The Complexity Landscape of Distributed Locally Checkable Problems on Trees

Yi-Jun Chang
ETH Zürich, Switzerland
yi-jun.chang@eth-its.ethz.ch

Abstract

Recent research revealed the existence of *gaps* in the complexity landscape of *locally checkable labeling* (LCL) problems in the LOCAL model of distributed computing. For example, the deterministic round complexity of any LCL problem on bounded-degree graphs is either $O(\log^* n)$ or $\Omega(\log n)$ [Chang, Kopelowitz, and Pettie, FOCS 2016]. The complexity landscape of LCL problems is now quite well-understood, but a few questions remain open.

For bounded-degree trees, there is an LCL problem with round complexity $\Theta(n^{1/k})$ for each positive integer k [Chang and Pettie, FOCS 2017]. It is conjectured that no LCL problem has round complexity $o(n^{1/(k-1)})$ and $\omega(n^{1/k})$ on bounded-degree trees. As of now, only the case of $k = 2$ has been proved [Balliu et al., DISC 2018].

In this paper, we show that for LCL problems on bounded-degree trees, there is indeed a gap between $\Theta(n^{1/(k-1)})$ and $\Theta(n^{1/k})$ for each $k \geq 2$. Our proof is *constructive* in the sense that it offers a sequential algorithm that decides which side of the gap a given LCL problem belongs to. We also show that it is EXPTIME-hard to distinguish between $\Theta(1)$-round and $\Theta(n)$-round LCL problems on bounded-degree trees. This improves upon a previous PSPACE-hardness result [Balliu et al., PODC 2019].

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed algorithms, LOCAL model, locally checkable labeling

Digital Object Identifier 10.4230/LIPIcs.DISC.2020.18

Related Version A full version of the paper is available at `https://arxiv.org/abs/2009.09645`.

1 Introduction

In this paper, we consider Linial's LOCAL model of distributed computing [17, 21], where the input graph $G = (V, E)$ and the communication network are identical. Each vertex $v \in V$ corresponds to a processor, each edge $e \in E$ corresponds to a communication link, and the computation proceeds in synchronized rounds. There is no restriction on the local computation power and the message size, and the main complexity measure for an algorithm is the number of rounds. We assume that the number of vertices $n = |V|$ and the maximum degree $\Delta = \max_{v \in V} \deg(v)$ are global knowledge.

There is a recent line of research [3, 4, 5, 6, 8, 10, 11, 12, 15, 16, 22] aiming to systematically understand the round complexity of distributed graph problems, with a focus on the *locally checkable labelings* (LCL) problems [19], which is the class of distributed problems whose solution is locally verifiable by examining a constant-radius neighborhood of each vertex. The class of LCL problems is sufficiently general that it encompasses many well-studied problems in the LOCAL model, such as maximal matching, maximal independent set (MIS), $(\Delta + 1)$-vertex coloring, and sinkless orientation. For example, in the $(\Delta + 1)$-vertex coloring problem, the output of each vertex v is a color $c(v) \in \{1, 2, \ldots, \Delta + 1\}$. The output is a legal solution if $c(u) \neq c(v)$ for each edge $e = \{u, v\} \in E$. Each vertex v can locally check if it has a neighbor $u \in N(v)$ with $c(u) = c(v)$ by examining the output within its radius-1 neighborhood.

1.1 The Spectrum of Distributed Complexities

Different from the sequential setting such as the Turing machine or the RAM model, in the complexity landscape of LCL problems in the LOCAL model, large *gaps* exist in the complexity landscape.

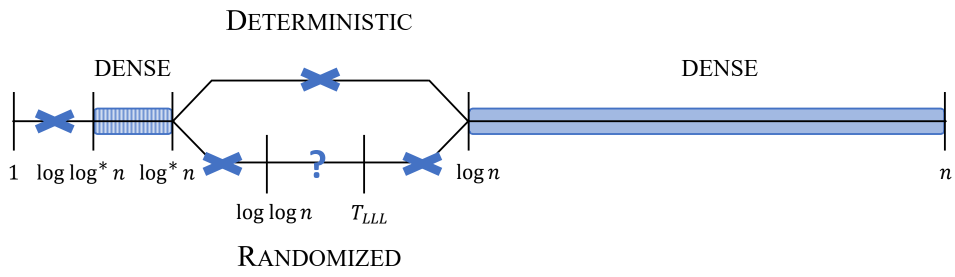

Figure 1 Complexity landscape of LCLs on bounded-degree general graphs.

General graphs. Chang, Kopelowitz, and Pettie [11] showed that for any LCL problem on bounded-degree graphs, its deterministic round complexity is either $O(\log^* n)$ or $\Omega(\log n)$, and its randomized round complexity is either $O(\log^* n)$ or $\Omega(\log\log n)$. Chang and Pettie [12] showed that any $o(\log n)$-round randomized algorithm for an LCL problem can be accelerated to run in $O(T_{\text{LLL}})$ rounds, where T_{LLL} is the randomized complexity of the distributed constructive Lovász Local Lemma (LLL) [14] under a polynomial criterion $pd^c = O(1)$ for any positive constant c. It was conjectured in [12] that $T_{\text{LLL}} = \Theta(\log\log n)$. Chang and Pettie also showed that the gap $\omega(1) - o(\log\log^* n)$ can be derived using the approach of Naor and Stockmeyer [19]. Balliu et al. showed that the two remaining regions $[\Theta(\log\log^* n), \Theta(\log^* n)]$ and $[\Theta(\log n), \Theta(n)]$ are *dense* in that many round complexity functions within these ranges can be realized by LCL problems [5]. See Figure 1 for an illustration of the complexity landscape of LCLs on bounded-degree general graphs.

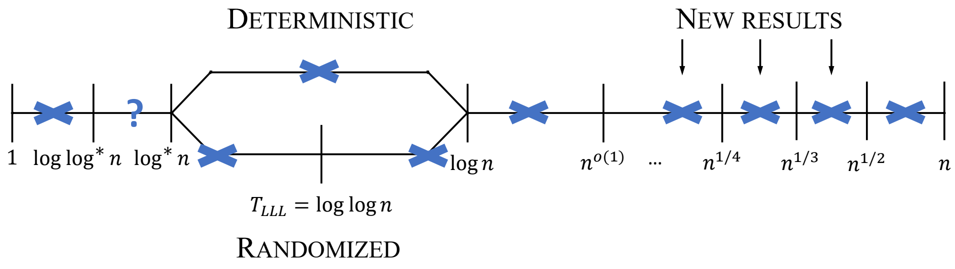

Figure 2 Complexity landscape of LCLs on bounded-degree trees.

Trees. The four gaps in the lower end of the spectrum $[\Theta(1), \Theta(\log n)]$ are the same as the setting of general graphs. It was conjectured in [12] that the $\omega(1) - o(\log\log^* n)$ gap on bounded-degree trees can be extended to $\omega(1) - o(\log^* n)$. So far this conjecture was proved only for the special case of *homogeneous* problems [6]. For the higher end of the spectrum $[\Theta(\log n), \Theta(n)]$, it was proved in [12] that any distributed algorithm that takes $n^{o(1)}$ rounds can be accelerated to run in just $O(\log n)$ rounds, and there exists an LCL problem with complexity $\Theta(n^{1/k})$ for each $k \geq 1$. It was left as an open problem to decide if there are gaps between them. Recently, Balliu et al. [3] showed that there is indeed a gap between $\Theta(\sqrt{n})$ and $\Theta(n)$, the other cases are still open. See Figure 2 for an illustration of the complexity landscape of LCLs on bounded-degree trees.

New result. In this paper, we prove the existence of the gap $\omega(n^{1/k}) - o(n^{1/(k-1)})$ of LCL problems on bounded-degree trees, for each integer $k \geq 2$.

▶ **Theorem 1.** *For any integer $k \geq 2$, for any LCL problem $\mathcal{P}$ on bounded-degree trees, either one of the following holds.*

- *The deterministic and randomized round complexities of $\mathcal{P}$ are $\Omega(n^{1/(k-1)})$.*
- *The deterministic and randomized round complexities of $\mathcal{P}$ are $O(n^{1/k})$.*

Furthermore, there is a sequential algorithm that given an integer $k \geq 2$ and a description of $\mathcal{P}$ decides which side of the gap $\mathcal{P}$ belongs to.

The proof of Theorem 1 is obtained by unifying the approach of Balliu et al. [3] and the approach of Chang and Pettie [12] using a generalized tree decomposition algorithm.

