $k = d_\Pi > 0$. Then any randomized online-*LOCAL *algorithm for Π has locality $\Omega(n^{1/k})$.*

Putting things together.   Combining the new results with results from prior work [3, 4, 9, 11, 15], we extend the characterization of LCLs in both unrooted and rooted regular trees all the way up to randomized online-LOCAL:

▶ **Corollary 2.4.** *Let Π be an LCL problem on unrooted regular trees. Then the following holds:*
- *If $d_\Pi = 0$, the problem is unsolvable.*
- *If $0 < d_\Pi < \infty$, then Π has locality $\Theta(n^{1/k})$ in both* LOCAL *and randomized online-*LOCAL.
- *If $d_\Pi = \infty$, then Π can be solved in* CONGEST *(and hence also in online-*LOCAL*, even deterministically) with locality $O(\log n)$.*

▶ **Corollary 2.5.** *If Π is a solvable LCL problem in rooted regular trees, then it belongs to one of the following four classes:*
1. *$O(1)$ in both deterministic* LOCAL *and randomized online-*LOCAL
2. *$\Theta(\log^* n)$ in deterministic* LOCAL *and $O(1)$ in randomized online-*LOCAL
3. *$\Theta(\log n)$ in both deterministic* LOCAL *and randomized online-*LOCAL
4. *$\Theta(n^{1/k})$ in both deterministic* LOCAL *and randomized online-*LOCAL *where $k = d_\Pi > 0$*

Comparison with prior work.   There is prior work [2] that took first steps towards classifying LCL problems in rooted regular trees; the main differences are as follows:
1. We extend the classification all the way to randomized online-LOCAL, while [2] only discusses deterministic online-LOCAL. While this may at first seem like a technicality, Figure 1 highlights the importance of this distinction: randomized online-LOCAL captures *all* models in this diagram. This includes in particular the quantum-LOCAL model and the non-signaling model. Note it is currently open if deterministic online-LOCAL captures either of these models.
2. We build on the classification of [3], while [2] builds on the (much simpler and weaker) classification of [4]. This has two direct implications:
   a. We can extend our work to *unrooted trees*, whereas the results in [2] only apply to rooted trees.
   b. We get an *exact classification* also for complexities $\Theta(n^{1/k})$. Meanwhile [2] essentially only shows that, if the complexity is $\Theta(n^{1/k})$ in deterministic LOCAL, then it is $\Theta(n^{1/\ell})$ in online-LOCAL for some $\ell \leq k$; hence it leaves open the possibility that these problems could be solved much faster (i.e., $\ell \gg k$) in online-LOCAL.

2.2   Speedup Arguments for General Trees

Second, we have also prior work based on speedup simulation arguments, see e.g. [5]. The high-level strategy there is as follows:

> Assume we are given some algorithm $\mathcal{A}$ that solves Π with locality $o(n)$ in **deterministic LOCAL**. Then we can construct a new algorithm $\mathcal{A}'$ that uses $\mathcal{A}$ as a black box to solve Π with locality $O(\sqrt{n})$ in **deterministic LOCAL**.

Unlike classification-style arguments, this does not (directly) yield an algorithm for determining the complexity of a given LCL problem. In essence, this can be seen as a black-box simulation of $\mathcal{A}$. For speedup-style arguments, the key idea for us to proceed is as follows:

Assume we are given some algorithm $\mathcal{A}$ that solves Π with locality $o(n)$ in **randomized online-LOCAL**. Then we can construct a new algorithm $\mathcal{A}'$ that uses $\mathcal{A}$ as a black box to solve Π with locality $O(\sqrt{n})$. Moreover, we can simulate $\mathcal{A}'$ efficiently in the **deterministic LOCAL** model.

Here a key challenge is that we need to explicitly change models; while *in general* deterministic LOCAL is not strong enough to simulate randomized online-LOCAL, we show that *in this case* such a simulation is possible. This is, in essence, what we do in Section 8: we start with the argument from [5] and show that sublinear-locality randomized online-LOCAL algorithms not only admit speedup inside randomized online-LOCAL, but also a simulation in deterministic LOCAL. Formally, we prove:

▶ **Theorem 2.6.** *Suppose Π is an LCL problem that can be solved with $o(n)$ locality in the online-LOCAL model (with or without randomness) in unrooted trees. Then Π can be solved in unrooted trees in online-LOCAL with $O(\sqrt{n})$ locality (with or without randomness, respectively).*

3   Preliminaries

We denote the set of natural numbers, including zero, by $\mathbb{N}$. For positive numbers integers, excluding zero, we use $\mathbb{N}_+$. For a set or multiset X and $k \in \mathbb{N}_+$, we write $\left(\!\binom{X}{k}\!\right)$ for the set of multisets over X of cardinality k.

In this work we are concerned exclusively with simple graphs. Unless stated otherwise, n always denotes the number of nodes in the graph. An algorithm solving some graph problem is said to succeed with high probability if, for all graphs except finitely many, it fails with probability at most $1/n$.

3.1   Locally Checkable Labeling Problems

In this section we define locally checkable labeling (LCL) problems as well as some of their key analytical concepts. We first recall the most general definition due to Naor and Stockmeyer [19].

▶ **Definition 3.1** (LCL problem in general graphs). *A locally checkable labeling (LCL) problem $\Pi = (\Sigma_{\mathrm{in}}, \Sigma_{\mathrm{out}}, \mathcal{C}, r)$ is defined as follows:*

- Σ_{in} *and Σ_{out} are finite, non-empty sets of input and output labels, respectively.*
- $r \in \mathbb{N}_+$ *is the* checkability radius*.*
- $\mathcal{C}$ *is the set of* constraints, *namely a finite set of graphs where:*
  - *Each graph $H \in \mathcal{C}$ is centered at some node v.*
  - *The distance of v from all other nodes in H is at most r.*
  - *Each node $u \in H$ is labeled with a pair $(i(u), o(u)) \in \Sigma_{\mathrm{in}} \times \Sigma_{\mathrm{out}}$.*

For a graph $G = (V, E)$ whose vertices are labeled according to Σ_{in}, a (node) labeling of a graph $G = (V, E)$ is a solution *to Π if it labels every node $v \in V$ with a label from Σ_{out} such that the r-neighborhood of v in G is identical to some graph of $\mathcal{C}$ (when we place v at the center of the respective graph in $\mathcal{C}$). Every node for which this requirement holds is said to be* correctly labeled*. If the nodes of G are labeled with a solution, then G itself is* correctly labeled*.*

We use this definition to refer to problems in the case of general, that is, not necessarily regular trees. For regular trees, we use two other formalisms, one for the case of rooted and one for that of unrooted trees. These are simpler to work with and require checking only

labels in the immediate vicinity of a node or half-edge. We note there is precedent in the literature for taking this approach (see, e.g., [3]).

▶ **Definition 3.2** (LCL problem on regular rooted trees). *An* LCL problem on regular rooted trees *is a tuple* $\Pi = (\Delta, \Sigma, \mathcal{V})$ *where:*

- $\Delta \in \mathbb{N}_+$ *and* Σ *is a finite, non-empty set.*
- *The (node)* constraints *form a set* $\mathcal{V}$ *of pairs* (σ, S) *where* $\sigma \in \Sigma$ *and* $S \subseteq \left(\!\!\binom{\Sigma}{\Delta}\!\!\right)$.

For a rooted tree $T = (V, E)$ *with nodes having degree in* $\{1, \Delta\}$, *a solution to* Π *is a labeling* $\ell \colon V \to \Sigma$ *such that, for each node* v *with* Δ *children* $u_1, \ldots, u_\Delta$, *we have* $(\ell(v), \{\ell(u_1), \ldots, \ell(u_\Delta)\}) \in \mathcal{V}$. *Again, this requirement allows us to speak of* correctly labeled *nodes and thus also* correctly labeled *trees.*

Under this formalism, leaves may be labeled arbitrarily. When Π is clear from the context, we refer to the elements of $\left(\!\!\binom{\Sigma}{\Delta}\!\!\right)$ as node configurations.

Meanwhile, in the case of unrooted trees, we work with LCL problems where the labels are placed on (instead of nodes) and are subject to node and edge constraints.

▶ **Definition 3.3** (LCL problem on regular unrooted trees). *An* LCL problem on regular unrooted trees *is a tuple* $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ *where:*

- $\Delta \in \mathbb{N}_+$ *and* Σ *is a finite, non-empty set.*
- *The* node constraints *form a set* $\mathcal{V} \subseteq \left(\!\!\binom{\Sigma}{\Delta}\!\!\right)$.
- *The* edge constraints *form a set* $\mathcal{E} \subseteq \left(\!\!\binom{\Sigma}{2}\!\!\right)$.

For an unrooted tree $T = (V, E)$ *with nodes having degree in* $\{1, \Delta\}$, *a solution to* Π *is a labeling of half-edges, that is, a map* $\ell \colon V \times E \to \Sigma$ *such that the following holds:*

- *For every node* $v \in V$ *with degree* Δ *that is incident to the edges* $e_1, \ldots, e_\Delta$, *we have* $\{\ell(v, e_1), \ldots, \ell(v, e_\Delta)\} \in \mathcal{V}$.
- *For every edge* $e = \{u, v\} \in E$, $\{\ell(u, e), \ell(v, e)\} \in \mathcal{E}$.

As before, if T *is labeled with a solution, then it is* correctly labeled. *In light of these two types of requirements, it is also natural to speak of* correctly labeled *nodes and edges.*

In the same vein as the preceding definition, leaves are unconstrained and may be labeled arbitrarily. As before, if such an LCL problem Π on regular unrooted trees is clear from the context, we refer to the elements of $\left(\!\!\binom{\Sigma}{\Delta}\!\!\right)$ as node configurations and to those of $\left(\!\!\binom{\Sigma}{2}\!\!\right)$ as edge configurations.

3.2 Models of Distributed Computing

Next we formally define the LOCAL model and its extensions.

▶ **Definition 3.4** (Deterministic LOCAL model). *The deterministic* LOCAL *model of distributed computing runs on a graph* $G = (V, E)$ *resembling a computer network, with each node* $v \in V$ *representing a computer, and each edge representing a connection channel. Each node is labeled with a unique identifier. All nodes run the same distributed algorithm in parallel. Initially, a node is only aware of its own identifier and degree. Computation proceeds in synchronous rounds, and in each round a node can send and receive a message to and from each neighbor and update its state. Eventually each node must stop and announce its local output (its part of the solution, e.g. in graph coloring its own color). The* running time, round complexity, *or* locality *of the algorithm is the (worst-case) number of rounds* $T(n)$ *until the algorithm stops in any* n-*node graph.*

▶ **Definition 3.5** (Randomized LOCAL model). *The randomized LOCAL model is defined identically to the (deterministic) LOCAL model, with the addition that each node has access to a private, infinite stream of random bits. Additionally, a randomized LOCAL algorithm is required to succeed with high probability.*

▶ **Definition 3.6** (Deterministic online-LOCAL model [2]). *In the (deterministic) online-LOCAL model, an adversary chooses a sequence of nodes, $\sigma = (v_1, v_2, \ldots, v_n)$, and reveals the nodes to the algorithm one at a time. The algorithm processes each node sequentially. Given an online-LOCAL algorithm with locality $T(n)$, when a node v_i is revealed, the algorithm must choose an output for v_i based on the subgraph induced by the radius-$T(n)$ neighborhoods of $v_1, v_2, \ldots, v_i$. In other words, the algorithm retains global memory.*

▶ **Definition 3.7** (Randomized online-LOCAL model [1]). *In the randomized online-LOCAL model, an adversary first commits to a sequence of nodes, $\sigma = (v_1, v_2, \ldots, v_n)$, and reveals the nodes to the algorithm one at a time, as in the online-LOCAL model. The algorithm runs on each v_i and retains global memory, as in the online-LOCAL model. Additionally, the algorithm has access to an infinite stream of random bits unbeknownst to the adversary; that is, the adversary* cannot *change the order they present the remaining nodes based on the intermediate output of the algorithm. As in the randomized LOCAL, the algorithm is required to succeed with high probability.*

4 Sub-logarithmic Gap for LCLs in Rooted Regular Trees

In this section, we show that there are no LCL problems on rooted trees with locality in the range $\omega(1)$—$o(\log n)$ in the randomized online-LOCAL model. Moreover, the problems that are solvable with locality $O(1)$ in randomized online-LOCAL are exactly the same problems that are solvable with locality $O(\log^* n)$ in the LOCAL model.

▶ **Theorem 2.1.** *Let Π be an LCL problem on rooted regular trees. If Π can be solved with $o(\log n)$ locality in randomized online-LOCAL, then it can be solved in LOCAL with $O(\log^* n)$ locality. Consequently, Π can be solved with $O(1)$ locality in SLOCAL (and thus also with the same locality in online-LOCAL, even deterministically).*

We show this by showing that a locality-$o(\log n)$ randomized online-LOCAL algorithm solving LCL problem Π implies the existence of a *coprime certificate for $O(\log^* n)$ solvability* (see Definition 4.6) that then implies that Π is solvable with locality $O(\log^* n)$ in the LOCAL model [4], and hence with locality $O(1)$ in the randomized online-LOCAL model.

Throughout this section, we consider an LCL problem $\Pi = (\Delta, \Sigma, \mathcal{V})$ and a randomized online-LOCAL algorithm $\mathcal{A}$ that solves Π with locality $T(n) = o(\log n)$ with high probability. We start by constructing a family of input instances. We then argue that $\mathcal{A}$ solving these instances must produce a canonical labeling regardless of the randomness. Finally, we show how we can use this canonical labeling to construct the coprime certificate for $O(\log^* n)$ solvability.

4.1 Construction of Input Instances

Let d be a depth parameter that we fix later, and let Δ be the number of children of internal nodes. We now construct the input instance as follows:

▶ **Definition 4.1** (Family of input instances). *We construct the family of input instances as follows:*

1. *Construct $(|\Sigma|+1)$ chunks of trees, each containing Δ^{d+1} complete rooted trees of height $2d$. Give the trees in each chunk an ordering. Let M be the set of nodes in these trees at distance d from the root; we call these nodes* middle nodes.

2. *Choose a bit b, and choose a node u which is at depth d in any of the trees created in the previous phase. Choose a chunk C such that node u is not contained in chunk C.*

3. *The subtree rooted at u has Δ^d leaf descendants.*

   If $b = 0$: *Identify the roots of the first Δ^d trees of chunk C with the leaf descendants of u in a consistent order (see bottom-left of Figure 4).*

   If $b = 1$: *Make the roots of the first Δ^{d+1} trees of chunk C the children of the leaf descendants of u in a consistent order (see bottom-right Figure 4).*

Observe that trees constructed in this way have $n = (|\Sigma| + 1) \cdot \Delta^{d+1} \cdot \frac{\Delta^{2d+1}-1}{\Delta-1}$ nodes. Moreover, choosing (b, u, C) uniquely fixes the construction. Let $\mathcal{P}$ be the set of choices for (b, u, C).

▶ **Observation 4.2.** *The number of instances is less than the number of nodes in each instance. In particular, $|\mathcal{P}| = 2|\Sigma|(|\Sigma| + 1)\Delta^{2d+1} < n$.*

We can now fix depth parameter d such that $d > \log_\Delta(2|\Sigma|)$ and $d > T(n)$. Recall that $T(n) = o(\log n)$, and hence such d exists.

4.2 Randomness of the Algorithm

Throughout this discussion, we let $\Pr_R(\cdot)$ denote the probability of an event when the randomness is over some source R for which the probability is being defined.

