Akbari, Amirreza; Eslami, Navid; Lievonen, Henrik; Melnyk, Darya; Särkijärvi, Joona; Suomela, Jukka

Locality in Online, Dynamic, Sequential, and Distributed Graph Algorithms

Published in:
50th International Colloquium on Automata, Languages, and Programming, ICALP 2023

DOI:
10.4230/LIPIcs.ICALP.2023.10

Published: 05/07/2023

Document Version
Publisher's PDF, also known as Version of record

Locality in Online, Dynamic, Sequential, and Distributed Graph Algorithms

Amirreza Akbari
Aalto University, Espoo, Finland

Navid Eslami
Aalto University, Espoo, Finland
Sharif University of Technology, Tehran, Iran

Henrik Lievonen
Aalto University, Espoo, Finland

Darya Melnyk
Aalto University, Espoo, Finland
TU Berlin, Germany

Joona Särkijärvi
Aalto University, Espoo, Finland

Jukka Suomela
Aalto University, Espoo, Finland

Abstract

In this work, we give a unifying view of locality in four settings: distributed algorithms, sequential greedy algorithms, dynamic algorithms, and online algorithms. We introduce a new model of computing, called the online-LOCAL model: the adversary presents the nodes of the input graph one by one, in the same way as in classical online algorithms, but for each node we get to see its radius-T neighborhood before choosing the output. Instead of *looking ahead* in time, we have the power of *looking around* in space.

We compare the online-LOCAL model with three other models: the LOCAL model of distributed computing, where each node produces its output based on its radius-T neighborhood, the SLOCAL model, which is the sequential counterpart of LOCAL, and the dynamic-LOCAL model, where changes in the dynamic input graph only influence the radius-T neighborhood of the point of change.

The SLOCAL and dynamic-LOCAL models are sandwiched between the LOCAL and online-LOCAL models. In general, all four models are distinct, but we study in particular *locally checkable labeling problems* (LCLs), which is a family of graph problems extensively studied in the context of distributed graph algorithms. We prove that for LCL problems in paths, cycles, and rooted trees, all four models are roughly equivalent: the locality of any LCL problem falls in the same broad class – $O(\log^* n)$, $\Theta(\log n)$, or $n^{\Theta(1)}$ – in all four models. In particular, this result enables one to generalize prior lower-bound results from the LOCAL model to all four models, and it also allows one to simulate e.g. dynamic-LOCAL algorithms efficiently in the LOCAL model.

We also show that this equivalence does not hold in two-dimensional grids or general bipartite graphs. We provide an online-LOCAL algorithm with locality $O(\log n)$ for the 3-coloring problem in bipartite graphs – this is a problem with locality $\Omega(n^{1/2})$ in the LOCAL model and $\Omega(n^{1/10})$ in the SLOCAL model.

2012 ACM Subject Classification Theory of computation → Online algorithms; Computing methodologies → Distributed algorithms; Theory of computation → Dynamic graph algorithms

Keywords and phrases Online computation, spatial advice, distributed algorithms, computational complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.10

Category Track A: Algorithms, Complexity and Games

50th International Colloquium on Automata, Languages, and Programming (ICALP 2023).
Editors: Kousha Etessami, Uriel Feige, and Gabriele Puppis; Article No. 10; pp. 10:1–10:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version *Full Version*: https://arxiv.org/abs/2109.06593

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

Acknowledgements We would like to thank Alkida Balliu, Sameep Dahal, Chetan Gupta, Fabian Kuhn, Dennis Olivetti, Jan Studený, and Jara Uitto for useful discussions. We would also like to thank the anonymous reviewers for the very helpful feedback they have provided for previous versions of this work.

1 Introduction

In *online graph algorithms*, the adversary reveals the input graph one node at a time: In step i, the adversary presents some node v_i. The algorithm gets to see the subgraph induced by the nodes $v_1, \ldots, v_i$, and the algorithm has to respond by labeling node v_i. For example, in *online graph coloring*, the algorithm has to pick a color for node v_i in such a way that the end result is a proper coloring of the input graph.

In this work, we consider a more general setting, which we call the online-LOCAL model: in step i, the algorithm gets to see the subgraph induced by all nodes that are within distance T from $v_1, \ldots, v_i$. That is, the algorithm can look T hops further in the input graph around the nodes presented by the adversary. For $T = 0$, this corresponds to the usual online model. For $T = n$, on the other hand, any graph problem (in connected graphs) is solvable in this setting. The key question is what value of T is sufficient for a given graph problem. Put otherwise, what is the *locality* of a given online problem?

It turns out that this question is very closely connected to questions studied in the context of *distributed* graph algorithms, and we can identify problem classes in which the online setting coincides with the distributed setting. However, we also see surprising differences, the prime example being the problem of 3-coloring bipartite graphs, which is a fundamentally *global* problem in the distributed setting, while we show that we can do much better in the online setting.

1.1 Contribution 1: landscape of models

In Section 2, we define the online-LOCAL model, and we also recall the definitions of three models familiar from the fields of distributed and dynamic graph algorithms:

- The LOCAL model [37, 44]: the nodes are processed *simultaneously* in parallel; each node looks at its radius-T neighborhood and picks its own output.
- The SLOCAL model [26]: the nodes are processed *sequentially* in an adversarial order; each node in its turn looks at its radius-T neighborhood and picks its own output (note that here the output of a node may depend on the outputs of other nodes that were previously processed).
- The dynamic-LOCAL model: the adversary *constructs* the graph by adding nodes and edges one by one; after each modification, the algorithm can only update the solution within the radius-T neighborhood of the point of change. While this is not one of the standard models, there is a number of papers [3, 9, 11, 21, 28, 34, 43] that implicitly make use of this model. We also occasionally consider the dynamic-LOCAL$^{\pm}$ model, in which we can have both additions and deletions.

In Section 3, we show that we can sandwich SLOCAL and both versions of dynamic-LOCAL between LOCAL and online-LOCAL, as shown in Figure 1. In particular, this implies that if we can prove that LOCAL and online-LOCAL are equally expressive for some family of graph problems, we immediately get the same result also for SLOCAL and dynamic-LOCAL. This is indeed what we achieve in our next contribution.