Theorem 1 implies that randomness does not help for LCL problems on bounded-degree trees in the regime of polynomial round complexity.

1.2 The Complexity of Classification

The complexity gaps in Figures 1 and 2 classify the distributed problems into complexity classes. A natural question to ask is whether this classification is *decidable*. Unfortunately, even for grids and tori, it is *undecidable* whether a given LCL problem can be solved in $O(1)$ rounds [9, 19], since LCL on grids can be used to simulate a Turing machine. This undecidability result does not apply to special graph classes such as paths, cycles, and trees. In fact, the proof of the $\omega(\log n) - n^{o(1)}$ gap on bounded-degree trees in [12] is *constructive* in the sense that it gives us an algorithm that can decide whether a given LCL problem on bounded-degree trees has complexity $O(\log n)$ or $n^{\Omega(1)}$.

Much progress has recently been made in understanding to what extent the design of distributed algorithms and the proof of distributed lower bounds can be automated [1, 2, 7, 9, 13, 20]. On paths or cycles, with or without input labels, only three complexity classes are possible: $\Theta(1)$, $\Theta(\log^* n)$, and $\Theta(n)$. Balliu et al. [1] showed that for any given LCL problem $\mathcal{P}$ on paths or cycles, it is *decidable* to check which class $\mathcal{P}$ belongs to, and there is a sequential algorithm that automates the design of an asymptotically optimal distributed algorithm for $\mathcal{P}$. For comparison, the previous proofs [11, 12, 19] establishing this classification did not offer such results.

On the negative side, Balliu et al. [1] showed that the problem of determining the optimal asymptotic distributed complexity is PSPACE-hard, even for paths and cycles with input labels. Since trees can be used to encode input labels, the same PSPACE-hardness result extends to the case of bounded-degree trees without input labels.

New result. Our proof of the existence of the gap $\omega(n^{1/k}) - o(n^{1/(k-1)})$ offers a sequential algorithm that decides which side of the gap a given LCL problem $\mathcal{P}$ belongs to. When the locality radius r of the LCL is a constant independent of the description length N of the LCL, the runtime $2^{2^{N^{O(1)}}}$ of our sequential algorithm is *doubly exponential* in $N^{O(1)}$. To complement this result, we show that this problem is inherently very hard by proving that this problem is EXPTIME-hard. Specifically, we say that a round complexity function $T(n)$ is *realizable* if there exists an LCL problem $\mathcal{P}$ whose round complexity is $\Theta(T(n))$ on bounded-degree trees. We prove the following theorem.

▶ **Theorem 2.** *Let $T_1(n) \ll T_2(n)$ be two realizable round complexity functions. Given an LCL problem $\mathcal{P}$ that is promised to have round complexity either $\Theta(T_1(n))$ or $\Theta(T_2(n))$ on bounded-degree trees, it is* EXPTIME*-hard to decide the round complexity of $\mathcal{P}$.*

1.3 Organization

In Section 2, we overview the basics of LCL problems and review the pumping lemma of Chang and Pettie [12]. In Section 3, we review the proof of the $\omega(n^{1/2}) - o(n)$ gap by Balliu et al. [3]. In Section 4, we consider a generalized version of the tree decomposition of Miller and Reif [18] that allows us to unify the approach of Balliu et al. [3] and the approach of Chang and Pettie [12]. In Section 5, we prove Theorem 1 for the case of deterministic algorithms. The complete proofs of Theorems 1 and 2 are left to the full version of the paper.

2 Preliminaries

In the deterministic variant of the LOCAL model, each vertex v has a distinct $O(\log n)$-bit identifier $\mathrm{ID}(v)$. In the randomized variant of the LOCAL model, there are no distinct identifiers, but each vertex has access to a stream of unbiased random bits, and the maximum tolerable global probability of failure is $1/n$. Note that a t-round LOCAL algorithm can be seen as a function that maps a radius-t subgraph centered at v to an output label assigned to v.

2.1 Locally Checkable Labeling

A distributed graph problem is locally checkable if there is some constant r such that the validity of a solution can be checked locally by having each vertex examine its radius-r neighborhood. For example, the maximal independent set (MIS) problem is locally checkable with locality radius $r = 1$, but the maximum independent set problem is not locally checkable.

Formal definition. Formally, an LCL problem $\mathcal{P}$ is specified by the following parameters: the locality radius r, the set of input labels Σ_{in}, the set of output labels Σ_{out}, and the set of allowed configurations $\mathcal{C}$. Each member of $\mathcal{C}$ is a radius-r subgraph H centered at a specific vertex v, where each vertex in H is assigned an input label from Σ_{in} and an output label from Σ_{out}. Note that $|\Sigma_{\mathrm{in}}| = 1$ corresponds to the special case where there is no input label.

An *instance* of an LCL problem $\mathcal{P}$ is a graph $G = (V, E)$ where each vertex is assigned an input label from Σ_{in}. A *solution* for $\mathcal{P}$ on G is a labeling function ϕ_{out} that assigns to each vertex in G an output label from Σ_{out}. We say that ϕ_{out} is *locally consistent* for a vertex $v \in V$ if its radius-r neighborhood $N^r(v)$ is an allowed configuration in $\mathcal{C}$ under the given input labeling and the output labeling ϕ_{out}. The output labeling ϕ_{out} is *legal* if it is locally consistent everywhere.

Graph terminology. Unless otherwise stated, all vertices in all graphs in this paper are assigned input labels from Σ_{in}, and the term *label* refers to *output label.* An *unlabeled graph* is a graph where no vertex is assigned an output label from Σ_{out}. A *partially labeled graph* is a graph with a labeling function $\mathcal{L}$ that maps each vertex v to an element of $\Sigma_{\mathrm{out}} \cup \{\perp\}$. A *completely labeled graph* or a *labeled graph* is a graph with a labeling function $\mathcal{L}$ that maps each vertex v to an element of Σ_{out}.

Description length. We assume that any given LCL problem $\mathcal{P}$ is specified by representing $\mathcal{C}$ as a truth table. Specifically, a *centered graph* is a graph $G = (V, E)$ with a distinguished vertex $s \in V$, and the *radius* of G is defined by $\max_{v \in V} \operatorname{dist}(v, s)$. The truth table representation of $\mathcal{P}$ is a mapping $\mathcal{G}_{r,\Delta,\Sigma_{\text{in}},\Sigma_{\text{out}}} \mapsto \{0, 1\}$, where $\mathcal{G}_{r,\Delta,\Sigma_{\text{in}},\Sigma_{\text{out}}}$ is the set of all centered graphs $G = (V, E)$ of radius at most r with maximum degree Δ where each vertex $v \in V$ is equipped with an input label from Σ_{in} and an output label from Σ_{out}.

If the graph class under consideration is the set of trees of maximum degree Δ, then the description length of $\mathcal{P}$ can be upper bounded by $(1 + |\Sigma_{\text{in}}| \cdot |\Sigma_{\text{out}}|)^{1+\Delta^r}$.

To derive this upper bound, consider the rooted tree T_r of height r where the root v has Δ children, all vertices u with $1 \leq \operatorname{dist}(u, v) \leq r - 1$ have $\Delta - 1$ children, and all vertices u with $\operatorname{dist}(u, v) = r$ are leaf vertices. The number of trees in $\mathcal{G}_{r,\Delta,\Sigma_{\text{in}},\Sigma_{\text{out}}}$ is at most the number of distinct labeling of the vertices in T_r by $(\Sigma_{\text{in}} \times \Sigma_{\text{out}}) \cup \{\star\}$, where $\star$ is a special symbol indicating the non-existence of a vertex. Hence the description length can be upper bounded by $(1 + |\Sigma_{\text{in}}| \cdot |\Sigma_{\text{out}}|)^{n_r}$, where n_r is the number of vertices in T_r. We have $n_0 = 1$, $n_1 = 1 + \Delta$, and $n_r = 1 + \Delta + \Delta \sum_{i=1}^{r-1} (\Delta - 1)^{r-1}$ for each $r \geq 2$. It is clear that $n_r \leq 1 + \Delta^r$ for all r.

