Locally Checkable Labeling Problems in Rooted Trees in the Online-LOCAL Model of Computation

Henrik Lievonen

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 28 February 2022

Supervisor

Prof. Jukka Suomela

Advisor

Dr. Darya Melnyk

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Henrik Lievonen

Title Locally Checkable Labeling Problems in Rooted Trees in the Online-LOCAL Model of Computation

Degree programme Master's Programme in Computer, Communication and Information Sciences

Major Computer Science **Code of major** SCI3042

Supervisor Prof. Jukka Suomela

Advisor Dr. Darya Melnyk

Date 28 February 2022 **Number of pages** 59 **Language** English

Abstract

There are many ways to classify algorithms. Online algorithms, for example, are algorithms that have to be able to handle input one element at a time. Offline algorithms, on the other hand, have access to the whole input. In the case of online graph algorithms, the structure of the underlying graph is fixed. The graph is revealed to the algorithm one node at a time. When a node is revealed, the algorithm has to decide its output for that node, and it cannot change its decision later.

Another way to classify algorithms is to divide them into centralized and distributed algorithms. In the case of graph algorithms, a centralized algorithm is a completely separate entity from the graph. When the nodes (or edges) of the graph are active parties in the execution of the algorithm, the algorithm is called a distributed algorithm. One commonly used model of distributed computation is the LOCAL model. In the LOCAL model, all nodes are computing their own part of the result in parallel. The nodes only see their own local neighborhood and need to base their decision only on this local view.

In this thesis, I introduce the online-LOCAL model, which combines the power of online graph algorithms and LOCAL algorithms. Like online graph algorithms, online-LOCAL algorithms need to react to nodes being revealed one at a time. Unlike online graph algorithms, online-LOCAL algorithms also get to see the local neighborhood around the nodes before needing to make their decisions.

The online-LOCAL model is a very strong model of computation. In general, there are problems that are trivial in the online-LOCAL model, but difficult to solve with online graph algorithms and LOCAL algorithms. In this thesis, I restrict my attention to the class of problems known as locally checkable labeling problems. These are a broad class of problems for which a solution is valid if it looks valid in all local neighborhoods. In particular, I show that for locally checkable labeling problems in rooted regular trees, the online-LOCAL model is approximately as powerful as the LOCAL model.

Keywords Online graph algorithms, Distributed algorithms, Locally checkable labeling problems, LOCAL model, Online-LOCAL model

Aalto University
School of Science

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Henrik Lievonen

Työn nimi Paikallisesti tarkistettavat merkitsemisongelmat juurellisissa puissa online-LOCAL-mallissa

Koulutusohjelma Master's Programme in Computer, Communication and Information Sciences

Pääaine Computer Science **Pääaineen koodi** SCI3042

Työn valvoja Prof. Jukka Suomela

Työn ohjaaja Dr. Darya Melnyk

Päivämäärä 28 February 2022 **Sivumäärä** 59 **Kieli** Englanti

Tiivistelmä

Algoritmeja voidaan luokitella monin eri perustein. Esimerkiksi online-algoritmit ovat algoritmeja, joiden pitää pystyä käsittelemään syötettä alkio kerrallaan. Niiden vastakohta on offline-algoritmit, joilla on pääsy koko syötteeseen. Online-verkkoalgoritmien tapauksessa verkon rakenne on kiinnitetty, mutta verkko paljastetaan algoritmille solmu kerrallaan. Solmun paljastuessa paljastuvat myös siitä lähtevät kaaret aiemmin paljastettuihin solmuihin. Algoritmin täytyy välittömästi päättää tuloste paljastetulle solmulle, eikä se voi enää myöhemmin muuttaa päätöstään.

Toinen tapa luokitella algoritmeja on jakaa ne hajautettuihin ja keskitettyihin algoritmeihin. Verkkoalgoritmien tapauksessa keskitetty algoritmi on verkosta irrallinen toimija. Jos sen sijaan verkon solmut (tai kaaret) ovat aktiivisia toimijoita algoritmin suorituksessa, on kyseessä hajautettu algoritmi. Yksi paljon tutkittu hajautettujen algoritmien malli on LOCAL-malli. LOCAL-mallissa jokainen verkon solmu laskee samanaikaisesti oman osansa ratkaisusta. LOCAL-mallissa algoritmi näkee vain solmun välittömän lähiympäristön, jonka pohjalta algoritmin täytyy päättää ratkaisu solmulle.

Tässä työssä esittelen online-LOCAL-mallin, joka yhdistää online-verkko- ja LOCAL-algoritmien tehokkuuden. Online-algoritmien tavoin online-LOCAL-algoritmien täytyy pystyä ratkaisemaan ongelma solmu kerrallaan. Toisin kuin online-verkkoalgoritmit, online-LOCAL-algoritmit näkevät myös solmun lähiympäristön ennen kuin niiden pitää päättää tuloste solmulle.

Online-LOCAL-malli on erittäin vahva laskennan malli. On olemassa ongelmia, jotka ratkeavat helposti online-LOCAL-mallissa, mutta joita ei pysty ratkaisemaan online- eikä LOCAL-algoritmeilla. Yleensä LOCAL-mallia tutkittaessa rajoittaudutaan kuitenkin paikallisesti tarkistettaviin merkitsemisongelmiin. Ne ovat verkko-ongelmia, joille ratkaisu on kelvollinen mikäli se näyttää kelvolliselta kaikkien solmujen lähiympäristöissä. Tässä työssä osoitan, että online-LOCAL-malli on vain hieman vahvempi kuin LOCAL-malli, kun tarkastelu rajoitetaan paikallisesti tarkistettaviin merkitsemisongelmiin säännöllisissä juurellisissa puissa.

Avainsanat online-verkkoalgoritmit, hajautetut algoritmit, paikallisesti tarkistettavat merkitsemisongelmat, LOCAL-malli, online-LOCAL-malli

Acknowledgements

I would like to thank my advisor Darya Melnyk for all the invaluable feedback I have received both on the technical results and on my writing. Without her help, my thesis would not have reached its current shape.

I would also like to thank my supervisor Jukka Suomela for letting me work as a teaching assistant on many of his courses during my studies. Therefore, it was very natural for him to also supervise my Master's thesis. I am looking forward to starting my doctoral studies in his group.

Many thanks to Amirreza Akbari and Joona Särkijärvi for helping me get the initial details of the proofs right. I will remember our summer together in the research group.

I would also like to thank Chetan Gupta, Juho Hirvonen, Rustam Latypov, Joonatan Saarhelo, Jan Studený and Jara Uitto for the conversations in the hallways and during lunches when those were possible. Thanks to Alkida Balliu, Fabian Kuhn and Dennis Olivetti for our discussions during your research visit at Aalto. I hope that our collaboration will be fruitful in the future.

Finally, I would like to thank my parents and my brother for supporting me in everything I do. Without them, I would not be here.

This work was supported in part by the Academy of Finland, Grant 333837.

Otaniemi, 28 February 2022

Henrik E. M. Lievonen

Contents

1 Introduction

Many natural problems around us can be understood as graph problems. For example, many problems related to logistics and transportation can be easily modelled as graph problems in the transportation network. There are also other problems that do not immediately look like graph problems but that can nevertheless be modelled as such. One such problem is the *channel assignment problem* in cellular networks [20].

For the channel assignment problem, consider a cellular network which consists of individual transmitters. Each transmitter needs to be assigned a channel which it can use for its transmissions. When a transmitter transmits a signal, other devices in its coverage area can receive it. If two transmitters, whose coverage areas overlap, try to transmit on the same channel, their signals interfere, rendering the transmission indecipherable. Therefore, the channels should be assigned to the transmitters such that the signals of two transmitters never interfere.

This problem of channel assignment can be modelled as a graph problem. Each node of the graph represents a transmitter. Two nodes of the graph are connected if their coverage areas overlap, and therefore their signals can interfere. Each node needs to be assigned a channel such that no two nodes that are connected by an edge can have the same channel.

When the channel assignment problem is formulated in this way as a graph problem, it begins to look like the classic *graph coloring problem*. In the coloring problem, the nodes of the graph need to be assigned colors such that no two neighboring nodes can have the same color. Here the colors represent the channels assigned to the transmitters.

The coloring problem has, for example, been studied extensively in the *centralized model* of computation. In the centralized model, an algorithm that is solving a problem has full information of the input. The main limitation of a centralized algorithm is the amount of time it needs to solve the given problem. For example, in the case of the channel allocation problem, a centralized algorithm knows where all the transmitters are and how their coverage areas overlap.

While the centralized model is very powerful, it assumes that the algorithm has full information of the input. For example, in a real cellular network, the transmitters are not static. More transmitters can be added, they can move around, and they may also be removed.

Online graph algorithms are a model which tries to answer to the case where nodes are added one-by-one. The input graph is revealed to an online graph algorithm one node at a time. When a node is revealed, the algorithm needs to immediately choose an output for that node, and the algorithm cannot change its choice later. In the case of channel allocation, the transmitters appear one-by-one. When a node appears, the algorithm needs to immediately assign it a permanent channel such that no current or future neighbor transmitter shares the same channel.

Both the centralized algorithms and the online graph algorithms presented above are centralized in the sense that a single entity is doing all the computations. This creates a single point of failure whose crash would cause the whole network to malfunction.

It is possible to alleviate the problem of centralization using *distributed algorithms*. In the distributed model, the nodes of the graph are doing the computation themselves instead of there being a single centralized entity doing it on their behalf. One much-studied distributed model is *the LOCAL model* of computation. In the LOCAL model, each node is running its own instance of the algorithm and needs to decide its own output. Each node needs to base its decision only on the structure of its local neighborhood. In particular, this neighborhood includes all nodes and edges that are within distance T of the node. The distance T is called the *locality* of the algorithm. The computation in the LOCAL model is fully parallelized, and the nodes do not share any memory.

In general, it is known that LOCAL algorithms cannot find the minimum coloring of a graph, unless the nodes of the network can see (almost) the whole network. Nevertheless, there exist efficient LOCAL algorithms for finding a $(\Delta + 1)$-coloring of a graph, where Δ is the maximum number of neighbors a node can have. While it is not possible to solve the $(\Delta + 1)$-coloring problem with a constant locality, already increasing the locality to $O(\log^* n)$ suffices. The function $\log^*$ is the iterated logarithm which is a very slowly growing function.

In this thesis, I study what happens if the online graph algorithms are empowered by giving them access to local information. In particular, I present *the online-LOCAL model* of computation. It differs from the online graph algorithms by letting the algorithm see the neighborhoods around the nodes before needing to decide their outputs.

It is pretty easy to see that the online-LOCAL model is at least as powerful as the LOCAL model. This is because an online-LOCAL algorithm could simulate a LOCAL algorithm in the neighborhood it sees to decide the output for the node. The main question is if the online-LOCAL model is more powerful, and if it is, by how much?

The answer turns out to be yes. Already in 2017, Ghaffari et al. [14] shortly considered the online-LOCAL model when introducing *the sequential-LOCAL model* but deemed it too powerful to be of interest. This is because an online-LOCAL algorithm can select a leader node in the graph with locality $T = 0$ while a LOCAL algorithm needs to see almost the whole graph to select a leader.

Contrary to their claim, I show in this thesis that the online-LOCAL model is not *too strong*. In particular, I show that the online-LOCAL model is approximately as powerful as the LOCAL model in rooted regular trees for the broad class of graph problems called *locally checkable labeling* (LCL) problems. These are problems for which a labeling of nodes is valid if it looks valid in all local neighborhoods. The coloring problem is, for example, an LCL problem because a coloring is valid when the coloring looks valid around all nodes. In particular, if no node of the graph has a neighbor with the same color, the coloring is valid.

One interesting consequence of the LOCAL and the online-LOCAL models being approximately equal for LCL problems in rooted trees is that it implies similar equality for other models too. In particular, the sequential-LOCAL [14] and the dynamic-LOCAL [1] models are also equal to these models. The sequential-LOCAL model is similar to the online-LOCAL model. The only difference is that the nodes

Table 1: Relation of complexity classes of LCL problems in directed paths and rooted trees in the LOCAL and the online-LOCAL models.

	LOCAL		online-LOCAL
LCL problems in directed paths	$O(\log^* n)$	$\Leftrightarrow$	$O(1)$
	$\Theta(n)$	$\Leftrightarrow$	$\Theta(n)$
LCL problems in rooted regular trees	$O(\log^* n)$	$\Leftrightarrow$	$O(1)$
	$\Theta(\log n)$	$\Leftrightarrow$	$\Theta(\log n)$
	$n^{\Theta(1)}$	$\Leftrightarrow$	$n^{\Theta(1)}$

do not have any shared global memory. For dynamic-LOCAL, consider a case where the underlying graph is changing over time. A dynamic-LOCAL algorithm is allowed to modify the labels in the neighborhoods of the changes to keep the solution valid.

1.1 Contributions

In this thesis, I provide an almost complete classification of locally checkable labeling problems in directed paths and rooted regular trees. In particular, I prove that in the online-LOCAL model the possible classes of locality for LCL problems in rooted regular trees are $O(1)$, $\Theta(\log n)$ and $n^{\Omega(1)}$. This classification corresponds to the previously known complexity classes $O(\log^* n)$, $\Theta(\log n)$ and $n^{\Omega(1)}$ in the LOCAL model [5]. I also prove that the same classification holds in directed paths, which can be seen as a special case of rooted trees. The main differences are that the complexity class $\Theta(\log n)$ is missing and the class $n^{\Omega(1)}$ simplifies to class $\Omega(n)$ in directed paths. The only missing part of the classification in rooted regular trees is the finer structure inside class $n^{\Omega(1)}$. Table 1 summarizes the main contributions of this thesis.

More formally, I prove the following theorem in directed paths:

Theorem 1.1 (Equivalence in directed paths)**.** *Let Π be an LCL problem in directed paths. If the problem is solvable with locality T in the online-LOCAL model, then it is solvable with $O(T + \log^* n)$ locality in the LOCAL model.*

In rooted trees, I prove the following two theorems which, together with the previously known classification in the LOCAL model [5], provide the full classification:

Theorem 1.2 (Equivalence in rooted trees, super-logarithmic region)**.** *Let Π be an LCL problem in rooted trees. Problem Π requires locality $n^{\Omega(1)}$ in the online-LOCAL model exactly when it requires $n^{\Omega(1)}$ locality in the LOCAL model.*

Theorem 1.3 (Equivalence in rooted trees, sub-logarithmic region)**.** *Let Π be an LCL problem in rooted trees. Problem Π requires locality $\Omega(\log n)$ in the online-LOCAL model exactly when it requires $\Omega(\log n)$ locality in the LOCAL model.*

The results in this thesis have become a part in a pre-published manuscript [1] that I have co-authored. I was primarily responsible for researching and writing up parts corresponding to Sections 4 and 5.

1.2 Roadmap

This thesis is structured as follows. In Section 2, I present previous work that has been done to understand distributed algorithms in the LOCAL and the sequential-LOCAL models. I also present previous work related to understanding online graph algorithms.

In Section 3, I give definitions for the theoretical concepts needed to understand the core results of this thesis. Readers who are familiar with graphs and automata in general are advised to skip Sections 3.1 and 3.2. In Sections 3.3 and 3.4, I first define online graph algorithms and then the main models of distributed computation that are used in this thesis. I recommend all readers to read Section 3.5 where I give the definition for the online-LOCAL model.

In Section 3.6, I give formal definitions for locally checkable labeling problems, and in Definition 3.20, I present a possible way to describe LCL problems that is used later in this thesis. In Section 3.7, I connect the locally checkable labeling problems with automata. At the end, in Section 3.8, I give a definition for the iterated logarithm.

In Section 4, I show that the equivalence between the LOCAL and the online-LOCAL models holds in directed paths by proving Theorem 1.1. In Section 5, I prove that the equivalence holds also for the more general class of rooted regular trees. Before stating the actual proves, I present an example LCL problem in Section 5.1 and show that it requires locality $\Omega(\sqrt{n})$ in the online-LOCAL model. In Section 5.2, I generalize this technique to all LCL problems and prove Theorem 1.2. In Section 5.3, I complete the classification of LCL problems in rooted regular trees by proving Theorem 1.3.

2 Related work

In this section, I provide a brief overview of what is already known about LOCAL algorithms, online graph algorithms and locally checkable labeling problems. I defer more formal definitions of these concepts to Section 3.

2.1 The LOCAL model of distributed computation

Linial [18] initiated the modern study of distributed graph algorithms by introducing *the LOCAL model*. The LOCAL model models a distributed system as a graph where each node represents a computer, and each edge represents a bidirectional communication link. All nodes of the graph are running the same algorithm.