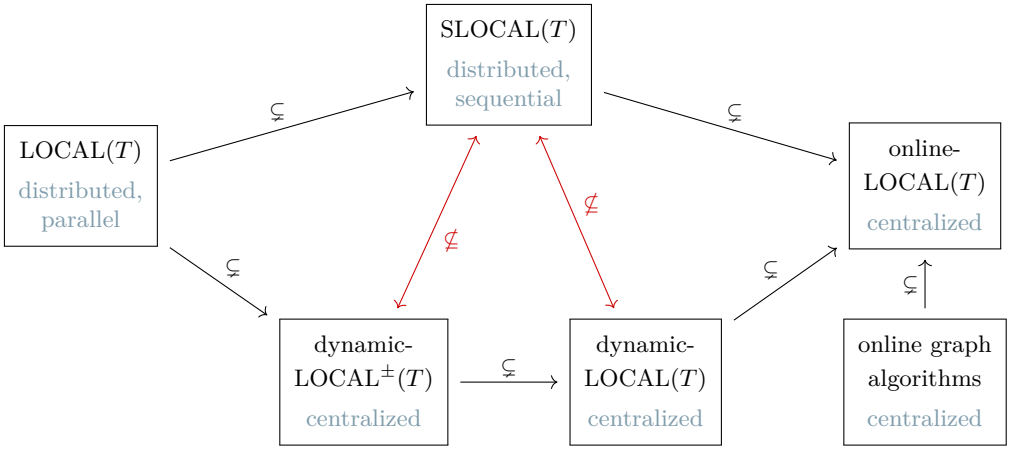

Figure 1 The landscape of models – see Section 2 for the definitions. Each box represents the set of problems solvable with locality $O(T)$ in the given model of computation (except for online graph algorithms, which do not have a notion of locality). For example, any problem with locality $O(T)$ in the LOCAL model can also be solved with locality $O(T)$ in both the SLOCAL and the online-LOCAL models. On the other hand, the SLOCAL and the dynamic-LOCAL models are incomparable, as there exist problems that are solvable with locality $O(T)$ in one of the models but that require $\omega(T)$ locality in the other model.

1.2 Contribution 2: collapse for LCLs in rooted regular trees

A lot of focus in the study of distributed graph algorithms and the LOCAL model has been on understanding *locally checkable labeling problems* (in brief, LCLs) [4,5,7,8,13,15,17,18,42]. These are problems where feasible solutions are defined with local constraints – a solution is feasible if it looks good in all constant-radius neighborhoods (see Definition 3). Coloring graphs of maximum degree Δ with k colors (for some constants Δ and k) is an example of an LCL problem.

In Section 5, we study LCL problems in *paths*, *cycles*, and *rooted regular trees*, and we show that all four models are approximately equally strong in these settings – see Table 1. For example, we show that if the locality of an LCL problem in rooted trees is $n^{\Theta(1)}$ in the LOCAL model, it is also $n^{\Theta(1)}$ in the dynamic-LOCAL, SLOCAL, and online-LOCAL models.

Table 1 In all four models, LCL problems have got the same locality classes in paths, cycles, and rooted trees. Here $n^{\Theta(1)}$ refers to locality $\Theta(n^\alpha)$ for some constant $\alpha > 0$. See Section 5 for more details.

	LOCAL		SLOCAL		dynamic-LOCAL		online-LOCAL
LCLs in paths and cycles	$O(\log^* n)$	$\Leftrightarrow$	$O(1)$	$\Leftrightarrow$	$O(1)$	$\Leftrightarrow$	$O(1)$
	$\Theta(n)$	$\Leftrightarrow$	$\Theta(n)$	$\Leftrightarrow$	$\Theta(n)$	$\Leftrightarrow$	$\Theta(n)$
LCLs in rooted regular trees	$O(\log^* n)$	$\Leftrightarrow$	$O(1)$	$\Leftrightarrow$	$O(1)$	$\Leftrightarrow$	$O(1)$
	$\Theta(\log n)$	$\Leftrightarrow$	$\Theta(\log n)$	$\Leftrightarrow$	$\Theta(\log n)$	$\Leftrightarrow$	$\Theta(\log n)$
	$n^{\Theta(1)}$	$\Leftrightarrow$	$n^{\Theta(1)}$	$\Leftrightarrow$	$n^{\Theta(1)}$	$\Leftrightarrow$	$n^{\Theta(1)}$

By previous work, we know that LCL complexities in paths, cycles, and rooted regular trees are *decidable* in the LOCAL model [4, 7, 18]. Our equivalence result allows us to extend this decidability to the SLOCAL, dynamic-LOCAL, and online-LOCAL models. For example, there is an algorithm that gets as input the description of an LCL problem in rooted trees and produces as output in which of the classes of Table 1 it is, for any of the four models.

1.3 Contribution 3: 3-coloring bipartite graphs in online-LOCAL

Given the equivalence results for LCLs in paths, cycles, and rooted regular trees, it would be tempting to conjecture that the models are approximately equal for LCLs in any graph class. In Section 4, we show that this is not the case: we provide an exponential separation between the SLOCAL and online-LOCAL models for the problem of 3-coloring bipartite graphs. By prior work it is known that in the LOCAL model, the locality of 3-coloring is $\Omega(n^{1/2})$ in two-dimensional grids [13], which are a special case of bipartite graphs; using this result we can derive a lower bound of $\Omega(n^{1/10})$ also for the SLOCAL model (see the full version). In Section 4, we prove the following:

▶ **Theorem 1.** *There is an online-LOCAL algorithm that finds a* 3*-coloring in bipartite graphs with locality* $O(\log n)$*.*

That is, in bipartite graphs, there is an LCL problem that requires locality $n^{\Omega(1)}$ in the LOCAL and SLOCAL models and is solvable with locality $O(\log n)$ in the online-LOCAL model.