Remarks on edge labeling and orientation. In general, an LCL might have edge labels and edge orientation. It is straightforward to encode edge labels and edge orientation as vertex labels. For example, given an input graph G, consider the following pre-processing. For each edge $e = \{u, v\} \in E$, subdivide it into a length-3 path $(u, x_{e,u}, x_{e,v}, v)$ by adding two new vertices $x_{e,u}$ and $x_{e,v}$. Each newly added vertex is assigned a special input label e indicating that it represents a half of an edge. Now an edge orientation $u \to v$ can be encoded as $\phi(x_{e,u}) = 0$ and $\phi(x_{e,v}) = 1$.

2.2 Pumping Lemma

We review the pumping lemma of Chang and Pettie [12], which plays a crucial role in establishing complexity gaps on trees.

Notation for partially labeled graphs. A *partially labeled graph* $\mathcal{G} = (G, \mathcal{L})$ is a graph $G = (V, E)$ together with a function $\mathcal{L} : V \to \Sigma_{\text{out}} \cup \{\perp\}$. The vertices in $\mathcal{L}^{-1}(\perp)$ are *unlabeled*. A *complete labeling* $\mathcal{L}' : V(G) \to \Sigma_{\text{out}}$ for $\mathcal{G}$ is one that labels all vertices and is consistent with $\mathcal{G}$'s partial labeling, i.e., $\mathcal{L}'(v) = \mathcal{L}(v)$ whenever $\mathcal{L}(v) \neq \perp$. A *legal labeling* is a complete labeling that is *locally consistent* for all $v \in V(G)$, i.e., the labeled subgraph induced by $N^r(v)$ is consistent with the given LCL problem $\mathcal{P}$. Here $N^r(v)$ is the set of all vertices within distance r of v. A subgraph of a partially labeled graph $\mathcal{G} = (G, \mathcal{L})$ is a pair $\mathcal{H} = (H, \mathcal{L}')$ such that H is a subgraph of G, and $\mathcal{L}'$ is $\mathcal{L}$ restricted to the domain $V(H)$. With a slight abuse of notation, we usually write $\mathcal{H} = (H, \mathcal{L})$.

An equivalence relation. A tree $\mathcal{H}$ with two distinguished vertices $s, t \in V(H)$ is called a *bipolar* tree. We call s and t the two *poles* of $\mathcal{H}$. We consider the equivalence relation $\overset{\star}{\sim}$ on bipolar trees defined in [12]. We write $\mathsf{Type}(\mathcal{H})$ to denote the equivalence class of the bipolar tree $\mathcal{H}$. The following property of $\overset{\star}{\sim}$ is crucial.

Suppose we are given the following.
- $\mathcal{G}$ is any graph.

- $\mathcal{H}$ is a bipolar subtree of $\mathcal{G}$ with two poles s and t such that the removal of s and t disconnects $\mathcal{H}$ from the rest of $\mathcal{G}$.
- $\mathcal{H}'$ is another bipolar subtree with two poles s' and t' such that $\mathsf{Type}(\mathcal{H}) = \mathsf{Type}(\mathcal{H}')$.
- $\mathcal{L}_\diamond$ is any complete legal labeling of $\mathcal{G}$.

Define the graph $\mathcal{G}'$ as the result of replacing the subgraph $\mathcal{H}$ of $\mathcal{G}$ with $\mathcal{H}'$. Then there exists a legal labeling $\mathcal{L}'$ of $\mathcal{H}'$ meeting the following conditions.

- The following complete labeling $\mathcal{L}'_\diamond$ of $\mathcal{G}'$ is a legal labeling.

$$\mathcal{L}'_\diamond(v) = \begin{cases} \mathcal{L}'(v) & \text{if } v \in \mathcal{H}, \\ \mathcal{L}_\diamond(v) & \text{if } v \in \mathcal{G} \setminus \mathcal{H}. \end{cases}$$

- Such a labeling $\mathcal{L}'$ of $\mathcal{H}'$ can be computed *solely* from $\mathcal{H}'$ and the given labeling $\mathcal{L}_\diamond$ restricted to $\mathcal{H}$.

In view of the above, the vertices in $\mathcal{H}'$ can compute their $\mathcal{L}'$-labels using only information within $\mathcal{H}'$ and the given labeling $\mathcal{L}_\diamond$ restricted to $\mathcal{H}$, without communicating with the vertices outside of $\mathcal{H}'$. Intuitively, this allows us to reduce the task of finding a legal labeling $\mathcal{L}'$ of $\mathcal{G}'$ to the task of finding a legal labeling $\mathcal{L}_\diamond$ of $\mathcal{G}$.

A pumping lemma for bipolar trees. The unique path $(s = u_1, u_2, \ldots, u_k = t)$ connecting the two poles s and t of a bipolar tree $\mathcal{H}$ is called the *core path* of $\mathcal{H}$. The tree $\mathcal{H}$ can be viewed as a string of subtrees $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$, where $\mathcal{T}_i$ is the subtree of $\mathcal{H}$ rooted at u_i. For the sake of convenience, we use the following string notation $\mathcal{H} = (\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k)$ to describe a bipolar tree $\mathcal{H}$. Viewing bipolar trees as strings, the following *pumping lemma* was proved in [12].

There exists a number ℓ_{pump} depending only on the given LCL problem $\mathcal{P}$ such that as long as $k \geq \ell_{\text{pump}}$, any bipolar tree $\mathcal{H} = (\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k)$ can be decomposed into three substrings $\mathcal{H} = x \circ y \circ z$ meeting the following conditions.

- $|xy| \leq \ell_{\text{pump}}$.
- $|y| \geq 1$.
- $\mathsf{Type}(x \circ y^j \circ z) = \mathsf{Type}(\mathcal{H})$ for each non-negative integer j.

Intuitively, the pumping lemma allows us to extend the length of $\mathcal{H} = (\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k)$ to arbitrarily long without changing its type, as long as $k \geq \ell_{\text{pump}}$.

3 A Review of the $\omega(n^{1/2}) - o(n)$ Gap

We review the proof of the $\omega(n^{1/2}) - o(n)$ gap by Balliu et al. [3]. Given an $o(n)$-round randomized or deterministic LOCAL algorithm $\mathcal{A}$ for the given LCL problem $\mathcal{P}$, the goal is to design a new randomized or deterministic LOCAL algorithm $\mathcal{A}'$ with round complexity $O(\sqrt{n})$. Within this section, we only apply the pumping lemma on unlabeled graphs, but we will see that when we extend the proof to other gaps, we need to deal with partially labeled graphs.

The skeleton tree. Let the tree $G = (V, E)$ be the underlying network. Let $\tau = \Theta(\sqrt{n})$ be a threshold to be determined. Define the *skeleton tree* G_{skel} as the result of iteratively removing all leaf vertices of G for τ iterations. Specifically, start with $G_0 = G$, and let G_i be the result of removing all leaf vertices of G_{i-1} for each $1 \le i \le \tau$, and then we have $G_{\mathsf{skel}} = G_\tau$.

If G_{skel} is empty, then we are already done, since this implies that the diameter of G is $O(\tau) = O(\sqrt{n})$, so $\mathcal{P}$ can be solved trivially in $O(\sqrt{n})$ rounds. In subsequent discussion we assume that G_{skel} is not empty. We will identify a set of disjoint paths $\mathcal{P}$ of G_{skel} meeting the following conditions.

1. Each $P = (v_1, v_2, \ldots, v_x) \in \mathcal{P}$ satisfies the following requirements.
   a. $x \in [\ell_{\text{pump}}, 2\ell_{\text{pump}}]$.
   b. Each v_i is of degree-2 in G_{skel}.
2. Let G' be the subgraph of G_{skel} resulting from removing all paths in $\mathcal{P}$. Let $\mathcal{S}$ denote the set of connected components in G'. Then each connected component $S \in \mathcal{S}$ in G_{skel} has diameter $O(\sqrt{n})$.

The proof of the existence of $\mathcal{P}$ can be found in [3]. We will also provide a proof in Section 4. In this section we only need to use the fact that the skeleton tree G_{skel} and the set of paths $\mathcal{P}$ can be computed in $O(\sqrt{n})$ rounds on G.

Since G_{skel} is constructed by iteratively removing all leaf vertices of G for τ iterations, each vertex v in $G \setminus G_{\mathsf{skel}}$ is reachable to a *unique* vertex u in G_{skel} via the vertices $G \setminus G_{\mathsf{skel}}$.

For any vertex subset U in G_{skel}, we define $U^* \supseteq U$ as the set of vertices in G resulting from adding to U all vertices in $G \setminus G_{\mathsf{skel}}$ reachable to U via the vertices in $G \setminus G_{\mathsf{skel}}$. A consequence of Condition 2 is that the diameter of S^* is $O(\tau + \sqrt{n}) = O(\sqrt{n})$, for each $S \in \mathcal{S}$.