The execution of an algorithm in the LOCAL model proceeds in synchronous rounds. In each round, each node gets to send a message to each one of its neighbors. The nodes then get to update their states based on the messages they have received.

The LOCAL model tries to capture the notion of *locality* by imposing restrictions only on the number of synchronous rounds of message exchange the algorithm is allowed to perform. In particular, the model imposes no restrictions on the amount of computations the algorithm can do, or on the size of messages the nodes exchange.

The number of synchronous round an algorithm is allowed perform is called the locality of the algorithm as, within T synchronous rounds of message exchange, each node can only receive information from distance of at most T. To allow nodes to distinguish themselves in their neighborhoods, the nodes are assigned unique identifiers. Linial was able to prove many lower bounds on the localities needed to color different classes of graphs in the LOCAL model, including that 3-coloring a cycle requires $\Omega(\log^* n)$ locality.

2.2 Locally checkable labeling problems

Naor and Stockmeyer [19] continued to study the LOCAL model, and focused especially on the problems which can be solved in constant locality. To do this, they introduced a class of problems known as *locally checkable labeling* (LCL) problems. These are problems for which the validity of a solution can be checked locally; a solution to an LCL problem is valid if it looks like a valid solution around every node of the graph.

It turns out that many interesting graph problems can be encoded as LCL problems: Vertex and edge coloring problems are canonical examples of LCL problems because for them a solution is valid if no vertex (resp. edge) of the graph is adjacent to another node (resp. edge) with the same color. Maximal matching problem is also an LCL problem because every node can locally check that they belong to at most one match, and that the other node they are matched to agrees on the match. In addition, the unmatched nodes can check that none of their neighbors are unmatched, as otherwise the matching would not be maximal.

Recently, the localities of LCL problems in the LOCAL model have been classified for many natural families of graphs. In paths, cycles and grids the possible localities

are $O(1)$, $\Theta(\log^* n)$ and $\Omega(n)$ [3, 11, 19, 8]. In rooted and unrooted trees the possible localities are $O(1)$, $\Theta(\log^* n)$, $\Theta(\log n)$ and $n^{\Omega(1)}$ in the deterministic LOCAL model [5, 10]. It is known that in the randomized LOCAL model, there exist also LCL problems in unrooted trees with complexities in ranges $\Omega(\log\log^* n) - O(\log^* n)$ and $\Omega(\log\log n) - O(\log n)$ [10].

In recent years, there has been aim to automate the process of deciding the complexity classes of LCL problems. Balliu et al. [4] provided practical algorithms for deciding the complexity of two-label LCL problems in trees. Balliu et al. [3] showed that the complexity of LCL problems in paths and cycles is decidable, albeit PSPACE-hard.

Chang et al. [11] restricted their attention to LCL problems in paths and cycles without inputs and provided efficient algorithms for deciding which complexity class LCL problems belong to. Most of the algorithms they provided run in polynomial time, but a couple of specific problems are NP-complete.

Balliu et al. [5] continued the classification work by classifying all LCL problems without input in rooted regular trees. They defined certificates for LCL problems, and showed that the existence of a certificate implies that the problem is solvable with specific locality. Moreover, they provided exponential-time algorithms for constructing these certificates. In practice, the algorithms they provided are efficient and can be implemented.

In this thesis, I show that the online-LOCAL model is no more powerful than the LOCAL model in solving LCL problems in directed paths and rooted regular trees. In particular, the existing classification results for the LOCAL model extend directly to the online-LOCAL model.

All of the above-mentioned algorithms have been implemented in practice. LCL Classifier[1] is a web-site developed by Tereshchenko [24] for classifying LCL problems in cycles, paths, trees and rooted trees automatically. Anyone can plug in the LCL problem of their choice using a similar formalism as presented in Definition 3.20, and the web-site can automatically tell the complexity of the given problem.

2.3 The sequential-LOCAL model

One of the main hurdles with the LOCAL model is symmetry breaking. To understand the problem, consider a long path of nodes with increasing identifiers and a LOCAL algorithm with constant locality. In the middle of the path, the algorithm only sees that it is in the middle of a path with increasing identifiers. Naor and Stockmeyer [19] showed that in this case the algorithm cannot label the nodes with any other label than a constant. In particular, this means that in cycles and paths, only LCL problems admitting a constant solution are solvable in constant locality in the LOCAL model.

To solve the problem of symmetry breaking, Ghaffari et al. [14] introduced *the sequential-LOCAL model* (also known as the SLOCAL model in the literature). The sequential-LOCAL model differs from the LOCAL model in a key way: The nodes of

[1] `https://lcl-classifier.cs.aalto.fi/`

the graph are processed in a sequential order. Every node has also a local memory. When a node is processed, it sees the structure of its local neighborhood along with the memories of the nodes in its neighborhood. The nodes can use their local memories to communicate with other nodes in their neighborhood.

Symmetry breaking is easy in the sequential-LOCAL model: the symmetry is broken by the order in which the nodes are processed. This makes some problems easier to solve in the sequential-LOCAL model than in the LOCAL model. In fact, any problem solvable with locality $O(\log^* n)$ in the LOCAL model is solvable with locality $O(1)$ in the sequential-LOCAL model.

2.4 Online graph algorithms

Online algorithms are algorithms which need to be able to solve problems without knowing the whole input beforehand. In particular, the input is revealed to the online algorithm one piece at a time. Each time a piece of the input is revealed, the algorithm needs to decide the output for that piece.

Online graph algorithms are online algorithms which operate on graphs. The underlying graph is fixed and is being revealed to the algorithm one node or edge at a time. When the algorithm gets to see a node or an edge, it needs to decide the output for it immediately.

The effectiveness of online algorithms is usually measured by *competitiveness ratio* [23]. The output produced by the algorithm is assigned a *cost* which measures how good or bad the output is; a lower cost is more desirable. An online algorithm is said to have competitiveness ratio c if for any possible sequence of inputs, the cost of the output it produces is at most c times higher than that of an optimal offline algorithm.

There have been some attempts at strengthening online algorithms [2, 15]. Most of them rely on the algorithm being able to defer the choice of its output in time: Sometimes the algorithm is allowed to delay its decisions a fixed amount of time. Other times the algorithm is allowed to dynamically choose for which pieces of the input it wants to defer its output, as long as there are not too many pieces being delayed at the same time. Albers and Schraink [2] have shown that for coloring problems, being able to delay the outputting does not help online algorithms unless the algorithm gets to delay its decisions for at least $\omega(n/\log n)$ nodes.

In this thesis, I study the online-LOCAL model which strengthens the online algorithms in another way. In the online-LOCAL model, the algorithm gets to see the neighborhood of the node before it needs to decide its output. This corresponds to the algorithm being able to look around in space, instead of being able to look ahead in time.

3 Definitions

In this section, I define the key concepts used in this thesis. I start by defining some commonly known concepts of graphs and automatons in Sections 3.1 and 3.2. In Sections 3.3 and 3.4, I define online graph algorithms and the LOCAL model of distributed computation. I then combine these models by introducing the online-LOCAL model in Section 3.5. In Section 3.6, I define locally checkable labeling problems, and in Section 3.7, I show how they relate to automata. Finally, in Section 3.8, I define the iterated logarithm.

3.1 Graphs

The models of computation studied in this thesis model a distributed system as a graph. In this section, I define directed graphs and some key concepts of them. In addition, I define two important families of graphs: directed paths and rooted trees.

Definition 3.1 (Graph)**.** A (directed) graph $G = (V, E)$ consists of a set of vertices V and set of directed edges $E \subseteq V \times V$ connecting them. A directed edge $(u, v) \in E$ consists of a *tail* u and a *head* v.

For every node v of the graph, the *indegree* of v is the number of neighboring edges oriented towards v, that is the number of edges of form (u, v) in E for some $u \in V$. Similarly, the *outdegree* of v is the number of oriented neighboring edges oriented away from v.

Definition 3.2 (Walk)**.** A walk $u \rightsquigarrow v$ in a graph G is sequence of vertices $u = s_1, s_2, \ldots, s_n = v$ such that for every $1 \leq i < n$, edge (s_i, s_{i+1}) belongs to the graph.

Definition 3.3 (Cycle)**.** Let $G = (V, E)$ be a graph. Graph G has a *cycle* if for some node v in V there exists a walk $v \rightsquigarrow v$ with at least one edge. The edge can be (v, v). A graph without a cycle is called *acyclic*.

Definition 3.4 (Rooted tree)**.** A *rooted tree* is an acyclic graph with a distinguished node r such that from every node v, there exist a unique walk $v \rightsquigarrow r$. The distinguished node r is called the *root* of the tree. A rooted tree is *regular* if the indegree of every node is either 0 or δ, where δ is a constant.

Definition 3.5 (Directed path)**.** A rooted tree G is a *directed path* if the indegree of every node is at most one.

Definition 3.6 (Induced subgraph)**.** Let $G = (V, E)$ be a graph, and let U be a subset of nodes V. The *subgraph induced by* U of G, denoted by $G[U] = (U, E')$, is the graph consisting of nodes U, and a subset E' of edges E where both endpoints of every edge reside in U. In other words, $E' = \{(u, v) \mid (u, v) \in E, u, v \in U\}$.

Definition 3.7 (Neighborhood)**.** Let $G = (V, E)$ be a graph, and let v be a node of V. Let $B(v, T)$ be the set of nodes in the radius-T neighborhood of v. It can be defined recursively as

$$B(v,T) = \begin{cases} \bigcup_{u \in B(v,1)} B(u, T-1) & T \geq 2 \\ \{u \mid (v,u) \in E\} \cup \{u \mid (u,v) \in E\} & T = 1 \end{cases}.$$

In other words, set $B(v, T)$ contains the nodes that are within distance T from v.

The *radius-T neighborhood of v* is the induced subgraph $G[B(v, T)]$, together with possible labelings.

3.2 Automata

In Section 3.7, I need nondeterministic unary semiautomata to define path-flexibility for locally checkable labeling problems. In the following, I define them.

The word *unary* in the definition means that the automaton is defined over input alphabet with only one symbol. This can be interpreted as the automaton not having an input. The word *semiautomaton* means that the automaton does not produce output either, and hence the automaton does not have any *accepting states* or a *start state*.

Definition 3.8 (Nondeterministic finite unary semiautomaton [17, p. 57])**.** A nondeterministic finite semiautomaton $\mathcal{M}$ over unary alphabet is a pair (Q, δ) where

1. Q is a finite set of *states*.
2. δ is the *transition function* which that takes a state in Q and returns a subset of Q. The returned subset represents the possible next states of the automaton.

The execution of semiautomaton $\mathcal{M}$ starts at a nondeterministic state q_0 of Q. In each subsequent step $i \geq 1$, the next state q_i of the automaton is chosen in a nondeterministic way from set $\delta(q_{i-1})$.

An automaton can be thought as a directed graph. Each state of the automaton corresponds to a node, and each possible state transition corresponds to an edge. In particular, each node q has outgoing edges to nodes of set $\delta(q)$. In later parts of this thesis, I visualize automata in this way.

3.3 Online graph algorithms

In this thesis, I study the online-LOCAL model of computation. The online-LOCAL model combines the power of online graph algorithms and LOCAL algorithms. In this section, I define online graph algorithms, and in the following two sections, I define the LOCAL model and the online-LOCAL model.

Online graph algorithms are algorithms which need to maintain a valid solution for a problem while an adversary reveals nodes one-by-one. When a node is revealed to the algorithm, the algorithm needs to decide an output label for it. The algorithm can base this decision only on the nodes it has seen so far, the order in which it has

seen the nodes, and the subgraph induced by those nodes. Once the algorithm has decided a label for the node, it cannot be changed anymore. The adversary reveals the next node to the algorithm only after all previously revealed nodes have been labeled.

Definition 3.9 (Online graph algorithms [2])**.** Let $G = (V, E)$ be a graph, and let $\pi = v_1, v_2, \ldots, v_n$ be an ordering of vertices V. Let $\pi_i = v_1, v_2, \ldots, v_i$ be the first i nodes of the sequence, and let

$$G_i = G\Big[\{v_1, v_2, \ldots, v_i\}\Big]$$

be the subgraph of G induced by π_i.

The adversary reveals the nodes one-by-one to the algorithm in the order defined by π. When a node v_i is revealed to the algorithm, the algorithm must decide the output for that node based on only the sequence of nodes π_i and the subgraph G_i induced by those nodes.

3.4 Distributed models of computation

In this section, I define the LOCAL model the sequential-LOCAL model which is an intermediate model between the LOCAL and the online-LOCAL models.

Definition 3.10 (The LOCAL model [18])**.** The LOCAL model, introduced by Linial [18], models a distributed system as a network of computers with unlimited processing capabilities that are connected to each other using unbounded communication links. More formally, the network of computers forms a graph $G = (V, E)$ where each computer is represented by a node, and each communication link by an edge. Each computer of the network is associated a unique identifier from set $\{1, 2, \ldots, \text{poly}(|V|)\}$. The unique identifiers are assigned to the nodes by an adversary.

The execution of a LOCAL algorithm proceeds in synchronous rounds. At the start of the execution, each nodes knows its local input, including its unique identifier, and a polynomial approximation of the size of the input graph. In each round of execution, every node sends a message to each of its neighbors and updates its state based on the messages it receives.

In T rounds of execution, every node collects all information that other nodes in their radius-T neighborhoods have—that is, each node establishes a view of its radius-T neighborhood. The nodes then decide their output based on this view of their neighborhoods. In other words, a LOCAL algorithm is a function which maps local neighborhoods to output labels.

LOCAL algorithms can be designed either from the view point of message passing or from the view point of mappings from neighborhoods to labels. In this thesis, I mostly use the latter view point. For a nice introduction to message passing, I recommend the lecture notes by Hirvonen and Suomela [16, pp. 37–88].

One of the challenges with the LOCAL model is that the nodes cannot break symmetry easily. To allow easier symmetry breaking for algorithms, the sequential-LOCAL model was introduced by Ghaffari et al. [13].

Definition 3.11 (The sequential-LOCAL model [13])**.** Let $G = (V, E)$ be a graph like in the LOCAL model, and let $\pi = v_1, v_2, \ldots, v_n$ be an adversarially-chosen ordering of nodes V. Each node is equipped with local memory. The nodes of the graph are processed in the order given by π. When a node gets processed, it gets to see its radius-T neighborhood along with the memories of all nodes that have been processed earlier in that neighborhood. The node then needs to decide its output and what to store in its memory based on this information.

It is easy to see that anything solvable in T rounds in the LOCAL model is also solvable with locality T in the sequential-LOCAL model: The sequential-LOCAL algorithm can just ignore the memories of other nodes in its neighborhood.

One interesting property of the sequential-LOCAL model, compared to the LOCAL model, is that the complexity class $\Theta(\log^* n)$ collapses to $O(1)$. This comes from the fact that any $o(\log n)$ LOCAL algorithm can be normalized into an algorithm that first finds a distance-k coloring of the graph in $O(\log^* n)$ locality, and then does $O(1)$ local processing. Because distance-k coloring can be found in $O(1)$ locality in the sequential-LOCAL model, any $o(\log n)$-locality LOCAL algorithm can be turned into an $O(1)$-locality sequential-LOCAL algorithm. This is formalized in the following observation:

Observation 3.12. *Let $\mathcal{A}$ be a LOCAL algorithm solving an LCL problem Π with locality $o(\log n)$. There exists a sequential-LOCAL algorithm $\mathcal{A}'$ solving problem Π with locality $O(1)$. This also implies that there exists an $O(\log^* n)$-locality LOCAL algorithm for solving Π.*

3.5 The online-LOCAL model

To unify the study of online and distributed algorithms, we introduced the online-LOCAL model in our manuscript [1]. Algorithms in the online-LOCAL model can be viewed as online graph algorithms which—instead of seeing the graph one node at a time—get to see the graph one neighborhood at a time. In particular, when a node is revealed to the algorithm, it also gets to see the whole neighborhood of that node before needing to decide the output for the node. Equivalently, the algorithm is shown nodes one node at a time, but it can delay its decision until it has seen the whole neighborhood.

Definition 3.13 (The online-LOCAL model [1])**.** Let $G = (V, E)$ be a graph, and let $\pi = v_1, v_2, \ldots, v_n$ be an ordering of vertices V. Let $\pi_i = v_1, v_2, \ldots, v_i$ denote the first i nodes of sequence π, and let

$$G_i = G\Big[\bigcup_{j=1}^{i} B(v_j, T)\Big]$$

be the subgraph induced by the radius-T neighborhoods of these nodes. When the adversary presents node v_i, the algorithm has to label v_i based on π_i and G_i.