The algorithm that we present for coloring bipartite graphs is also interesting from the perspective of competitive analysis of online algorithms. With locality $O(\log n)$, the online-LOCAL algorithm can compute a 3-coloring. Since bipartite graphs are 2-colorable, this gives us a 1.5-competitive online-LOCAL algorithm. On the other hand, it has been shown that any online algorithm for coloring bipartite graphs is at least $\Omega(\log n)$-competitive [10], with a matching algorithm presented in [38]. This result shows how much the competitive ratio of an algorithm can be improved by increasing the view of each node.

1.4 Contribution 4: locality of online coloring

As a corollary of our work, together with results on distributed graph coloring from prior work [13, 19, 37], we now have a near-complete understanding of the locality of graph coloring in paths, cycles, rooted trees, and grids in both distributed and online settings. Table 2 summarizes our key results. For the proofs of the localities in the online-LOCAL model, see Sections 4 and 5.

1.5 Motivation

Before we discuss the key technical ideas, we briefly explain the practical motivation for the study of online-LOCAL and dynamic-LOCAL models. As a running example, consider the challenge of providing public services (e.g. local schools) in a rapidly growing city. The future is unknown, depending on future political decisions, yet the residents need services every day.

The offline solution would result in a city-wide redesign of e.g. the entire school network every time the city plan is revised; this is not only costly but also disruptive. On the other hand, a strict online solution without any consideration of the future would commit to a solution that is far from optimal. The models that we study in this work capture the essence of two natural strategies for coping with such a situation:

Table 2 The locality of the vertex coloring problem in distributed vs. online settings, for two graph families: rooted trees and paths (with n nodes) and 2-dimensional grids (with $\sqrt{n} \times \sqrt{n}$ nodes). Note that most results for the online-LOCAL model follow from the equivalence results discussed in Section 1.2. See Sections 4 and 5 and the full version for more details.

	colors	competitive ratio	LOCAL	SLOCAL	online-LOCAL	references
Rooted trees and paths	2	1	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	trivial
	3	1.5	$\Theta(\log^* n)$	$O(1)$	$O(1)$	[19, 37]
	4	2	$\Theta(\log^* n)$	$O(1)$	$O(1)$	[19, 37]
	...					
Grids	2	1	$\Theta(n^{1/2})$	$\Theta(n^{1/2})$	$\Theta(n^{1/2})$	trivial
	3	1.5	$\Theta(n^{1/2})$	$\boldsymbol{\Omega(n^{1/10})}$	$\boldsymbol{O(\log n)}$	Section 4, [13]
	4	2	$\Theta(\log^* n)$	$O(1)$	$O(1)$	[13]
	5	2.5	$\Theta(\log^* n)$	$O(1)$	0	[13]
	...					

- Redesign the public service network only in the local neighborhoods in which there are new developments. This corresponds to the dynamic-LOCAL model, and the locality parameter T captures the redesign cost and the disruption it causes.
- Wait until new developments in a neighborhood are completed before providing permanent public services in the area. This corresponds to the online-LOCAL model, and the locality parameter T captures the inconvenience for the residents (the width of the "buffer zone" without permanent public services around areas in which the city plan is not yet finalized).

These two models make it possible to formally explore trade-offs between the quality of the solution in the long term vs. the inconvenience of those living close to the areas where the city is changing. In these kinds of scenarios, the key challenge is not related to the computational cost of finding an optimal solution (which is traditionally considered in the context of dynamic graph algorithms) but to the quality of the solution (which is typically the focus in online algorithms). The key constraint is not the availability of information on the current state of the world (which is traditionally considered in distributed graph algorithms), but the cost of changing the solution.

1.6 Techniques and key ideas

For the equivalence in paths and cycles (Section 5.1), we first make use of *pumping-style arguments* that were introduced by Chang and Pettie [17] in the context of distributed algorithms. We show that such ideas can be used to also analyze locality in the context of online algorithms: we start by showing that we can "speed up" (or "further localize") online-LOCAL algorithms with a *sublinear* locality to online-LOCAL algorithms with a *constant* locality in paths and cycles. Then, once we have reached constant locality in the online-LOCAL model, we show how to turn it into a LOCAL-model algorithm with locality $O(\log^* n)$. In this part, the key insight is that *we cannot directly simulate* online-LOCAL in LOCAL. Instead, we can use an online-LOCAL algorithm with a constant locality to find a *canonical labeling* for each possible input-labeled fragment, and use this information to design a LOCAL-model algorithm. The main trick is that we first present only disconnected path fragments to an online-LOCAL algorithm, and force it to commit to some output labeling in each fragment without knowing how the fragments are connected to each other.

In the case of rooted regular trees (Section 5.2), we face the same fundamental challenge: we cannot directly simulate black-box online-LOCAL algorithms in the LOCAL model. Instead, we need to look at the combinatorial properties of a given LCL problem Π. We proceed in two steps: (1) Assume that the locality of Π is $n^{\Theta(1)}$ in the LOCAL model; we need to show that the locality is $n^{\Theta(1)}$ also in the online-LOCAL model. Using the result of [7], high LOCAL-model locality implies that the structure of Π has to have certain "inflexibilities", and we use this property to present a strategy that the adversary can use to force any online-LOCAL algorithm with locality $n^{o(1)}$ to fail. (2) Assume that we have an online-LOCAL algorithm A for Π with locality $o(\log n)$; we need to show that the locality is $O(\log^* n)$ in the LOCAL model. Here we design a family of inputs and a strategy of the adversary that forces algorithm A to construct a "certificate" (in the sense of [7]) that shows that Π is efficiently solvable in the LOCAL model.

For 3-coloring bipartite graphs in online-LOCAL (Section 4), we make use of the following ideas. We maintain a collection of graph fragments such that each of the fragments has got a boundary that is properly 2-colored. Each such fragment has got one of two possible parities (let us call them here "odd" and "even") with respect to the underlying bipartition. We do not know the global parity of a given graph fragment until we have seen almost the entire graph. Nevertheless, it is possible to merge two fragments and maintain the invariant: if two fragments A and B have parities that are not compatible with each other, we can surround either A or B with a *barrier* that uses the third color, and thus change parities. Now we can merge A and B into one fragment that has got a properly 2-colored boundary. The key observation here is that we can make a *choice* between surrounding A vs. B, and if we always pick the one with the smallest number of nested barriers, we never need to use more than a logarithmic number of nested barriers. It turns out that this is enough to ensure that seeing up to distance $O(\log n)$ suffices to color any node chosen by the adversary.