The virtual tree. Consider the *virtual tree* G_{virt} defined as the result of applying the pumping lemma on P^* for each $P = (v_1, v_2, \ldots, v_x) \in \mathcal{P}$ to the graph G. The definition of P^* is in the paragraph above. Here P^* is seen as a bipolar tree with the poles $s = v_1$ and $t = v_x$. Specifically, the pumping lemma allows us to replace each bipolar tree $P^* = (T_1, T_2, \ldots, T_k)$ is by some other bipolar tree $P' = (T'_1, T'_2, \ldots, T'_{x'})$ such that $\mathsf{Type}(P') = \mathsf{Type}(P^*)$, and $x' \in [w, w + \ell_{\text{pump}}]$, where w is some very large number to be determined.

The $O(\sqrt{n})$-round algorithm $\mathcal{A}'$. We are ready to describe our $O(\sqrt{n})$-round algorithm $\mathcal{A}'$. The first step of the algorithm is to compute the skeleton tree G_{skel} and the set of paths $\mathcal{P}$ in $O(\sqrt{n})$ rounds. After that, we can simulate the virtual tree G_{virt} by having the vertices in each $P \in \mathcal{P}$ simulate the virtual bipolar tree P' resulting from the pumping lemma. We compute a legal labeling $\mathcal{L}_{\mathsf{virt}}$ of G_{virt} by a simulation of $\mathcal{A}$ on G_{virt}. We will later see that the simulation can also be done in $O(\sqrt{n})$ rounds. Finally, we will show that the labeling $\mathcal{L}_{\mathsf{virt}}$ can be transformed into a legal labeling $\mathcal{L}$ of G using another $O(\sqrt{n})$ rounds.

Simulation of $\mathcal{A}$ on the virtual tree. It is clear that the number of vertices in G_{virt} can be upper bounded by $O(n^2 w)$, since $|\mathcal{P}| \le n$ and the number of vertices in each P' is $O(nw)$. We simulate the given algorithm $\mathcal{A}$ on the virtual tree G_{virt} assuming that the number of vertices is $n' = O(n^2 w)$.

Since the round complexity of $\mathcal{A}$ on an n'-vertex graph is $o(n')$, by selecting w as a sufficiently large number depending on n, the round complexity of $\mathcal{A}$ can be made much smaller than $0.1w$. Therefore, to simulate $\mathcal{A}$ on G_{virt}, each vertex v in G_{virt} only needs to gather all information within radius $0.1w$ to v. We make the following observations.

- For each $S \in \mathcal{S}$, the subgraph S^* has diameter $O(\sqrt{n})$.

- For each $P \in \mathcal{P}$, the number of vertices in the core path of the bipolar subtree P' is within $[w, w + \ell_{\text{pump}}]$.

By these facts, it is straightforward to see that each vertex v in G_{virt} is able to gather all information within radius $0.1w$ to v in $O(\sqrt{n})$ rounds of communication in the underlying network G. For example, if $v \in S^*$ for some $S \in \mathcal{S}$, then v only need to learn the following.

- The subgraph induced by the set S^*.
- The virtual bipolar tree P', for each path $P \in \mathcal{P}$ adjacent to S.

Remember that P' can be computed from P^*. Since the diameter of S^* (for each $S \in \mathcal{S}$) and the diameter of P^* (for each $P \in \mathcal{P}$) are $O(\sqrt{n})$, this information gathering can be done in $O(\sqrt{n})$ rounds in G.

Computing a legal labeling of G. Suppose we have computed a legal labeling $\mathcal{L}_{\mathsf{virt}}$ of G_{virt}. We show how to use this legal labeling $\mathcal{L}_{\mathsf{virt}}$ to obtain a legal labeling $\mathcal{L}$ of G in $O(\sqrt{n})$ rounds. For each $S \in \mathcal{S}$, the labeling of the vertices in S^* is unchanged, i.e., $\mathcal{L}(v) = \mathcal{L}_{\mathsf{virt}}(v)$. For each path $P \in \mathcal{P}$, the $\mathcal{L}$-labels of the vertices in P^* are computed as follows.

Remember that G_{virt} is the result of replacing P^* with P', for each $P \in \mathcal{P}$, and the two bipolar trees P' and P^* have the same type. In view of the property of $\overset{\star}{\sim}$ described in Section 2.2, there exists a labeling $\mathcal{L}'$ of P^* such that if we replace the bipolar subtree P' (labeled with $\mathcal{L}_{\mathsf{virt}}$) by the bipolar subtree P^* (labeled with $\mathcal{L}'$), the legality of the labeling of the underlying graph is maintained. Moreover, such a labeling $\mathcal{L}'$ of P^* can be computed from the labeling $\mathcal{L}_{\mathsf{virt}}$ restricted to P', without using any information outside of P'. Thus, we can carry out this procedure, in parallel for each $P \in \mathcal{P}$, and this takes $O(\sqrt{n})$ rounds, since the diameter of P^* is at most $2\tau + 2\ell_{\text{pump}} - 1 = O(\sqrt{n})$, for each $P \in \mathcal{P}$. After that, we obtain a desired legal labeling $\mathcal{L}$ of G.

4 A Generalized Tree Decomposition

Miller and Reif [18] considered the following decomposition algorithm. Start with a tree $G = (V, E)$; remove the vertices in V by repeatedly doing the following two operations alternately: Rake (removing all leaf vertices) and Compress (removing all degree-2 vertices). It is known that $O(\log n)$ iterations suffice to remove all vertices in the tree [18]. Variants of this decomposition have turned out to be useful in the design of LOCAL algorithms [10, 12].

In this section, we consider a generalized version of this decomposition, which allows us to show the existence of $\mathcal{P}$ needed in Section 3, and to extend the proof idea in Section 3 to other gaps.

We start with a formal definition of our decomposition, which is parameterized by two integers $\ell \geq 1$ and $\gamma \geq 1$, and it decomposes the vertices in the tree G into

$$V = V_1^{\mathsf{R}} \cup V_1^{\mathsf{C}} \cup V_2^{\mathsf{R}} \cup V_2^{\mathsf{C}} \cup V_3^{\mathsf{R}} \cup V_3^{\mathsf{C}} \cup \cdots.$$

Let L denote the highest number i such that $V_i^{\mathsf{C}} \cup V_{i+1}^{\mathsf{R}} \cup V_{i+1}^{\mathsf{C}} \cup \cdots$ is empty. We define G_i^{C} as the subgraph induced by the vertices $\left(\bigcup_{j=i+1}^{L} V_j^{\mathsf{R}}\right) \cup \left(\bigcup_{j=i}^{L-1} V_j^{\mathsf{C}}\right)$, which is the set of all vertices that are in V_i^{C} or higher layers. Similarly, we define G_i^{R} as the subgraph induced by the vertices $\left(\bigcup_{j=i}^{L} V_j^{\mathsf{R}}\right) \cup \left(\bigcup_{j=i}^{L-1} V_j^{\mathsf{C}}\right)$.

We require the sets V_i^{R} and V_i^{C} to satisfy some requirements. Each connected component of the subgraph induced by V_i^{R} must be a rooted tree with height at most $\gamma - 1$, and only the root can possibly have neighbors in $V_i^{\mathsf{C}} \cup V_{i+1}^{\mathsf{R}} \cup V_{i+1}^{\mathsf{C}} \cup \cdots$. Each connected component of the subgraph induced by V_i^{C} must be a path with $x \in [\ell, 2\ell]$ vertices, and only the endpoints can possibly have neighbors in $V^{\mathsf{R}}_{i+1} \cup V^{\mathsf{C}}_{i+1} \cup V^{\mathsf{R}}_{i+2} \cup \cdots$. See Figure 3 for an illustration, where each triangle represents a rooted tree. The precise requirements are as follows.

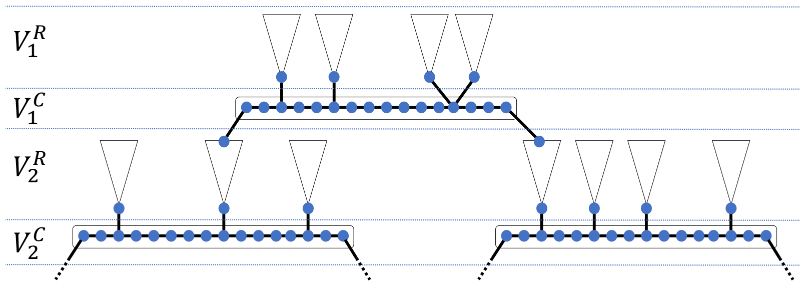

Figure 3 Top layers in a generalized tree decomposition.