Again, it is easy to see that anything solvable with locality T in the sequential-LOCAL model is also solvable with locality T in the online-LOCAL model. It is also easy to come up with a labeling problem that is solvable in the online-LOCAL model with locality 0 but requires $\Omega(n)$ locality in the LOCAL model. One such problem is "Label at least one node with the number of nodes in the graph". However, this problem is not locally verifiable and therefore is not a locally checkable labeling problem (see the next section for the definition).

One interesting property of the online-LOCAL model is that the unique identifiers provide no help:

Lemma 3.14. *Consider an online-LOCAL algorithm $\mathcal{A}$ with locality T. There exists another online-LOCAL algorithm $\mathcal{A}'$ with locality T which does not use identifiers of nodes.*

Proof. Let $\mathcal{A}$ be as in the statement of the lemma. Algorithm $\mathcal{A}'$ relabels all nodes it sees for the first time by consecutive integers. It then runs $\mathcal{A}$ using these new labels as identifiers. As $\mathcal{A}$ must work for any graph in any order and with any assignment of unique identifiers, it must also work correctly when $\mathcal{A}'$ uses it. □

When restricting attention to LCL problems, it is no longer clear that the online-LOCAL model is stronger than the LOCAL model. In fact, I prove in this thesis that the online-LOCAL model is approximately equally powerful to the LOCAL model. In Section 4, I provide a simplified version of the proof shown in our manuscript [1] for directed paths. Note that directed paths can be seen as a special case of rooted trees with $\delta = 1$. In Section 5, I extend the result to hold also in general rooted trees with $\delta \geq 2$.

3.6 Locally checkable labeling problems

The family of graph problems studied in this thesis are *locally checkable labeling* (LCL) problems. The LCL problems can be defined by first defining labeling problems, then restricting those to locally verifiable problems, and finally restricting those to locally checkable problems.

Even though the definition of LCL problems may seem restrictive at first, many natural graph problems can be encoded as LCL problems. These include vertex coloring and edge coloring with constant number of colors, maximal independent set, maximal matching, and vertex and edge cover, among others. Also combinations of LCL problems are LCL problems: "Find a maximal independent set *and* a 3-coloring of the graph", and "Find a vertex cover *or* an edge cover of the graph" are both LCL problems.

In this thesis, the LCL problems are considered to be input-free. In the general case, LCL problems can have inputs, but Balliu et al. [3] have shown that it is PSPACE-hard to decide whether an LCL problem with inputs in paths can be solved in $O(1)$ locality in the LOCAL model, or whether it requires $\Omega(n)$ locality. Nevertheless, Chang et al. [11] have shown that there exist efficient algorithms for deciding the complexity of input-free LCL problems in directed and undirected paths, and Balliu et al. [5] have extended these results for rooted trees.

Definition 3.15 (Labeling problem)**.** Consider a family of graphs $\mathcal{G}$ and an (output) label set Σ. A labeling of a graph $G = (V, E) \in \mathcal{G}$ is a function $L\colon V \to \Sigma$. A *labeling problem* Π associates a set of valid output labelings L for every possible graph $G \in \mathcal{G}$. This assignment of valid output labelings must be invariant under graph isomorphism.

Definition 3.16 (Locally verifiable labeling problem)**.** A labeling problem Π is *locally verifiable* with checking radius r if there exists a set of valid local neighborhoods $\mathcal{T}$ such that L is a valid labeling if and only if, for every node v, the radius-r neighborhood of node v in (G, L) is in $\mathcal{T}$. In other words, a labeling L is valid if every local neighborhood of G looks valid.

Definition 3.17 (Locally checkable labeling problem)**.** A labeling problem Π is *locally checkable* if it is locally verifiable, label set Σ is finite, and the degree of all nodes in all graphs of $\mathcal{G}$ is at most some constant Δ.

A really interesting LCL problem is the sinkless orientation problem. Introduced by Brandt et al. [7], it was the first LCL problem for which an exponential separation between deterministic and randomized LOCAL model could be proved [9]. The problem requires that the edges of the graph are oriented such that no vertex is a sink. In other words, every vertex has at least one outgoing edge. In trees, the leaves are allowed to be sinks.

Not all graph problems are locally checkable labeling problems, though. For example *maximum* independent set is not locally verifiable as checking whether an independent set is the largest possible is a global property of the graph. Another inherently global problem is finding a spanning tree of a graph. This is because it is not possible to locally check whether the formed candidate spanning tree does not contain a loop. It is also difficult to check that all nodes belong to the same spanning tree.

For another example of a non-LCL problem, consider the maximal fractional matching problem. The maximal fraction matching problem requires that each edge of the graph is assigned a nonnegative rational weight. The weight of a node is the sum of the weights of its adjacent edges. The weights must be assigned so that the weight of every node is at most 1, and for every edge, at least one of the endpoints has weight 1. The maximal fractional matching problem is a locally verifiable problem but is not a locally checkable labeling problem because the set of possible weights is not finite.

As a final example, consider the vertex coloring problem. The vertex coloring problem in bounded-degree graphs with finite number of colors is an LCL problem. Allowing the coloring include an infinite number of colors makes the problem a non-LCL. Similarly, relaxing the setting to include graphs having nodes with unbounded degrees also makes the problem a non-LCL. This is because the LCL problems require that the maximum degree of the graph is bounded.

Observe that the checking radius r of an LCL problem can be made 1 without loss of generality:

Observation 3.18 (Checking radius speedup)**.** *Consider an LCL problem* Π *with checking radius* r*. There exists another LCL problem* Π' *with checking radius 1 such that a solution to* Π' *can be turned into a solution of* Π *by a local mapping.*

Proof. The idea is that each label $\sigma \in \Sigma_{\Pi'}$ encodes a radius-r neighborhood of a solution of Π. A labeling L' is a valid solution for Π' if the neighborhoods encoded by labels of adjacent nodes are compatible and the underlying labeling L is valid according to Π.

A valid labeling L' for Π' can be turned into a valid labeling L for Π by removing the information of the neighborhood from the labels of L'. This can be done locally by each node checking its own label in the neighborhood given by L'.

Note that problem Π' is an LCL problem: The underlying family of graphs is the same as for Π, and hence the degrees of nodes are bounded, the labeling is locally verifiable, and the number of possible radius-r neighborhoods is finite and therefore label set $\Sigma_{\Pi'}$ is finite. □

This thesis focuses on solving LCL problems in rooted trees. To allow deciding the solvability of LCL problems in various models of computation, a formal definition for LCL problems in rooted trees is needed. I borrow the definition from Balliu et al. [5]:

Definition 3.19 (LCL problem in rooted trees [5])**.** An LCL problem Π in rooted trees is a triple (δ, Σ, C) where:

- δ is the number of allowed children.
- Σ is a finite set of (output) labels.
- C is a set of tuples of size $\delta + 1$ from $\Sigma^{\delta+1}$ called *allowed configurations*.

Each allowed configuration $(a\colon b_1, b_2, \ldots, b_\delta) \in C$ states that a node with label a is allowed to have children with labels $b_1, b_2, \ldots b_\delta$ in some order. Nodes having more than or less than δ children are unconstrained and any configuration of labels is valid for them; otherwise input labels could be encoded into the structure of the graph.

Note that this definition is compatible with the previous definition of LCL problems, with the minor change that only the configurations of nodes with exactly δ children are constrained.

Any LCL problem in rooted trees can be described by listing the set of allowed configurations; the number of children δ and the label set Σ can be inferred from the configurations. For the purposes of describing concrete LCL problems, the following succinct notation for listing configurations is adopted from LCL Classifier tool [24], which in turn adopts it from the notation used by the Round Eliminator tool developed by Olivetti [21].

Definition 3.20 (LCL notation)**.** An LCL problem Π is described by a set of lines where each line encodes one or more allowed configuration. The line

```
a: bc de
```

reads as "nodes having label `a` can have their children be labeled with any combination of labels containing either `b` or `c`, and either `d` or `e`." More concretely, this includes the following configurations:

$$(\texttt{a: bd}) \quad (\texttt{a: be}) \quad (\texttt{a: cd}) \quad (\texttt{a: ce})$$

For example, the 3-coloring problem in binary trees can be encoded as

```
A: BC BC
B: AC AC
C: AB AB
```

Here a node with label `A` can have its children be labeled with any combination of labels `B` and `C`. In particular, a node with label `A` cannot have a child labeled with `A`. The other two lines provide similar restrictions for nodes with labels `B` and `C`.

Maximal matching in rooted trees with $\delta = 3$ can be encoded as

```
D: U O0 O0
U: O0 O0 O0
O: D O0 O0
```

Here label `D` denotes that a node is matched with its child, label `U` denotes that a node is matched with its parent, and label `O` denotes that a node is unmatched. Note that configuration (`O: OOO`) is omitted as a node with such configuration could be matched with one of its children and hence the matching would not be maximal.

3.7 LCLs as automata: path-form and path-flexibility

There are many ways to study the properties of LCL problems. One of the more interesting ones was introduced by Chang et al. [11]. They showed how LCL problems in paths and cycles can be turned into finite automata, and how the properties of the automata can be used to infer the complexity of the corresponding LCL problem.

In this thesis, I study rooted trees and not just paths. Nevertheless, the tools developed for paths can also be extended to rooted trees by defining a *path-form* of an LCL problem. The path-form can be seen as a relaxation of the original problem defined in rooted trees; instead of looking at all of the children together, in the path-form every child of a node is considered separately. In other words, every child of a node must have a compatible label with their parent, but the combination of the labels of the children does not matter. It is therefore clear that any lower bound for the path-from also extends to become a lower bound for the original problem.

Definition 3.21 (Path-form of an LCL problem [5])**.** Let $\Pi = (\delta, \Sigma, C)$ be an LCL problem in rooted trees. The *path-form* of Π is an LCL problem $\Pi^{\text{path}} = (1, \Sigma, C')$ in directed paths, where a configuration $(a\colon b)$ belongs to C' if and only if there exists a configuration $(a\colon b_1, b_2, \ldots, b_\delta)$ in C with $b = b_i$ for some i.

The path-form of an LCL problem constructed this way can be turned into a nondeterministic finite automaton over unary alphabet. The intuition is that a path is correctly labeled if and only if the automaton can visit the states in the same order as they appear on the path.

Definition 3.22 (Automaton associated with path-form of an LCL problem [11])**.** Let $\Pi = (\delta, \Sigma, C)$ be an LCL problem, and let Π^{path} be the corresponding path-form. Automaton $\mathcal{M}(\Pi)$ associated with the path-form of Π is a nondeterministic unary semiautomaton. The set of states is Σ, and the automaton has a transition $a \to b$ if configuration $(a\colon b)$ is a configuration of Π^{path}.

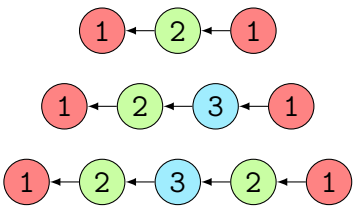
An important property of a state of an automaton is *flexibility*. The flexibility of a state corresponds to *path-flexibility* of a label of an LCL.

Definition 3.23 (Flexible state of an automaton [8])**.** Let σ be a state of automaton $\mathcal{M}$. State σ is a *flexible state* if there exists a constant K such that for every $d \geq K$, there exists a walk $\sigma \rightsquigarrow \sigma$ in $\mathcal{M}$ with length d.

Definition 3.24 (Path-flexibility [5])**.** Let $\Pi = (\delta, \Sigma, C)$ be an LCL problem in rooted trees. A label σ in Σ is *path-flexible* if σ is a flexible state of automaton $\mathcal{M}(\Pi)$. Otherwise, the label is said to be *path-inflexible*.

The intuition behind the concept of path-flexibility relates to how the label can be used when labeling a long path. If a label is path-flexible, then it can be used to label nodes that are at least distance K apart without knowing the exact distance between them. This is because for any distance of at least K, there exists a sequence of transitions in the automaton from the state corresponding to the label back to itself with that length. Therefore, the path can be labeled with labels corresponding to the visited states. If, on the other hand, a label is path-inflexible, then two nodes having that label can exist only at certain possible distances from each other.

For example, consider the problem of 2-coloring a path with labels 1 and 2. The corresponding automaton is visualized in Figure 1, along with the automaton for the 3-coloring problem. Both of the labels are path-inflexible. This means that no matter how far two nodes with label 1 (or 2) are from each other, the distance between them must be even. On the other hand, the labels of 3-coloring problem are path-flexible. For example, all the following 3-colorings of a path are valid:



It is easy to convince oneself that for any distance of at least 2, there exists a valid 3-coloring of a path having nodes with label 1 that far apart.

The path-flexibility of a label can be extended to include two labels. The intuition is the same as with path-flexibility of individual labels; both of the labels of a path-flexible pair can be used to label far-away nodes without knowing their exact distance.

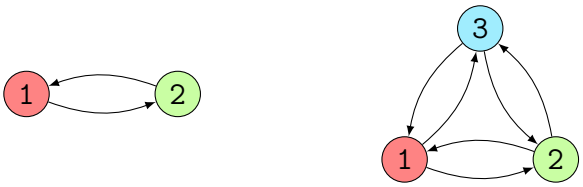


Figure 1: Automata associated with the 2-coloring and the 3-coloring problems. Both of the states in the left automaton are inflexible because there exists only walks with even length back to themselves. In the right automaton, all of the states are flexible because for every length of at least 2, there exist a path with that length from any state back to itself.

Lemma 3.26 relates the path-flexibility of a pair to the path-flexibility of individual labels forming the pair.

Definition 3.25 (Path-flexible pair)**.** Let Π be an LCL problem in rooted trees. A label pair (σ_1, σ_2) is path-flexible if there exists some constant K such that for every $d \geq K$, there exists walks $\sigma_1 \rightsquigarrow \sigma_2$ and $\sigma_2 \rightsquigarrow \sigma_1$ in $\mathcal{M}(\Pi)$ such that the lengths of both walks are d. Otherwise the pair is path-inflexible.

Lemma 3.26. *Let Π be an LCL problem in rooted trees. A label pair (σ_1, σ_2) is path-flexible if and only if both σ_1 and σ_2 are separately path-flexible and belong to the same strongly connected component of $\mathcal{M}(\Pi)$.*

Proof. It is easy to see that if both σ_1 and σ_2 belong to the same connected component and are path-flexible, then the label pair (σ_1, σ_2) is also path-flexible. If the labels do not belong to the same strongly connected component, then at least one of paths $\sigma_1 \rightsquigarrow \sigma_2$ or $\sigma_2 \rightsquigarrow \sigma_1$ does not exist, and hence the pair cannot be path-flexible.

The only case left to show is that if the pair is path-flexible, then both σ_1 and σ_2 are separately path-flexible. For contradiction, assume w.l.o.g. that σ_1 is path-inflexible but pair (σ_1, σ_2) is path-flexible. Then there exists some constant K such that for every $d_1, d_2 \geq K$ there exists walks $\sigma_1 \rightsquigarrow \sigma_2$ and $\sigma_2 \rightsquigarrow \sigma_1$ with lengths d_1, d_2, respectively. But then for every $d \geq 2K$, there exists a walk $\sigma_1 \rightsquigarrow \sigma_2 \rightsquigarrow \sigma_1$ with length exactly d by keeping walk $\sigma_1 \rightsquigarrow \sigma_2$ constant and varying the length of walk $\sigma_2 \rightsquigarrow \sigma_1$ accordingly. This means that σ_1 is path-flexible, and this is a contradiction. Hence the lemma must be true. □

The final lemma regarding path-flexibility states that if there exist nodes with path-inflexible labels, then it is easy to find a way to connect the nodes in such a way that the labeling cannot be completed. Hence if an algorithm labels two distant nodes with labels that form a path-inflexible pair, an adversary can force the algorithm to fail by making sure that the distance between the nodes is such that the path connecting the nodes cannot be labeled.

Lemma 3.27. *Let Π be an LCL problem in rooted trees, and let (σ_1, σ_2) be a path-inflexible pair in Π. Then for every pair of constants $p_1, p_2 \in \mathbb{N}$, not all of the following walks can exist in $\mathcal{M}(\Pi)$:*

I *a walk* $\sigma_1 \rightsquigarrow \sigma_2$ *of length* p_1,
II *a walk* $\sigma_1 \rightsquigarrow \sigma_2$ *of length* $p_1 + 1$,
III *a walk* $\sigma_2 \rightsquigarrow \sigma_1$ *of length* p_2, *and*
IV *a walk* $\sigma_2 \rightsquigarrow \sigma_1$ *of length* $p_2 + 1$.

Proof. Assume for contradiction that all of the walks existed. Then the claim is that there exists some K such that for every $d \geq K$ there exists walks $\sigma_1 \rightsquigarrow \sigma_2$ and $\sigma_2 \rightsquigarrow \sigma_1$ of length exactly d, and therefore (σ_1, σ_2) is a path-flexible pair.