1.7 Open questions

Our work gives rise to a number of open questions. First, we can take a more fine-grained view of the results in Tables 1 and 2:

1. Is there any problem in rooted trees with locality $\Theta(n^\alpha)$ in the online-LOCAL model and locality $\Theta(n^\beta)$ in the LOCAL model, for some $\alpha < \beta$?
2. Is it possible to find a 3-coloring in 2-dimensional grids in the dynamic-LOCAL model with locality $O(\log n)$?
3. Is it possible to find a 3-coloring in bipartite graphs in the online-LOCAL model with locality $o(\log n)$?

Perhaps even more interesting is what happens if we consider *unrooted* trees instead of rooted trees. In unrooted trees we can separate *randomized* and *deterministic* versions of the LOCAL model [16], and SLOCAL is strong enough to derandomize randomized LOCAL-model algorithms [25]; hence the key question is:

4. Does randomized-LOCAL $\approx$ SLOCAL $\approx$ dynamic-LOCAL $\approx$ online-LOCAL hold for LCL problems in unrooted trees?

Finally, our work shows a trade-off between the competitive ratio and the locality of coloring: With locality $O(\log n)$, one can achieve $O(1)$-coloring of a bipartite graph, and to achieve locality 0, one needs to use $\Omega(\log n)$ colors. This raises the following question:

5. What trade-offs exist between the locality and number of colors needed to color a (bipartite) graph in the online-LOCAL model?

2 Definitions and related work

Throughout this work, graphs are simple, undirected, and finite, unless otherwise stated. We write $G = (V, E)$ for a graph G with the set of nodes V and the set of edges E, and we use n to denote the number of nodes in the graph. For a node v and a natural number T, we use $B(v, T)$ to denote the set of all nodes in the radius-T neighborhood of node v. For a set of nodes U, we write $G[U]$ for the subgraph of G induced by U. By radius-T neighborhood of v we refer to the induced subgraph $G[B(v, T)]$, together with possible input and output labelings.

We use the following notation for graph problems. We write $\mathcal{G}$ for the family of graphs, Σ for the set of input labels, and Γ for the set of output labels. For a graph $G = (V, E)$, we write $I\colon V \to \Sigma$ for the input labeling and $L\colon V \to \Gamma$ for the output labeling. We consider here node labelings, but edge labelings can be defined in an analogous manner. A *graph problem* Π associates with each possible *input* (G, I) a set of feasible *solutions* L; this assignment must be invariant under graph isomorphism.

Locality. In what follows, we define five models of computing: LOCAL, SLOCAL, two versions of dynamic-LOCAL, and online-LOCAL. In all of these models, an algorithm is characterized by a *locality* T (a.k.a. locality radius, local horizon, time complexity, or round complexity, depending on the context). In general, T can be a function of n. We assume that the algorithm knows the value of n.

In each of these models $\mathcal{M}$, we say that algorithm $\mathcal{A}$ solves problem Π if, for each possible input (G, I) and for each possible adversarial choice, the labeling L produced by $\mathcal{A}$ is a feasible solution. We say that problem Π has locality T in model $\mathcal{M}$ if T is the pointwise smallest function such that there exists an $\mathcal{M}$-model algorithm $\mathcal{A}$ that solves Π with locality at most T.

LOCAL model. In the LOCAL model of distributed computing [37, 44], the adversary labels the nodes with unique identifiers from $\{1, 2, \ldots, \mathrm{poly}(n)\}$. In a LOCAL model algorithm, each node in parallel chooses its local output based on its radius-T neighborhood (the output may depend on the graph structure, input labels, and the unique identifiers).

Naor and Stockmeyer [42] initiated the study of the locality of LCL problems (see Definition 3) in the LOCAL model. Today, LCL problems are well classified with respect to their locality for the special cases of paths [4, 5, 13, 18, 42], grids [13], directed and undirected trees [5, 7, 8, 15, 17] as well as general graphs [13, 42], with only a few unknown gaps [7].

SLOCAL model. In the SLOCAL model [26], we have got adversarial unique identifiers similar to the LOCAL model, but the nodes are processed sequentially with respect to an adversarial input sequence $\sigma = v_1, v_2, v_3, \ldots, v_n$. Each node v is equipped with an unbounded local memory; initially, all local memories are empty. When a node v is processed, it can query the local memories of the nodes in its radius-T neighborhood, and based on this information, it has to decide what is its own final output and what to store in its own local memory.

The SLOCAL model has been used as a tool to e.g. better understand the role of randomness in the LOCAL model [25, 26]. It is also well-known that SLOCAL is strictly stronger than LOCAL. For example, it is trivial to find a maximal independent set greedily in the SLOCAL model, while this is a nontrivial problem in the general case in the LOCAL model [6, 36]. There are many LCL problems with LOCAL-locality $\Theta(\log^* n)$ [19, 37], and

all of them have SLOCAL-locality $O(1)$. There are also LCL problems (e.g. the so-called *sinkless orientation* problem), where the locality in the (deterministic) LOCAL model is $\Theta(\log n)$, while the locality in the (deterministic) SLOCAL model is $\Theta(\log \log n)$ [16, 25].

Dynamic-LOCAL model. To our knowledge, there is no standard definition or name for what we call dynamic-LOCAL here; however, the idea has appeared implicitly in a wide range of work. For example, many efficient dynamic algorithms for graph problems, such as vertex or edge coloring, maximal independent set, and maximal matching, also satisfy the property that the solution is only modified in the (immediate) local neighborhood of a point of change [3, 9, 11, 21, 28, 34, 43], and hence all of them fall in the class dynamic-LOCAL.

We use the following definition for dynamic-LOCAL: Computation starts with an empty graph G_0. In step i, the adversary constructs a supergraph G_i of G_{i-1} such that G_i and G_{i-1} differ in only one edge or one node; let C_i denote the set of nodes v in G_i with $G_i[B(v,T)] \neq G_{i-1}[B(v,T)]$, i.e., nodes that are within distance at most T from the point of change. In each step, the algorithm has to produce a feasible labeling L_i for problem Π in graph G_i, and the labeling can only be modified in the local neighborhood of a point of change, i.e., $L_i(v) = L_{i-1}(v)$ for all $v \notin C_i$.