Requirements for V^{R}_i. Let S be a connected component of the subgraph induced by V^{R}_i. Then there is a root vertex $z \in S$ such that the following conditions are met.
- z has at most one neighbor in G^{C}_i, and each $v \in S \setminus \{z\}$ has no neighbor in G^{C}_i.
- Each $v \in S \setminus \{z\}$ satisfies $\mathrm{dist}(v, z) \leq \gamma - 1$.

Note that for the special case of $\gamma = 1$, the set V^{R}_i is an independent set.

Requirements for V^{C}_i. Let S be a connected component of the subgraph induced by V^{C}_i. Then S is a path $(u_1, u_2, \ldots, u_x)$ with $x \in [\ell, 2\ell]$ such that the following is true for each $u_j \in S$.
- For the case $1 < j < x$ (i.e., u_j is an intermediate vertex), u_j has no neighbor in G^{R}_{i+1}.
- Consider the case $j = 1$ or $j = x$ (i.e., u_j is an endpoint). If $x \geq 2$, then u_j has exactly one neighbor in G^{R}_{i+1}. If $x = 1$, then u_j has exactly two neighbors in G^{R}_{i+1}.

A decomposition $V = V^{\mathsf{R}}_1 \cup V^{\mathsf{C}}_1 \cup V^{\mathsf{R}}_2 \cup V^{\mathsf{C}}_2 \cup V^{\mathsf{R}}_3 \cup V^{\mathsf{C}}_3 \cup \cdots$ satisfying the above requirements is called a *(γ, ℓ)-decomposition.* We will see in Lemma 7 that for any positive integers $k = O(1)$ and $\ell = O(1)$, and for any

$$\gamma \geq n^{1/k}(\ell/2)^{1-1/k},$$

an (γ, ℓ)-decomposition with $L = k$ can be computed in $O(n^{1/k})$ rounds deterministically.

4.1 The Decomposition Algorithm

Our algorithm constructing the above decomposition uses the following modified Rake and Compress operations defined in [12]. Here U is a subset of V representing the set of vertices that are not yet removed.

Rake: Each $v \in U$ removes itself if one of the following conditions is met.
1. $\deg_U(v) = 0$.
2. $\deg_U(v) = 1$ and the unique neighbor u of v in U has $\deg_U(u) > 1$.
3. $\deg_U(v) = 1$ and the unique neighbor u of v in U has $\deg_U(u) = 1$ and $\mathrm{ID}(v) > \mathrm{ID}(u)$.

Compress: Each $v \in U$ removes itself if v belongs to a path P such that $|V(P)| \geq \ell$ and $\deg_U(u) = 2$ for each $u \in V(P)$.

The purpose of Condition 3 in the Rake operation is to break tie for the special case where v is in a component of U that is a length-1 path. This is to ensure that we remove an independent set of vertices in a Rake operation.

▶ **Definition 3** ([12]). *Let P be a path. A subset $I \subset V(P)$ is called an (α, β)-independent set if the following conditions are met: (i) I is an independent set that does not contain either endpoint of P, and (ii) each connected component of the subgraph induced by $V(P) \setminus I$ has at least α vertices and at most β vertices, unless $|V(P)| < \alpha$, in which case $I = \emptyset$.*

It is a folklore that an $(\ell, 2\ell)$-independent set of a path graph can be computed in $O(\log^* n)$ rounds deterministically when $\ell = O(1)$ [1,12,17].

The algorithm. The decomposition algorithm begins with $U = V(G)$ and $i = 1$. In iteration i, we do the following.
1. Do γ Rake operations.
2. Do one Compress operation.
3. Update the iteration number $i \leftarrow i + 1$.

Repeatedly do this until $U = \emptyset$, and then we proceed to the following post-processing step.

The post-processing step. Let R_i (resp., C_i) be the set of vertices removed during a Rake (resp., Compress) operation in the ith iteration. For each path P that is a connected component of the subgraph induced by C_i, Find an $(\ell, 2\ell)$-independent set I_P of P. Define C_i^* as the subset of C_i that is the union of I_P for each P that is a connected component of the subgraph induced by C_i. Let L be the highest number i such that $R_i \cup C_{i-1} \neq \emptyset$. Then a partition $V = \left(\bigcup_{i=1}^{L} V_i^{\mathsf{R}}\right) \cup \left(\bigcup_{i=1}^{L-1} V_i^{\mathsf{C}}\right)$ is defined by setting $V_i^{\mathsf{R}} = R_i \cup C_{i-1}^*$ and $V_i^{\mathsf{C}} = C_i \setminus C_i^*$. What we have done in the post-processing step is promoting each vertex in the independent set I_P to the next layer, and this ensures that the requirement on the size of paths for V_i^{C} is met.

Analysis. We analyze the decomposition $V = \left(\bigcup_{i=1}^{L} V_i^{\mathsf{R}}\right) \cup \left(\bigcup_{i=1}^{L-1} V_i^{\mathsf{C}}\right)$ produced using the above algorithm. The proofs of the following two lemmas follow immediately from the description of the decomposition algorithm.

▶ **Lemma 4** (Properties of V_i^{R}). *Let S be a connected component of the subgraph induced by V_i^{R}. Then there is a root vertex $z \in S$ such that the following conditions are met.*
- *z has at most one neighbor in G_i^{C}, and each $v \in S \setminus \{z\}$ has no neighbor in G_i^{C}.*
- *Each $v \in S \setminus \{z\}$ satisfies $\operatorname{dist}(v, z) \leq \gamma - 1$.*

Proof. The first case is when S contains a vertex u that is in I_P for some P during the post-processing step, we must have $S = \{u\}$, and u has no neighbor in G_i^{C}. In this case, setting $z = u$ works.

The second case is when $S \subseteq R_i$. We select $z \in S$ as the last vertex removed from U during the decomposition algorithm, among all vertices in S. Since we do γ Rake operations in each iteration, each $v \in S \setminus \{z\}$ satisfies $\operatorname{dist}(v, z) \leq \gamma - 1$. It is straightforward to see that z is the only vertex in S that may have a neighbor in G_i^{C}; and z can have at most one such neighbor. ◀

▶ **Lemma 5** (Properties of V_i^{C}). *Let S be a connected component of the subgraph induced by V_i^{C}. Then S is a path $(u_1, u_2, \ldots, u_x)$ with $x \in [\ell, 2\ell]$ such that the following is true for each $u_j \in S$.*
- *For the case $1 < j < x$ (i.e., u_j is an intermediate vertex), u_j has no neighbor in G_{i+1}^{R}.*
- *Consider the case $j = 1$ or $j = x$ (i.e., u_j is an endpoint). If $x \geq 2$, then u_j has exactly one neighbor in G_{i+1}^{R}. If $x = 1$, then u_j has exactly two neighbors in G_{i+1}^{R}.*

Proof. In view of the post-processing step and the definition of an (α, β)-independent set, S is a path $(u_1, u_2, \ldots, u_x)$ with $x \in [\ell, 2\ell]$. It is straightforward to verify that the conditions specified in the lemma are met. ◀

Next, we analyze the round complexity of the decomposition algorithm and the number L in the decomposition. We remark that the case of $\gamma = 1$ is considered and analyzed in [12].

▶ **Lemma 6** ([12]). *Suppose $\gamma = 1$ and $\ell \geq 1$ is a constant. An (γ, ℓ)-decomposition with $L = O(\log n)$ of a tree G can be computed in $O(\log n)$ rounds deterministically.*

In this paper, we are only interested in the case of $\gamma \gg 1$.

▶ **Lemma 7.** *Suppose $\gamma \geq n^{1/k}(\ell/2)^{1-1/k}$, for some positive integers $k = O(1)$ and $\ell = O(1)$. An (γ, ℓ)-decomposition with $L = k$ of a tree G can be computed in $O(n^{1/k})$ rounds deterministically.*

Proof. Consider an arbitrary vertex $v \in V$, and root the tree G at v. Define S_i as the connected component containing v in the subgraph induced by the set U at the beginning of the ith iteration. Define S'_i as the connected component containing v in the subgraph induced by the set U at the beginning of the Compress operation during the ith iteration. Note that $S_1 = V$. To prove the lemma, it suffices to show that $S'_k = \emptyset$.