Such walks can be constructed. Consider the following compositions of walks from σ_1 back to σ_1:

W_1 Walk along I to go from σ_1 to σ_2, and then take walk III to go back to σ_1.
W_2 Walk along I to go from σ_1 to σ_2, and then take walk IV to go back to σ_1.

Walk W_1 has length $p_1 + p_2$, and walk W_2 has length $p_1 + p_2 + 1$. Combining walks W_1 and W_2 repeatedly makes it possible to construct a walk from σ_1 back to itself for any length at least $K' = (p_1 + p_2)(p_1 + p_2 - 1)$ [22].

A similar construction can be used to construct a walk from σ_2 back to itself for any length at least K'. Combining these with walks I and III, it is possible to construct walks $\sigma_1 \rightsquigarrow \sigma_2$ and $\sigma_2 \rightsquigarrow \sigma_1$ for any length d at least $K = K' + \max(p_1, p_2)$. Hence pair (σ_1, σ_2) is path-flexible. □

3.8 Iterated logarithm

Iterated logarithm is a very slowly growing function that is often encountered in the analysis of LOCAL algorithms. It appears especially in cases where the sizes of sets can be repeatedly made exponentially smaller, such as in the color reduction algorithm by Cole and Vishkin [12]. The iterated logarithm measures how many times a logarithm needs to be taken before the value reaches 1.

Definition 3.28 (Iterated logarithm $\log^*$)**.** Iterated logarithm $\log^*$ is a function from positive real numbers $\mathbb{R}_+$ to natural numbers $\mathbb{N}$ defined recursively as follows:

$$\log^* n = \begin{cases} 0 & \text{if } n \leq 1 \\ 1 + \log^*(\log_2 n) & \text{otherwise} \end{cases}$$

For all practical purposes, the value of iterated logarithm can be considered to be a constant. In particular, $\log^* n \leq 5$ for all $n \leq 2^{65536}$.

4 Locally checkable labeling problems in directed paths

In this section, I study the locality of LCL problems in the online-LOCAL model by looking at directed paths. The directed paths are an important special case of rooted trees: They are rooted trees where every node has at most one child.

In directed paths, it is well known that the possible complexities of locality for LCL problems in the traditional LOCAL model are $O(1)$, $\Theta(\log^* n)$ and $\Omega(n)$ [3]. In this section, I prove that an online-LOCAL algorithm with locality $o(n)$ implies the existence of a LOCAL algorithm with locality $O(\log^* n)$. This, combined with Observation 3.12, shows that the only possible locality complexities in directed paths in the online-LOCAL model are $O(1)$ and $\Omega(n)$. Moreover, the LCL problems solvable in locality $O(1)$ in the online-LOCAL model coincide with the problems that are solvable in $O(\log^* n)$ in the LOCAL model.

Formally, I prove the following theorem:

Theorem 1.1 (Equivalence in directed paths)**.** *Let Π be an LCL problem in directed paths. If the problem is solvable with locality T in the online-LOCAL model, then it is solvable with $O(T + \log^* n)$ locality in the LOCAL model.*

The theorems and proofs in this section are simplifications from our manuscript [1]. In particular, only LCL problems without inputs and in directed paths are considered. Moreover, any constraints near the ends of the paths are ignored. This is because I want to be more consistent with the definition of LCL problems in rooted trees. For the more complicated proof that includes both cycles and paths, special conditions for endpoints, and inputs, I refer the reader to read our manuscript [1].

Before moving to prove Theorem 1.1, I define what I mean by LCL problems in directed paths.

Definition 4.1 (LCL problems in directed paths)**.** An LCL problem Π in directed paths is a pair (Σ, C) where:

- Σ is a finite set of labels.
- C is a set of pairs from Σ^2 called *allowed configurations*.

Each allowed configuration $(a\colon b) \in C$ states that a node with label a can have a *predecessor* with label b.

Note that this definition orders the labels in the opposite order to the definition of the edges. Consider two nodes u and v connected by an edge (u, v). Configuration $(a\colon b)$ means that node u can be labeled with b and node v can be labeled with a. I do this in order to make this definition compatible with the definition of LCL problems in rooted trees, presented in Definition 3.19.

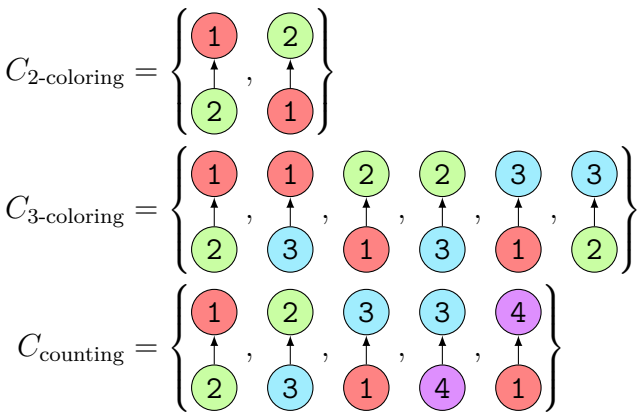


Figure 2: Visualizations of allowed states for the 2-coloring, the 3-coloring, and the counting problems.

Consider the counting problem that is defined as follows:

1: 2
2: 3
3: 14
4: 1

In the counting problem, every node labeled with 1 must be separated by distance 3 or 4. This constraint is encoded by ensuring that the labels count to either 3 or 4 before another 1 is allowed to occur. Figure 2 visualizes the allowed configurations for the counting problem along with the configurations for the 2-coloring and the 3-coloring problems.

The 2-coloring problem requires $\Omega(n)$ locality in both the LOCAL and the online-LOCAL models while the 3-coloring and the counting problem are solvable in $O(\log^* n)$ locality in the LOCAL model and in $O(1)$ locality in the online-LOCAL model.

Instead of proving Theorem 1.1 directly, I prove the following lemma:

Lemma 4.2 (LOCAL simulation of sub-linear online-LOCAL algorithms)**.** *Let* Π *be an LCL problem in directed paths, and let* $\mathcal{A}$ *be an online-LOCAL algorithm solving problem* Π *with locality* $T(n) = o(n)$*. Then, there exists a LOCAL algorithm* $\mathcal{A}'$ *solving* Π *with locality* $O(\log^* n)$*.*

It is easy to see that this directly implies Theorem 1.1: If the problem requires $\Omega(n)$ locality in the online-LOCAL model, then it clearly does so also in the LOCAL model. On the other hand, if the problem is solvable with sub-linear locality in the online-LOCAL model, it can be solved with locality $O(\log^* n)$ in the LOCAL model. This then implies that the same problem is solvable with locality $O(1)$ in the online-LOCAL model.

The high-level idea of the proof is to use the sub-linear-locality online-LOCAL algorithm to find a label that can be used to label far-away nodes on the path such that the labeling is always completable. A LOCAL algorithm can then shatter the path into approximately constant-length segments and label the endpoints of those segments with this label. The choice of the label ensures that the labeling on the connecting segments is completable.

Before the proof, I formalize the shattering that the LOCAL algorithm does by defining ruling sets.

Definition 4.3 ((α, β)-ruling set [5])**.** An (α, β)-ruling set R of a graph $G = (V, E)$ is a subset of nodes V such that for every pair of nodes v, u in R such that $v \neq u$, it holds that $\text{dist}(v, u) \geq \alpha$, and for every node v in $V \setminus R$, there exists at least one node u in R such that $\text{dist}(v, u) \leq \beta$.

The ruling set is useful because nodes belonging to the set are apart at a distance of at least α, but they cannot be too far apart. This is formalized in the following observation:

Observation 4.4. *The distance between two elements of an (α, β)-ruling set is at most $\alpha + \beta$.*

Lemma 4.2 can now be proven.

Proof of Lemma 4.2. Let $\Pi = (\Sigma, C)$ be an LCL problem in directed paths, and let $\mathcal{A}$ be an online-LOCAL algorithm solving Π with locality $T(n) = o(n)$. Then there exists some constant n_0 such that for every $n > n_0$ the following holds

$$(|\Sigma| + 2)(2T(n) + 3) < n.$$

Fix such n, and let $\beta = T(n) + 1$.

Construct a graph P consisting of $|\Sigma| + 1$ disjoint directed path segments, each of length $2\beta + 1$, and let the middlemost nodes on those paths be v_i. Note that the size of graph P is at this point $(|\Sigma| + 1)(2\beta + 1) < n$. Execute algorithm $\mathcal{A}$ on nodes v_i in an arbitrary order. The structure of graph P at this point is under-specified, but the execution of algorithm $\mathcal{A}$ on nodes v_i is well-defined. This is because the locality of $\mathcal{A}$ is $\beta - 1$ but the length of the path segments is $2\beta + 1$, and therefore the algorithm does not see the endpoints of the segments.

The algorithm must produce some labels for all nodes v_i. By the pigeonhole principle, at least two nodes are labeled with the same label σ. Label σ is the canonical label which the LOCAL algorithm can use to label far-away nodes of the graph. In the following, I present a three-step LOCAL algorithm $\mathcal{A}'$ for solving Π with locality $O(\log^* n)$.

1. Algorithm $\mathcal{A}'$ starts by constructing a distance-2β coloring of the path; this can be done with locality $O(\log^* n)$, as shown by Cole and Vishkin [12]. The algorithm then uses this coloring to construct a $(2\beta, 2\beta)$-ruling set R of the input graph. This can be done with constant locality by using the above-constructed coloring as a schedule for the greedy algorithm.

2. Algorithm $\mathcal{A}'$ labels the nodes in the ruling set R with σ.

3. By the construction of label σ, there exists a valid labeling for the segments between the nodes of the ruling set R. There is only a constant number of possible lengths that a path segment connecting two nodes of R can have. Hence the labelings for all different path lengths can be constructed offline, and algorithm $\mathcal{A}'$ can recall the labeling with constant locality.

   To see why such labelings exist, consider two nodes u and v that are adjacent in the ruling set R. The distance between nodes u and v is at least 2β but at most 4β. Let u' and v' be two nodes that algorithm $\mathcal{A}$ labeled with label σ in graph P. Identify u and v with u' and v', respectively. Add nodes between u' and v' such that their distance is equal to the distance of u and v. Because the distance is at least 2β, the radius-$(\beta - 1)$ neighborhoods of u' and v' do not change and hence the execution of $\mathcal{A}$ remains valid. Now algorithm $\mathcal{A}$ must be able to produce a valid labeling for the path connecting u' and v'. As this path is isomorphic with the path connecting u and v, the labeling must also be valid in the original graph.

The existence of algorithm $\mathcal{A}'$ is sufficient to prove the lemma. □

This lemma directly implies Theorem 1.1, thus concluding this section. In the next section, I classify all LCL problems in the more general case of rooted regular trees with $\delta \geq 2$.

5 Locally checkable labeling problems in rooted regular trees

In this section, I show that all LCL problems in rooted regular trees fall into one of the following complexity classes in the online-LOCAL model: $O(1)$, $\Theta(\log n)$, or $n^{\Omega(1)}$. I base my results on the previous work by Balliu et al. [5] who have shown that in the LOCAL model, the possible complexity classes for LCL problems in rooted regular trees are $O(1)$, $\Theta(\log^* n)$, $\Theta(\log n)$, and $n^{\Omega(1)}$. The upper bounds for the LOCAL model presented in their paper directly give upper bounds for the online-LOCAL model. I extend their proofs for the lower bounds to work also in the online-LOCAL model. This shows that the complexity classes for LCL problems in both models are exactly the same, apart from classes $O(1)$ and $\Theta(\log^* n)$ that are collapsed into $O(1)$ in the online-LOCAL model, as per Observation 3.12. Note that the result also directly applies to the sequential-LOCAL model.

Formally, I show the following two theorems:

Theorem 1.2 (Equivalence in rooted trees, super-logarithmic region)**.** *Let Π be an LCL problem in rooted trees. Problem Π requires locality $n^{\Omega(1)}$ in the online-LOCAL model exactly when it requires $n^{\Omega(1)}$ locality in the LOCAL model.*

Theorem 1.3 (Equivalence in rooted trees, sub-logarithmic region)**.** *Let Π be an LCL problem in rooted trees. Problem Π requires locality $\Omega(\log n)$ in the online-LOCAL model exactly when it requires $\Omega(\log n)$ locality in the LOCAL model.*

This section is structured as follows: In Section 5.1, I present an example LCL problem and show that it requires locality $\Omega(\sqrt{n})$ in the online-LOCAL model. I then generalize the proof technique to all LCL problems and prove Theorem 1.2 in Section 5.2. Both of these sections rely on the path-flexibility defined in Section 3.7. In Section 5.3, I show that all problems solvable in $o(\log n)$ locality in the online-LOCAL model are solvable in $O(\log^* n)$ locality in the LOCAL model. This can be combined with Observation 3.12 to get a locality-$O(1)$ online-LOCAL algorithm.

5.1 $2\frac{1}{2}$-coloring requires $\Omega(\sqrt{n})$ locality in the online-LOCAL model

In this section, I present an example LCL problem that requires $\Omega(\sqrt{n})$ locality in the online-LOCAL model. The problem is so-called $2\frac{1}{2}$-coloring problem, first introduced by Chang and Pettie [10].

Informally, the $2\frac{1}{2}$-coloring problem requires 2-coloring the tree near the root with labels `A` and `B`, and near the leaves with `1` and `2`. The different 2-colored parts can be combined using an intermediate label `X`. Figure 3 visualizes the allowed configurations for the $2\frac{1}{2}$-coloring problem. I define the $2\frac{1}{2}$-coloring problem formally in the following.

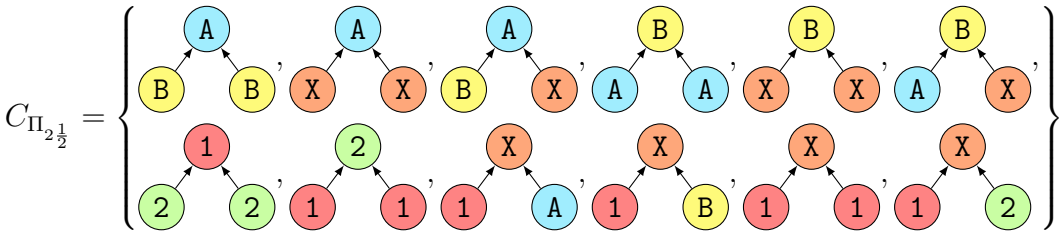


Figure 3: Allowed configurations for the $2\frac{1}{2}$-coloring problem $\Pi_{2\frac{1}{2}}$. Only the first row of configurations are allowed for the restricted $2\frac{1}{2}$-coloring problem $\Pi'_{2\frac{1}{2}}$.

Definition 5.1 ($\Pi_{2\frac{1}{2}}$)**.** $2\frac{1}{2}$-coloring is defined in the formalism presented in Definition 3.20 as

```
A: BX BX
B: AX AX
X: 1 AB12
1: 2 2
2: 1 1
```

The reason for naming the problem $2\frac{1}{2}$-coloring is that any valid 2-coloring is also a valid $2\frac{1}{2}$-coloring, and any valid $2\frac{1}{2}$-coloring can be transformed into a valid 3-coloring by mapping `A` $\mapsto$ `1` and `B` $\mapsto$ `2`. Hence the $2\frac{1}{2}$-coloring problem is at least as hard as the 3-coloring problem, but at most as hard as the 2-coloring problem.

By analyzing the problem description of $\Pi_{2\frac{1}{2}}$, and the automaton associated with its path-form shown in Figure 4, the following observation can be made:

Observation 5.2. *The labels* `1` *and* `2` *are path-inflexible in problem* $\Pi_{2\frac{1}{2}}$*. This is because any path between two nodes labeled with labels* `1` *and* `2` *must form a valid 2-coloring. Denote the set of path-inflexible labels by*

$$\Sigma_1 = \{\texttt{1}, \ \texttt{2}\}.$$

Problem $\Pi_{2\frac{1}{2}}$ can be restricted by removing these labels from the set of possible labels. This gives the following restricted problem $\Pi'_{2\frac{1}{2}}$:

Definition 5.3 ($\Pi'_{2\frac{1}{2}}$)**.** The restricted $2\frac{1}{2}$-coloring is defined in the formalism presented in Definition 3.20 as

```
A: BX BX
B: AX AX
X:
```

Here again analyzing the problem description of $\Pi'_{2\frac{1}{2}}$, and the automaton associated with its path-form shown in Figure 4, gives rise to the following observation:

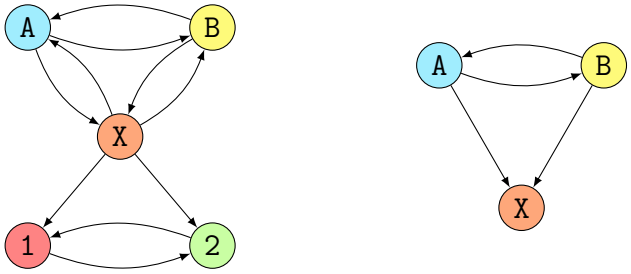


Figure 4: The automaton associated with path form of problem $\Pi_{2\frac{1}{2}}$ on the left, and the automaton associated with the path from of $\Pi'_{2\frac{1}{2}}$ on the right. In the automaton on the left, the states `A`, `B` and `X` are path-flexible as there exist walks from them back to themselves for all lengths of at least 2. The states `1` and `2` are path-inflexible as only self walks with even lengths are possible. On the right, all states `A`, `B` and `X` are path-inflexible. States `A` and `B` have only even-length self-walks. The state `X` has no self-walk.