For each instance (b, u, C), we fix the order middle nodes M such that it is consistent across all instances. We then reveal these middle nodes to $\mathcal{A}$ in this order. Let $S \in \overline{\Sigma} = \Sigma^{\Delta^{2d+1}(|\Sigma|+1)}$ be the random sequence of labels generated by $\mathcal{A}$ for the middle nodes M. Since the locality of $\mathcal{A}$ is $T(n) < d$, the algorithm does not get to see the roots or the leaves of these height-$2d$ trees. In particular, the algorithm does not get to know which instance (b, u, C) we have chosen. Hence S must be independent of the choice of (b, u, C).
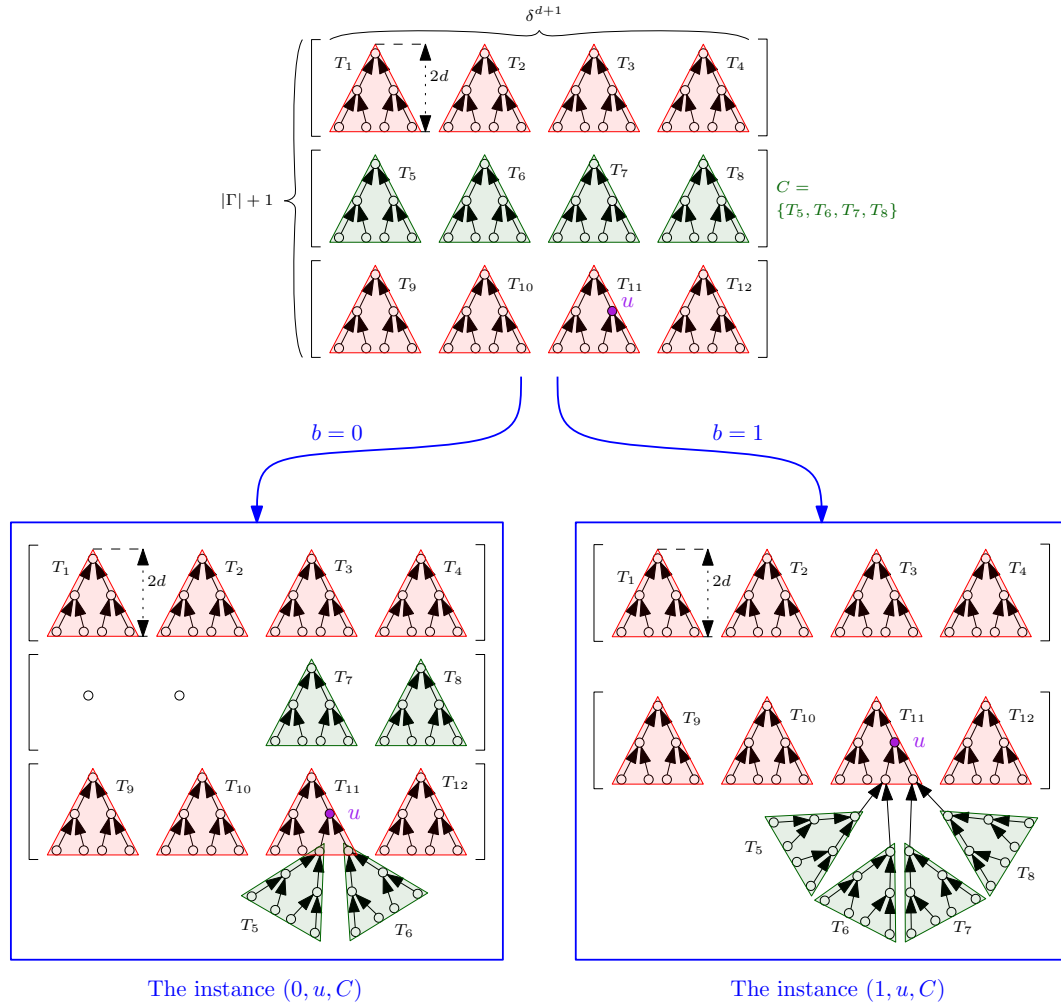
▶ **Observation 4.3.** *For every choice $\overline{\sigma} \in \overline{\Sigma}$ of labeling middle nodes M, and every instance $(b, u, C) \in \mathcal{P}$, we must have*

$$\Pr_S(S = \overline{\sigma}) = \Pr_S(S = \overline{\sigma} \mid (b, u, C))$$

where $\Pr_S(S = \overline{\sigma})$ denotes the probability that $\mathcal{A}$ produces output $\overline{\sigma}$ for nodes M, and $\Pr_S(S = \overline{\sigma} \mid (b, u, C))$ denotes the same given that the input instance was (b, u, C).

Note that while the output S for nodes M is independent of (b, u, C), the output that $\mathcal{A}$ produces for the rest of the nodes of the input may be dependent of (b, u, C). We denote this random variable by $\mathcal{A}^+(b, u, C)$.

We now analyze the failure probability of $\mathcal{A}$ to show that there exists a fixed labeling $\overline{\sigma *}$ for middle nodes M such that the labeling is still completable for each input instance (b, u, C). Let $\mathcal{F}$ denote the event that algorithm $\mathcal{A}$ fails to solve the problem. As, by assumption, $\mathcal{A}$ succeeds with high probability, we must have that $\Pr_{S, \mathcal{A}^+(b,u,C)}(\mathcal{F}) \leq \frac{1}{n}$ for all choices of $(b, u, C) \in \mathcal{P}$. We can now pick a labeling of the middle nodes that minimizes the average failure probability:

Figure 4 Visualization of two trees in the family of input instances. At the top we have $|\Sigma| + 1$ chunks of complete trees of depth $2d$, each containing Δ^{d+1} trees. The green trees in the middle row represent the chosen chunk C. On the bottom-left, we have $b = 0$, in which case we identify the roots of the first Δ^d trees of C with the leaf descendants of node u. On the bottom-right, we have $b = 1$, in which case we make all trees in chunk C the children of the leaf descendants of node u. See Definition 4.1 for full construction.

▶ **Definition 4.4.** *Let $\overline{\sigma*}$ be defined as follows:*

$$\overline{\sigma*} \in \underset{\overline{\sigma} \in \overline{\Sigma}, \Pr_S(S=\overline{\sigma})>0}{\arg\min} \sum_{(b,u,C)\in\mathcal{P}} \Pr_{\mathcal{A}^+(b,u,C)}(\mathcal{F} \mid (b,u,C), S = \overline{\sigma})$$

where ties are broken deterministically.

This definition ensures the following: Give that the algorithm labels nodes M with $\overline{\sigma*}$, the total probability of failure over all instances (b, u, C) is minimized. In some sense, $\overline{\sigma*}$ is the best labeling for M when the algorithm know nothing about the instance. Note that $\overline{\sigma*}$ is a concrete element of $\overline{\Sigma}$ and hence does not depend on the randomness of the algorithm. We can formalize this idea:

▶ **Lemma 4.5.** *Regardless of the instance $(b, u, C) \in \mathcal{P}$, if the labeling S of nodes M is $\overline{\sigma*}$, there exists a valid way to label the remaining nodes of the instance.*

Proof. We start by noting that

$$\frac{1}{n} \geq \Pr_{S,\mathcal{A}^+(b,u,C)}(\mathcal{F})$$

since $\mathcal{A}$ works correctly with high probability. We can now expand the probability to be conditional over the initial labeling S:

$$\Pr_{S,\mathcal{A}^+(b,u,C)}(\mathcal{F}) = \sum_{\overline{\sigma} \in \overline{\Sigma}} \Pr_{\mathcal{A}^+(b,u,C)}(\mathcal{F} \mid (b,u,C), S = \overline{\sigma}) \Pr_{S}(S = \overline{\sigma} \mid (b,u,C)).$$

Next, we apply Observation 4.3 to get

$$\frac{1}{n} \geq \sum_{\overline{\sigma} \in \overline{\Sigma}} \Pr_{\mathcal{A}^+(b,u,C)}(\mathcal{F} \mid (b,u,C), S = \overline{\sigma}) \Pr_{S}(S = \overline{\sigma}),$$

and then we sum over all choices of (b,u,C) on both sides to get

$$\frac{|\mathcal{P}|}{n} = \sum_{(b,u,C) \in \mathcal{P}} \frac{1}{n} \geq \sum_{(b,u,C) \in \mathcal{P}} \sum_{\overline{\sigma} \in \overline{\Sigma}} \Pr_{\mathcal{A}^+(b,u,C)}(\mathcal{F} \mid (b,u,C), S = \overline{\sigma}) \Pr_{S}(S = \overline{\sigma})$$

Exchanging the order of summation and noting that $\overline{\sigma*}$ minimizes

$$\sum_{(b,u,C) \in \mathcal{P}} \Pr_{\mathcal{A}^+(b,u,C)}(\mathcal{F} \mid (b,u,C), S = \overline{\sigma}),$$

we get

$$\frac{|\mathcal{P}|}{n} \geq \sum_{(b,u,C) \in \mathcal{P}} \Pr_{\mathcal{A}^+(b,u,C)}(\mathcal{F} \mid (b,u,C), S = \overline{\sigma*})$$

Recalling that $\frac{|\mathcal{P}|}{n} < 1$ by Observation 4.2 and that all probabilities are non-negative numbers, we complete our calculation:

$$1 > \Pr_{\mathcal{A}^+(b,u,C)}(\mathcal{F} \mid (b,u,C), S = \overline{\sigma*}) \quad \text{for each } (b,u,C) \in \mathcal{P}.$$

Hence for each instance (b,u,C), the algorithm fails with a probability strictly less than 1. In other words, it must succeed with non-zero probability, and hence a correct labeling must also exist. ◀

4.3   Getting a Coprime Certificate as a Subset of All Such Instances

We are now ready to extract from algorithm $\mathcal{A}$ a coprime certificate for $O(\log^* n)$ solvability. The idea is to pick a subset of input instances $\mathcal{P}$ that—along with the labeling $\overline{\sigma*}$—form the pairs of sequences of trees of the certificate. We defer the proof of why this implies the existence of a locality-$O(\log^* n)$ LOCAL algorithm to previous work [4].

▶ **Definition 4.6** (Coprime certificate for $O(\log^* n)$ solvability [4])**.** *Let $\Pi = (\Delta, \Sigma, C)$ be an LCL problem. A certificate for $O(\log^* n)$ solvability of Π consists of labels $\Sigma_{\mathcal{T}} = \{\sigma_1, \ldots, \sigma_t\} \subseteq \Sigma$, a depth pair (d_1, d_2) and a pair of sequences $\mathcal{T}^1$ and $\mathcal{T}^2$ of t labeled trees such that*
1. *The depths d_1 and d_2 are coprime.*
2. *Each tree of $\mathcal{T}^1$ (resp. $\mathcal{T}^2$) is a complete Δ-ary tree of depth $d_1 \geq 1$ (resp. $d_2 \geq 1$).*
3. *Each tree is labeled by labels from Σ correctly according to problem Π.*

4. Let $\bar{\mathcal{T}}_i^1$ (resp. $\bar{\mathcal{T}}_i^2$) be the tree obtained by starting from $\mathcal{T}_i^1$ (resp. $\mathcal{T}_i^2$) and removing the labels of all non-leaf nodes. It must hold that all trees $\bar{\mathcal{T}}_i^1$ (resp. $\bar{\mathcal{T}}_i^2$) are isomorphic, preserving the labeling. All the labels of the leaves of $\bar{\mathcal{T}}_i^1$ (resp. $\bar{\mathcal{T}}_i^2$) must be from set $\Sigma_{\mathcal{T}}$.
5. The root of tree $\mathcal{T}_i^1$ (resp. $\mathcal{T}_i^2$) is labeled with label σ_i.

Let $\Sigma_{\mathcal{T}} = \{\sigma_1, \sigma_2, \ldots, \sigma_t\} \subseteq \Sigma$ be the set of labels appearing in $\overline{\sigma*}$, and let $(u_1, u_2, \ldots, u_t)$ be some middle nodes in the input instances such that node u_i has label σ_i according to $\overline{\sigma*}$. Note that there are $|\Sigma| + 1$ chunks whereas $t = |\Sigma_{\mathcal{T}}| < |\Sigma| + 1$. Therefore, we get the following result by the pigeonhole principle:

▶ **Observation 4.7.** *There must be a chunk C_0 which does not contain any node in* $\{u_1, \ldots, u_t\}$.

With these definitions of labels $\Sigma_{\mathcal{T}} = (\sigma_1, \ldots, \sigma_t)$, chunk C_0, and nodes $(u_1, \ldots, u_t)$, we are ready to prove our main result:

▶ **Theorem 4.8.** *For an LCL problem Π without inputs on rooted regular trees, if there is a randomized online-LOCAL algorithm $\mathcal{A}$ with locality $T(n) = o(\log n)$, then there exists a coprime certificate of $O(\log^* n)$ solvability for Π, with $\Sigma_{\mathcal{T}}$ as the subset of labels.*

Proof. For each $i \in \{1, \ldots, t\}$, consider the instance $(0, u_i, C_0)$ with $S = \overline{\sigma*}$ and some valid way to label the remaining nodes; such a labeling exists due to Lemma 4.5. The subtree rooted at u_i contains the first Δ^d trees of the chunk C_0. Let the depth-$2d$ subgraph rooted at u_i be tree $\mathcal{T}_i^1$. See Figure 5a for a visualization.

Note that for each $i \in \{1, \ldots, t\}$, tree $\mathcal{T}_i^1$ has a root labeled with σ_i, and the leaves are labeled with labels from set $\Sigma_{\mathcal{T}}$ in an identical way. Hence $\mathcal{T}^1$ form the first sequence of the certificate.

Similarly, we consider $(1, u_i, C_0)$ for all $i \in \{1, \ldots, t\}$. This time we let the depth-$(2d+1)$ subtree rooted at u_i be tree $\mathcal{T}_i^2$. See Figure 5b for a visualization. The sequence $\mathcal{T}^2$ forms the second sequence of the certificate, again by similar arguments.

It is easy to check that $\Sigma_{\mathcal{T}}$, $2d$ and $2d + 1$ and sequences $\mathcal{T}^1$ and $\mathcal{T}^2$ indeed form a coprime certificate for $O(\log^* n)$ solvability for problem Π. ◀
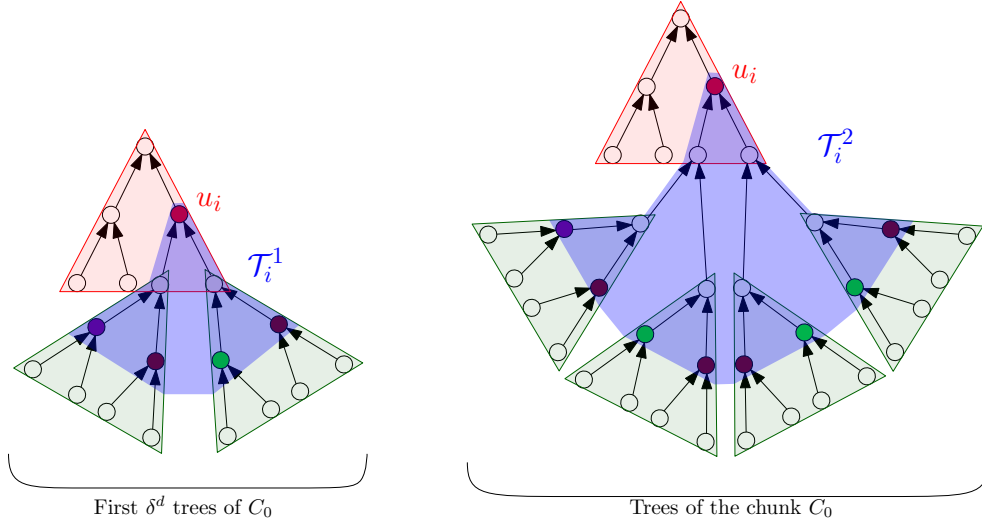
We can now proof the main result of this section:

▶ **Theorem 2.1.** *Let Π be an LCL problem on rooted regular trees. If Π can be solved with $o(\log n)$ locality in randomized online-LOCAL, then it can be solved in LOCAL with $O(\log^* n)$ locality. Consequently, Π can be solved with $O(1)$ locality in SLOCAL (and thus also with the same locality in online-LOCAL, even deterministically).*

Proof. A locality-$o(\log n)$ randomized online-LOCAL algorithm for problem Π implies the existence of certificate of $O(\log^* n)$ solvability of Π by Theorem 4.8. This further implies existence of LOCAL algorithm solving Π with locality $O(\log^* n)$ [4], which in turn implies the existence of SLOCAL algorithm solving Π with locality $O(1)$ [13]. Since the randomized online-LOCAL is stronger than the SLOCAL model [1], this implies the existence of a locality-$O(1)$ randomized online-LOCAL algorithm that solves Π. Therefore, there can be no LCL problem on rooted regular trees where the optimal algorithm has a locality of $T(n)$ which is both $o(\log n)$ and $\omega(1)$. ◀

5 Definition of Depth

Before we continue, we recall notions pertaining to the analysis of LCL problems in the regular trees case (for both rooted and unrooted trees). We stress the tools that we present

(a) Trees $\mathcal{T}_i^1$ of depth $2d$ for certificate from instance $(0, u_i, C_0)$.