Let A be the number of degree-2 vertices in S'_i that are not removed during the ith Compress. Let B be the number of vertices in S'_i whose degree is not 2 at the beginning of the ith Compress. Note that $|S_{i+1}| = A + B$.

We observe that $A \leq (\ell - 1)(B - 1)$, as the degree-2 vertices in S'_i that are not removed during the ith Compress form connected components of at most $\ell - 1$ vertices. Specifically, consider the tree T resulting from contracting each degree-2 vertex in S'_i. The number of vertices in T equals B, and the number of edges in T is at least $A/(\ell - 1)$. Hence $A/(\ell - 1) \leq (B - 1)$.

We also observe that the number of degree-1 vertices in S'_i at the beginning of the ith Compress is at least $B/2$. As each degree-1 vertex of S'_i must be adjacent to a connected component of $S_i \setminus S'_i$ of size at least γ, we have $|B|/2 < |S_i|/\gamma$. Therefore,

$$|S_{i+1}| = A + B < \ell B < \frac{\ell}{2\gamma} \cdot |S_i|.$$

In order to have $S'_k = \emptyset$, it suffices that $|S_k| \leq \gamma$. Indeed, we have

$$|S_k| \leq \left(\frac{\ell}{2\gamma}\right)^{(k-1)} \cdot n \leq \gamma.$$

For the round complexity, the main part of the algorithm costs $O((\gamma + \ell)k) = O(\gamma) = O(n^{1/k})$ rounds. The post-processing step costs $O(\log^* n)$ rounds, as $\ell = O(1)$. ◀

The set of paths $\mathcal{P}$. We revisit the proof in Section 3 and prove that the required set of paths $\mathcal{P}$ can be computed in $O(\sqrt{n})$ rounds. We run our algorithm for constructing a (γ, ℓ)-decomposition with the parameters $\gamma = \tau = n^{1/2}(\ell/2)^{1/2} = \Theta(\sqrt{n})$ and $\ell = \ell_{\text{pump}} = \Theta(1)$. Here τ is the parameter in the definition of the skeleton tree G_{skel} in Section 3. Remember that G_{skel} is the result of iteratively removing all leaf vertices of G for τ iterations.

By Lemma 7, our (γ, ℓ)-decomposition satisfies $L = 2$, and it decomposes V into three sets V_1^{R}, V_1^{C}, and V_2^{R}, and the decomposition can be computed in $O(\sqrt{n})$ rounds. It is clear from the description of the algorithm that $G_{\mathsf{skel}} = G_1^{\mathsf{C}}$ is exactly the subgraph induced by $V_1^{\mathsf{C}} \cup V_2^{\mathsf{R}}$. Selecting $\mathcal{P}$ as the set of all connected components of V_1^{C} satisfies all the requirements of $\mathcal{P}$ stated in Section 3.

5 Extension to Other Gaps

In this section, we prove Theorem 1 for the case of deterministic algorithms by extending the proof of the $\omega(n^{1/2}) - o(n)$ gap by Balliu et al. [3] described in Section 3. The complete proof of Theorem 1 is left to the full version of the paper.

5.1 Proof Idea

The main idea of the proof of Theorem 1 is as follows. Let k be any positive constant, To prove the existence of the gap $\omega(n^{1/k}) - o(n^{1/(k-1)})$, for any given $o(n^{1/(k-1)})$-round deterministic algorithm $\mathcal{A}$ for a given LCL problem $\mathcal{P}$, we need to be able to design a new deterministic algorithm $\mathcal{A}'$ with round complexity $O(n^{1/k})$.

We compute a (γ, ℓ)-decomposition, with $\gamma = \Theta(n^{1/k})$ and $\ell \geq \ell_{\text{pump}}$ to decompose V into the subsets $V_1^{\mathsf{R}}, V_1^{\mathsf{C}}, V_2^{\mathsf{R}}, \ldots, V_{k-1}^{\mathsf{R}}, V_{k-1}^{\mathsf{C}}, V_k^{\mathsf{R}}$, and then apply the pumping lemma to extend each path in $V_1^{\mathsf{C}}, V_2^{\mathsf{C}} \ldots, V_{k-1}^{\mathsf{C}}$ to a path of length within $[w, w + \ell_{\text{pump}}]$, in order to produce a virtual tree with $O(w^{k-1})$ vertices, omitting the dependency on n. By Lemma 7, such a decomposition can be computed in $O(n^{1/k})$ rounds.

If we select w to be sufficiently large, the execution of a given $o(n^{1/(k-1)})$-round algorithm $\mathcal{A}$ takes less than $0.1w$ rounds on the virtual tree. As each connected component induced by V_i^{R} is a rooted tree with diameter $O(n^{1/k})$, the simulation of $\mathcal{A}$ can be done in $O(n^{1/k})$ rounds in the underlying network G. Hence we obtain an $O(n^{1/k})$-round algorithm $\mathcal{A}'$ for the same problem.

This approach does not work immediately, as we will encounter some issues described below, but these issues can be overcome using the graph operations defined in [12].

An issue in pumping bipolar subtrees. The reason that we can apply the pumping lemma for V_1^{C} in Section 3 is that each connected component P of V_1^{C} is naturally associated with a bipolar tree P^*. We do not have this property for the connected components of V_i^{C} for $i > 1$, since each vertex $v \in V_1^{\mathsf{R}} \cup V_1^{\mathsf{C}} \cup \cdots \cup V_i^{\mathsf{R}} = V(G) \setminus V(G_i^{\mathsf{C}})$ might be reachable to more than one connected component of V_i^{C} via the vertices in $V(G) \setminus V(G_i^{\mathsf{C}})$. See Figure 3.

Let us recall the virtual tree construction in Section 3. Define G' as the graph resulting from pumping the paths in V_1^{C}. Formally, for each connected component P of V_1^{C}, replace P^* by a much longer bipolar tree P' with $\mathsf{Type}(P^*) = \mathsf{Type}(P')$, where P^* is the bipolar subtree of G induced by the vertices in P and all vertices in V_1^{R} that are reachable to a vertex in P via the vertices in V_1^{R}. Note that G' is the same as the virtual tree G_{virt} in Section 3, if we use $G_{\mathsf{skel}} = G_1^{\mathsf{C}}$ and let $\mathcal{P}$ be the set of connected components of V_1^{C}.

As we are in the $k > 2$ case, we would like to also pump the paths in V_2^{C} in this graph G' in a way similar to the case of V_1^{C}. As discussed above, a difference between V_1^{C} and V_2^{C} is that it is possible that a vertex $v \in V(G') \setminus V(G_2^{\mathsf{C}})$ is reachable to multiple connected components in V_2^{C} via the vertices in $V(G') \setminus V(G_2^{\mathsf{C}})$, so we are unable to associate a bipolar tree P^* to each connected component P of V_2^{C}.

Recall that in our high-level proof idea we will ultimately simulate an algorithm $\mathcal{A}$ on a virtual tree, and the runtime of $\mathcal{A}$ will be less than $0.1w$. Again consider the graph G' and one of its bipolar subtree P' resulting from pumping P^* for some connected component P of V_1^{C}. The virtual bipolar tree P' separates the graph G' into two parts, and the vertices in one part does not need to communicate with the vertices in the other part in the simulation of $\mathcal{A}$, as its runtime is less than $0.1w$. Recall that the core path of P' has at least w vertices.

Motivated by the above discussion, we consider the graph G'' defined as the result of applying the following operation on G' for each virtual bipolar tree P'. Let u and v be the two vertices in $V(G') \setminus V(P')$ adjacent to the two poles s and t of P' via the edges $\{u, s\}$ and $\{v, t\}$. We *duplicate* P' into two identical bipolar subtrees, one is attached to u via $\{u, s\}$, the other is attached to v via $\{v, t\}$. Note that this is the Duplicate-Cut operation defined in [12]. See Figure 4 for an illustration.