Observation 5.4. *All labels `A`, `B` and `X` are path-inflexible in problem $\Pi'_{2\frac{1}{2}}$. This is because the label `X` cannot be used to label any internal nodes of the tree, and therefore problem $\Pi'_{2\frac{1}{2}}$ is (almost) a regular 2-coloring problem with labels `A` and `B`. Denote the set of inflexible labels by*

$$\Sigma_2 = \{\texttt{A},\ \texttt{B},\ \texttt{X}\}.$$

The lower-bound construction for $2\frac{1}{2}$-coloring relies on the fact that an algorithm solving $\Pi_{2\frac{1}{2}}$ cannot use labels `1` and `2` to label two nodes without seeing how the nodes are connected to each other. Otherwise, the adversary could alter the length of the path connecting the nodes such that the labeling cannot be completed. This is because labels `1` and `2` are path-inflexible, as noted in Observation 5.2.

Nevertheless, any sufficiently local algorithm can be forced to use labels `1` and `2` in far-away parts of the graph using the following trick: Show the algorithm two fragments of the graph such that the algorithm does not know how they are connected. Based on the previous observation, the algorithm must use labels `A`, `B` and `X` to label the shown nodes. By Observation 5.4, the labels `A`, `B` and `X` are path-inflexible in the restricted $2\frac{1}{2}$-coloring problem $\Pi'_{2\frac{1}{2}}$. Hence the adversary can connect the fragments in such a way that the no valid labeling satisfying the restricted problem $\Pi'_{2\frac{1}{2}}$ exists for the path connecting the two fragments, and therefore the algorithm must use labels `1` and `2` somewhere along the connecting path. The construction can be repeated to create another node with label `1` or `2`. By being careful with the construction, it can be ensured that the algorithm has not seen how the two nodes having labels `1` and `2` are connected to each other, and hence the adversary can force the distance between them to be such that no valid labeling for $\Pi_{2\frac{1}{2}}$ exists for the resulting tree.

Informally, the components that are shown to the algorithm are formed in the following way: Let P be a directed path with x nodes from t to s, and let c be the middlemost node on the path. Collectively call the nodes in path P *layer-2 nodes*, and call node t the *connector node*. Identify each node in path P with the root of another path with $x + 1$ nodes, and call the newly-created nodes *layer-1 nodes*. As

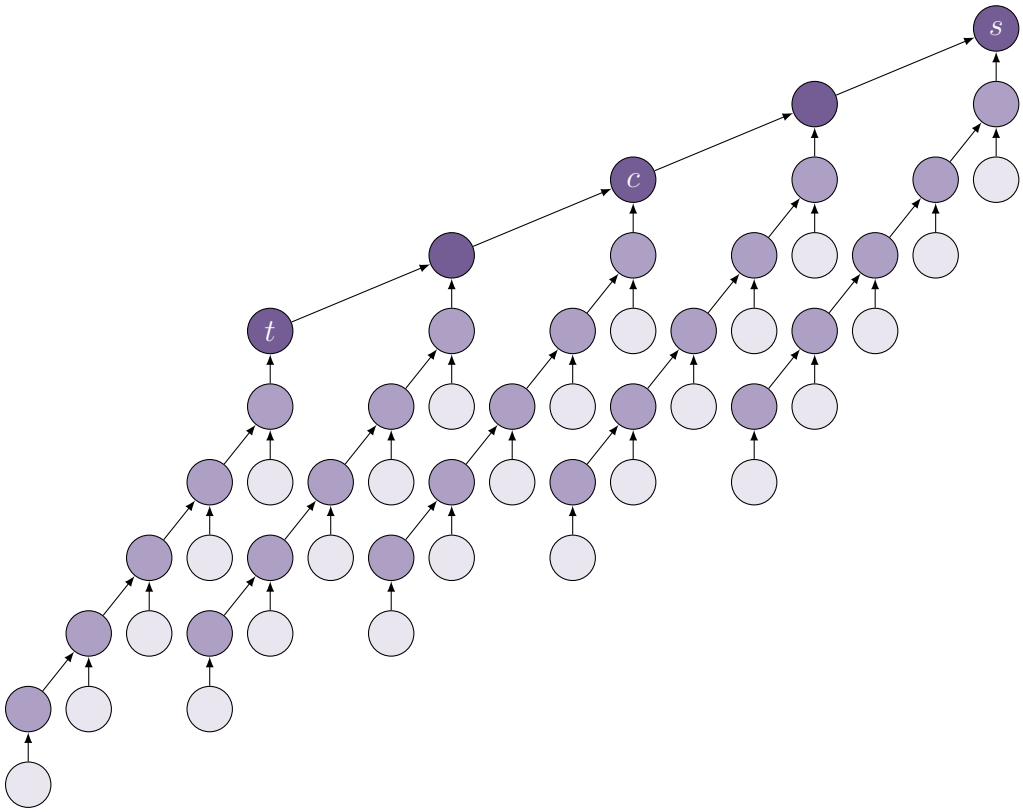


Figure 5: An example of a layered tree T_2^5. The darkest purple nodes belong to layer 2, the middle purple nodes belong to layer 1, and the lightest leaf nodes belong to layer 0. Node s is the root, node t is the connector node, and node c is the middlemost node on the core path.

the last step, for each layer-1 node add one child node, and call these nodes *layer-0 nodes*. The tree formed in this way is called T_2^x.

I illustrate the lower bound construction in this section using an algorithm with locality $T(n) = 1$ and path length $x = 5$. The values are chosen to be small enough for illustrations to fit on a page. Figure 5 shows an example of tree T_2^5.

More formally, the layered tree is a form of a *bipolar tree* and is formed using *tree layering*, both of which I define next:

Definition 5.5 (Bipolar tree [5])**.** A rooted tree T is a *bipolar tree* with pole nodes s_T and t_T if the following holds:

1. s_T is the root of T, and
2. the indegree of t_T is $\delta - 1$, that is node t_T is missing one child.

The unique path connecting s_T and t_T is the *core path* of T. Node t_T is called the *connector node* of T.

Definition 5.6 (Tree layering [5])**.** Given a rooted tree T, define tree layering $\oplus^x T$ as follows: Construct a path $P = (v_1, v_2, \ldots, v_x)$ and $x(\delta - 1)$ copies of T. For each node v_i, identify $\delta - 1$ of its children with a root of a copy of T such that every v_i has exactly $\delta - 1$ copies of T as its children. Note that the last node v_x has one missing child. To make the tree bipolar, nodes $s_{\oplus^x T}$ and $t_{\oplus^x T}$ can be identified with v_x and v_1, respectively.

A *layered tree* can now be defined in a recursive fashion. Note that the definition constructs the tree from "bottom-up", while in the example above the tree was constructed in "top-down" order.

Definition 5.7 (Layered tree [5])**.** For each natural number constant x, define a family T_k^x of layered trees recursively as follows:

- T_0^x is a single node.
- T_k^x for $k \geq 1$ is defined as $T_k^x = \oplus^x T_{k-1}^x$.

The recursive definition gives a natural notion of *layer* for each node: the layer of a node is the index of the tree in which the node is first present. For example, the leaves always belong to layer 0 and the root belongs to the highest layer.

With these definitions, the exact lower bound can be proven. The proof is based on a simulation of an online-LOCAL algorithm $\mathcal{A}$ which supposedly solves the $2\frac{1}{2}$-coloring problem with locality $o(\sqrt{n})$. An adversary simulates algorithm $\mathcal{A}$ on disjoint graph fragments, and only after the algorithm has committed the labels for some nodes, the adversary decides how the fragments are connected to each other. In the following, I show how exactly the adversary can use the commitments of the algorithm to construct an instance on which the algorithm fails.

Assume for contradiction that there existed an online-LOCAL algorithm $\mathcal{A}$ solving the $2\frac{1}{2}$-coloring problem $\Pi_{2\frac{1}{2}}$ with locality $T(n) = o(\sqrt{n})$. The adversary can construct a failing instance $G_{\mathcal{A}}$ as follows:

1. Construct 4 copies of tree T_2^x, namely G_1, G_2, G_3, G_4, with sufficiently large constant $x \gg 2T(n)$ where n is the size of the graph. This is possible because each copy of tree T_2^x has $x+2x^2$ nodes, and hence the whole graph has $4x+8x^2 < 16x^2$ nodes. By assumption, the locality of algorithm $\mathcal{A}$ is $T(n) = o(\sqrt{n})$, and hence there exists some x such that

   $$x \gg 2T(16x^2)$$

   holds.

2. Reveal the center nodes on the core paths of trees G_i to algorithm $\mathcal{A}$. The algorithm must commit to some labels for these nodes without seeing the roots or the connector nodes of the trees. This means that the algorithm does not know how trees G_i are connected to each other. This is because x is larger than the diameter of the view that is revealed to the algorithm. Figure 6 visualizes an example of this situation.

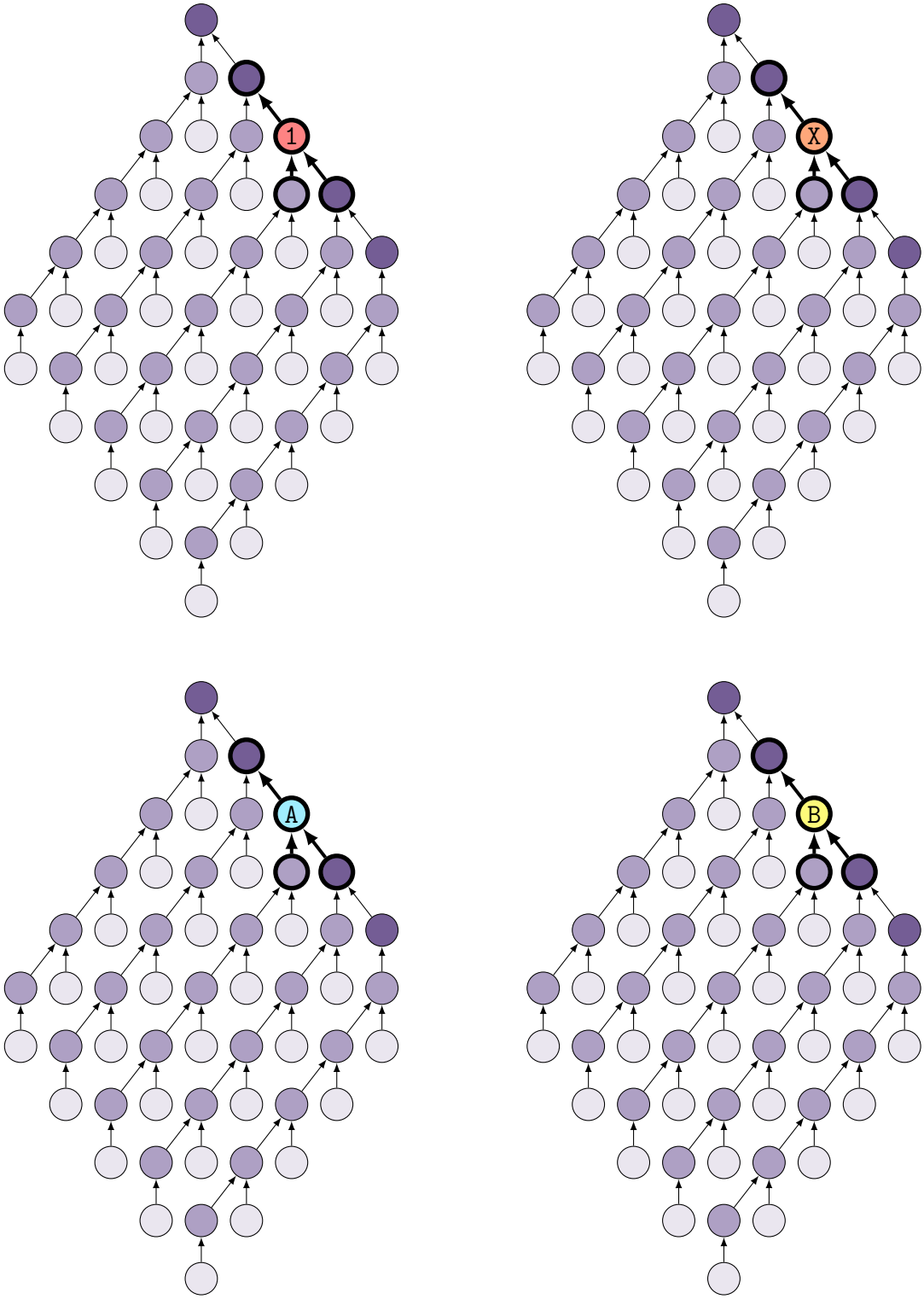


Figure 6: Trees G_1, G_2, G_3 and G_4. The algorithm has labeled the center nodes on the core paths with `1`, `X`, `A` and `B`. The neighborhoods that the algorithm has seen are visualized by thicker lines around nodes.

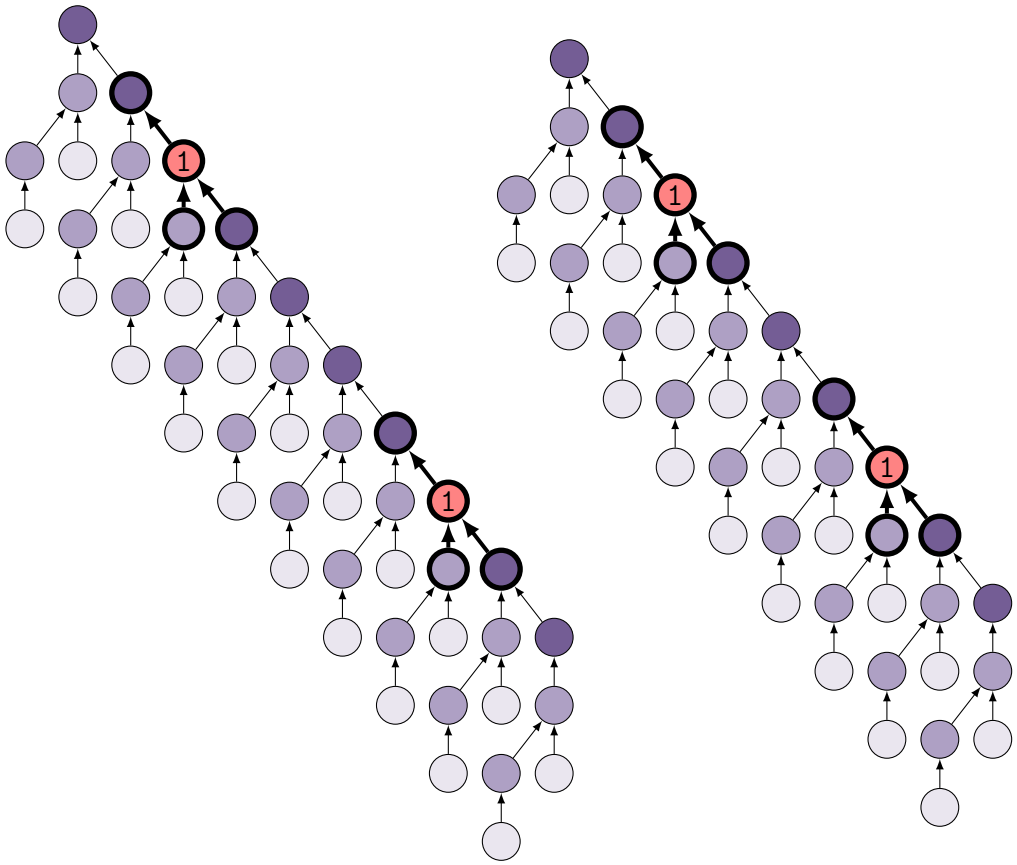


Figure 7: Example for case I. Two ways to connect trees with labels from set Σ_1. In this example, the algorithm has decided to label both center nodes with label `1`. Note that in this visualization the chain of layer-1 nodes has been made shorter to draw attention to layer 2; in reality there are more layer-1 and layer-0 nodes. On the left, the root of the lower tree has been made a child of the connector node of the upper tree, while on the right they have been identified as one node. The distance between labeled nodes is 5 in the left tree and 4 in the right tree. Because the distance between nodes labeled with `1` must be even, the labeling on the left tree cannot be completed.