(b) Trees $\mathcal{T}_i^1$ of depth $2d + 1$ for certificate from instance $(1, u_i, C_0)$.

■ **Figure 5** Construction of trees of coprime certificate for $O(\log^* n)$ solvability. The blue shaded region represents the tree of the sequence and extends from the labeled middle node u_i to the middle nodes of the trees hanging from its leaf descendants.

here concern only the *description* of the problems and thus may be applied no matter what is our computational model of interest. Most definitions are morally the same for both rooted and unrooted trees (and, indeed, are even named the same), though the latter turns out to be more convoluted. Hence we will always introduce the relevant concept first for rooted trees. In fact, to obtain a clearer understanding of the concepts involved, an unfamiliar reader may prefer to postpone the parts concerning unrooted trees to a reread of the text altogether.

Automata.   The definitions of LCL problems we introduced above are useful in that they allow us to associate an LCL problem Π with an *automaton* that encodes correct solutions to Π. Conceptually speaking, the automaton describes the labels that we observe as we are traversing a path in a correctly labeled tree. For rooted trees this is relatively straightforward to specify; in the unrooted case we do not have a sense of direction and hence we must encode that in the nodes by using two components, one specifying which label we see when entering and the other which label we see when exiting the node. Alternatively, it may also be instructive to imagine the automaton as describing paths on the *line graph* of the original tree (i.e., we are traversing the tree *edge by edge* instead of node by node).

▶ **Definition 5.1** (Automaton for an LCL on regular trees)**.** *Let Π be an LCL problem on regular trees.*
- *If $\Pi = (\Delta, \Sigma, \mathcal{V})$ is a problem on rooted trees, then the* automaton $\mathcal{M}_\Pi$ *associated with Π is the digraph with Σ as its set of nodes and where we have an edge (σ, σ') if and only if there is $(\sigma, S) \in \mathcal{V}$ such that $\sigma' \in S$.*
- *If $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ is a problem on unrooted trees, then the* automaton $\mathcal{M}_\Pi$ *associated with Π is the digraph defined as follows:*
  - *The nodes of $\mathcal{M}_\Pi$ are the elements $(x, y) \in \Sigma \times \Sigma$ for which $\{x, y\} \in \left(\binom{\Sigma}{2} \right)$ is a sub-multiset of some node configuration in $\mathcal{V}$.*
  - *There is an edge between (x_1, x_2) and (y_1, y_2) in $\mathcal{M}_\Pi$ if and only if $\{x_2, y_1\} \in \mathcal{E}$.*

Preliminaries. For every LCL problem Π on regular trees, there is a fixed quantity d_Π that depends only on the problem description and which essentially captures its complexity. This quantity is called the *depth* of Π. The precise definition is rather involved, so here we attempt to keep it brief and self-contained; for a more extensive treatment, see for instance [3]. As mentioned before, the definitions for rooted and unrooted trees differ in the details but are otherwise conceptually very similar to one other.

Let us now fix an LCL problem Π on regular trees. The following is our roadmap in order to define the depth d_Π:

1. First we define what is meant by taking the restriction of Π to a set of labels (in the rooted case) or set of node configurations (in the unrooted case).
2. Next we define two operations trim and flexible-SCC that induce two different kinds of restrictions that we may take.
3. By alternating between the two, we obtain so-called good sequences of problem restrictions.
4. Finally, d_Π is defined to be exactly the maximum length of such a good sequence.

Restrictions. In the case of rooted trees, a restriction is defined in terms of a *subset* of labels; that is, we are simply considering the same problem but allowing only a subset of labels to be used in the solution. In contrast, in unrooted trees, restrictions are defined by a *set of permissible pairs* of labels and we allow only node configurations to be used where *every* pair of labels present in the configuration is permissible.

▶ **Definition 5.2** (Restriction of an LCL on regular trees). *Let Π be an LCL on regular trees.*
- *When $\Pi = (\Delta, \Sigma, \mathcal{V})$ is a problem on rooted trees, the* restriction *of Π to a subset $\Sigma' \subseteq \Sigma$ of labels is the problem $\Pi \restriction_{\Sigma'} = (\Delta, \Sigma', \mathcal{V}')$ where $\mathcal{V}'$ consists of all pairs $(\sigma, S) \in \mathcal{V}$ for which $\sigma \in \Sigma'$ and $S \subseteq \Sigma'$.*
- *If $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ is a problem on unrooted trees, then the* restriction *of Π to $\mathcal{D} \subseteq \left(\binom{\Sigma}{2}\right)$ is the problem $\Pi \restriction_{\mathcal{D}} = (\Delta, \Sigma, \mathcal{V}', \mathcal{E})$ where $\mathcal{V}' \subseteq \mathcal{V}$ is maximal such that $\left(\binom{C}{2}\right) \subseteq \mathcal{D}$ holds for every $C \in \mathcal{V}'$.*

The trim operation. Next we define the trim operation, which restricts the set of labels (or, in the case of unrooted trees, node configurations) to those that can be used at the root of a complete tree of arbitrary depth. Although this sounds like a precise definition, we stress that "complete tree" is in fact an ambiguous term in this context since it has different meanings depending on which LCL formalism we are (i.e., rooted or unrooted trees). We settle this matter upfront in a separate definition before turning to that of trim proper.

▶ **Definition 5.3** (Complete regular tree). *Let $i \in \mathbb{N}_+$. With T_i we denote the tree of depth i where the root has degree $\Delta - 1$ and every other inner vertex has degree Δ. Meanwhile, T_i^* denotes the tree of depth i where every inner vertex (including the root) has degree Δ.*

Note that, technically, T_i is not regular. Nevertheless we can add a single parent node of degree 1 to the root in order to make it so. (Since it has degree 1, this new node is unconstrained.) We avoid doing so in order to be able to refer to the root of T_i more easily.

▶ **Definition 5.4** (trim). *Let Π be an LCL on regular trees of degree Δ.*
- *If $\Pi = (\Delta, \Sigma, \mathcal{V})$ is a problem on rooted trees, then trim($\cdot$) maps subsets of Σ again to subsets of Σ. For $\Sigma' \subseteq \Sigma$, trim(Σ') is the set of all $\sigma \in \Sigma'$ for which, for every $i \in \mathbb{N}_+$, there is a solution of Π to T_i such that:*
  - *The root is labeled with σ.*
  - *Every other vertex (including the leaves) is labeled with a label from Σ'.*

- When $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ is a problem on unrooted trees, $\mathsf{trim}(\cdot)$ maps subsets of $\mathcal{V}$ again to subsets of $\mathcal{V}$. Namely, for $\mathcal{D} \subseteq \mathcal{V}$, $\mathsf{trim}(\mathcal{D})$ is the set of node configurations $C \in \mathcal{D}$ for which, for every $i \in \mathbb{N}_+$, there is a solution of Π to T_i^* such that:
  - The root has C as its node configuration.
  - Every other vertex (except for the leaves, which are always unrestricted) has an element of $\mathcal{D}$ as its node configuration.

Hence trim restricts the sets of labels (or node configurations, in the case of unrooted trees) that can be placed at the root of the tree. Note that the rest of the tree does not need to be labeled according to what ends up being placed in the trim set but only to the set Σ' of labels (or set $\mathcal{D}$ of node configurations) that we started with.

The flexible-SCC operation.   In contrast to the trim operation, which is defined on the basis of labelings of the complete tree, $\mathsf{flexible}\text{-}\mathsf{SCC}$ is computed solely by analyzing (the graph of) the automaton $\mathcal{M}_\Pi$ associated with Π.

For concreteness, let us focus on the case of rooted trees. Given some subset $\Sigma' \subseteq \Sigma$ of labels, we start by considering the restriction of Π to Σ', in which case we obtain a restricted LCL problem $\Pi' = \Pi \restriction_{\Sigma'}$. Notice the automaton $\mathcal{M}_{\Pi'}$ associated with Π is contained in $\mathcal{M}_\Pi$. We are interested in the strongly connected components of $\mathcal{M}_{\Pi'}$ and in particular those whose nodes can be reached from one another in a *flexible* way. Flexibility here is meant in the sense of the walk's length, namely that one may reach any vertex from any other one in any desired (large enough) number of steps. Conceptually, each such component of $\mathcal{M}_{\Pi'}$ gives us a strategy for locally filling in labels for arbitrarily long paths using just the set $\Lambda \subseteq \Sigma'$ of labels in the component.

This notion of flexibility is specified in the next definition.

▶ **Definition 5.5** (Path-flexibility)**.** *Let Π be an LCL problem on regular trees (of either kind), and let $\mathcal{M}_\Pi$ be the automaton associated with it. A subset U of vertices of $\mathcal{M}_\Pi$ is said to be* path-flexible *if there is a constant $K \in \mathbb{N}_+$ such that, for any choice of vertices $u, v \in U$, there is a walk of length $k \geq K$ from u to v that only visits vertices in U.*

With this notion we now define $\mathsf{flexible}\text{-}\mathsf{SCC}$. As already stressed several times, the definition for unrooted trees is more involved since we have to manipulate multisets, but morally it yields the same operation as in the rooted case.

▶ **Definition 5.6** (flexible-SCC)**.** *Let Π be an LCL on regular trees.*
- *If $\Pi = (\Delta, \Sigma, \mathcal{V})$ is a problem on rooted trees, then $\mathsf{flexible}\text{-}\mathsf{SCC}(\cdot)$ maps subsets of Σ to sets of subsets of Σ. Namely, for $\Sigma' \subseteq \Sigma$, $\mathsf{flexible}\text{-}\mathsf{SCC}(\Sigma')$ is obtained as follows: First take the restriction $\Pi' = \Pi \restriction_{\Sigma'}$ of Π to Σ' and construct the automaton $\mathcal{M}_{\Pi'}$ associated with it. Then $\mathsf{flexible}\text{-}\mathsf{SCC}(\Sigma')$ is the set of strongly connected components of $\mathcal{M}_{\Pi'}$ that are path-flexible.*
- *When $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ is a problem on unrooted trees, $\mathsf{flexible}\text{-}\mathsf{SCC}(\cdot)$ maps subsets of $\mathcal{V}$ to sets of elements of $\left(\binom{\Sigma}{2}\right)$. For $\mathcal{V}' \subseteq \mathcal{V}$, $\mathsf{flexible}\text{-}\mathsf{SCC}(\mathcal{V}')$ is constructed as follows: First determine the restriction $\Pi' = \Pi \restriction_{\mathcal{V}'}$ of Π to $\mathcal{V}'$ as well as the automaton $\mathcal{M}_{\Pi'}$ associated with it. Then $\mathsf{flexible}\text{-}\mathsf{SCC}(\mathcal{V}')$ is "morally" the set of strongly connected components of $\mathcal{M}_{\Pi'}$ that are path-flexible, but lifted back to a set over elements of $\left(\binom{\Sigma}{2}\right)$ (instead of elements of $\Sigma \times \Sigma$). Concretely we have that $\mathsf{flexible}\text{-}\mathsf{SCC}(\mathcal{V}')$ is the set that contains every $\mathcal{D} \in \left(\binom{\Sigma}{2}\right)$ for which $U_\mathcal{D} = \{(x, y) \in V(\mathcal{M}_\Pi) \mid \{x, y\} \in \mathcal{D}\}$ is a path-flexible component in $\mathcal{M}_{\Pi'}$.*

Good sequences and depth. With the above we may define so-called *good sequences* that are obtained by applying the two operations trim and flexible-SCC alternatingly.

▶ **Definition 5.7** (Good sequences). *Let Π be an LCL problem on regular trees.*

- *When $\Pi = (\Delta, \Sigma, \mathcal{V})$ is a problem on rooted trees, a* good sequence *is a tuple $s = (\Sigma_1^{\mathsf{R}}, \Sigma_1^{\mathsf{C}}, \ldots, \Sigma_k^{\mathsf{R}})$ of subsets of Σ where:*
  - *$\Sigma_i^{\mathsf{R}} = \mathsf{trim}(\Sigma_{i-1}^{\mathsf{C}})$, where $\Sigma_0^{\mathsf{C}} = \Sigma$.*
  - *$\Sigma_i^{\mathsf{C}} \in \mathsf{flexible\text{-}SCC}(\Sigma_i^{\mathsf{R}})$ is a path-flexible component in the automaton $\mathcal{M}_{\Pi_i}$ associated with $\Pi_i = (\Delta, \Sigma_i^{\mathsf{R}}, \mathcal{V})$.*
- *When $\Pi = (\Delta, \Sigma, \mathcal{V}, \mathcal{E})$ is a problem on unrooted trees, a* good sequence *is a tuple $s = (\mathcal{V}_1, \mathcal{D}_1, \ldots, \mathcal{V}_k)$ alternating elements of $\left(\binom{\Sigma}{\Delta}\right)$ and $\left(\binom{\Sigma}{2}\right)$ where:*
  - *$\mathcal{V}_i$ is obtained by the following procedure:*
    1. *Restrict the problem $\Pi_{i-1} = (\Delta, \Sigma, \mathcal{V}_{i-1}, \mathcal{E})$ to $\mathcal{D}_i$ (where $\mathcal{V}_0 = \mathcal{V}$ and $\mathcal{D}_0 = \left(\binom{\Sigma}{2}\right)$) to obtain a problem $\Pi'_{i-1} = \Pi_{i-1} \upharpoonright_{\mathcal{D}_i}$ with node constraints $\mathcal{V}'_{i-1}$.*
    2. *Apply trim, that is, set $\mathcal{V}_i = \mathsf{trim}(\mathcal{V}'_{i-1})$.*
  - *$\mathcal{D}_i \in \mathsf{flexible\text{-}SCC}(\mathcal{V}_i)$ is a path-flexible component in the automaton $\mathcal{M}_{\Pi_i}$ associated with $\Pi_i = (\Delta, \Sigma, \mathcal{V}_i, \mathcal{E})$.*

In both cases we refer to k as the length *of s.*

Note that the step of performing trim (in both contexts) is deterministic, whereas picking a path-flexible strongly connected component might be a non-deterministic one.

Finally the depth d_Π is defined as the length of the longest good sequence.

▶ **Definition 5.8** (Depth). *Let Π be an LCL problem on regular trees (of either kind). The depth d_Π of Π is the largest integer k for which there is a good sequence for Π of length k. If no good sequence exists, then $d_\Pi = 0$. If there is a good sequence of length k for every k, then $d_\Pi = \infty$.*

6 Super-logarithmic Gap for LCLs on Rooted Regular Trees

In this section, we show that if an LCL problem on rooted regular trees has complexity $\omega(\log n)$, then its complexity is $\Omega(n^{1/k})$ for some $k \in \mathbb{N}_+$ in the randomized online-LOCAL model. Moreover, this complexity coincides with the existing upper bounds in CONGEST and LOCAL [3], providing a tight classification. Formally, we prove the following theorem, restated for the reader's convenience:

▶ **Theorem 2.2.** *Let Π be an LCL problem on rooted regular trees with finite depth $k = d_\Pi > 0$. Then any algorithm $\mathcal{A}$ solving Π in randomized online-LOCAL must have locality $\Omega(n^{1/k})$.*

For the rest of this section, we consider $\Pi = (\Delta, \Sigma, \mathcal{V})$ to be an LCL problem on rooted trees with $d_\Pi = k \in \mathbb{N}_+$. We show that problem Π has locality $\Omega(n^{1/k})$ by constructing a randomized input instance that no locality-$o(n^{1/k})$ randomized online-LOCAL algorithm can solve with high probability. We do this by adapting the previous construction by Balliu et al. [3]. We augment this construction by providing a randomized processing order of nodes. We then show that the labeling produces by the algorithm for a specific set of nodes must be independent of the exact order. Finally, we prove by induction that either the algorithm must fail with probability more than $\frac{1}{n}$, or there exists a good sequence longer than d_Π.