Note that we defined the dynamic-LOCAL model for the *incremental* case, where nodes and edges are only added. If we do not require that G_i is a supergraph of G_{i-1}, we arrive at what we call the dynamic-LOCAL$^{\pm}$ model with both additions and deletions.

Online graph algorithms. In online graph algorithms, nodes are processed sequentially with respect to an adversarial input sequence $\sigma = v_1, v_2, \ldots, v_n$. Let $\sigma_i = v_1, v_2, \ldots, v_i$ denote the first i nodes of the sequence, and let $G_i = G[\{v_1, v_2, \ldots, v_i\}]$ be the subgraph induced by these nodes. When the adversary presents a node v_i, the algorithm has to label v_i based on σ_i and G_i.

Online algorithms on graphs have been studied for many problems such as matching [35] and independent set [30], but closest to our work is the extensive literature on online graph coloring [1, 10, 29, 31, 33, 38, 47]. There is also prior work that has considered various ways to strengthen the notion of online algorithms; the performance of online algorithms can be improved by letting the algorithm know the input graph [20, 32], by giving it an advice string [12, 14, 22] with knowledge about the request sequence, or allowing the algorithm to delay decisions [23]. The online-LOCAL model can be interpreted as online graph algorithms with spatial advice, and it can also be interpreted as a model where the online algorithm can delay its decision for node v until it has seen the whole neighborhood around v (this interpretation is equivalent to the definition we give next).

Online-LOCAL model. We define the online-LOCAL model as follows. The nodes are processed sequentially with respect to an adversarial input sequence $\sigma = v_1, v_2, \ldots, v_n$. Let $\sigma_i = v_1, v_2, \ldots, v_i$ denote the first i nodes of the sequence, 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 a node v_i, the algorithm has to label v_i based on σ_i and G_i.

Observe that any online graph algorithm is an online-LOCAL algorithm with locality 0. Further note that in the online-LOCAL model, unique identifiers would not give any additional information. This is because the nodes can always be numbered with respect to the point in time when the algorithm first sees them in some G_i.

Yet another way to interpret the online-LOCAL model is that it is an extension of the SLOCAL model, where the algorithm is equipped with unbounded *global memory* where it can store arbitrary information on what has been revealed so far. When they introduced

the SLOCAL model, Ghaffari, Kuhn, and Maus [26] mentioned the possibility of such an extension but pointed out that it would make the model "too powerful", as just one bit of global memory would already make it possible to solve e.g. leader election (and this observation already shows that the online-LOCAL model is indeed strictly stronger than the SLOCAL model). In our work, we show that even though online-LOCAL can trivially solve e.g. leader election thanks to the global memory, it is not that easy to exploit this extra power in the context of LCL problems. Indeed, online-LOCAL turns out to be as weak as SLOCAL when we look at LCL problems in paths, cycles, and rooted trees.

Local computation algorithms. We do not discuss local computation algorithms (LCAs) [2, 24, 39, 40, 46, 46] in this work in more detail, but we briefly point out a direct connection between the online-LOCAL model and LCAs. It is known that for a broad family of graph problems (that includes LCLs), we can w.l.o.g. assume that whenever the adversary queries a node v, the LCA makes probes to learn a *connected* subgraph around node v [27]. For such problems, an online-LOCAL algorithm with locality T is at least as strong as an LCA that makes T probes per query: an LCA can learn some *subgraph* of the radius-T neighborhood of v and, depending on the size of the state space, remember some part of that, while in the online-LOCAL model the algorithm can learn the entire radius-T neighborhood of v and remember all of that. We leave a more detailed exploration of the distinction between distance (how far to see) and volume (how much to see), in the spirit of e.g. [41, 45], for future work.

3 Landscape of models

As an introduction to the models, we first check that all relations in Figure 1 indeed hold. In each case, we are interested in *asymptotic* equivalence: for example, when we claim that $A \subsetneq B$, the interpretation is that locality T in model A implies locality $O(T)$ in model B, but the converse is not true. Note that the relation between the online-LOCAL problems and the online graph algorithms has already been discussed in Sections 1 and 2.

Inclusions. Let us first argue that the subset relations in Figure 1 hold. These cases are trivial:

- Any LOCAL algorithm can be simulated in the SLOCAL model, and any SLOCAL algorithm can be simulated in the online-LOCAL model (this is easiest to see if one interprets online-LOCAL as an extension of SLOCAL with the global memory).
- Any dynamic-LOCAL$^{\pm}$ algorithm can be directly used in the dynamic-LOCAL model (an algorithm that supports both additions and deletions can handle additions).

These are a bit more interesting cases:

- To simulate a LOCAL algorithm A in the dynamic-LOCAL$^{\pm}$ model, we can simply recompute the entire output with A after each change. If the locality of A is T, then the output of A only changes within distance T from a point of change.
- To simulate a dynamic-LOCAL algorithm A in the online-LOCAL model, we proceed as follows: When the adversary reveals a node v, we feed v along with the new nodes in its radius-$O(T)$ neighborhood to A edge by edge. Now there will not be any further changes within distance T from v, and hence A will not change the label $L(v)$ of v anymore. Hence the online-LOCAL algorithm can also label v with $L(v)$.

Table 3 Problems that we use to separate the models, and the bounds that we show for their locality.