3. Consider now the following cases. In each one of them, the adversary can force algorithm $\mathcal{A}$ to fail to produce a valid labeling:

   I **Algorithm $\mathcal{A}$ labels at least two of the center nodes of the trees with labels from set Σ_1:**

   Let v_1 and v_2 be two distinct center nodes labeled with labels $\sigma_1, \sigma_2 \in \Sigma_1$, respectively. Without loss of generality, it may be assumed that nodes v_1 and v_2 are the center nodes of trees G_1 and G_2. By Observation 5.2, labels σ_1 and σ_2 are path-inflexible in $\Pi_{2\frac{1}{2}}$, and hence pair (σ_1, σ_2) is a path-inflexible pair by Lemma 3.26.

   Consider two different ways of combining trees G_1 and G_2: The root of tree G_2 can be identified with the connector node of G_1, or the root of

tree G_2 can be made a child of the connector node. Figure 7 visualizes both of these cases. In the former case, the length of the path between nodes v_1 and v_2 is p, and in the latter case, it is $p+1$. As p and $p+1$ have different parity, only one of the walks $\sigma_1 \rightsquigarrow \sigma_2$ of length p and $p+1$ can exist in $\mathcal{M}(\Pi_{2\frac{1}{2}})$. The adversary can choose the option for which the walk of that length does not exist. This implies that there is no way to label the path between v_1 and v_2 such that the labeling would be valid according to the $2\frac{1}{2}$-coloring problem $\Pi_{2\frac{1}{2}}$. Hence the algorithm must fail.

II **Algorithm $\mathcal{A}$ labels all center nodes of the trees with labels from set Σ_2:**

In this case, two trees with labels from set Σ_2 can be combined into one tree with a label from set Σ_1. By repeating this for both pairs of trees, the adversary can construct two trees with labels from set Σ_1 such that the algorithm has not seen how the trees are connected. A contradiction can then be derived using the construction from case I on these trees.

To see how to construct a tree with a label from set Σ_1, consider two of the trees, say G_1 and G_2. The corresponding center nodes are v_1 and v_2, and their labels are σ_1 and σ_2, respectively. By Observation 5.4, labels σ_1 and σ_2 are path-inflexible in $\Pi'_{2\frac{1}{2}}$. Using a similar construction as in case I, the adversary can construct a tree in which algorithm $\mathcal{A}$ cannot solve the restricted problem $\Pi'_{2\frac{1}{2}}$. In particular, algorithm $\mathcal{A}$ must fail to solve $\Pi'_{2\frac{1}{2}}$ on the path connecting v_1 and v_2.

However, algorithm $\mathcal{A}$ is not actually trying to solve $\Pi'_{2\frac{1}{2}}$ but $\Pi_{2\frac{1}{2}}$, and hence it can also use labels from set Σ_1 to label some of the revealed nodes. By revealing the nodes between v_1 and v_2, as well as their children, to algorithm $\mathcal{A}$, the adversary can force the algorithm to use a label from set Σ_1 for some revealed node v without showing the root of the tree to the algorithm. Moreover, revealing node v does not let the algorithm see the whole subtree rooted at v. In particular, the node at the end of a path consisting of only layer-1 nodes is not shown to the algorithm. Therefore, the adversary can use that node to connect the tree to another tree.

Note that this is the case where the structure of trees G_i comes into play. The layered tree structure ensures that when node v is revealed to the algorithm, it has not seen all layer-1 nodes in the subtree rooted at v. This is because the algorithm has locality $T(n)$, but the length of the path consisting of layer-1 nodes has length $x \gg 2T(n)$. The layered tree structure is also needed for proving the general result in Section 5.2.

Figure 8 shows an example of how two trees can be connected to force the algorithm to use a label from set Σ_1.

III **Algorithm $\mathcal{A}$ labels exactly one of the center nodes of the trees with label from set Σ_1, and the rest of the center nodes from set Σ_2:**

In this case, two of the trees whose center nodes have been labeled with labels from set Σ_2 can be combined like in case II to produce a tree with

a label from set Σ_1. To force the algorithm to fail, this resulting tree can then be combined with the tree original whose center node is from set Σ_1, just like in case I. Figure 9 shows an example of this.

This is an exhaustive list of cases. In all cases the adversary can force algorithm $\mathcal{A}$ to fail to produce a valid labeling. Therefore, the assumption that such an algorithm can exist must be false. The only assumption made about algorithm $\mathcal{A}$ in step 1 is that it has locality $T(n) = o(\sqrt{n})$, and hence the $2\frac{1}{2}$-coloring problem $\Pi_{2\frac{1}{2}}$ must require locality $\Omega(\sqrt{n})$ in the online-LOCAL model.

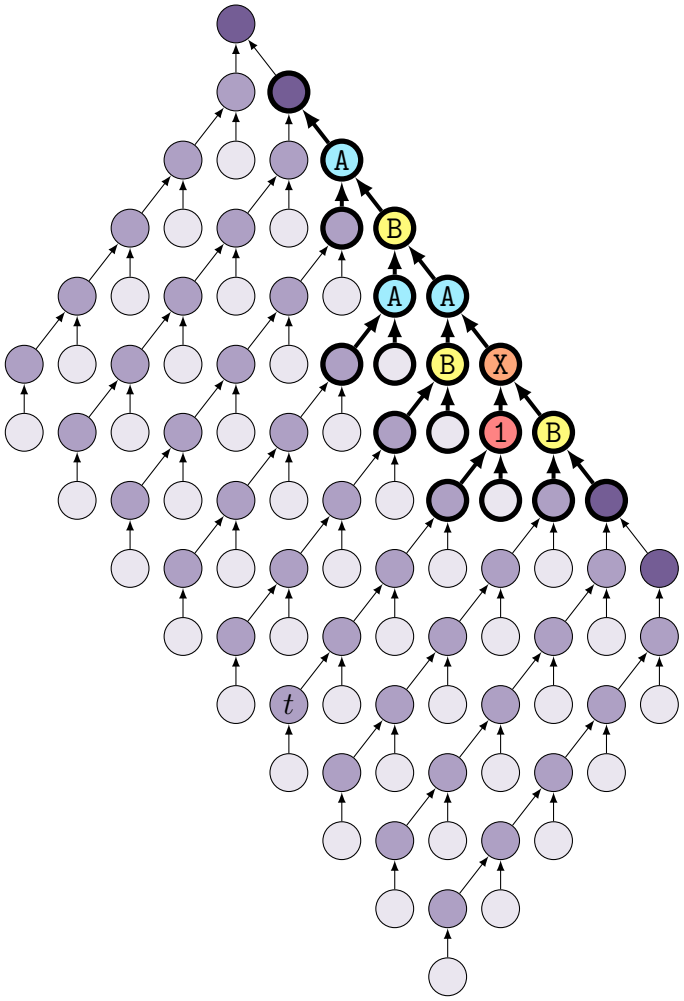


Figure 8: Example for case II. The adversary has connected trees G_3 and G_4, having labels `A` and `B`, in such a way that no valid labeling for the restricted problem $\Pi'_{2\frac{1}{2}}$ exists for the tree. The algorithm has then labeled the nodes on the connecting path, as well as their children. Because there is no valid labeling to the restricted problem $\Pi'_{2\frac{1}{2}}$, the algorithm had to use a label from set Σ_1 to label at least one node. In this case, the algorithm has decided to use label `1`. The node labeled with `1` is node v from the text, and the node marked with t is the new connector node.

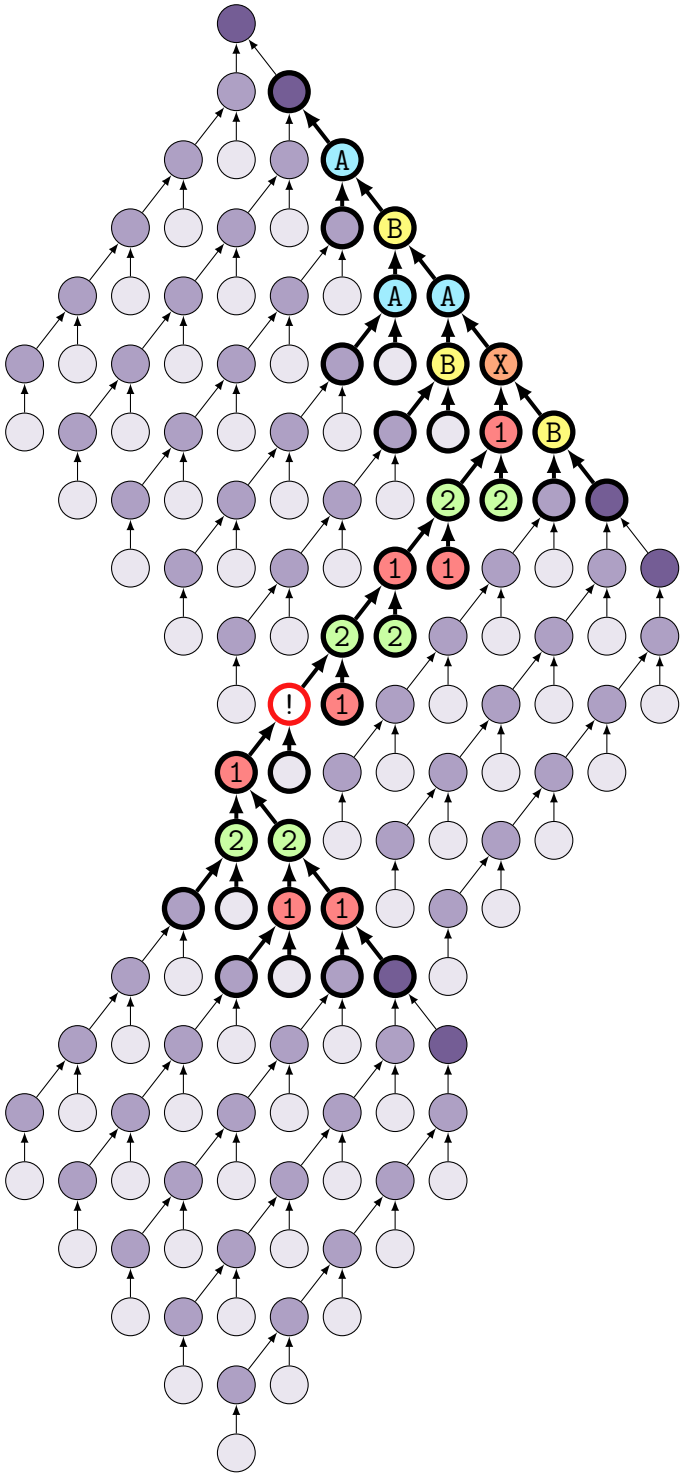


Figure 9: Example for case III. In this case, the adversary has first combined trees G_3 and G_4 to get a node with label `1`. The adversary has then made tree G_1 a child of the connector node of the newly-formed tree. The algorithm has tried to label all nodes on the path connecting the two labeled parts, but as there exists no valid labeling for the path, the algorithm has failed to label the node marked with `!`.

5.2 Equivalence in super-logarithmic region

In the previous section, I showed an example on how to prove that an LCL problem requires at least polynomial locality in the online-LOCAL model. In this section, I generalize the proof for all LCL problems and prove the following theorem:

Theorem 1.2 (Equivalence in rooted trees, super-logarithmic region)**.** *Let* Π *be an LCL problem in rooted trees. Problem* Π *requires locality* $n^{\Omega(1)}$ *in the online-LOCAL model exactly when it requires* $n^{\Omega(1)}$ *locality in the LOCAL model.*

By the previous work of Balliu et al. [5], it is known that LCL problems requiring $n^{\Omega(1)}$ locality in the LOCAL model have a very specific structure. In particular, such problems can be decomposed into a hierarchical sequence of restricted problems and the corresponding sequence of path-inflexible labels. The decomposition can be constructed by repeatedly removing path-inflexible labels from the problem. More formally, the decomposition is defined as follows:

Definition 5.8 (Path-inflexible decomposition)**.** Let Π be an LCL problem in rooted trees. Its *path-inflexible decomposition* consists of a sequence of problems $(\Pi_0, \Pi_1, \ldots, \Pi_k)$ and a sequence of labels $(\Sigma_1, \Sigma_2, \ldots, \Sigma_k)$. The sequences are defined as follows:

1. $\Pi_0 = \Pi$.
2. For each $1 \leq i \leq k$, let Σ_i be the set of path-inflexible labels in Π_{i-1}.
3. For each $1 \leq i \leq k$, let Π_i be the restriction of Π to $\Sigma \setminus (\Sigma_1 \cup \Sigma_2 \cup \cdots \cup \Sigma_i)$.
4. The problem Π_k is either an empty problem, or all of its labels are path-flexible.

Observation 5.9. *Every problem in the sequence of problems* $(\Pi_0, \Pi_1, \ldots, \Pi_k)$*, as defined above, is a restriction of the previous one. In particular, a solution to problem* Π_i *is also a valid solution to any problem* Π_j *for* $j \leq i$*.*

Note that, for the $2\frac{1}{2}$-coloring problem presented in Section 5.1, the label sequence in its path-inflexible decomposition consists of sets Σ_1 and Σ_2 defined in Observations 5.2 and 5.4. The restricted $2\frac{1}{2}$-coloring problem $\Pi'_{2\frac{1}{2}}$ was also defined such that it is compatible with the above definition.

Balliu et al. [5] showed that any LCL problem having only path-flexible labels is solvable with locality $O(\log n)$ in the LOCAL model, and therefore also in the online-LOCAL model. This holds even when the LCL problem contains path-inflexible labels, but it has a restriction with only path-flexible labels. In particular, if such restriction exists, then the last problem in the path-inflexible decomposition is such a restriction. In this case, Balliu et al. call the last problem Π_k a *certificate for* $O(\log n)$ *solvability* for the original problem.

If, on the other hand, the last problem Π_k in the path-inflexible decomposition is an empty problem, then the problem requires locality $\Omega(n^{1/k})$ in the LOCAL model. I prove that this lower bound holds also in the online-LOCAL model.

In the proof of Theorem 1.2, I use layered trees. Recall their definitions from Section 5.1:

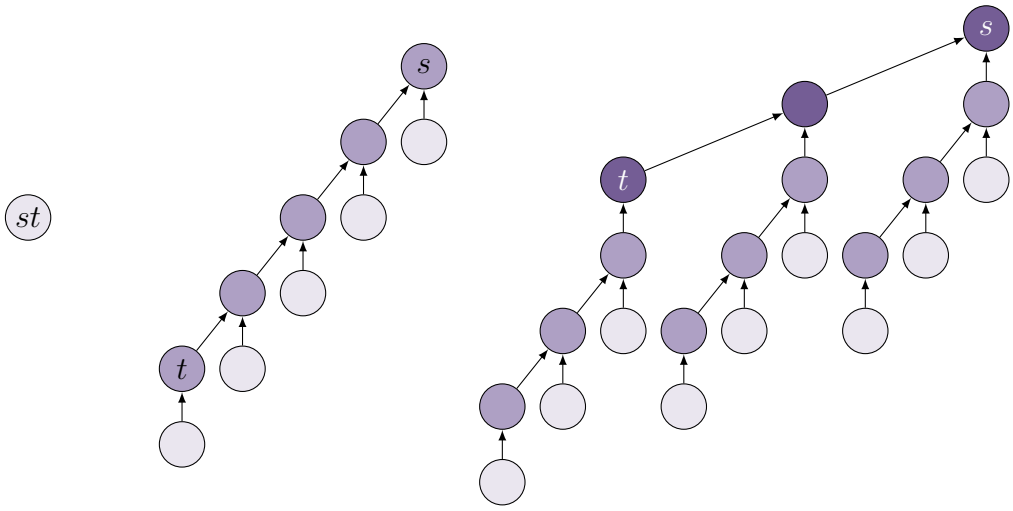


Figure 10: Examples of layered trees. From left to right: T_0^x for any x, T_1^5, and T_2^3. The lightest nodes belong to layer 0, the medium purple to layer 1, and the darkest nodes to layer 2. The nodes marked with s are the roots, and the nodes marked with t are the connector nodes. Note that in tree T_0^x these are the same node.

Definition 5.7 (Layered tree [5])**.** For each natural number constant x, define a family T_k^x of layered trees recursively as follows:

- T_0^x is a single node.
- T_k^x for $k \geq 1$ is defined as $T_k^x = \oplus^x T_{k-1}^x$.