6.1   Lower-Bound Graph Construction

We start by setting constant β to be the smallest natural number such that for each subset of labels $\tilde{\Sigma} \subseteq \Sigma$ and for each label $\sigma \in \tilde{\Sigma} \setminus \mathsf{trim}(\tilde{\Sigma})$, there exists no correct labeling of the complete tree T_β using only labels in $\tilde{\Sigma}$ and having root labeled with σ. We also let t be a constant that we fix later. We now define the parts of the lower bound graph followed by the main lower bound construction:

▶ **Definition 6.1** (Lower bound graph part [3])**.** *Let $s = 4t + 4$.*
- *Let $G_{\mathsf{R},1}$ be the complete rooted tree T_β. We say that all nodes of $G_{\mathsf{R},1}$ are in layer $(\mathsf{R}, 1)$.*
- *For each integer $i \geq 1$, let $G_{\mathsf{C},i}^\circ$ be the tree formed by an s-node directed path $v_1 \leftarrow v_2 \leftarrow \cdots \leftarrow v_s$ such that we append $\Delta - 1$ copies of $G_{\mathsf{R},i}$ as the children of all nodes v_i. We call the path $v_1 \leftarrow v_2 \leftarrow \cdots \leftarrow v_s$ the core path of $G_{\mathsf{C},i}^\circ$, and say that they are in layer (C, i).*
- *Let $G_{\mathsf{C},i}$ be defined like $G_{\mathsf{C},i}^\circ$ except that we append another copy of $G_{\mathsf{R},i}$ as a child of v_s.*
- *For each $i \geq 2$, let $G_{\mathsf{R},i}$ be the tree formed by taking the complete rooted tree T_β and appending Δ copies of $G_{\mathsf{C},i-1}$ to each leaf of T_β. All nodes of T_β are said to be in layer (R, i).*

This construction is visualized in Figure 6.

▶ **Definition 6.2** (Main lower bound graph [3])**.** *We define our main lower bound graph G as the rooted tree formed by the following construction:*
- *Construct rooted trees $G_{\mathsf{C},1}^\circ, G_{\mathsf{C},2}^\circ, \ldots, G_{\mathsf{C},k}^\circ$ 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 $G_{\mathsf{C},i}^\circ$, and let r be the root of $G_{\mathsf{R},k+1}$.*
- *Add 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 $v_s^k \leftarrow r$.*

This construction is visualized in Figure 7.

The nodes of the main lower bound graph G are partitioned into layers by the recursive construction. We order these nodes into the following order:

$$(\mathsf{R}, 1) \prec (\mathsf{C}, 1) \prec (\mathsf{R}, 2) \prec (\mathsf{C}, 2) \prec \cdots \prec (\mathsf{R}, k+1)$$
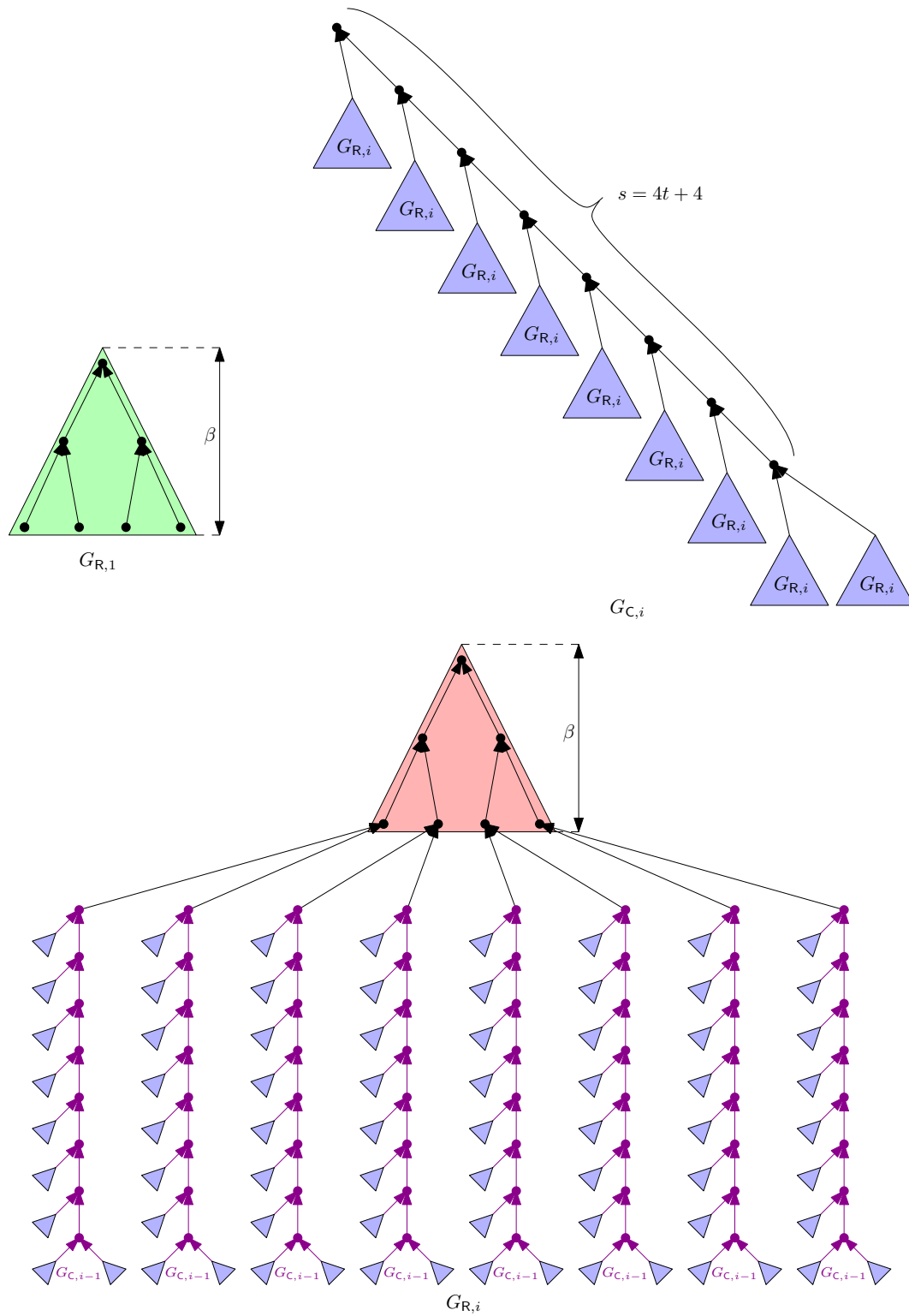
Intuitively, nodes closer to leaves come before nodes closer to the center of the tree.

It is easy to see that each node in G has degree either 0 or Δ, and that the number of nodes in G is $n = O(t^k)$. To show that Π requires locality $\Omega(n^{1/k})$ to solve, it suffices to show that no randomized online-LOCAL algorithm can solve Π on G with locality t or less.

Next, we define a randomized order Z in which the adversary reveals nodes of G to the randomized online-LOCAL algorithm:

▶ **Definition 6.3** (Randomized adversarial sequence and nodes $u_0^i, u_1^i, \ldots, u_{c_i}^i$)**.** *A randomized adversarial sequence Z is a random sequence of all nodes in G, sampled by the followed process.*
1. *Start with an empty sequence Z.*
2. *For each $i \in \{1, \ldots, k\}$, do the following:*
    a. *Let $Q_0^i, Q_1^i, Q_2^i, \ldots, Q_{c_i}^i$ be all core paths in layer (C, i), where Q_0^i is P_i, that is the core path closest to the root.*
    b. *For each $l \in \{1, \ldots, c_i\}$, do the following:*
        i. *Let r_l^i be the node in Q_l^i that is closest to the root of G.*
        ii. *Sample d_l^i from the set $\{2t + 1, 2t + 2\}$ uniformly at random.*
        iii. *Let u_l^i be the node of Q_l^i that is at the distance of d_l^i from r_l^i.*
    c. *Append a uniform random permutation of $u_0^i, u_1^i, \ldots, u_{c_i}^i$ to Z.*
3. *Append all other nodes in any fixed order to Z.*

Figure 6 Visualization of lower bound trees; see Definition 6.1. The tree on top-left shows the structure of $G_{\mathsf{R},1}$, the tree on top right shows $G_{\mathsf{C},i}$ for $i \geq 1$, and the three at the bottom shows $G_{\mathsf{R},i}$ for $i \geq 2$.

Figure 7 The figure shows the lower bound graph G. The orange triangles represent complete trees T_β. Each purple oval represents a core path Q_l^k of some $G_{\mathsf{C},k}$. Each blue pentagon represents copies of $G_{\mathsf{R},k}$. The zoom-in shows the structure of the internal connections of $G_{\mathsf{R},k}$. The turquoise chains are core paths $Q_{l'}^{k-1}$ of $G_{\mathsf{C},k-1}$ where each yellow pentagon represents copies of $G_{\mathsf{R},k-1}$. The topmost red oval represents the core path P_1 (which is same as Q_0^1) of the topmost copy of $G_{\mathsf{C},1}$. Near the zoom-in, the figure shows how u_l^k is chosen in Q_l^k from r_l^k and d_l^k. See Definitions 6.2 and 6.3 for more information.

The randomized adversarial sequence Z therefore also defines the nodes $u_0^i, u_1^i, \ldots, u_{c_i}^i$.

Note that for either choice of $d_l^i \in \{2t+1, 2t+2\}$, the distance of u_l^i from either end of the core path it belongs to is at least $2t+1$. This is the reason behind the choice of $s = 4t+4$. In particular, this ensures that any locality-t randomized online-LOCAL algorithm $\mathcal{A}$ must label nodes u_l^i for $i \in \{1, 2, \ldots, k\}$ and $l \in \{0, 1, \ldots, c_i\}$ independently of the adversarial sequence Z:

▶ **Lemma 6.4** (Independence lemma). *For $i \in \{1, 2, \ldots k\}$, the labelling of the nodes $u_0^i, u_1^i, \ldots, u_{c_i}^i$ produced by any randomized online-LOCAL algorithm $\mathcal{A}$ with locality t is independent of the randomness of the randomized adversarial sequence Z.*

Proof. For any i, the t-neighborhoods of $u_0^i, u_1^i, \ldots, u_{c_i}^i$ are identical and disjoint. For $i_1 \neq i_2$, the t-neighborhoods of $u_{j_1}^{i_1}$ and $u_{j_2}^{i_2}$ are also disjoint as their lowest common ancestor is at a distance of at least $2t$ from one of them. Therefore, for any u_l^i, its t-neighborhood is unlabeled for all randomized adversarial sequences Z. Since the t-neighborhoods of $u_0^i, u_1^i, \ldots, u_{c_i}^i$ are the identical, disjoint and unlabeled at the time of revealing, the output distribution of $u_0^i, u_1^i, \ldots, u_{c_i}^i$ produced by $\mathcal{A}$ is independent of the randomized adversarial sequence Z for all $i \in \{1, 2, \ldots k\}$. ◀

Next, we define a sequence of subsets of the nodes in G. This definition is very similar to the subsets defined in previous works of Balliu et al. [3].

▶ **Definition 6.5** (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-β is isomorphic to T_β.*
- *For $2 \leq i \leq k+1$, 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-β is isomorphic to T_β and contains only nodes in set $S'_{\mathsf{C},i-1}$.*
- *For $1 \leq i \leq k$, define $S'_{\mathsf{C},i}$ as the set of nodes v in G meeting one of the following conditions.*
  - *v is in one of these layers: $(\mathsf{R}, i+1), (\mathsf{C}, i+1), (\mathsf{R}, i+2), \ldots, (\mathsf{R}, k+1)$.*
  - *$v \in P_i$ is either the node u_0^i or its descendant in layer (C, i).*
  - *$v \notin P_i$ is either the node u_l^i or an ancestor of it in layer (C, i), for some $l \neq 0$.*

The intuition behind the definition is the following: $S'_{\mathsf{R},1}$ basically consists of all nodes of the tree G apart from some layer $(\mathsf{R}, 1)$ nodes which are very close to leaves. $S'_{\mathsf{C},i}$ contains the nodes which are descendants of u_0^i but also an ancestor of some node u_l^i. It is evident from the definition that $S'_{\mathsf{C},i}$ contains nodes only from layer (C, i) or above. And finally, $S'_{\mathsf{R},i}$ contains of only nodes in layer (R, i) or above. In particular, it contains the nodes, and their immediate children, that are roots of subgraphs isomorphic to T_β that are fully in layer (R, i).

We prove some basic properties of the sets in Definition 6.5. The proof is again similar to that in the previous works of Balliu et al. [3], we restate it for the convenience of the reader:

▶ **Lemma 6.6** (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 the 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-β neighborhood of v is isomorphic to T_β 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, which is the root of $G_{\mathsf{R},k+1}$. The subgraph induced by r and its descendants within radius-β neighborhood of r is isomorphic to T_β 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 $r \in S'_{\mathsf{R},k+1}$.   ◄

It is also evident that the children of the nodes in $S'_{\mathsf{C},i}$ are also contained in $S'_{\mathsf{R},i}$, for all $i \in \{1, 2, \ldots, k\}$:

▶ **Observation 6.7.** *If v is a child of some node in $S'_{\mathsf{C},i}$, then $v \in S'_{\mathsf{R},i}$, for all $i \in \{1, 2, \ldots, k\}$.*

Furthermore, due to the recursive structure of the input instance G, we get the following result:

▶ **Observation 6.8.** *For each $i \in \{1, 2, \ldots, k\}$, for all $v \in S'_{\mathsf{C},i}$ there exists a path P from u^i_l (for some l) to u^i_0, such that P contains v. Moreover, all nodes of P, and their children, belong to $S'_{\mathsf{R},i}$.*

6.2   Running Algorithm on the Lower-Bound Graph

We are now ready to show that no locality-t randomized online-LOCAL algorithm can solve problem Π on graph G and processing order Z with high probability. For contradiction, we assume that such algorithm $\mathcal{A}$ exists. We then show that algorithm $\mathcal{A}$ fails to find a proper labeling for lower bound graph G with randomized input sequence Z with probability larger than $\frac{1}{n}$. In particular, this shows that $\mathcal{A}$ does not work with high probability. Hence the locality any randomized online-LOCAL algorithm solving Π must be strictly larger than $t = \Omega(n^{1/k})$.

We start by proving the following result that holds for every valid output labelling:

▶ **Lemma 6.9.** *Let L be a valid labeling of the nodes according to Π. For each $i \in \{1, 2, \ldots, k\}$, let $\tilde{\Sigma}^L_i \subseteq \Sigma$ be any subset of labels such that for each node $v \in S'_{\mathsf{R},i}$, the corresponding label $L(v)$ is in $\tilde{\Sigma}^L_i$. Let $\mathcal{M}$ be the automaton associated with the restricted problem $\Pi \restriction_{\tilde{\Sigma}^L_i}$. Then for all nodes $u', u'' \in S'_{\mathsf{C},i}$ such that $u' \leftarrow u''$ is an edge in G, edge $L(u') \to L(u'')$ exists in $\mathcal{M}$.*

Note that the result talks about existence of the edge $L(u') \to L(u'')$ in the automaton of the restricted problem $\Pi \restriction_{\tilde{\Sigma}^L_i}$. The result would trivially hold for the automaton of the original problem Π.

Proof. By Observation 6.8, there exists path $P = (u^i_0 = v_0 \leftarrow v_1 \leftarrow \cdots \leftarrow v_\alpha = u^i_l)$ for some l containing node u'', and hence also u'. Moreover, all nodes of P and their children belong to set $S'_{\mathsf{R},i}$. This implies that the labels of all nodes of P, and labels of their children, belong to set $\tilde{\Sigma}^L_i$.