Problem	LOCAL	SLOCAL	dynamic-LOCAL$^{\pm}$	dynamic-LOCAL	online-LOCAL
3-coloring paths	$\Omega(\log^* n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
weak reconstruction	$\Omega(n)$	$\Omega(n)$	$O(1)$	$O(1)$	$O(1)$
cycle detection	$\Omega(n)$	$\Omega(n)$	$\Omega(n)$	$O(1)$	$O(1)$
component-wise leader election	$\Omega(n)$	$\Omega(n)$	$\Omega(n)$	$\Omega(n)$	$O(1)$
nested orientation	$\omega(1)$	$O(1)$	$\omega(1)$	$\omega(1)$	$O(1)$

Separations. To prove the separations of Figure 1, we make use of the classic distributed graph problem of 3-*coloring paths*, as well as the following problems that are constructed to highlight the differences between the models:

- *Weak reconstruction:* in each connected component C there has to be at least one node v such that its label $L(v)$ is an encoding of a graph isomorphic to C.
- *Cycle detection:* for each cycle there has to be at least one node that outputs "yes", and each node that outputs "yes" has to be part of at least one cycle.
- *Component-wise leader election:* in each connected component exactly one node has to be marked as the leader.
- *Nested orientation:* find an acyclic orientation of the edges and label each node recursively with its own identifier, the identifiers of its neighbors, and the labels of its in-neighbors (see the full version for the precise definition).

We can prove the bounds shown in Table 3 for the locality of these problems in the five models; see the full version for the details. Now each separation in Figure 1 follows from one of the rows of Table 3.

4 3-coloring bipartite graphs

In this section, we present our Contribution 3: we design an algorithm for 3-coloring bipartite graphs in the online-LOCAL model and show that this gives us an exponential separation between the SLOCAL and online-LOCAL models. This section also serves as an introduction into the algorithmic techniques that work in online-LOCAL. Equipped with this understanding, in Section 5, we start to develop more technical tools that we need for our Contribution 2.

By prior work [13], it is known that the locality of 3-coloring in $\sqrt{n} \times \sqrt{n}$ grids is at least $\Omega(\sqrt{n})$ in the LOCAL model. The aforementioned paper considers the case of *toroidal* grid graphs, but the same argument can be applied for non-toroidal grids (in essence, if you could color locally anywhere in the middle of a non-toroidal grid, you could also apply the same algorithm to color a toroidal grid). We can easily extend this result to show a polynomial lower bound for 3-coloring grids in the SLOCAL model:

▶ **Theorem 2.** *There is no SLOCAL algorithm that finds a 3-coloring in 2-dimensional grids with locality $o(n^{1/10})$.*

To prove the result, we show that we can simulate SLOCAL algorithms sufficiently efficiently in the LOCAL model. We use the standard technique of first precomputing a distance-$o(n^{1/10})$ coloring, and then using the colors as a schedule for applying the SLOCAL algorithm. Such a simulation can be done efficiently and would lead to a LOCAL algorithm running in $o(\sqrt{n})$ time, which is a contradiction. The full proof of the lower bound is presented

in the full version of the paper. As grids are bipartite graphs, the problem of 3-coloring in grids already gives an exponential separation between the SLOCAL and online-LOCAL models. For the special case of grids, we discuss the known locality bounds for the coloring problem in the full version. A summary of these results can be found in Table 2.

In Section 4.1, we introduce the 3-coloring algorithm in the online-LOCAL, and we use the special case of grids in order to visualize it. Besides providing a natural separation between the SLOCAL and online-LOCAL models, the 3-coloring problem also shows the advantage of allowing online algorithms to look around: while the best online coloring algorithm on bipartite graphs is $\Theta(\log n)$-competitive, our algorithm in the online-LOCAL model achieves a competitive ratio of 1.5.

Note that an optimal solution would be to color a bipartite graph with 2 colors. In all models that we consider here, we know it is not possible to solve 2-coloring with locality $o(n)$, the worst case being a path with n nodes. We show that allowing an online-LOCAL algorithm to use only one extra color makes it possible to find a valid coloring with locality $O(\log n)$:

▶ **Theorem 1.** *There is an online-LOCAL algorithm that finds a 3-coloring in bipartite graphs with locality $O(\log n)$.*

4.1 Algorithm for 3-coloring bipartite graphs in online-LOCAL

Algorithm overview. The high-level idea of our online-LOCAL algorithm is to color the presented nodes of the graph with 2 colors until the algorithm sees two areas where the 2-colorings are not compatible. In essence, when the adversary presents a node far from any other node the algorithm has seen, the algorithm blindly start constructing a 2-coloring. When the adversary presents nodes in the neighborhood of already colored nodes, the algorithm simply expands the 2-colored component – we call such a component a *group*. The algorithm keeps expanding such properly 2-colored groups until, eventually, two groups with incompatible 2-colorings meet (i.e., groups that have different *parities*). Then, the algorithm uses the third color in order to create a *barrier* around one of the groups, effectively flipping its parity. Our algorithm thereby makes use of the knowledge of previously queried neighborhoods that are given by the online-LOCAL model: the algorithm is *committing* to colors for nodes in the revealed subgraphs before they are queried.

Algorithm in detail. At the beginning, no nodes are revealed to the algorithm, and we therefore say that all nodes are *unseen*. We refer to connected components of the subgraph G_i of G as *groups*. With each of these groups, we associate a *border count*, which is a natural number that is initially 0. The algorithm uses colors 0 and 1 for the 2-coloring, reserving color 2 as the barrier color. Each time the adversary points at a node v_i, the algorithm gets to see the radius-T neighborhood $B(v_i, T)$ of this node. Now consider different types of nodes in $B(v_i, T+1)$. There are three different cases that the algorithm needs to address (we visualize them in Figures 2–4 using grids as an example):

1. *All nodes in $B(v_i, T+1)$ are unseen.* In this case, the nodes in $B(v_i, T)$ form a new connected component, i.e. a new *group*. This group has a *border count* of 0. The algorithm colors v_i with 0, thus fixing the parity for this group (see Figure 2).
2. *The algorithm has already seen some nodes in $B(v_i, T+1)$, but all of them belong to the same group.* In this case, the adversary has shown an area next to an existing group. If v_i was already committed to a color, the algorithm uses that color. Otherwise, the algorithm colors v_i according to the 2-coloring of the group. All nodes in $B(v_i, T)$ are now considered to be in this group (see Figure 3).