The recursive definition gives a natural notion of *layer* for each node: the layer of a node is the index of the tree in which the node is first present. For example, the leaves always belong to layer 0 and the root belongs to the highest layer.

Figure 10 shows three examples of layered trees for different layer counts and path lengths.

The structure of the layered tree has some nice properties that are taken advantage of in the proof of Theorem 1.2.

Observation 5.10. *For every node v of tree T_k^x belonging to layer $i > 1$, there exists some other node u in the subtree rooted at v such that the layer of u is $i - 1$ and the distance between u and v is at least x.*

This holds despite the fact that tree T_k^x has only a polynomial number of nodes, as formalized in the observation below:

Observation 5.11. *Let T_k^x be a k-layer layered tree with path length x. Then tree T_k^x has $O(x^k)$ nodes.*

To see why these observations are useful, consider the case where x is larger than the locality of the online-LOCAL algorithm. Based on the former observation, the algorithm gets no information about node u when shown node v. This allows the

adversary to change the structure of the graph near node u even after the algorithm has already decided the label for node v. This can then be used to alter the distance between two nodes the algorithm has already labeled, like in the example.

On the other hand, the latter observation states that the size of the tree is only polynomial. As the Theorem 1.2 concerns a separation between polynomial and sub-polynomial locality, this allows the construction of layered trees with arbitrarily large x compared to the locality of the algorithm, as long as the layer count k is small enough.

With the help of these observations, it is possible to prove Theorem 1.2:

Proof of Theorem 1.2. Let $\Pi = (\delta, \Sigma, C)$ be an LCL problem. Let the sequence of problems $(\Pi_0, \ldots, \Pi_k)$ and the sequence of labels $(\Sigma_1, \ldots, \Sigma_k)$ be the path-inflexible decomposition of problem Π. If problem Π_k is non-empty, then problem Π can be solved with locality $O(\log n)$ in the LOCAL model [5], and therefore also in the online-LOCAL model. Hence from now onwards, assume that the problem Π_k is an empty problem. Assume also that there existed an online-LOCAL algorithm $\mathcal{A}$ solving Π with locality $T(n) = o(n^{1/k})$. Note that here the k in the exponent is the same as the number of label sets in the path-inflexible decomposition of Π.

The idea of the proof is following: The adversary uses algorithm $\mathcal{A}$ to construct an input instance $G_{\mathcal{A}}$ on which the algorithm fails. The adversary starts by constructing many copies of tree T_k^x and uses algorithm $\mathcal{A}$ to label one layer-k node on each of them. Every label the algorithm uses belongs to some set Σ_i. The adversary puts all trees having a node with a label from set Σ_i to collection C_i.

The adversary repeatedly combines two trees from collection C_i, using a construction similar to what was shown in the example in Section 5.1, to get a new tree belonging to some collection C_j for $j < i$. Repeating this sufficiently many times, the adversary constructs two trees with labels belonging to set Σ_1. Recall that set Σ_1 is the set of are path-inflexible labels in the original problem Π. The adversary then combines these trees in such a way that no valid labeling exists for the resulting tree, showing that algorithm $\mathcal{A}$ must fail to solve the problem. Therefore an algorithm with locality $T(n) = o(n^{1/k})$ cannot exist.

This division of trees into collections C_i is not dissimilar to what was done in the example in Section 5.1. In the example, the path-inflexible decomposition of the $2\frac{1}{2}$-coloring problem contained only two sets of labels: Σ_1 and Σ_2. In the general case, the path-inflexible decomposition can divide the set of labels into more than two partitions. Therefore, more structured approach to combining trees than just enumerating cases is needed. In particular, the combining of trees in step 4 corresponds to case II in the example. The final step 5, where two trees are combined to arrive at the contradiction, corresponds to cases I and III in the example.

The adversary constructs a failing input instance $G_{\mathcal{A}}$ in the following way:

1. The adversary constructs 2^k copies of tree T_k^x with a sufficiently large constant parameter $x \gg 2T(n)$.

   By Observation 5.11, each of copy of T_k^x has $O(x^k)$ nodes. The number of constructed trees is a constant dependent only on k (a constant determined by

problem Π), and hence the total number of nodes in all trees is $O(x^k)$. This, combined with the assumption that the locality of $\mathcal{A}$ is $T(n) = o(n^{1/k})$, implies that there exists some sufficiently large x.

2. For each copy of T_k^x, the adversary reveals the center node on the core path to algorithm $\mathcal{A}$. The algorithm must commit a label for the center node without seeing the root or the connector node of the tree, that is the endpoints of the core path. Hence the algorithm does not know how the trees are connected to each other. The adversary can use this to combine trees without the algorithm noticing.

3. The adversary divides the trees into collections $C_1, \ldots, C_k$ based on the label produced by the algorithm for each tree: If the algorithm produced label σ for the center node of tree G such that $\sigma \in \Sigma_i$, then the adversary puts tree G into collection C_i. In other words, the adversary uses the index of set $\Sigma_i \ni \sigma$ to decide the index of collection C_i.

   There are four properties that hold for every tree G in collection C_i:

   (a) There exists a node v in tree G such that the label of v belongs to set Σ_i.
   (b) The algorithm has not committed a label for any node whose layer is less than i.
   (c) The algorithm has not seen the root s_G nor the connector node t_G of G.
   (d) The layer of the connector t_G of G is at least i.

   All of these hold trivially for the initial trees: The layer of the center nodes, the roots and the connector nodes is always k. The algorithm has committed labels only for the center nodes. The algorithm has not seen the whole core path because the value of parameter x has been chosen to be larger than the diameter of visibility of the algorithm.

4. The adversary iteratively combines two trees from collection C_i into a tree of some other collection C_j for $j < i$. This step corresponds to case II in the example in Section 5.1.

   The combination of trees proceeds as follows: Consider two trees A and B from collection C_i. By the construction of C_i, there exist nodes v_A and v_B in trees A and B, respectively, such that the labels of nodes v_A and v_B belong to set Σ_i. Moreover, the layers of v_A and v_B are at least i.

   There are four different ways in which trees A and B can be combined:

   (a) The root s_B of B is identified with the connector node t_A of A.
   (b) The root s_A of A is identified with the connector node t_B of B.
   (c) The root s_B of B is made a child of the connector node t_A of A.
   (d) The root s_A of A is made a child of the connector node t_B of B.

   Note that the adversary could choose any one of these combinations without the algorithm knowing which one it chose. This is because the algorithm has

not seen the roots s_A and s_B or the connector nodes t_A and t_B of trees A and B, and hence the adversary can change the structure of the graph in their neighborhoods freely.

Among these four combinations, the adversary chooses one for which there exists no valid labeling for problem Π_{i-1}. Such tree must exist by Lemma 3.27. In particular, no valid labeling exists for the nodes and their children on the path connecting nodes v_A and v_B. Call the resulting tree R. Any attempt to label the nodes and their children on the path between nodes v_A and v_B in tree R according to problem Π_{i-1} must fail.

Once tree R is constructed, the adversary makes algorithm $\mathcal{A}$ commit labels for all nodes, as well as their children, on the path between v_A and v_B in R. Note that all of these nodes belong to a layer whose index is at least $i-1$. By the choice of R, the algorithm cannot label all nodes using only labels present on problem Π_{i-1}. Hence there must exist a node v on the path between v_A and v_B such that it and its children do not form a valid configuration in problem Π_{i-1}. This is possible only if either v or at least one of its children has label $\sigma \in \Sigma_j$ for some $j < i$. Denote that node by v'.

By Observation 5.10, there exists a layer-j node u in the subtree rooted at node v such that the distance between v and u is at least x. Hence the distance between v' and u is at least $x - 1 > T(n)$. This means that the algorithm has not seen node u yet. Make node u the connector node t_R of tree R.

With this construction, it is easy to check that tree R fulfills all properties of a tree of collection C_j, as described in step 3. Hence the adversary can add tree R to collection C_j and remove trees A and B from collection C_i. The adversary repeats this step until collection C_1 contains at least two trees, at which point the adversary moves to the next step and forces the algorithm to fail.

5. In this last step, the adversary forces the algorithm to fail. This step corresponds to cases I and III in the example in Section 5.1.

   The adversary does this by taking two trees A and B from collection C_1. By the construction of collection C_1, there exists nodes v_A and v_B in trees A and B with labels σ_A and σ_B from set Σ_1, respectively. Recall that set Σ_1 is the set of path-inflexible labels in problem $\Pi_0 = \Pi$, and hence the label pair (σ_A, σ_B) is a path-inflexible pair of Π.

   The adversary combines trees A and B in a way that is similar to the previous step. As the result, the adversary gets tree R. This time, there does not exist a valid labeling for the original problem $\Pi_0 = \Pi$ in tree R. Hence, when the adversary uses algorithm $\mathcal{A}$ to label all nodes of tree R, the algorithm must fail. Therefore this is the tree $G_{\mathcal{A}}$ which the adversary aimed to construct.

As the adversary can force algorithm $\mathcal{A}$ to fail, it can be concluded that such algorithm cannot exist. The only technical requirement of algorithm $\mathcal{A}$ was that it

has locality $T(n) = o(n^{1/k})$. Hence problem Π must require locality $\Omega(n^{1/k})$ to be solved, even in the online-LOCAL model.

As a final remark, the number of trees the adversary constructs in step 1 is chosen to be sufficiently large. In particular, even in the worst case where all trees initially belong to collection C_k, there are enough trees to get at least two trees into collection C_1.

This concludes the proof of Theorem 1.2. □

In this section, I have shown that any LCL problem whose path-inflexible decomposition ends in an empty problem requires locality $n^{\Omega(1)}$ in the online-LOCAL model. This matches the lower bound Balliu et al. [5] showed for the LOCAL model. Hence the problems requiring polynomial locality in the online-LOCAL model are exactly the same problems which require polynomial locality in the LOCAL model. This completes the first half of the classification of LCL problems in rooted trees in the online-LOCAL model.

5.3 Equivalence in sub-logarithmic region

In this section, I classify the rest of the LCL problems in rooted trees. I show that any online-LOCAL algorithm $\mathcal{A}$ solving an LCL problem Π with locality $o(\log)$ can be sped up to an $O(\log^* n)$-locality LOCAL algorithm. Observation 3.12 then implies that the there exists an online-LOCAL algorithm solving problem Π with locality $O(1)$. This, in turn, implies that there are no problems with locality $\omega(1)$ and $o(\log n)$ in the online-LOCAL model. Moreover, the problems that are solvable with locality $O(1)$ in the online-LOCAL model are exactly the same problems that are solvable in $O(\log^* n)$ locality in the LOCAL model.

I show the gap by extending the previous results of Balliu et al. [5] to work also in the online-LOCAL model. In particular, I provide a construction for a *certificate for* $O(\log^* n)$ *solvability* for any LCL problem having a locality-$o(\log n)$ online-LOCAL algorithm.

I make use of the coprime certificate for $O(\log^* n)$ solvability that was defined by Balliu et al. [5]. The high-level idea of the certificate is to construct a set of balanced, labeled trees such that each tree has the same set of labels at the leaves, and different labels at the roots. The certificate can be used to label a rooted tree by dividing the tree into approximately constant-height subtrees, labeling the leaves of each subtree by these labels, and then using the certificate to complete the labeling between the leaves of two subtrees. Figure 11 shows an example certificate and how that can be used to 3-color a tree.

Formally, the coprime certificate for $O(\log^* n)$ solvability is defined as follows:

Definition 5.12 (Coprime certificate for $O(\log^* n)$ solvability [5])**.** Let $\Pi = (\delta, \Sigma, C)$ be an LCL problem. *A coprime 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 the following conditions hold:

1. Depths d_1 and d_2 are coprime.

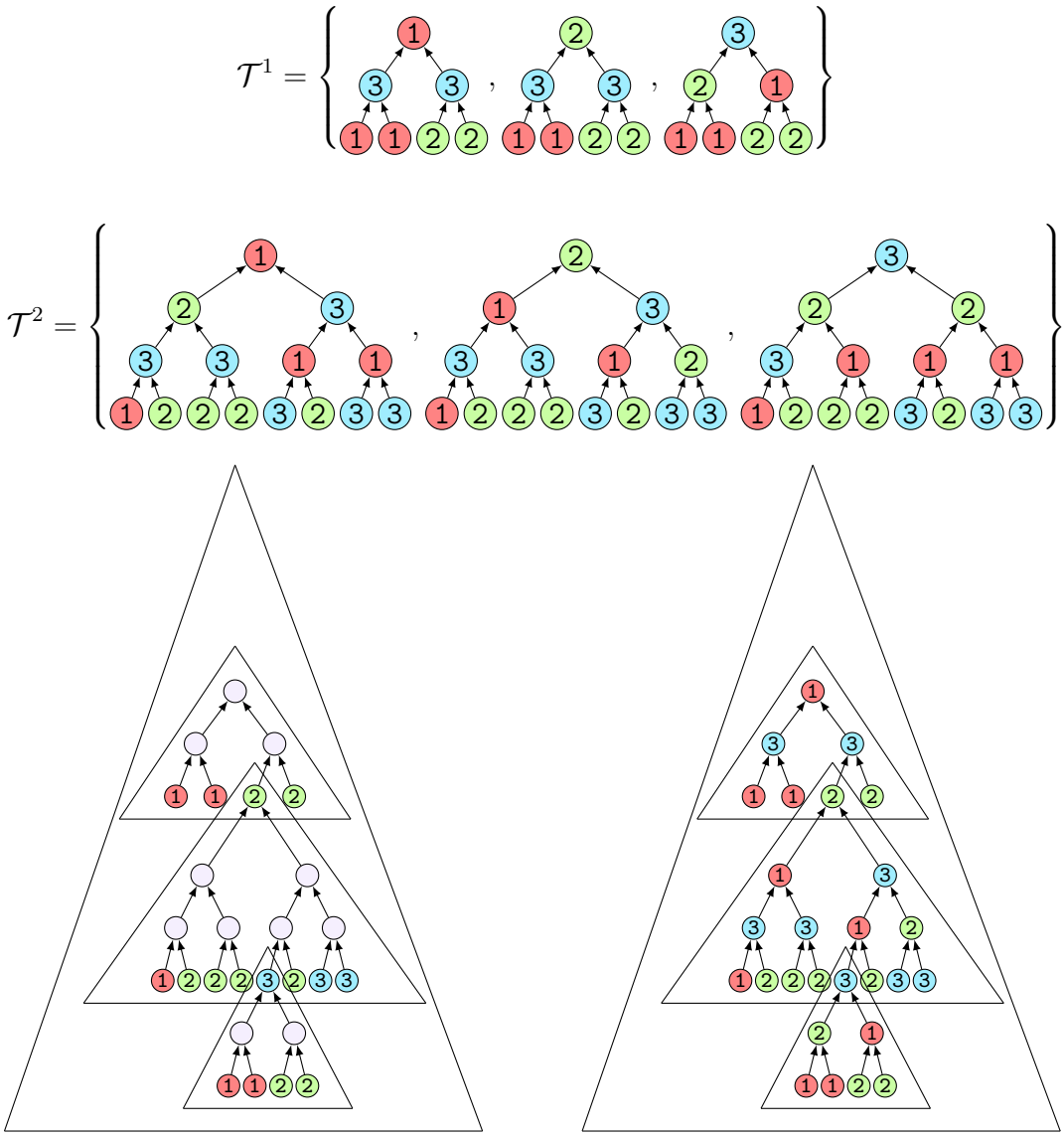


Figure 11: A certificate for $O(\log^* n)$ solvability of 3-coloring and an example on how it can be used to label a tree. The certificate consists of two sequences of trees, $\mathcal{T}^1$ and $\mathcal{T}^2$. For every color, there exists a tree with root labeled with that color in both sequences. The leaves of each tree in both sequences are labeled with the same labels in the same order. On the bottom left, the algorithm has shattered a large tree into subtrees with heights 2 and 3, and it has labeled the leaves of those subtrees with the leaf labels of sequences $\mathcal{T}^1$ and $\mathcal{T}^2$. On the right, the algorithm has filled in the labels for rest of the trees by copying the intermediate labels from sequences $\mathcal{T}^1$ and $\mathcal{T}^2$.

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.

Balliu et al. [5] also define a *uniform certificate for $O(\log^* n)$ solvability*, but it is not needed for the purposes of this thesis. Hence, when using a *certificate for $O(\log^* n)$ solvability*, I refer to *coprime* certificate for $O(\log^* n)$ solvability.