Let $\sigma \in \tilde{\Sigma}^L_i$ be the label of u', and let $\sigma_1, \sigma_2, \ldots, \sigma_\Delta$ be the labels of its children. As the children of u' belong to $S'_{\mathsf{R},i}$, all σ_j belong to $\tilde{\Sigma}^L_i$. Therefore $(\sigma : \sigma_1, \ldots, \sigma_\Delta)$ must be a valid configuration in the restricted problem $\Pi \restriction_{\tilde{\Sigma}^L_i}$. This then implies that all $\sigma \to \sigma_1, \ldots, \sigma \to \sigma_\Delta$ are edges in $\mathcal{M}$. As the label of node u'' is one of $\sigma_1, \ldots, \sigma_\Delta$, edge $\sigma = L(u') \to L(u'') = \sigma_j$ belongs to $\mathcal{M}$.   ◄

We now restrict our attention to a particular run of algorithm $\mathcal{A}$ on the randomized sequence Z on graph G. We start by defining the set of labels the algorithm uses to label nodes $u^i_0, u^i_1, u^i_2, \ldots, u^i_{c_i}$ (see Definition 6.3):

► **Definition 6.10.** *For $i \in \{1, 2, \ldots k\}$, let $\Sigma'_i \subseteq \Sigma$ be the set of labels used by $\mathcal{A}$ to label the nodes $u^i_0, u^i_1, u^i_2, \ldots, u^i_{c_i}$.*

Note that the set Σ'_i, $i \in \{1, 2, \ldots, k\}$ depends on a particular run of $\mathcal{A}$, and can change with different runs of $\mathcal{A}$. Whenever we talk about Σ'_i, $i \in \{1, 2, \ldots, k\}$, we consider a fixed run of $\mathcal{A}$ and analyze it.

We now prove the following result on the structure of these sets of labels:

► **Lemma 6.11.** *Let j be an integer in $\{0, 1, 2, \ldots, k\}$, and let $\Sigma'_1, \Sigma'_2, \ldots, \Sigma'_k$ be the sets of nodes $\mathcal{A}$ used to label nodes $u^i_0, u^i_1, u^i_2, \ldots, u^i_{c_i}$. If there now exists sets $\Sigma = \Sigma''_0, \Sigma''_1, \Sigma''_2, \ldots, \Sigma''_j$ satisfying $\Sigma''_i \in$ flexible-SCC$(\mathsf{trim}(\Sigma''_{i-1}))$ and $\Sigma'_i \subseteq \Sigma''_i$ for all $i \in \{1, 2, \ldots j\}$, then the following must hold for the full labeling L produced by $\mathcal{A}$:*

- **Induction hypothesis for $S'_{\mathsf{R},i}$:** *for all $i \in \{1, 2, \ldots, j, j+1\}$ and $v \in S'_{\mathsf{R},i}$ we have $L(v) \in \mathsf{trim}(\Sigma''_{i-1})$.*
- **Induction hypothesis for $S'_{\mathsf{C},i}$:** *for all $i \in \{1, 2, \ldots, j\}$ and $v \in S'_{\mathsf{C},i}$ we have $L(v) \in \Sigma''_i$.*

The idea behind defining $\Sigma''_0, \Sigma''_1, \ldots, \Sigma''_j$ is to ensure that $(\Sigma^{\mathsf{R}}_1 = \mathsf{trim}(\Sigma''_0), \Sigma^{\mathsf{C}}_1 = \Sigma''_1, \Sigma^{\mathsf{R}}_2 = \mathsf{trim}(\Sigma''_1), \ldots, \Sigma^{\mathsf{C}}_j = \Sigma''_j, \Sigma^{\mathsf{R}}_{j+1} = \mathsf{trim}(\Sigma''_{j-1}))$ forms a good sequence and the corresponding nodes follow some specific structure.

Proof. We prove this lemma with induction on i for a fixed $j \leq k$:

Base Case: induction hypothesis for $S'_{\mathsf{R},1}$. By definition, we have $\Sigma''_0 = \Sigma$. Moreover, by definition, for every node $v \in S'_{\mathsf{R},1}$, the radius-β descendants of v are isomorphic to T_β. For any valid assignments, the labels of v and its radius-β descendants come from the set $\Sigma = \Sigma''_0$. By definition of β, this must mean that the label of v must come from $\mathsf{trim}(\Sigma) = \mathsf{trim}(\Sigma''_0)$. Therefore, for all $v \in S'_{\mathsf{R},i}$, we have $L(v) \in \mathsf{trim}(\Sigma''_{i-1})$ for $i = 1$.

Induction: induction hypothesis for $S'_{\mathsf{C},i}$. We assume that for some $1 \leq i \leq j$, the induction hypothesis for $S'_{\mathsf{R},i}$ holds true, i.e. for all $v \in S'_{\mathsf{R},i}$ we have $L(v) \in \mathsf{trim}(\Sigma''_{i-1})$. Let v be any node in $S'_{\mathsf{C},i}$. By Observation 6.8, there exists a path P from u^i_l for some l to u^i_0 containing v where all nodes of P and the children of all such nodes belong to $S'_{\mathsf{R},i}$. Let the path P be $u^i_0 = v_0 \leftarrow v_1 \leftarrow \cdots \leftarrow v_\alpha = u^i_l$. Then v_y and its children are in $S'_{\mathsf{R},i}$, for all $y \in \{0, 1, \ldots, \alpha\}$. Moreover, by Lemma 6.9, for all $y \in \{0, 1, \ldots, \alpha - 1\}$, $L(v_y) \to L(v_{y+1})$ is a directed edge in $\mathcal{M}$, where $\mathcal{M}$ is the automaton associated with the restricted problem $\Pi \restriction_{\mathsf{trim}(\Sigma''_{i-1})}$.

Now, $L(u^i_0), L(u^i_l) \in \Sigma'_i$ and $\Sigma'_i \subseteq \Sigma''_i$ where Σ''_i is a flexible strongly connected component in $\mathcal{M}$ by the definition of Σ''_i. Since $\mathcal{M}$ has the directed edge $L(v_y) \to L(v_{y-1})$ for all $y \in \{0, 1, \ldots, \alpha - 1\}$, $L(u^i_0) = L(v_0) \to L(v_1) \to \cdots \to L(v) \to \cdots \to L(v_\alpha) = L(u^i_l)$ is a walk in $\mathcal{M}$ that starts and ends in the strongly connected component Σ''_i. This must mean $L(v) \in \Sigma''_i$, for all $v \in S'_{\mathsf{C},i}$.

Induction: induction hypothesis for $S'_{\mathsf{R},i}$. We assume that for some $2 \leq i \leq j + 1$, the induction hypothesis for $S'_{\mathsf{C},i-1}$ holds true, that is for all $v \in S'_{\mathsf{C},i-1}$ we have $L(v) \in \Sigma''_{i-1}$. Moreover, by definition, each node $v \in S'_{\mathsf{R},i}$ is a root of a subtree isomorphic to T_β such that all nodes reside in $S'_{\mathsf{C},i-1}$, each of which has a label from set Σ''_{i-1} by induction hypothesis for $S'_{\mathsf{C},i-1}$. Hence, the label of v must be from $\mathsf{trim}(\Sigma''_{i-1})$, implying for all $v \in S'_{\mathsf{R},i}$, we have $L(v) \in \mathsf{trim}(\Sigma''_{i-1})$. ◄

We can now combine this result with the independence lemma (Lemma 6.4) to provide a lower bound for the failure probability of $\mathcal{A}$ on the randomized adversarial sequence Z, even after fixing the labels produced by $\mathcal{A}$ on nodes $u^i_0, u^i_1, \ldots, u^i_{c_i}$ for all $i \in \{1, 2, \ldots, k\}$:

▶ **Lemma 6.12.** *For all choices of $\Sigma_1', \Sigma_2', \ldots, \Sigma_k'$, algorithm $\mathcal{A}$ fails to produce a correct labelling for the randomized adversarial sequence Z of G with probability more than $\frac{1}{n}$.*

Proof. Let j be the largest integer in $\{0, 1, 2, \ldots, k\}$ such that there exists $\Sigma_0'', \Sigma_1'', \Sigma_2'', \ldots, \Sigma_j''$ satisfying the conditions of Lemma 6.11, that is:

- $\Sigma_0'' = \Sigma$,
- for all $i \in \{1, 2, \ldots j\}$, $\Sigma_i'' \in \mathsf{flexible\text{-}SCC}(\mathsf{trim}(\Sigma_{i-1}''))$ and $\Sigma_i' \subseteq \Sigma_i''$.

Note that such j always exists as these conditions trivially hold true for $j = 0$.

We now do a case analysis on j and show that in each case the algorithm fails to produce a correct labeling with probability more than $\frac{1}{n}$:

Case I: $j < k$. This means that there does not exist $\Sigma_{j+1}'' \supseteq \Sigma_{j+1}'$ such that $\Sigma_{j+1}'' \in \mathsf{flexible\text{-}SCC}(\mathsf{trim}(\Sigma_j''))$. Let $\mathcal{M}$ be the automaton associated with the restricted problem $\Pi \restriction_{\mathsf{trim}(\Sigma_j'')}$. Now one of the following must hold:

(a) $\Sigma_{j+1}' \not\subseteq \mathsf{trim}(\Sigma_j'')$,

(b) $\Sigma_{j+1}' \subseteq \mathsf{trim}(\Sigma_j'')$ but Σ_{j+1}' does not form a strongly connected component of $\mathcal{M}$,

(c) $\Sigma_{j+1}' \subseteq \mathsf{trim}(\Sigma_j'')$ and Σ_{j+1}' forms an inflexible strongly connected component of $\mathcal{M}$.

Observe that if all three are false, then we could take the strongly connected component containing the elements of Σ_{j+1}' in $\mathcal{M}$ as Σ_{j+1}''. This would contradict our assumption that there does not exist $\Sigma_{j+1}'' \in \mathsf{flexible\text{-}SCC}(\mathsf{trim}(\Sigma_j''))$.

We now show that for all three cases, the probability that $\mathcal{A}$ eventually fails to output a correct solution is more than $\frac{1}{n}$:

Case I(a): $\Sigma_{j+1}' \not\subseteq \mathsf{trim}(\Sigma_j'')$. There must be some l such that label $L(u_l^{j+1})$ generated by $\mathcal{A}$ for node u_l^{j+1} is not in set $\mathsf{trim}(\Sigma_j'')$. By Lemma 6.11, we know that in any valid labelling we have $L(v) \in \mathsf{trim}(\Sigma_j'')$ for every node $v \in S_{\mathsf{R},j+1}'$. However, this is violated in this particular run as $u_l^{j+1} \in S_{\mathsf{C},j+1}' \subseteq S_{\mathsf{R},j+1}'$ but $L(u_l^{j+1}) \notin \mathsf{trim}(\Sigma_j'')$. Therefore, irrespective of the other output labels, $\mathcal{A}$ must fail to generate a valid labelling for G. Thus, the probability that $\mathcal{A}$ eventually fails is indeed 1.

Case I(b): Σ_{j+1}' does not from a strongly connected component of $\mathcal{M}$. There must be two labels $\sigma_1, \sigma_2 \in \Sigma_{j+1}'$ such that σ_1 and σ_2 are not in the same strongly connected component in $\mathcal{M}$. This means that at least one of walks $\sigma_1 \rightsquigarrow \sigma_2$ and $\sigma_2 \rightsquigarrow \sigma_1$ does not exist in $\mathcal{M}$. Without loss of generality, we may assume that walk $\sigma_1 \rightsquigarrow \sigma_2$ is missing from $\mathcal{M}$. Algorithm $\mathcal{A}$ uses label σ_1 to label at least one of nodes $u_0^{j+1}, u_1^{j+1}, \ldots, u_{c_{j+1}}^{j+1}$. As the choices of the algorithm are independent of the input sequence Z by Lemma 6.4, and the sequence Z contains a uniformly random permutation of the nodes $u_0^{j+1}, u_1^{j+1}, \ldots, u_{c_{j+1}}^{j+1}$, the probability that $\mathcal{A}$ uses label σ_1 on node u_0^{j+1} is at least $\frac{1}{c_{j+1}+1} > \frac{1}{n}$.

We now show that algorithm $\mathcal{A}$ is bound to fail in this case. As label $\sigma_2 \in \Sigma_{j+1}'$, algorithm $\mathcal{A}$ must use it to label at least one of the nodes $u_1^{j+1}, \ldots, u_{c_{j+1}}^{j+1}$. Let that node be u_l^{j+1}. Consider path $P = u_0^{j+1} \leftarrow v_1 \leftarrow \cdots \leftarrow v_\alpha \leftarrow u_l^{j+1}$. Note that all nodes of P, and their children, belong to set $S_{\mathsf{R},j}'$. Lemma 6.9 now gives us that walk $\sigma_1 = L(u_0^{j+1}) \rightarrow L(v_1) \rightarrow \cdots \rightarrow L(v_\alpha) \rightarrow L(u_l^{j+1}) = \sigma_2$ exists in $\mathcal{M}$, which contradicts our assumption that σ_1 and σ_2 don't belong to the same strongly connect component.

As this happens with probability at least $\frac{1}{c_{j+1}+1} > \frac{1}{n}$, algorithm $\mathcal{A}$ must fail with at least the same probability.

Case I(c): Σ_{j+1}' forms an inflexible strongly connected component of $\mathcal{M}$. Similar to the previous case, there must exist a walk $L(u_0^{j+1}) \rightsquigarrow L(u_l^{j+1})$ in $\mathcal{M}$ for each $l \in \{1, 2, \ldots, c_{j+1}\}$. For any pair of labels $(L(u_0^{j+1}), L(u_l^{j+1}))$ which are in an inflexible strongly

connected component in $\mathcal{M}$, automaton $\mathcal{M}$ cannot contain walks $L(u_0^{j+1}) \rightsquigarrow L(u_l^{j+1})$ of length both d and $d+1$; otherwise the component containing the pair would be flexible. However, the distance between nodes u_0^{j+1} and u_l^{j+1} is even with probability $\frac{1}{2}$ and odd with probability $\frac{1}{2}$ by construction of sequence Z. As the labels chosen by $\mathcal{A}$ for nodes u_0^{j+1} and u_l^{j+1} are independent of sequence Z by Lemma 6.4, it must fail to choose a label with the correct parity with probability at least $\frac{1}{2}$. Hence the algorithm must fail with probability at least $\frac{1}{2} > \frac{1}{n}$.

This proves that if $j < k$, the probability that the algorithm eventually fails is more than $\frac{1}{n}$. The other case $j = k$ is relatively simpler:

Case II: $j = k$. Let v be a node in $S'_{\mathsf{R},k+1}$; note that $S'_{\mathsf{R},k+1} \neq \emptyset$ by Lemma 6.6. Due to Lemma 6.11, the label of node v must be in $\mathsf{trim}(\Sigma''_k)$ in any correct labelling of the graph, which implies that $\mathsf{trim}(\Sigma''_k) \neq \emptyset$. However, this cannot be the case since

$$\left(\mathsf{trim}(\Sigma''_0), \Sigma''_1, \mathsf{trim}(\Sigma''_1), \dots, \mathsf{trim}(\Sigma''_{k-1}), \Sigma''_k, \mathsf{trim}(\Sigma''_k) \right)$$

would be a good sequence of length $k+1$, contradicting $d_\pi = k$. Therefore, algorithm $\mathcal{A}$ must eventually fail (i.e. with probability 1).

Since the algorithm $\mathcal{A}$ fails with probability more than $\frac{1}{n}$ in all cases, the failure probability of $\mathcal{A}$ in total must be more than $\frac{1}{n}$. ◀

The main result now follows as a simple corollary of the previous lemma:

Proof of Theorem 2.2. Let $\mathcal{A}$ be a randomized online-LOCAL algorithm solving LCL problem Π on rooted regular trees having depth $d_\Pi = k \in \mathbb{N}_+$ with locality $t = o(n^{1/k})$. Then by Lemma 6.12 algorithm $\mathcal{A}$ must fail with probability more than $\frac{1}{n}$. Hence any randomized online-LOCAL algorithm solving Π with high probability must have locality $\Omega(n^{1/k})$. ◀

7 Lower Bounds in the Super-Logarithmic Region in Unrooted Regular Trees

In this section, we restate and prove the following result:

▶ **Theorem 2.3.** *Let Π be an LCL problem on unrooted regular trees with finite depth $k = d_\Pi > 0$. Then any randomized online-LOCAL algorithm for Π has locality $\Omega(n^{1/k})$.*

Let us thus fix such a problem Π and its depth k. We show that the existence of an $o(n^{1/k})$ algorithm which solves Π will force a contradiction. The procedure is analogous to the rooted case.