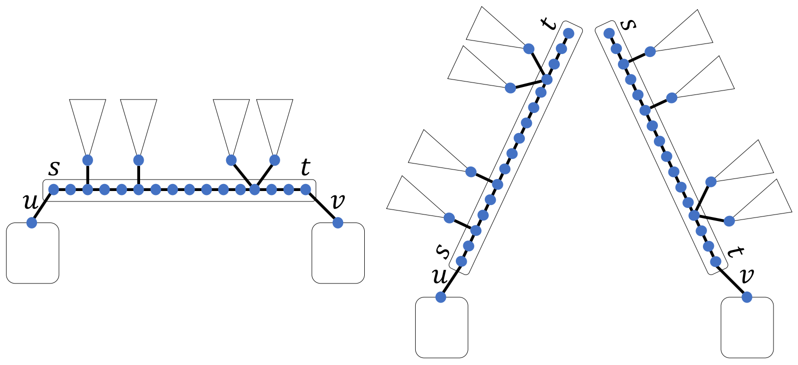

Figure 4 The Duplicate-Cut operation.

Let P be a connected component of V_2^{C} in the graph G''. With respect to G'', we are able to define P^* in the same way as the case of V_1^{C}. Specifically, we define P^* as the bipolar subtree of G'' induced by the vertices in P and all vertices in $V(G'') \setminus V(G_2^{\mathsf{C}})$ that are reachable to a vertex in P via the vertices in $V(G'') \setminus V(G_2^{\mathsf{C}})$. Using this approach recursively, we can pump the paths of all layers $V_1^{\mathsf{C}}, V_2^{\mathsf{C}}, \ldots, V_{k-1}^{\mathsf{C}}$.

An issue caused by duplicating bipolar trees. The duplication of bipolar subtrees in Duplicate-Cut also causes an issue. Consider the graphs G, G', and G'' defined above. As discussed in Section 3, given a legal labeling of G', we can obtain a legal labeling of G using a property of $\overset{\star}{\sim}$ and the fact that pumping does not alter the type of a bipolar tree. However, when we try to obtain a legal labeling $\mathcal{L}'$ of G' from a given legal labeling $\mathcal{L}''$ of G'', we encounter an issue that the two copies of a bipolar subtree P' resulting from applying Duplicate-Cut in G' might be labeled differently in $\mathcal{L}''$.

To resolve this issue, before the duplication of P' in the construction of G'' from G', we let some vertices near the middle of P' to first commit to a certain labeling. Such a labeling is computed by simulating the given $o(n^{1/(k-1)})$-round algorithm $\mathcal{A}$, pretending that the number of vertices is $O(w^{k-1})$, omitting the dependence on n. We can assume that the runtime of $\mathcal{A}$ on P' is at most $0.1w$ by selecting w to be sufficiently large.

Specifically, let P^* be a bipolar tree that we would like to apply the pumping lemma. We give a different way of constructing P' from P^*. We write P^* as a string of subtrees $(\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_x)$. Let $(v_1, v_2, \ldots, v_x)$ be the core path of P^* and $e = \{v_{\lfloor x/2 \rfloor}, v_{\lfloor x/2 \rfloor+1}\}$ be the middle edge of the core path. Consider the decomposition $P^* = \mathcal{X} \circ \mathcal{Y} \circ \mathcal{Z}$, where $\mathcal{Y} = (\mathcal{T}_{\lfloor x/2 \rfloor - r+1}, \ldots, \mathcal{T}_{\lfloor x/2 \rfloor + r})$ is the middle part. We apply the pumping lemma on $\mathcal{X}$ and $\mathcal{Z}$ to extend them to longer bipolar trees whose whose size of core path is within $[w, w+\ell_{\text{pump}}]$, and then we assign output labels to the vertices in $N^{r-1}(e) = N^{r-1}(v_{\lfloor x/2 \rfloor}) \cup N^{r-1}(v_{\lfloor x/2 \rfloor+1})$ by simulating the algorithm $\mathcal{A}$, whose runtime is less than $0.1w$. The resulting partially labeled bipolar tree is P'. Note that this construction of P' from P^* is the same as the one in [12] using the operations Label and Extend. In this paper, we call this operation Label-Extend. See Figure 5 for an illustration.

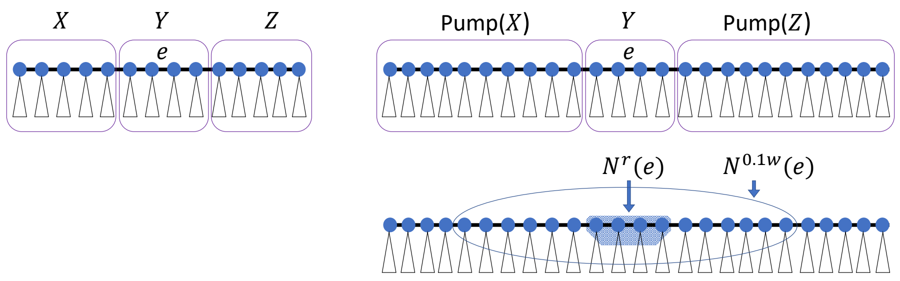

Figure 5 The Label-Extend operation.

We briefly explain why doing this labeling of middle vertices resolves the issue. Suppose G_a is the result of apply Duplicate-Cut to some bipolar subtree P' in G_b, where this bipolar tree P' is constructed as above and its middle vertices have been assigned output labels. In a given legal labeling of G_a, the two copies of P' might be labeled differently, but their middle vertices must be labeled the same. We decompose $P' = P_s \circ P_t$ into two parts by cutting along the middle edge e. The pole s is in P_s, and the other pole t is in P_t. We name the two copies of P' in G_a by P'_s and P'_t based on the poles they use to connect to the rest of the graph. To obtain a legal labeling of G_b from a given legal labeling of G_a, we simply label P_s by adapting the labeling of P'_s in G_a, and label P_t by adapting the labeling of P'_t in G_a. The legality of the resulting labeling of G_b is easy to verify.

See Figure 6 for an example. The top-left figure illustrates the bipolar subtree P' in G_b, where its middle vertices have been assigned output labels. The top-right figure illustrates the graph G_a, which results from applying Duplicate-Cut to P' in G_b. The down-right figure illustrates the given legal labeling of G_a. The down-left figure shows the legal labeling of G_b obtaining from the given legal labeling of G_a.

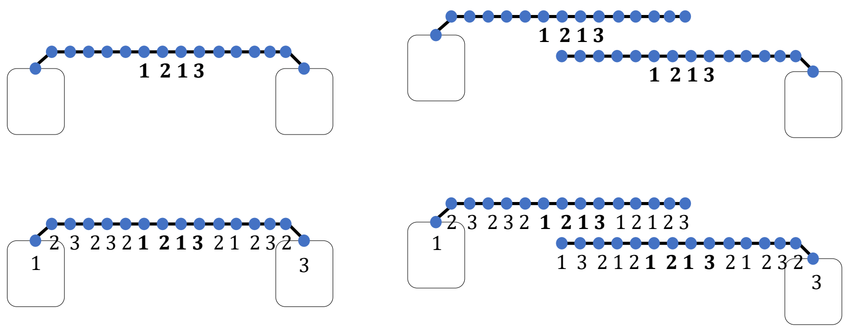

Figure 6 Obtaining a legal labeling.

5.2 A Sequence of Virtual Graphs

The approach discussed above naturally leads to a sequence of partially labeled virtual graphs $\mathcal{R}_1^{\mathsf{R}}, \mathcal{R}_1^{\mathsf{C}}, \mathcal{R}_2^{\mathsf{R}}, \mathcal{R}_2^{\mathsf{C}}, \ldots, \mathcal{R}_k^{\mathsf{R}}$. Each virtual graph has a *real* and an *imaginary* part. Each real vertex corresponds to a vertex in the underlying network G. The graph $\mathcal{R}_i^{\mathsf{R}}$ will have G_i^{R} as its real part, and the graph $\mathcal{R}_i^{\mathsf{C}}$ will have G_i^{C} as its real part. The imaginary part of these graphs are subtrees attached to the real vertices. In the actual distributed implementation, the simulation of the imaginary subtrees attached to a real vertex v are handled by v in the underlying network G.

Construction of $\mathcal{R}_1^{\mathsf{R}}$. The graph $\mathcal{R}_1^{\mathsf{R}}$ is unlabeled and it equals the underlying network G.