The proof of Theorem 1.3 is based on constructing a certificate for $O(\log^* n)$ solvability using a sufficiently fast online-LOCAL algorithm as a black box. The following lemma formalizes this idea:

Lemma 5.13. *Let $\mathcal{A}$ be an online-LOCAL algorithm with locality $T(n) = o(\log n)$ solving an LCL problem Π. Then there exists a certificate for $O(\log^* n)$ solvability for problem Π. Moreover, algorithm $\mathcal{A}$ can be used to construct the certificate.*

The high-level idea of the certificate construction is to create many large trees and use the online-LOCAL algorithm to label the nodes in the middle of the trees. The trees need to be sufficiently large such that the algorithm does not get to see the roots or the leaves of the trees. Even though it has not been specified how the trees are connected to each other, the execution of the algorithm in the middle of these trees is well defined. In particular, the algorithm must be able to complete the labeling, no matter how the trees are later connected to each other. This allows the adversary to connect the trees in different ways to produce different parts of the certificate.

The following definition formalizes the concept of "nodes in the middle of the tree" as *middle nodes*:

Definition 5.14 (Middle nodes of a rooted tree)**.** Let T be a complete, balanced δ-ary rooted tree with depth $2d$. The *middle nodes* of T are the nodes that are at distance d from the root and the leaves.

With this definition, it is possible to prove the above lemma:

Proof of Lemma 5.13. Let $\Pi = (\delta, \Sigma, C)$ be an LCL problem, and let $\mathcal{A}$ be an online-LOCAL algorithm solving problem Π with locality $T(n) = o(\log n)$.

A certificate for $O(\log^* n)$ solvability of Π can be constructed as follows:

1. Let n be large enough to satisfy
$$(\delta^{T(n)+2} + |\Sigma|)(\delta^{2T(n)+3} - 1) \ll n.$$
Such n must exist because $T(n) = o(\log n)$ by assumption.
Construct $\delta^{T(n)+2} + |\Sigma|$ complete δ-ary rooted trees, each with depth $2T(n) + 2$. Each one of the trees has $\delta^{2T(n)+3} - 1$ nodes.

2. Use algorithm $\mathcal{A}$ to label the middle nodes of each tree. Note that at this point, it has not been specified how the trees are connected to each other. Nevertheless, the execution of the algorithm is well-defined. This is because the depth of the tree is larger than the diameter of visibility of algorithm, and hence the algorithm does not see the roots or the leaves of the trees.

   Let $\Sigma_{\mathcal{T}} = \{\sigma_1, \sigma_2, \ldots, \sigma_t\}$ be the set of labels that the algorithm used to label the middle nodes of the trees.

3. Divide the trees into two sets $\mathcal{L}$ and $\mathcal{U}$ as follows: Consider each tree in turn. If there exists a labeled middle node in the tree such that no tree in set $\mathcal{U}$ has a node with that label, add the tree to set $\mathcal{U}$. Otherwise add the tree to set $\mathcal{L}$.

   In other words, the trees are divided into sets $\mathcal{L}$ and $\mathcal{U}$ such that for every label used by algorithm $\mathcal{A}$ to label a middle node of some tree, there exists a tree in $\mathcal{U}$ that has a middle node labeled with that label. The construction ensures that the size of set $\mathcal{U}$ remains small. In particular, the size of set $\mathcal{U}$ is at most $|\Sigma|$. This is because for each label, a tree is added to set $\mathcal{U}$ at most once. This also means that the size of set $\mathcal{L}$ is at least $\delta^{T(n)+2}$.

   Give the trees of $\mathcal{L}$ a consistent ordering.

4. The sets $\mathcal{L}$ and $\mathcal{U}$ can now be used to construct the individual trees of the sequences $\mathcal{T}^1$ and $\mathcal{T}^2$ of the certificate.

   Consider each label σ_i in set $\Sigma_{\mathcal{T}}$ in turn. There exists a tree in set $\mathcal{U}$ that has a node labeled with σ_i by the construction of the set. Let that tree be U, and let the node having label σ_i be v. Consider the subtree of U rooted at v, and denote that by U_v. The tree U_v has $\delta^{T(n)+1}$ leaves. Identify the leaves with the roots of the first $\delta^{T(n)+1}$ trees of set $\mathcal{L}$.

   Now the root of subtree U_v has label σ_i, and the nodes at depth $2T(n)+2$ have also been labeled with labels from set $\Sigma_{\mathcal{T}}$. Algorithm $\mathcal{A}$ can now be used to label all the nodes in between. The result is that the depth-$(2T(n)+2)$ subtree of U_v is a complete labeled tree with depth $2T(n)+2$. Let that be tree $\mathcal{T}_i^1$ of sequence $\mathcal{T}^1$.

   The trees of for sequence $\mathcal{T}^2$ can be constructed in an analogous way. The only difference is that instead of identifying the roots of the first $\delta^{T(n)+1}$ trees of $\mathcal{L}$ with the leaves of subtree U_v, the first $\delta^{T(n)+2}$ trees of $\mathcal{L}$ are made the children of those leaves. Again, algorithm $\mathcal{A}$ can be used to label all the nodes in between. Now the depth-$(2T(n)+3)$ subtree of U_v can be made tree $\mathcal{T}_i^2$ of sequence $\mathcal{T}^2$.

It is easy to verify that the sequences $\mathcal{T}^1$ and $\mathcal{T}^2$ constructed in this way actually form a valid certificate for $O(\log^* n)$ solvability for problem Π. In particular, the depths $2T(n)+2$ and $2T(n)+3$ are coprime, the leaves of each tree in both sets are labeled similarly using labels from set $\Sigma_{\mathcal{T}}$, and for every label of set $\Sigma_{\mathcal{T}}$, there exists a tree in both sequences having a root labeled with that label. □

Lemma 5.13 directly implies Theorem 1.3:

Proof of Theorem 1.3. Let Π be an LCL problem in rooted trees. There are now two cases:

1. Problem Π is solvable with locality $o(\log n)$ in online-LOCAL. By Lemma 5.13, it is possible to construct a certificate for $O(\log^* n)$ solvability for Π, and hence Π is solvable in $O(\log^* n)$ in LOCAL.

2. Problem Π requires locality $\Omega(\log n)$ in online-LOCAL. Then problem Π also requires locality $\Omega(\log n)$ in LOCAL as online-LOCAL model is at least as strong as LOCAL. □

This completes the classification of all LCL problems in rooted trees in the online-LOCAL model.

6 Conclusion

In this thesis, I have shown that the LOCAL and the online-LOCAL models are approximately equally powerful for solving LCL problems in directed paths and rooted regular trees. Based on this result, it may seem likely that the result could generalize to all graph classes. Unfortunately, this is not the case, as we have shown in our manuscript [1]. In particular, we have shown that in 2-dimensional grids, the problem of 3-coloring can be solved with locality $O(\log n)$ in the online-LOCAL model while it requires $\Omega(\sqrt{n})$ locality in the LOCAL model [8].

It is also known that the *randomized* LOCAL model is exponentially more powerful than the deterministic LOCAL model for some LCL problems [9]. There exist problems whose locality is $\Theta(\log n)$ in the deterministic LOCAL model but $\Theta(\log \log n)$ in the randomized LOCAL model. Moreover, it is known that the sequential-LOCAL model is powerful enough to derandomize LOCAL algorithms, and hence those problems are solvable in $O(\log \log n)$ also in the deterministic sequential-LOCAL model [13].

One interesting problem exhibiting this exponential separation between the deterministic and randomized LOCAL models is *sinkless orientation*. In the sinkless orientation problem, the edges of the graph need to be oriented such that no node is a sink of the graph. A node is a sink if it has only incoming edges, and it is not a leaf. What makes sinkless orientation an especially interesting problem is that it is known to be solvable in $\Theta(\log \log \log n)$ locality in the *randomized* sequential-LOCAL model [13]. This raises the following natural question:

Question 6.1. What is the locality of the sinkless orientation problem in the online-LOCAL model?

The two natural answers could be that the online-LOCAL model is approximately equally powerful with the deterministic sequential-LOCAL model and therefore the locality is $\Theta(\log \log n)$, or that the online-LOCAL model is powerful enough to derandomize the sequential-LOCAL model and hence the locality is $\Theta(\log \log \log n)$. I think it would be surprising if there would a third answer to this question.

Chang and Pettie [10] have shown that all LCL problems in trees fall into one of the following three complexity classes: $O(\log^* n)$, $\Theta(\log n)$ and $n^{\Omega(1)}$. While it is known that the complexity in the LOCAL and the online-LOCAL models differs in the logarithmic region, I postulate that all LCL problems in trees fall into one of the following complexity classes in both the LOCAL and online-LOCAL models:

- Problems solvable in $O(\log^* n)$ in the LOCAL model and in $O(1)$ in the online-LOCAL model. In these problems, the main obstacle in the LOCAL model is symmetry breaking, which is broken by the adversarial order in the online-LOCAL model.

- Problems that are solvable in $\Theta(\log n)$ in the LOCAL model, and that require locality $\Omega(\log \log \log n)$ but are solvable in $O(\log n)$ in the online-LOCAL model. These problems are akin to the sinkless orientation and the main obstacle is finding an irregularity in the graph, such as a leaf.

- Problems requiring $n^{\Omega(1)}$ locality both in the LOCAL and in the online-LOCAL models. These are the problems that require global coordination in both models.

In particular, I postulate that the gap between $O(\log^* n)$ and $\Omega(\log n)$ in the LOCAL model extends to become a gap between $O(1)$ and $\Omega(\log \log \log n)$ in the online-LOCAL model. I also postulate that the gap between $O(\log n)$ and $n^{\Omega(1)}$ in the LOCAL model extends to the online-LOCAL model. Showing these gaps would prove the above classification correct.

The discussion about randomized models brings up the question of how randomness affects the online-LOCAL model. To answer this question, it must first be decided what the adversary knows about the randomness that the online-LOCAL algorithm is using. In the online algorithm literature, there are three types of adversaries [6]:

- **Oblivious adversaries** know nothing about the randomness that the algorithm is using. Hence they need to choose all of their actions beforehand without seeing what the algorithm does.
- **Adaptive online adversaries** know the choices the algorithm has made so far, and can therefore base their future decisions on them.
- **Adaptive offline adversaries** also know the future values of randomness that the algorithm is using, and can therefore make optimal decisions against the algorithm.

The proofs for Theorems 1.2 and 1.3 are based on the adversary reacting to the decisions of the algorithm. Hence it seems likely that randomness would not help against adaptive adversaries in rooted trees. This leaves open one final question on what happens against oblivious adversaries:

Question 6.2. Does randomness help solving LCL problems in rooted trees in the online-LOCAL model against an oblivious adversary?

References

[1] Amirreza Akbari, Henrik Lievonen, Darya Melnyk, Joona Särkijärvi, and Jukka Suomela. "Online Algorithms with Lookaround". *arXiv:2109.06593 [cs]* (Feb. 15, 2022). DOI: 10.48550/arXiv.2109.06593.

[2] Susanne Albers and Sebastian Schraink. "Tight Bounds for Online Coloring of Basic Graph Classes". *Algorithmica* 83.1 (Jan. 2021), pp. 337–360. DOI: 10.1007/s00453-020-00759-7.

[3] Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. "The Distributed Complexity of Locally Checkable Problems on Paths is Decidable". In: *Proc. 38th Symposium on Principles of Distributed Computing (PODC 2019)*. Toronto ON Canada: ACM, July 16, 2019, pp. 262–271. DOI: 10.1145/3293611.3331606.

[4] Alkida Balliu, Sebastian Brandt, Yuval Efron, Juho Hirvonen, Yannic Maus, Dennis Olivetti, and Jukka Suomela. "Brief Announcement: Classification of Distributed Binary Labeling Problems". In: *Proc. 39th Symposium on Principles of Distributed Computing (PODC 2020)*. Virtual Event Italy: ACM, July 31, 2020, pp. 349–351. DOI: 10.1145/3382734.3405703.

[5] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, Jan Studený, Jukka Suomela, and Aleksandr Tereshchenko. "Locally Checkable Problems in Rooted Trees". In: *Proc. 40th Symposium on Principles of Distributed Computing (PODC 2021)*. Virtual Event Italy: ACM, July 21, 2021, pp. 263–272. DOI: 10.1145/3465084.3467934.

[6] Shai Ben-David, Allan Borodin, Richard Karp, Gabor Tardos, and Avi Wigderson. "On the Power of Randomization in Online Algorithms". *Algorithmica* 11.1 (Jan. 1994), pp. 2–14. DOI: 10.1007/BF01294260.

[7] Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. "A Lower Bound for the Distributed Lovász Local Lemma". In: *Proc. 48th Annual Symposium on Theory of Computing (STOC 2016)*. Cambridge MA USA: ACM, June 19, 2016, pp. 479–488. DOI: 10.1145/2897518.2897570.

[8] Sebastian Brandt, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Patric R.J. Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela, and Przemysław Uznański. "LCL Problems on Grids". In: *Proc. 36th Symposium on Principles of Distributed Computing (PODC 2017)*. Washington DC USA: ACM, July 25, 2017, pp. 101–110. DOI: 10.1145/3087801.3087833.

[9] 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 (Jan. 2019), pp. 122–143. DOI: 10.1137/17M1117537.

[10] Yi-Jun Chang and Seth Pettie. "A Time Hierarchy Theorem for the LOCAL Model". *SIAM Journal on Computing* 48.1 (Jan. 2019), pp. 33–69. DOI: 10.1137/17M1157957.

[11] Yi-Jun Chang, Jan Studený, and Jukka Suomela. “Distributed Graph Problems Through an Automata-Theoretic Lens”. In: *Structural Information and Communication Complexity*. Ed. by Tomasz Jurdziński and Stefan Schmid. Vol. 12810. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 31–49. DOI: 10.1007/978-3-030-79527-6_3.

[12] Richard Cole and Uzi Vishkin. “Deterministic Coin Tossing With Applications to Optimal Parallel List Ranking”. *Information and Control* 70.1 (July 1986), pp. 32–53. DOI: 10.1016/S0019-9958(86)80023-7.

[13] Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. “On Derandomizing Local Distributed Algorithms”. In: *Proc. 59th Annual Symposium on Foundations of Computer Science (FOCS 2018)*. Paris: IEEE, Oct. 2018, pp. 662–673. DOI: 10.1109/FOCS.2018.00069.

[14] Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. “On the Complexity of Local Distributed Graph Problems”. In: *Proc. 49th Annual Symposium on Theory of Computing (STOC 2017)*. Montreal Canada: ACM, June 19, 2017, pp. 784–797. DOI: 10.1145/3055399.3055471.

[15] Magnus M. Halldórsson and Mario Szegedy. “Lower Bounds for Online Graph Coloring”. *Theoretical Computer Science* 130.1 (Aug. 1994), pp. 163–174. DOI: 10.1016/0304-3975(94)90157-0.

[16] Juho Hirvonen and Jukka Suomela. *Distributed Algorithms 2020*. Finland: Aalto University, Dec. 11, 2021. 221 pp. URL: https://jukkasuomela.fi/da2020/.

[17] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. 2nd ed. Boston Munich: Addison-Wesley, 2001. 521 pp.

[18] Nathan Linial. “Locality in Distributed Graph Algorithms”. *SIAM Journal on Computing* 21.1 (Feb. 1992), pp. 193–201. DOI: 10.1137/0221015.

[19] Moni Naor and Larry Stockmeyer. “What Can be Computed Locally?” *SIAM Journal on Computing* 24.6 (Dec. 1995), pp. 1259–1277. DOI: 10.1137/S0097539793254571.

[20] Lata Narayanan. “Channel Assignment and Graph Multicoloring”. In: *Wiley Series on Parallel and Distributed Computing*. Ed. by Ivan Stojmenović. New York, USA: John Wiley & Sons, Inc., Feb. 1, 2002, pp. 71–94. DOI: 10.1002/0471224561.ch4.

[21] Dennis Olivetti. “Brief Announcement: Round Eliminator: A Tool for Automatic Speedup Simulation”. In: *Proc. 39th Symposium on Principles of Distributed Computing (PODC 2020)*. Virtual Event Italy: ACM, July 31, 2020, pp. 352–354. DOI: 10.1145/3382734.3405694.

[22] Jeffrey Shallit. “The Frobenius Problem and Its Generalizations”. In: *Developments in Language Theory*. Ed. by Masami Ito and Masafumi Toyama. Vol. 5257. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 72–83. DOI: 10.1007/978-3-540-85780-8_5.

[23] Daniel D. Sleator and Robert E. Tarjan. “Amortized Efficiency of List Update and Paging Rules”. *Communications of the ACM* 28.2 (Feb. 1985), pp. 202–208. DOI: 10.1145/2786.2793.

[24] Aleksandr Tereshchenko. “Automated Classification of Distributed Graph Problems”. MA thesis. Aalto University, May 17, 2021. 76 pp. URL: http://urn.fi/URN:NBN:fi:aalto-202105236941.