We pick γ to be the smallest integer satisfying: for each set $S \subseteq \mathcal{V}$ and each $C \in S \backslash \mathsf{trim}(S)$, there exists no correct labeling of T_γ^* where the node configuration of the root node is C and the node configurations of the remaining Δ-degree nodes are in S. By definition of trim, such a γ will always exist. We note that in any algorithm, γ is a constant, depending only on the underlying LCL problem Π.

▶ **Definition 7.1** (Lower bound graphs). *Let t be any positive integer, and choose $s = 4t + 4$.*
- *$G_{\mathsf{R},1}$ is the rooted tree T_γ, and $G_{\mathsf{R},1}^*$ is the rooted tree T_γ^*. We say all nodes in $G_{\mathsf{R},1}$ or $G_{\mathsf{R},1}^*$ are in layer $(\mathsf{R},1)$.*
- *For each integer $i \geq 1$, $G_{\mathsf{C},i}$ is constructed as follows. Begin with an s-node path $(v_1, v_2, \dots, v_s)$, 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. We say nodes $v_1, v_2, \dots, v_s$ are in layer (C,i).*

- *For each integer $i \geq 2$, $G_{\mathsf{R},i}$ is constructed as follows. Begin with a rooted tree T_γ. Append $\Delta - 1$ copies of $G_{\mathsf{C},i-1}$ to each leaf in T_γ. We say all nodes in T_γ are in layer (R, i). The rooted tree $G_{\mathsf{R},i}^*$ is defined analogously by replacing T_γ with T_γ^* in the construction.*

The choice of $s = 4t + 4$ will be motivated later. Note that although the trees $G_{\mathsf{R},i}, G_{\mathsf{R},i}^*$, and $G_{\mathsf{C},i}$ are defined as rooted trees, we can treat them as unrooted trees. We define our main lower bound graph to be $G = G_{\mathsf{R},k+1}^*$. We will use this graph to show that if $d_\Pi = k$, there can be no algorithm faster than $\Omega(n^{1/k})$ solving Π on G.

From the construction, it can easily be seen that $G = G_{\mathsf{R},k+1}^*$ has $O(t^k)$ nodes. So to show that Π requires a locality of $\Omega(n^{1/k})$, it suffices to show that solving Π requires locality of at least t on G.

We now define a sequence of subsets of nodes in G.

▶ **Definition 7.2** (Subsets of nodes in G: $S_{\mathsf{R},i}, S_{\mathsf{C},i}, U_i$). *We define the following sets:*
- *Let $S_{\mathsf{R},1}$ consist of all nodes in G with their radius-γ neighborhood isomorphic to T_γ.*
- *For each path $(v_1, v_2, \ldots, v_s)$ in layer (C, i), choose one element of the set $\{v_{2t+1}, v_{2t+2}\}$ uniformly at random. Define the set of all such elements as U_i.*
- *For $1 \leq i \leq k$, construct $S_{\mathsf{C},i}$ as follows: Initialize $S_{\mathsf{C},i}$ as U_i. For any $u_j, u_k \in U_i$, there exists a path joining them, $P = (u_j, s_1, s_2, \ldots, u_k)$. Add each node s_l in such a path to $S_{c,i}$.*
- *For $2 \leq i \leq k + 1$, let $S_{\mathsf{R},i}$ consist of all nodes in $S_{\mathsf{C},i-1}$ with radius-γ neighborhood contained entirely within $S_{\mathsf{C},i-1}$ and isomorphic to T_γ.*

Intuitively, $S_{\mathsf{C},i}$ consists of all nodes in U_i and all nodes "above" any node in U_i. The layers (R, i) and (C, i) form a partition of nodes in G; we impose an order on these layers for subsequent discussion: let $(\mathsf{R}, 1) \preceq (\mathsf{C}, 1) \preceq \cdots \preceq (\mathsf{R}, k) \preceq (\mathsf{C}, k) \preceq (\mathsf{C}, k + 1)$. For example, "layer (R, i) or higher" refers to layers $(\mathsf{R}, i), (\mathsf{C}, i), \ldots, (\mathsf{R}, k + 1)$.

We now prove some properties of the sets $S_{\mathsf{R},i}$ and $S_{\mathsf{C},i}$.

▶ **Lemma 7.3** (Subset containment). *It holds that $S_{\mathsf{R},1} \supseteq S_{\mathsf{C},1} \supseteq S_{\mathsf{R},2} \supseteq \cdots \supseteq S_{\mathsf{R},k+1}$.*

Proof. Each $S_{\mathsf{C},i} \supseteq S_{\mathsf{R},i+1}$ follows immediately from the definition, as we require each node in $S_{\mathsf{R},i+1}$ to be in $S_{\mathsf{C},i}$ in the construction.

To see that $S_{\mathsf{R},i} \supseteq S_{\mathsf{C},i}$, observe that $S_{\mathsf{R},i}$ contains all nodes in layers (R, i) and higher, for all $i \geq 2$: each node in layer (R, i) or higher is a distance at least $2t + 1 \gg \gamma$ from any node in U_i, and therefore its radius-τ neighborhood is contained entirely within $S_{\mathsf{C},i}$. It can easily be seen from the construction that this neighborhood is isomorphic to T_γ. So we have that $S_{\mathsf{R},i} \supseteq S_{\mathsf{C},i}$. ◀

We now define our randomized adversarial order. Given a randomized online-LOCAL algorithm $\mathcal{A}$ which has locality t, we will show that if $\mathcal{A}$ is given nodes in this order, it will fail with probability asymptotically greater than $\frac{1}{n}$.

▶ **Definition 7.4** (Randomized adversarial order). *We define a randomized adversarial order of nodes as follows: begin with an empty sequence $\mathcal{S}$.*
1. *For i from 1 to k, append the nodes of U_i to $\mathcal{S}$ in any randomized order.*
2. *Add all the remaining nodes (those in $V(G) \setminus \bigcup_i U_i$) to $\mathcal{S}$ in any order.*
Also, define $\mathcal{S}^U$ to be the subsequence of $\mathcal{S}$ consisting only of nodes in some U_i. Note that $\mathcal{S}^U$ is a prefix of $\mathcal{S}$.

Now, we argue that when labeling the nodes of $\bigcup_i U_i$, the label which $\mathcal{A}$ assigns the ith node of $\mathcal{S}^U$ is independent of the randomness in $\mathcal{S}^U$'s order.

▶ **Lemma 7.5** (Independence of labels of nodes in each U_i). *For $i \in \{1, 2, \ldots, k\}$, the labels assigned by $\mathcal{A}$ to the nodes of U_i are independent of the way the nodes of U_i are ordered in $\mathcal{S}^U$.*

Proof. Let $U_i = \{u_1^i, u_2^i, \ldots, u_i^{c_i}\}$. The radius-$t$ neighborhoods of any u_p^i and u_q^i are identical and disjoint: since each node of U_i is chosen as the $2t + 1$ or $2t + 2$th node in a path of length $4t + 4$, its radius-t neighborhood cannot see into any other such paths, and because $U_i \subseteq S_{\mathsf{C},i} \subseteq S_{\mathsf{R},i}$ for all $i \in \{1, 2, \ldots, k\}$, each u_p^i's radius-t neighborhood is isomorphic to T_γ. Since no nodes within the radius-t neighborhood of any u_p^i are labeled when u_p^i is revealed to the algorithm, the algorithm will be unable to distinguish between u_p^i and u_q^i for any $p \neq q$. For $i_1 \neq i_2$, the radius-t neighborhoods of any two $u_{p_1}^{i_1}$ and $u_{p_2}^{i_2}$ are also disjoint as their nearest common ancestor is a distance of at least $2t$ from at least one of them. So every u_p^i's radius-t neighborhood is identical, disjoint, unlabeled at the time of revealing to the algorithm, for all orders of $\mathcal{S}$. Thus, the algorithm is unable to distinguish between the nodes of U_i, and thus, the labeling of U_i is independent of the randomness of the sequence $\mathcal{S}$, for all $i \in \{1, 2, \ldots, k\}$. ◀

Note that this also means the orientation—in particular, the half-edge labels which connect a given u_j^i to the rest of the nodes in a path of layer (C, i)—is indistinguishable to the algorithm.

We now have all necessary lemmas and definitions for the proof of Theorem 2.3.

Proof outline. We provide a high-level overview of the proof before discussing it with more technicality. We first prove by induction that all edges in $S_{\mathsf{C},i}$ must be labeled with labels in a flexible-SCC of the labels used in $S_{\mathsf{R},i}$, and that all nodes in $S_{\mathsf{R},i}$ must be labeled with labels in the trim of the labels used in $S_{\mathsf{R},i-1}$. This forces the labels of these sets to be contained within a corresponding set of our "good sequence"—that is, the labels of the nodes in $S_{\mathsf{R},i}$ must belong to $\mathcal{V}_i$, and the labels of $S_{\mathsf{C},i}$ must belong to $\mathcal{D}_i$. But this requires that $S_{\mathsf{R},k+1}$ be labeled with elements of some $\mathcal{V}_{k+1}$— which, because the depth of Π is $d_\Pi = k$ by assumption, is empty. This allows us to conclude that $\mathcal{A}$ is unable to generate a valid labeling of G with a sufficiently high probability of success.

Throughout the proof, we use the following notation. For each node $v \in G$, we define $\mathcal{V}_v^{\mathcal{A}}$ as the set of all possible node configurations of v that can occur in any run of $\mathcal{A}$. Similarly, for any two edges e_1 and e_2 incident to a node v, we define $\mathcal{D}_{v,e_1,e_2}^{\mathcal{A}}$ to be the set of all multisets $\{a, b\} \in \left(\binom{\Sigma}{2}\right)$ such that $\{a, b\}$ is a possible outcome of labeling the two half-edges (v, e_1) and (v, e_2) when we run $\mathcal{A}$.

We now define our induction hypotheses.

■ **Induction hypothesis for $S_{\mathsf{R},i}$.** For each $1 \leq i \leq k+1$, any node $v \in S_{\mathsf{R},i}$ satisfies $\mathcal{V}_v^{\mathcal{A}} \subseteq \mathcal{V}_i$.

■ **Induction hypothesis for $S_{\mathsf{C},i}$.** For each $1 \leq i \leq k$, for each 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 $\mathcal{D}_{v,e_1,e_2}^{\mathcal{A}} \subseteq \mathcal{D}_i$.

Base case: $S_{\mathsf{R},1}$. We show that the induction hypothesis for $S_{\mathsf{R},1}$ holds. We recall that γ was chosen to be the smallest integer such that for each set $S \subseteq \mathcal{V}$ and each $C \in S \setminus \mathsf{trim}(S)$, there exists no correct labeling of T_γ^* where the node configuration of the root node is C and the node configurations of the remaining Δ-degree nodes are in S. Now consider any node $v \in S_{\mathsf{R},1}$: by construction, the radius-γ neighborhood of v is isomorphic to T_γ^*. By setting $S = \mathcal{V}$, we can conclude there is no correct labeling of G such that the labeling chosen for v

is in $\mathcal{V} \setminus \text{trim}(\mathcal{V}) = \mathcal{V} \setminus \mathcal{V}_1$. So for any run of $\mathcal{A}$, the node configuration of v must be contained within $\mathcal{V}_i$—that is, $\mathcal{V}_v^{\mathcal{A}} \subseteq \mathcal{V}_i$. ◀

▶ **Lemma 7.6** (Inductive step: $S_{\mathsf{R},i}$). *For $2 \leq i \leq k$, if the induction hypotheses for $S_{\mathsf{R},i-1}$ and $S_{\mathsf{C},i-1}$ hold, then the induction hypothesis for $S_{\mathsf{R},i}$ holds.*

Proof. Consider any $v \in S_{\mathsf{R},i}$. We recall that by the construction of $S_{\mathsf{R},i}$, v's radius-γ neighborhood is isomorphic to T_γ^*. We note that the Δ-degree nodes in this T_γ^* are the nodes within the radius-$(\gamma - 1)$ neighborhood of v. Consider any node u within this neighborhood. Since $u \in S_{\mathsf{C},i-1} \subseteq S_{\mathsf{R},i-1}$, we can apply the inductive hypothesis for $S_{\mathsf{R},i-1}$ to conclude that $\mathcal{V}_u^{\mathcal{A}} \subseteq \mathcal{V}_{i-1}$. Further, since all neighbors of u are in $S_{\mathsf{C},i-1}$, we can apply the inductive hypothesis on all edges e_1, e_2 incident to u, and conclude that $\mathcal{D}_{u,e_1,e_2}^{\mathcal{A}} \subseteq \mathcal{D}_i$. From this, it is clear that $\mathcal{V}_u^{\mathcal{A}} \subseteq \mathcal{S}$, where $\mathcal{S}$ consists of node configurations $V \in \mathcal{V}_{i-1}$ such that for all $\{a, b\} \in \left(\binom{\Sigma}{2}\right)$ such that $\{a, b\} \subseteq V$, $\{a, b\} \in \mathcal{D}_i$. (Intuitively, this is the restriction of $\mathcal{V}_{\rangle -\infty}$ to node configurations which have every pair of edge labels $\{a, b\} \in V$ contained in $\mathcal{D}_i$.)

So all Δ-degree nodes in the T_γ^* rooted at v have labels in $\mathcal{S}$. So by the definition of γ, no valid labeling of this T_γ^* can assign v a node configuration from $\mathcal{S} \setminus \text{trim}(\mathcal{S})$. Thus, $\mathcal{V}_v^{\mathcal{A}} \subseteq \text{trim}(\mathcal{S}) = \mathcal{V}_i$. ◀

We now prove the inductive hypothesis for $\mathcal{D}_i$. Note that this step is significantly more involved.

▶ **Lemma 7.7** (Inductive step: $S_{\mathsf{C},i}$). *For $1 \leq i \leq k$, if the induction hypothesis holds for $S_{\mathsf{R},i}$, then the induction hypothesis for $S_{\mathsf{C},i}$ holds.*

Proof. We first show that all $u_1, u_2 \in U_i$ must belong to the same path-flexible strongly connected component. Then, we show that all $v \in S_{\mathsf{C},i}$ must also belong to this flexible-SCC. Finally, we argue that this flexible-SCC is indeed $\mathcal{D}_i$.

Let $u_1, u_2 \in U_i$. Recalling that each u_i's distance from layer $(\mathsf{R}, i + 1)$ was chosen uniformly at random from the set $\{2t + 1, 2t + 2\}$, let α denote the distance between u_1 and u_2 if $2t + 1$ was the chosen distance from layer $(\mathsf{R}, i + 1)$ for both u_1 and u_2. So the actual distance between u_1 and u_2 is either α (with probability $1/4$), $\alpha + 1$ (with probability $1/2$), or $\alpha + 2$ (with probability $1/4$). Further, by Lemma 7.5, at the time they are revealed to the algorithm $\mathcal{A}$, it does not know which of these 3 values is the true distance between u_1 and u_2, nor does it know which half-edge of each node is contained in the path between u_1 and u_2.

Let $\mathcal{D}_{u_1}$ denote the set of all elements of $\left(\binom{\Sigma}{2}\right)$ that can be assigned to 2 half-edges of u_1 in some fixed correct run of $\mathcal{A}$. Define $\mathcal{D}_{u_2}$ analogously for u_2. Since the path between u_1 and u_2 can be of length α, $\alpha + 1$, or $\alpha + 2$, and this length is unknown to the algorithm, any $\{a_1, b_1\} \in \mathcal{D}_{u_1}$ and $\{a_2, b_2\} \in \mathcal{D}_{u_2}$ must have walks of all of these lengths in $\mathcal{M}_{\mathcal{V}_i}$. Since $\alpha + 1$ and $\alpha + 2$ are clearly coprime, there exists some $N \in \mathbb{N}$ (the Frobenius number of $\alpha + 1$ and $\alpha + 2$) such that for all $n > N$, there exist $x, y \in \mathbb{N}$ such that $n = x(\alpha + 1) + y(\alpha + 2)$. By definition, then, all such $\{a_1, b_1\}$ and $\{a_2, b_2\}$ must belong to the same path-flexible strongly-connected component of $\mathcal{V}_i$. We call this component $\mathcal{D}_U$.