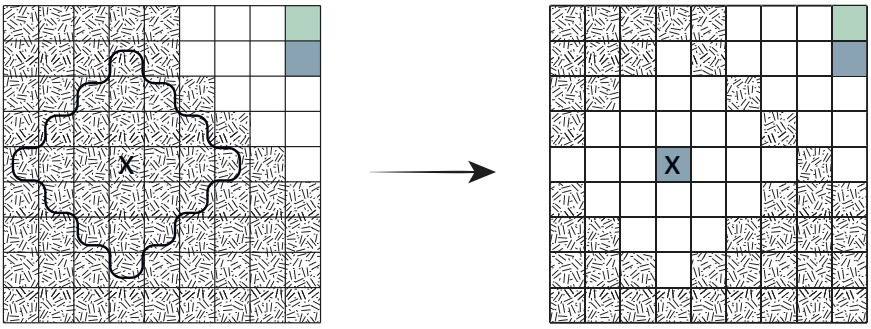

Figure 2 3-coloring algorithm, case 1/3. The adversary queries node x. Here node x is in the middle of an unseen region (shaded). The algorithm creates a new group (white) and fixes the color of node x arbitrarily.

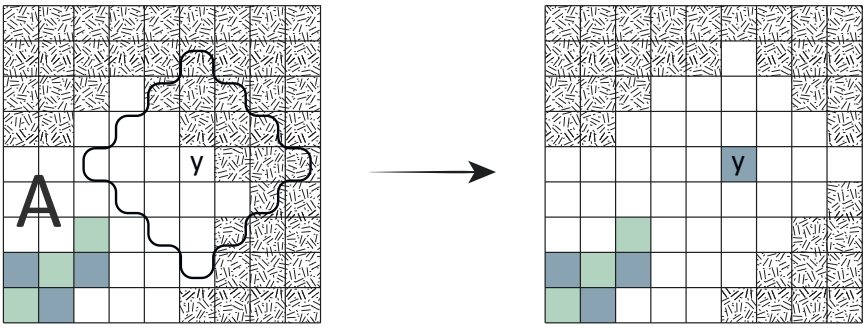

Figure 3 3-coloring algorithm, case 2/3. The adversary queries node y. Some nodes in the local neighborhood of y are already part of a group (white), and hence y joins this group. The algorithm fixes the color of node y so that it is consistent with the coloring of the group.

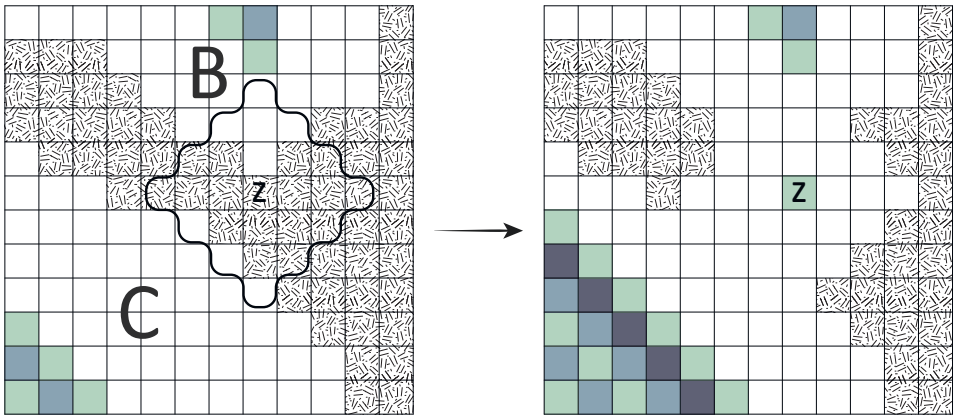

Figure 4 3-coloring algorithm, case 3/3. The adversary queries node z. Some nodes in the local neighborhood of z belong to two different groups, B and C. The algorithm merges the groups. As they have incompatible parities, the algorithm adds a new border around one of the groups, in this case C, as both groups have the same number of borders around them and the algorithm can choose arbitrarily. Nodes in the local neighborhood of z join the group, and z is colored in a way compatible with the coloring of the newly created group.

Algorithm 1 join_groups(A, B).

Input: Groups A, B
Output: Group X
if *A and B have different parities* **then**
  Let S be the group with the smaller border count. If they are equal, $S = A$;
  For all nodes of color 0 in S, commit all uncolored neighbors to color 1;
  For all nodes of color 1 in S, commit all uncolored neighbors to color 2;
  For all nodes of color 2 in S, commit all uncolored neighbors to color 0;
  Increase border count of S by 1.
end
Set all nodes in groups A, B to be in group X;
Set the border count for X to be the maximum of border counts for A, B and S;
return X

3. *There are nodes in $B(v_i, T+1)$ that belong to different groups.* In this case, the algorithm has to join groups. Here, we only define the join of two groups A and B; if there are more groups, this join can be applied iteratively.
   If A and B have different parities (i.e., the 2-colorings at their boundaries are not compatible), the algorithm takes the group with the smaller border count and uses a layer of nodes of color 2 to create a barrier that changes its parity, and then it increases the group's border count; see Algorithm 1 for the details. Then, the algorithm joins the groups, that are now compatible, and sets the border count of the newly created group to the maximum of the border counts of A and B.
   By merging all groups in the local neighborhood of v_i, the algorithm eventually ends up in a situation where v_i only sees nodes in a single group, and we are in a scenario similar to case 2 above: nodes in the local neighborhood of v_i also join the newly created group, and if v_i has not already committed to a color, the algorithm colors it according to the 2-coloring of this group (see Figure 4).

4.2 Analysis of the 3-coloring Algorithm in online-LOCAL

In order to show the correctness of the coloring algorithm, we first prove that this process creates a valid 3-coloring provided that all our commitments remain within the visible area, that is, inside subgraph G_i. Next, we show that by choosing $T(n) = O(\log n)$, all our commitments indeed remain inside the visible area. Together, these parts prove Theorem 1.

Validity of the 3-coloring. We first prove that our algorithm always continues a valid 3-coloring, as long as it does not need to make commitments to unseen nodes. We consider all three cases of the algorithm individually.