Construction of $\mathcal{R}_1^{\mathsf{C}}$. The graph $\mathcal{R}_1^{\mathsf{C}}$ is almost identical to $G = \mathcal{R}_1^{\mathsf{R}}$. In $\mathcal{R}_1^{\mathsf{C}}$, only the vertices in G_1^{C} are real. For each connected component S of $V_1^{\mathsf{R}} = \mathcal{R}_1^{\mathsf{R}} \setminus G_1^{\mathsf{C}}$, there is at most one vertex $v \in G_1^{\mathsf{C}}$ that is adjacent to S. If such a vertex v exists, then S becomes an imaginary subtree stored in the real vertex v. Otherwise, S is not included in $\mathcal{R}_1^{\mathsf{C}}$.

Construction of $\mathcal{R}_2^{\mathsf{R}}$. The graph $\mathcal{R}_2^{\mathsf{R}}$ is the graph G' in Section 5.1. Formally, the graph $\mathcal{R}_2^{\mathsf{R}}$ is constructed by applying the following operation to each connected component $P = (v_1, v_2, \ldots, v_x)$ of V_1^{C} in $\mathcal{R}_1^{\mathsf{C}}$. Let $P^* = (\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_x)$ be the bipolar subtree induced by P and the imaginary subtrees therein. Replace P^* by the partially labeled bipolar tree P' which is the result of applying Label-Extend to P^*, and then apply Duplicate-Cut to P^*.

Construction of $\mathcal{R}_2^{\mathsf{C}}$. The graph $\mathcal{R}_2^{\mathsf{C}}$ is almost identical to $\mathcal{R}_2^{\mathsf{R}}$. In $\mathcal{R}_2^{\mathsf{C}}$, only the vertices in G_2^{C} are real. Due to the Duplicate-Cut operation, for each connected component S of $\mathcal{R}_2^{\mathsf{R}} \setminus G_2^{\mathsf{C}}$, there is at most one vertex $v \in G_2^{\mathsf{C}}$ that is adjacent to S. If such a vertex v exists, then S becomes an imaginary subtree stored in the real vertex v. Otherwise S is not included in $\mathcal{R}_2^{\mathsf{C}}$.

Construction of the other graphs. The rest of the partially labeled graphs $\mathcal{R}_3^{\mathsf{R}}$, $\mathcal{R}_3^{\mathsf{C}}$, $\mathcal{R}_4^{\mathsf{R}}$, $\mathcal{R}_4^{\mathsf{C}}$, ..., $\mathcal{R}_k^{\mathsf{R}}$ are constructed analogously. In the end, $\mathcal{R}_k^{\mathsf{R}}$ is a virtual graph with $O(w^{k-1})$ vertices, omitting the dependence on n. It is straightforward to see that the sequence of virtual graphs $\mathcal{R}_1^{\mathsf{R}}, \mathcal{R}_1^{\mathsf{C}}, \mathcal{R}_2^{\mathsf{R}}, \mathcal{R}_2^{\mathsf{C}}, \ldots, \mathcal{R}_k^{\mathsf{R}}$ can be constructed in $O(n^{1/k})$ rounds.

Completing the labeling. Recall that the partial labelings of $\mathcal{R}_1^{\mathsf{R}}, \mathcal{R}_1^{\mathsf{C}}, \mathcal{R}_2^{\mathsf{R}}, \mathcal{R}_2^{\mathsf{C}}, \ldots, \mathcal{R}_k^{\mathsf{R}}$ are computed using the operation Label-Extend, which is based on simulating $\mathcal{A}$ while assuming that the number of vertices is $n' = O(w^{k-1})$, omitting the dependence on n. By the correctness of $\mathcal{A}$, each of these partial labelings can be completed into a complete legal labeling. Since each connected component of the real part of $\mathcal{R}_k^{\mathsf{R}}$ has at most $O(n^{1/k})$ vertices, a complete legal labeling of $\mathcal{R}_k^{\mathsf{R}}$ can be found in $O(n^{1/k})$ rounds by a brute-force information gathering. Once we have a complete labeling of $\mathcal{R}_k^{\mathsf{R}}$, we can start from this complete labeling to obtain a complete legal labeling for $\mathcal{R}_{k-1}^{\mathsf{C}}, \mathcal{R}_{k-1}^{\mathsf{R}}, \mathcal{R}_{k-2}^{\mathsf{C}}, \ldots, \mathcal{R}_1^{\mathsf{R}} = G$ in $O(n^{1/k})$ rounds in view of the discussion in Section 5.1, as these graphs are constructed by applying Label-Extend and then applying Duplicate-Cut to the bipolar trees resulting from Label-Extend.

The round complexity for finding a legal labeling of $G = \mathcal{R}_1^{\mathsf{R}}$ using this approach is $O(n^{1/k})$ because the size of each connected component of V_i^{R} is $O(n^{1/k})$.

Hence we have the $\omega(n^{1/k}) - o(n^{1/(k-1)})$ gap for LCL problems on bounded-degree trees for the case of deterministic algorithms. That is, given a deterministic $o(n^{1/(k-1)})$-round algorithm $\mathcal{A}$ for $\mathcal{P}$, we can construct another deterministic $O(n^{1/k})$-round algorithm $\mathcal{A}'$.

A note about unique identifiers. A subtle issue about the simulation of $\mathcal{A}$ is that the simulation needs distinct identifiers. Specifically, to guarantee the correctness of a deterministic τ-round algorithm for an LCL problem with locality radius r, it suffices that any two vertices within distance $2\tau + 2r$ have distinct identifiers [11].

We only simulate $\mathcal{A}$ when we apply Label-Extend. When we do the simulation of $\mathcal{A}$, we can locally generate distinct identifiers of length $O(\log n')$ for all vertices in $N^{0.1w+r}(e)$, where e is the middle edge of the core path of the bipolar tree on which we run $\mathcal{A}$, and $n' = O(w^{k-1})$, omitting the dependence on n. This partial ID assignment satisfies the requirement that, for any two vertices u and v that are assigned identifiers and are within distance $2 \cdot 0.1w + 2r$, their identifiers are distinct.

References

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

2 Alkida Balliu, Sebastian Brandt, Yuval Efron, Juho Hirvonen, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Classification of distributed binary labeling problems. In *Proceedings of the 34th International Symposium on Distributed Computing (DISC)*, 2020.

3 Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. Almost global problems in the LOCAL model. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC)*, 2018.

4 Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. How much does randomness help with locally checkable problems? In *Proceedings of the 39th Symposium on Principles of Distributed Computing (PODC)*, pages 299–308. ACM, 2020.

5 Alkida Balliu, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Dennis Olivetti, and Jukka Suomela. New classes of distributed time complexity. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1307–1318. ACM, 2018.

6 Alkida Balliu, Juho Hirvonen, Dennis Olivetti, and Jukka Suomela. Hardness of minimal symmetry breaking in distributed computing. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 369–378, 2019.

7 Sebastian Brandt. An automatic speedup theorem for distributed problems. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 379–388, 2019.

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 *Proceedings of the 48th ACM Symposium on the Theory of Computing (STOC)*, pages 479–488, 2016.

9 Sebastian Brandt, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Patric R.J. Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela, and Przemysław Uznaundefinedski. LCL problems on grids. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 101–110, 2017.

10 Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. Distributed edge coloring and a special case of the constructive lovász local lemma. *ACM Trans. Algorithms*, 16(1), 2019.

11 Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the local model. *SIAM J. Comput.*, 48(1):122–143, 2019.

12 Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the local model. *SIAM J. Comput.*, 48(1):33–69, 2019.

13 Yi-Jun Chang, Jan Studený, and Jukka Suomela. Distributed graph problems through an automata-theoretic lens. *arXiv:2002.07659*, 2020.

14 Kai-Min Chung, Seth Pettie, and Hsin-Hao Su. Distributed algorithms for the Lovász local lemma and graph coloring. *Distributed Computing*, 30:261–280, 2017.

15 Manuela Fischer and Mohsen Ghaffari. Sublogarithmic distributed algorithms for Lovász local lemma with implications on complexity hierarchies. In *Proceedings of the 31st International Symposium on Distributed Computing (DISC)*, pages 18:1–18:16, 2017.

16 Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. On derandomizing local distributed algorithms. In *Proceedings of 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 662–673, 2018.

17 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

18 Gary L. Miller and John H. Reif. Parallel tree contraction–Part I: fundamentals. *Advances in Computing Research*, 5:47–72, 1989.
19 Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995.
20 Dennis Olivetti. Brief announcement: Round eliminator: A tool for automatic speedup simulation. In *Proceedings of the 39th Symposium on Principles of Distributed Computing (PODC)*, pages 352–354. ACM, 2020.
21 David Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, 2000.
22 Václav Rozhoň and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2020.