Now, consider any $v \in S_{\mathsf{C},i}$, and any two of its incident edges $e_1 = \{v, w_1\}$ and $e_2 = \{v, w_2\}$ such that w_1, w_2 are in layer (C, i) or higher. Note that w_1, w_2 are therefore in $S_{\mathsf{C},i}$ as well. By construction of $S_{\mathsf{C},i}$, there is a path $P_1 = (u_1, \ldots, w_1, \ldots, u_2)$ and a path $P_2 = (u_3, \ldots, w_2, \ldots, u_4)$ for some $u_1, u_2, u_3, u_4 \in U_i$ (with $u_1 \neq u_2, u_3 \neq u_4$). Without loss of generality, suppose $u_1 \neq u_4$. Then we can construct a path $P' = (u_1, \ldots, w_1, v, w_2, \ldots, u_4)$ between u_1 and u_4 which passes through e_1 and e_2. Let a_1 be the edge adjacent to u_1 which is traversed in P', and b_1 any other edge adjacent to u_1. Define a_4 and b_4 analogously for u_4.

Let $L(v, e)$ denote the label assigned to the half-edge (v, e). By the previous observations, $L(a_1, b_1)$ and $L(a_4, b_4)$ belong to the same flexible strongly-connected component, so $L(a_1, b_1) \to \cdots \to L(e_1, e_2) \to \cdots \to L(a_4, b_4)$ is a walk in the automaton $\mathcal{M}$ which starts and ends in the same flexible strongly connected component. Thus, $L(e_1, e_2)$ must also belong to this component—that is, $\mathcal{D}_{v,e_1,e_2} \subseteq \mathcal{D}_U$.

Finally, it can be seen that $\mathcal{D}_U$ is indeed $\mathcal{D}_i$ by noting that we must be able to label $S_{\mathsf{C},i}$ for $i \in \{1, \ldots, k\}$. We have seen that $\mathcal{D}_U$ must be a path-flexible strongly connected component of $\mathcal{V}_i$. Thus, if $\mathcal{V}_i$ is a component of a good sequence of length less than k, then $\mathsf{trim}(\mathsf{flexible\text{-}SCC}(\mathcal{V}_{k-1})) = \emptyset$ by definition, and we would thus have that $\mathcal{D}_U = \mathsf{flexible\text{-}SCC}(\mathcal{V}_{i-1}) = \emptyset$, and would not be able to label $S_{\mathsf{C},k}$. ◄

We now have sufficient tools to prove Theorem 2.3:

Proof of Theorem 2.3. Given a randomized online-LOCAL algorithm $\mathcal{A}$ with depth $k = d_\Pi$ which runs with $o(n^{1/k})$ locality, we can derive a contradiction. Taking G to be defined as above, with subsets $S_{\mathsf{R},1}, S_{\mathsf{C},1}, \ldots, S_{\mathsf{C},k}, S_{\mathsf{R},k+1}$, the previous induction argument tells us that $S_{\mathsf{R},k+1}$ must be labeled with elements of some $\mathcal{V}_{k+1} = \mathsf{trim}(\mathsf{flexible\text{-}SCC}(\mathcal{V}_k))$. But if $\mathcal{V}_{k+1} \neq \emptyset$, then we have a good sequence of length $k+1$—a contradiction to the assumption that $d_\Pi = k$. Thus, no such $o(n^{1/k})$ algorithm can exist, and Π has locality $\Omega(n^{1/k})$. ◄

8 Extending the Gap Between $o(n)$ and $\omega(\sqrt{n})$ in Unrooted Trees

Here, we are concerned with the following result:

▶ **Theorem 2.6.** *Suppose Π is an LCL problem that can be solved with $o(n)$ locality in the online-LOCAL model (with or without randomness) in unrooted trees. Then Π can be solved in unrooted trees in online-LOCAL with $O(\sqrt{n})$ locality (with or without randomness, respectively).*

Our proof is based on the work of Balliu et al. [5]. In said paper the authors show that, in the LOCAL model, any LCL with sublinear locality $o(n)$ can be transformed into an LCL with locality $O(\sqrt{n})$.
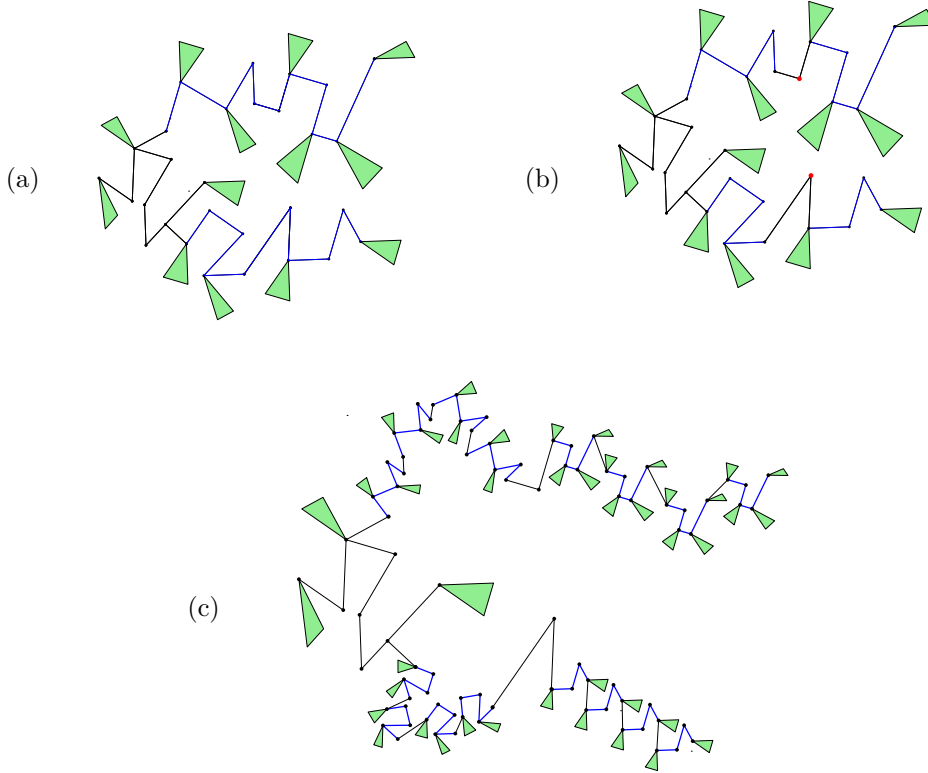
8.1 Previous Work

We provide a brief overview of the proof described in [5].

Suppose we have an LCL Π which is solved by an algorithm $\mathcal{A}$ which is known to run with sublinear ($o(n)$) locality. For any graph $\mathcal{G}$, we can distributedly construct a "virtual graph" $\mathcal{G}'$ with similar properties to $\mathcal{G}$ by viewing a radius-$O(\sqrt{n})$ neighborhood. $\mathcal{G}'$ is constructed to be much bigger than $\mathcal{G}$, and has its size upper-bounded by $N \gg n$. It will be possible to simulate the execution of $\mathcal{A}$ on $\mathcal{G}'$ by viewing only its corresponding radius-$O(\sqrt{n})$ neighborhoods in $\mathcal{G}$.

Given a tree T, we will now describe how we distributively construct T' using radius $O(\sqrt{n})$ neighborhoods of T.

Let $\tau = c\sqrt{n}$ for a constant c which depends only on the LCL problem Π (we will determine its precise value later). Given a node $v \in T$, we consider its radius-2τ neighborhood. We construct the *skeleton tree* T_{skel} by performing the following operation τ times: If a node within the radius-2τ neighborhood of v is a leaf node, remove it. Repeating this operation τ times results in removing all subtrees of height τ or less.

Now from T_{skel} we construct a set of paths $\mathcal{Q}_T$ by first removing all nodes of degree greater than 2. We are left with a set of paths which we will call T'_{skel}. We find a $(c+1, c)$ ruling set

Figure 8 (a) T with height-τ trees (green) and long paths (blue) identified. (b) T with ruling set nodes (red) and paths to pump (blue) (c) S, with pumped paths.

$\mathcal{R}$ for these paths. Note that this can be done with constant locality in the online-LOCAL model. Remove all nodes from $\mathcal{R}$ from T'_{skel} to obtain $\mathcal{Q}_T$.

So we have a set $\mathcal{Q}_T$ of paths of length l such that $c \leq l \leq 2c$. From here, we construct T'.

We now define $c = l_{\mathsf{pump}} + 4r$. Since $c > l_{\mathsf{pump}}$, we can apply the pumping lemma to make each path in $\mathcal{Q}$'s corresponding path in T arbitrarily long.

8.2   Constructing a Virtual Tree

At a high level, we use the following procedure, nearly identical to that described by Balliu et al. [5]; the steps are visualized in Figure 8. For each node we are given, we construct a portion of a virtual graph with many more nodes than our original graph. This graph consists of radius-$O(\sqrt{n})$ neighborhoods connected by very long paths, which cannot be seen in full by the algorithm. We show we can run the algorithm on this graph (that is, without knowledge of the parts of the original graph which are outside of the selected node's radius-$O(\sqrt{n})$ neighborhood), and that the results of this algorithm will give us a valid labeling of the original graph.

We begin by defining our new virtual graph. Suppose we are given a tree $T = (V, E)$ and an online-LOCAL algorithm $\mathcal{A}$ with locality $o(n)$.

8.2.1 Skeleton Tree

Our first step is to "prune" T, by removing any subtrees of height less than or equal to $\tau = c\sqrt{n}$ (for a constant c which we will define later). We call this the *skeleton tree T'* of T.

More formally, initialize T' as a copy of T. For τ rounds, we remove all leaf nodes from T'. After this, T' is our skeleton tree.

Note that we perform this construction distributedly and sequentially: given a node v, we check its radius-2τ neighborhood, and prune the leaf nodes for τ rounds. This ensures that every node within a radius-τ neighborhood will be completely pruned.

For the remaining nodes in T' (those which were not pruned), we define a function $\psi \colon V(T') \to V(T)$ which maps nodes of T' to their corresponding node in T.

8.2.2 A Set of Long Paths

We now want to construct a set of paths of T' which are of length $l \in [c, 2c]$ for some constant c.

We begin by removing any node with degree greater than 2 from the skeleton tree T'. From here, we are left with a set of long paths (note we only worry about those within a radius-τ neighborhood of whichever node we have selected at a time). We construct a $(c+1, c)$ ruling set for the remaining nodes. (Recall that an (α, β) ruling set is a set of nodes which are a distance at least α apart, with the property that every other node in the graph is no more than distance β from at least one ruling set node.)

Removing all the ruling set nodes, we obtain a set of paths with length at most $2c$. We discard any paths shorter than c (which occur at the ends of the original long paths we obtain after removing the high-degree nodes). We now have a set of paths with length $l \in [c, 2c]$, which we call $\mathcal{Q}$.

8.2.3 Virtual Tree

Before we can finish our definition of the final virtual tree S, we need to introduce some additional concepts. We do this in the next section, then resume the construction of S in Section 8.4.1.

8.3 An Equivalence Relation on Paths and the Pumping Lemma

We take an "intermission" to discuss properties of the paths of $\mathcal{Q}$.

Given a graph G with a subgraph H, we define the *poles* of H to be nodes $v \in V(H)$ which are adjacent to some node in $V(G) \setminus V(H)$. Now we can define the following equivalence relation on trees (introduced originally in [11]):

▶ **Definition 8.1** (Equivalence relation $\overset{*}{\sim}$)**.** *Given a graph H and its poles F, define $\xi(H, F) = (D_1, D_2, D_3)$ to be a tripartition of $V(H)$ where:*
- $D_1 = \bigcup_{v \in F} N^{r-1}(v)$
- $D_2 = \bigcup_{v \in D_1} N^r(v) \setminus D_1$
- $D_3 = V(H) - (D_1 \cup D_2)$

Let Q and Q' be the subgraphs of H and H' induced by the vertices $D_1 \cup D_2$ and $D'_1 \cup D'_2$ respectively. The equivalence holds, i.e., $(H, F) \overset{*}{\sim} (H', F')$, if and only if there is a 1 to 1 correspondence $\varphi \colon (D_1 \cup D_2) \to (D'_1 \cup D'_2)$ satisfying:
- *Q and Q' are isomorphic under φ, preserving the input labels of the LCL problem (if any) and the order of the poles.*

- Let $\mathcal{L}_*$ be any assignment of the output labels to vertices in $D_1 \cup D_2$, and let $\mathcal{L}'_*$ be the corresponding labeling of $D'_1 \cup D'_2$ under φ. Then $\mathcal{L}_*$ is extendible to $V(H)$ if and only if $\mathcal{L}'_*$ is extendible to $V(H')$.

It is proven by Chang and Pettie [11] that if the number of poles is constant, there is a constant number of equivalence classes under $\overset{*}{\sim}$. We also inherit the following lemma from Chang and Pettie [11], where $\mathsf{Replace}(G, (H, F), (H', F'))$ is the graph obtained by replacing a subgraph H of G with poles F with a new graph H' with poles F'.

▶ **Lemma 8.2.** *Let* $G' = \mathsf{Replace}(G, (H, F), (H', F'))$. *Suppose* $(H, F) \overset{*}{\sim} (H', F')$. *Let* $D_0 = V(G) \setminus V(H)$. *Let* $\mathcal{L}_\diamond$ *be a complete labelling of* G *that is locally consistent for all vertices in* $D_2 \cup D_3$. *Then there exists a complete labelling* $\mathcal{L}'_\diamond$ *satisfying the following:*

- $\mathcal{L}_\diamond = \mathcal{L}'_\diamond$ *for all* $v \in D_0 \cup D_1 \cup D_2$ *and their corresponding vertices in* $D'_0 \cup D'_1 \cup D'_2$. *Also, if* $\mathcal{L}_\diamond$ *is locally consistent for a node* v, *then* $\mathcal{L}'_\diamond$ *is locally consistent for* $\varphi(v)$.
- $\mathcal{L}'_\diamond$ *is locally consistent for all nodes in* $D'_2 \cup D'_3$.

We now inherit the notation introduced by Balliu et al. [5]: given a tree rooted at v, which we denote $\mathcal{T}_v$, let $\mathsf{Class}(\mathcal{T}_v)$ denote the equivalence class of $\mathcal{T}_v$ with v as its unique pole. Given a path $Q \in \mathcal{Q}$ with length k, we can consider Q's image in the original tree T as a sequence of trees, $(\mathcal{T}_i)_{i \in [k]}$, since each node $v \in Q$ is the root of some (possibly empty, aside from v itself) tree which was deleted in the first stage of processing. Let $\mathsf{Type}(Q)$ denote the equivalence class of the path Q with its 2 endpoints as poles.

We now restate 2 lemmas:

▶ **Lemma 8.3.** *Each node* u *can determine the type of* $\mathcal{T}_v$ *for all* v *contained within* u's *radius-τ neighborhood.*

This lemma allows us to ignore the deleted subtrees of $T \setminus T'$, as they can be filled in at the end.

▶ **Lemma 8.4.** *Let* $H = (\mathcal{T}_i)_{i \in [k]}$ *and let* $H' = (\mathcal{T}_i)_{i \in [k+1]}$ *be identical to* H *in its first* k *trees. Then* $\mathsf{Type}(H')$ *is a function of* $\mathsf{Type}(H)$ *and* $\mathsf{Class}(\mathcal{T}_{k+1})$.

8.3.1 The Pumping Lemma

We now have the necessary pieces to define a crucial lemma: the pumping lemma for paths.

▶ **Lemma 8.5** (Pumping lemma for paths)**.** *Let* $H = (\mathcal{T}_i)_{i \in [k]}$ *be a chain of trees with* $k \geq l_{\mathsf{pump}}$. *Then* H *can be decomposed into three subpaths* $H = x \circ y \circ z$ *such that:*

- $|xy| \leq l_{\mathsf{pump}}$,
- $|y| \geq 1$,
- $\mathsf{Type}(x \circ y^j \circ z) = \mathsf{Type}(x \circ y \circ z)$ *for all* $j \in \mathbb{N}$.

Now, the pumping lemma tells us that in every path $Q \in \mathcal{Q}$, there is some subpath which can be "pumped"—repeated an arbitrary number of times—and leave the type of the graph unchanged. This means that a valid solution of the graph with pumped paths, S, can be mapped to the original graph and be used to produce a valid solution.

8.4 Properties of the Virtual Tree

We first finish our construction of the virtual tree, which we began in Section 8.2. Then we examine some important properties of S which will allow us to simulate $\mathcal{A}$ on T by knowing only a radius-$O(\sqrt{n})$ neighborhood of T.

8.4.1 Finishing the Virtual Tree Construction

We now are able to finish the construction of the virtual tree S.

We choose a parameter B which can be an arbitrarily large function of n (which we will define more precisely later). At a high level, we will duplicate the pumpable subpath of each path $Q \in \mathcal{Q}$ so that its new length l' satisfies the inequality: $cB \leq l' \leq c(B+1)$. We maintain a mapping of nodes of T which are not a part of any pumped subpath of some $Q \in \mathcal{Q}$ to their new nodes in S: Let $S_o \subset V(S)$ denote the set of nodes of S which are not part of any path in $\mathcal{Q}$, or are at distance at most $2r$ from a node not contained in any $Q \in \mathcal{Q}$ (recall r is the checkability radius of our LCL). Informally, S_o is the set of "real" nodes of S—they are not part of the pumped paths, and they have corresponding nodes in T. Let $\eta : S_o \to T$ be a mapping of nodes in S_o to their corresponding nodes in T. We also define $T_o = \{\eta(v) \mid v \in S_o\}$. So T_o is the set of nodes far enough from the pumped regions of T which were not removed in the construction of T'.

So we have a virtual tree S which is a function of T and two parameters c and B (which will be defined more precisely soon).

8.4.2 Properties of S

We state some properties of S, proved originally by Balliu et al. [5]:

▶ **Lemma 8.6.** *S has at most $N = c(B+1)n$ nodes, where $n = |V(T)|$.*

This is because each node in a pumped path or a subtree of a node in a pumped path is duplicated at most $c(B+1)$ times. Note that N is an *upper* bound on the number of nodes of S, but the actual number of nodes $|V(S)|$ will be much smaller.

▶ **Lemma 8.7.** *For any path $P = (x_1, \ldots, x_k)$ of length $k \geq c\sqrt{n}$ which is a subgraph of T', at most $\sqrt{n}/c$ nodes in $V(P)$ have degree greater than 2.*

This lemma gives us an upper bound on the number of high-degree nodes, and is the point where this procedure demands $\Omega(\sqrt{n})$ locality, rather than allowing us to obtain an even better locality such as $O(\sqrt[3]{n})$.

Now, we can prove a new lemma which is helpful in the online-LOCAL model. This tells us no matter what path we take from a given node in S, we will either hit a pumped path or a leaf node within a radius-$c\sqrt{n}$ neighborhood.

▶ **Corollary 8.8.** *Any path $P = (x_1, \ldots, x_k) \subset T'$ with length $k \geq c\sqrt{n}$ will contain a subpath $Q \subset \mathcal{Q}$.*

Proof. By the previous lemma, P can have at most $\sqrt{n}/c$ nodes which have degree greater than 2 in T'. By the pigeonhole principle, there must be at least one subpath $(v_i, \ldots, v_{i+l})$ of P between 2 high-degree nodes (or the endpoints of P) with length $l \geq c^2 - 1 > c$ (this will always be true by our choice of c), so after selecting ruling set nodes from this subpath, we will be left with at least one path with length between c and $2c$—that is, a path $Q \subset \mathcal{Q}$. ◀

This corollary tells us that any path in T' starting from some vertex $v \in T'$ will either be short enough that it is contained within v's τ-radius neighborhood, or that it contains some subpath which will be pumped.

We now have all necessary ingredients to construct a new algorithm, $\mathcal{A}'$, which can find a labeling for T with $O(\sqrt{n})$ locality.

▶ **Lemma 8.9.** *Let $v \in T'$. For any path in T starting at $\psi(v)$ and ending at a leaf node, one of the following holds:*

- *The path has length less than or equal to 2τ.*
- *The path contains some path $Q \in \mathcal{Q}$ within its first τ nodes.*

Furthermore, this can be determined with knowledge of only the radius-2τ neighborhood of v.

Proof. Let $P = (v = v_1, v_2, \ldots, v_l)$ be a path in T, where $\psi^{-1}(v)$ is defined, and where v_l is some leaf node which is reachable from v. l denotes the length of the path. If $l \leq 2\tau$, then we are done. So suppose $l > 2\tau$. We consider $P' = (v = v_1, v_2, \ldots, v_\tau)$—the first τ nodes of P. We note that even if we have only seen v at this point in our algorithm, we can be certain of the degree of the first τ nodes of P after trimming their height-τ subtrees (that is, in T'), since we can see the 2τ-radius neighborhood of v (and thus the τ-radius neighborhood of all nodes in P'). So $\psi^{-1}(P') \subset T'$. P' has length τ, so we can apply the previous lemma to determine that at most $\frac{\sqrt{n}}{c}$ nodes of P' have degree greater than 2. Define $P'' \subset P'$ as the set of paths obtained by removing all nodes with degree greater than 2 from P'. By the pigeonhole principle, at least one path in P'' must have at least $c^2 - 1$ nodes. $c^2 - 1 > 2c$ (this will always be true by our choice of c). This path therefore must contain a path between 2 ruling set nodes with length between c and $2c$—that is, one of the paths of $\mathcal{Q}$. ◀

8.5 Speeding up $\mathcal{A}$

Given a node $v \in T'$ (a node which will not be removed in the construction of the skeleton tree T'), we can construct a local portion of S by viewing only v's radius-2τ neighborhood (which we will call T_v). If every node in this neighborhood is removed in the construction of T', then the connected component containing v has an $O(\sqrt{n})$ diameter and the problem can trivially be solved with $O(\sqrt{n})$ locality. So suppose T'_v is nonempty.

Let $t_{\mathsf{orig}}(n)$ denote the original runtime of $\mathcal{A}$ on a graph of n nodes. The following lemma, reproduced from work by Balliu et al. [5], is essential in determining the value of B:

▶ **Lemma 8.10.** *There exists some constant c such that, if nodes $u, v \in T_o$ are at distance at least $c\sqrt{n}$ in T, then their corresponding nodes $\eta^{-1}(u)$ and $\eta^{-1}(v)$ are at distance at least $cB\sqrt{n}/3$ in S.*

In particular, let $c = 4r + l_{\mathsf{pump}}$.

So we choose B such that $t_{\mathsf{orig}}(N) \leq cB\sqrt{n}/6$. Such a B will always exist, as $t_{\mathsf{orig}}(x) = o(x)$. (Recall that B may be an arbitrarily large function of n.) This choice of B, by the previous lemma, implies that if 2 nodes $u, v \in T_o$ are a distance at least $c\sqrt{n}$ in T, their radius-$t_{\mathsf{orig}}(N)$ neighborhoods are entirely disjoint, and thus one can have no influence on the other's labeling.

We run $\mathcal{A}$ on $\eta^{-1}(v)$, but tell $\mathcal{A}$ that the size of S is (exactly) N. Note that since N is an upper bound, it is not possible that $\mathcal{A}$ sees more than N nodes.

Finally, we can prove the following result:

▶ **Lemma 8.11.** *For nodes in T_o, it is possible to execute $\mathcal{A}$ on S by knowing only the neighborhood of radius $2c\sqrt{n}$ in T.*

Proof. Since B satisfies $t_{\mathsf{orig}}(N) \leq cB\sqrt{n}/6 \; (\leq cB\sqrt{n}/3)$, and since by the previous lemma, nodes outside of a radius-$2c\sqrt{n}$ ball in T_o are at distance at least $cB\sqrt{n}/3$ in S, when $\mathcal{A}$ runs on S and is processing v, it cannot see any nodes u such that the distance between $\eta(u)$ and $\eta(v)$ is greater than $2c\sqrt{n}$. So the locality of $\mathcal{A}$ on S is less than the radius of the subtree of S which $\eta(v)$ computed. This also implies that the nodes in T_0 do not see the whole graph and thus $\mathcal{A}$ cannot notice that N is not the actual size of the graph S. Thus, $\mathcal{A}$ can execute on S by knowing only the radius-$2c\sqrt{n}$ neighborhood of a given node in T. ◀

8.6   LOCALizing $\mathcal{A}$

We now show that $\mathcal{A}$ can be transformed not only to an online-LOCAL $O(\sqrt{n})$ algorithm, but a LOCAL one.

We note that the procedure followed in constructing S does not rely on any properties unique to the online-LOCAL model–and, in fact, can be done in the LOCAL model, as is shown by Balliu et al. [5]. So we can inherit this construction in the LOCAL model; it only remains to eliminate potential dependencies on global memory and sequential processing. We do this using a procedure similar to that used by Akbari et al. [1], where we create an "amnesiac" algorithm $\mathcal{A}'$. This algorithm effectively will see so many nodes and paths of the same type that global memory is of no use, and can be effectively disregarded.

8.6.1   Preprocessing

The preprocessing phase involves enumerating all possible pumpable subpaths of the paths in $\mathcal{Q}$, and feeding them to $\mathcal{A}$ many times. Recall that every such path is a chain of trees of height $\leq c\sqrt{n}$ The pumpable portions of a path are bounded by a constant—in particular, they must have length $\leq l_{\mathsf{pump}}$. Further, there is a constant number of equivalence classes ("Types") which tree in the path can take on—let us call this number ξ. Then the number of possible pumpable subpaths is bounded by $\xi^{l_{\mathsf{pump}}}$—a constant.

Let $\mathcal{P}$ be a sequence of all possible such paths, in an arbitrary order. We proceed by feeding each path $P \in \mathcal{P}$ to $\mathcal{A}$. We repeat this many times, until we notice a repeated labeling on each path P. This is guaranteed to happen since there is a constant number of input labels and thus a constant number of possible labelings for each path. More formally, for any constant Δ, if we feed a given path $P \in \mathcal{P}$ to the original algorithm $\xi^{l_{\mathsf{pump}}}\Delta$ times, by the pigeonhole principle, we are guaranteed to see at least one labeling $\mathcal{L}_0^P$ appear Δ many times. We call $\mathcal{L}_0^P$ the *canonical labeling* of P. Finally, we define a function $f : \mathcal{P} \to \{\mathcal{L}_0^P\}_{P \in \mathcal{P}}$, which maps each P to $\mathcal{L}_0^P$.

8.6.2   Running the Algorithm

From here, we will construct a LOCAL algorithm $\mathcal{A}'$ which solves Π in $O(\sqrt{n})$ rounds.

Given a tree T, we first construct S as described above. This can be done in the LOCAL model with $O(\sqrt{n})$ locality, as proven by Balliu et al. [5].

Let $\mathcal{Q}_S$ denote the set of all (pumped) paths in S. Each path $Q \in \mathcal{Q}_S$ takes the form $Q = x \circ y^j \circ z$ for subpaths x, y, z and some constant j, with $|y| \leq l_{\mathsf{pump}}$, since we duplicate a middle portion of the original path from T some number j times to obtain Q. $y \in \mathcal{P}$. We find the canonical labeling $\mathcal{L}_0^y$ of y, and will label each copy of $y \subset Q$ with this labeling. We then label x and z with brute force. This is doable in constant locality, since $|x \circ y \circ z| \leq 2c$, a constant.

From here, we can fill in the remaining portions of the graph—those not included in any path in $\mathcal{Q}$. By lemma 4.3, the remaining nodes not in any deleted tree or pumped path can be labeled with knowledge of a τ-radius neighborhood. Similarly, by lemma 3.3, each node in some deleted tree can be labeled with only knowledge of the tree itself. Since each of these trees has height at most τ, a locality of τ is sufficient to label each of these nodes using $\mathcal{A}$.

So we are able to run $\mathcal{A}$ with $O(\sqrt{n})$ locality, without reliance on any properties of the online-LOCAL model—and in particular, through use of the LOCAL model. With this procedure, we can transform any online-LOCAL $o(n)$ algorithm to a LOCAL algorithm which runs with $O(\sqrt{n})$ locality.

Since the paths are bounded by a constant length, this can be done with constant locality. Further, there is a (likely very large) constant number of paths, so this process will terminate. We will repeat this many times, and notice that our algorithm will begin to repeat labelings of a given path. From here, we can paste the labelings to the paths of $\mathcal{Q}$.

More formally, we notice that the paths of $\mathcal{Q}$ have their lengths bounded by a constant, namely $2c$. Further, the set of input labels Σ_{in} has constant size. So there is a constant number of possible paths which can occur in $\mathcal{Q}$.

References

1   Amirreza Akbari, Xavier Coiteux-Roy, Francesco D'Amore, François Le Gall, Henrik Lievonen, Darya Melnyk, Augusto Modanese, Shreyas Pai, Marc-Olivier Renou, Václav Rozhon, and Jukka Suomela. Online locality meets distributed quantum computing. *CoRR*, abs/2403.01903, 2024. `arXiv:2403.01903`, `doi:10.48550/ARXIV.2403.01903`.

2   Amirreza Akbari, Navid Eslami, Henrik Lievonen, Darya Melnyk, Joona Särkijärvi, and Jukka Suomela. Locality in online, dynamic, sequential, and distributed graph algorithms. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 10:1–10:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPICS.ICALP.2023.10`.

3   Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Jan Studený, and Jukka Suomela. Efficient classification of locally checkable problems in regular trees. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*, volume 246 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPICS.DISC.2022.8`.

4   Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Jan Studený, Jukka Suomela, and Aleksandr Tereshchenko. Locally checkable problems in rooted trees. *Distributed Computing*, 36(3):277–311, 2023. `doi:10.1007/S00446-022-00435-9`.

5   Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. Almost global problems in the LOCAL model. *Distributed Computing*, 34(4):259–281, 2021. `doi:10.1007/S00446-020-00375-2`.

6   Alkida Balliu, Mohsen Ghaffari, Fabian Kuhn, Augusto Modanese, Dennis Olivetti, Mikaël Rabie, Jukka Suomela, and Jara Uitto. Shared randomness helps with local distributed problems. *CoRR*, abs/2407.05445, 2024. `arXiv:2407.05445`, `doi:10.48550/ARXIV.2407.05445`.

7   Alkida Balliu, Janne H. Korhonen, Fabian Kuhn, Henrik Lievonen, Dennis Olivetti, Shreyas Pai, Ami Paz, Joel Rybicki, Stefan Schmid, Jan Studený, Jukka Suomela, and Jara Uitto. Sinkless orientation made simple. In Telikepalli Kavitha and Kurt Mehlhorn, editors, *2023 Symposium on Simplicity in Algorithms, SOSA 2023, Florence, Italy, January 23-25, 2023*, pages 175–191. SIAM, 2023. `doi:10.1137/1.9781611977585.CH17`.

8   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 Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 479–488. ACM, 2016. `doi:10.1145/2897518.2897570`.

9   Yi-Jun Chang. The complexity landscape of distributed locally checkable problems on trees. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 18:1–18:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPICS.DISC.2020.18`.

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

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

12 Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. On derandomizing local distributed algorithms. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 662–673. IEEE Computer Society, 2018. `doi:10.1109/FOCS.2018.00069`.

13 Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the complexity of local distributed graph problems. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 784–797. ACM, 2017. `doi:10.1145/3055399.3055471`.

14 Mohsen Ghaffari and Hsin-Hao Su. Distributed degree splitting, edge coloring, and orientations. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2505–2523. SIAM, 2017. `doi:10.1137/1.9781611974782.166`.

15 Christoph Grunau, Václav Rozhoň, and Sebastian Brandt. The landscape of distributed complexities on trees and beyond. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 37–47. ACM, 2022. `doi:10.1145/3519270.3538452`.

16 Alexander E. Holroyd and Thomas M. Liggett. Finitely dependent coloring. *Forum of Mathematics, Pi*, 4, 2016. `doi:10.1017/fmp.2016.7`.

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

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

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