1. *All nodes in $B(v_i, T+1)$ are unseen.* In this case, the algorithm colors v_i with 0. As all neighboring nodes were unseen, they have not been committed to any color, and thus this case causes no errors.
2. *The algorithm has already seen some nodes in $B(v_i, T+1)$, but all of them belong to the same group.* In this case, the algorithm would either use the committed color or the parity of the group. As previously committed colors do not cause errors, and the group has consistent parity, this case cannot cause any errors.

3. *There are nodes in $B(v_i, T+1)$ that belong to different groups.* In this case, we want to join groups without breaking the coloring. If the two groups have the same parity, clearly, no errors can be caused by continuing the 2-coloring. The interesting case is when the two groups have different parities. Then, we need to show that the new commitments made by Algorithm 1 do not create any errors.

   Let S be the group with the smaller border count. By examining Algorithm 1, we can see that all colored nodes that have uncolored neighbors are either of color 0 or color 1: only in line 4, nodes can be colored with color 2, and all of those nodes' neighbors are then colored in line 5. Thus, in order for an error to occur, there either needs to be two nodes of colors 0 and 1 that have uncolored neighbors and different parities in S, or the algorithm commits to a color of a node that it has not yet seen. This could cause an error, as two groups could commit a single node to two different colors.

   As for the first case, we assume that all nodes in S that have uncolored neighbors also have consistent parity. This trivially holds for a group that has border count 0, as all colored nodes in it have the same parity. From the assumption, it follows that all nodes colored with 1 in line 3 have the same parity, so they cannot create an error. After this, all colored nodes with uncolored neighbors in the group have the same parity, and are colored with 1. Thus all nodes colored with 2 in line 4 also have the same parity, as do the nodes colored with 0 in line 5. As these are the only lines where nodes are colored, this procedure cannot create any errors. It also ensures that, after the procedure, the only colored nodes in the group that have uncolored neighbors are the nodes colored in line 5, which have the same parity. Therefore, our assumption holds for all groups. Those nodes also have a parity different from the nodes in S that had uncolored neighbors before this procedure, so in essence, we have flipped the parity of group S to match the parity of the other group.

   As for the second case, this can be avoided by choosing a large enough T, so that all commitments remain within the visible area of G_i. Next, we discuss how to choose such a T.

Locality of the 3-coloring algorithm. In this part, we prove that by choosing locality $T(n) = 3\lceil \log_2 n \rceil = O(\log n)$, no nodes outside the visible area of G_i need to be committed.

We first make the observation that a group with border count b contains at least 2^b nodes; this is a simple induction:

$b = 0$: A newly created group contains at least 1 node.

$b > 0$: Consider the cases in which Algorithm 1 returns a group X with border count b. One possibility is that A or B already had border count b, and hence by assumption it already contained at least 2^b nodes. The only other possibility is that both A and B had border count exactly $b-1$, they had different parities, one of the border counts was increased, and hence X has now got a border count of b. But, in this case, both A and B contained at least 2^{b-1} nodes each.

Hence the border count is bounded by $b \leq \log_2 n$ in a graph with n nodes.

We next consider the maximum distance between a node that the adversary has queried and a node with a committed color. Note that the only place where the algorithm commits a color to a node that the adversary has not queried yet is when building a border around a group. There are three steps (lines 3–5) where the algorithm commits to the color of a neighbor of a committed node, and thus effectively extends the distance by at most one in each step. Therefore, if the border count is b, in the worst case, the algorithm commits a color for a node that is within distance $3b$ from a node that was queried by the adversary.

As we have $b \leq \log_2 n$, a locality of $3\lceil \log_2 n \rceil \geq 3b$ suffices to ensure that all the commitments of the algorithm are safely within the visible region. This concludes the proof of Theorem 1.

5 LCL problems in paths, cycles, rooted regular trees

We just showed that the online-LOCAL model is much more powerful than LOCAL and SLOCAL for an LCL on bipartite graphs and grids. In this section, we discuss what happens when we restrict our attention to LCL problems in paths, cycles, and trees. We start by defining LCL problems more formally.

We say that Π is a *locally verifiable problem* with verification radius r if the following holds: there is a collection of labeled local neighborhoods $\mathcal{T}$ such that L is a feasible solution for input (G, I) if and only if for all nodes v, the radius-r neighborhood of v in (G, I, L) is in $\mathcal{T}$. Informally, a solution is feasible if it looks good in all radius-r neighborhoods.

▶ **Definition 3** (Locally checkable labeling [42]). *A locally verifiable problem Π is a* locally checkable labeling (LCL) *problem if the set of the input labels Σ is finite, the set of the output labels Γ is finite, and there is a natural number Δ such that maximum degree of any graph $G \in \mathcal{G}$ is at most Δ.*

Note that in LCL problems, $\mathcal{T}$ is also finite since there are only finitely many possible non-isomorphic labeled local neighborhoods.

It turns out that in the case of paths, cycles, and rooted regular trees, the LOCAL, SLOCAL, dynamic-LOCAL, and online-LOCAL models are all approximately equally expressive for LCL problems. In particular, all classification and decidability results related to LCLs in paths, cycles, and rooted regular trees in the LOCAL model [4, 7, 18] directly apply also in the online-LOCAL model, the SLOCAL model, and both versions of the dynamic-LOCAL model.

We show first that the LOCAL and online-LOCAL models are equivalent in the case of paths and cycles, even when the LCL problems can have inputs. We then continue to prove that the models are equivalent also in the more general case of LCL problems rooted regular trees, but in this case we do not consider the possibility of having input labels.

Formally, we prove the following theorem for cycles and paths:

▶ **Theorem 4.** *Let Π be an LCL problem in paths or cycles (possibly with inputs). If the locality of Π is T in the online-LOCAL model, then its locality is $O(T + \log^* n)$ in the LOCAL model.*

For the case of rooted trees, we prove the following two theorems:

▶ **Theorem 5.** *Let Π be an LCL problem in rooted regular trees (without inputs). Problem Π has locality $n^{\Omega(1)}$ in the LOCAL model if and only if it has locality $n^{\Omega(1)}$ in the online-LOCAL model.*

▶ **Theorem 6.** *Let Π be an LCL problem in rooted regular trees (without inputs). Problem Π has locality $\Omega(\log n)$ in the LOCAL model if and only if it has locality $\Omega(\log n)$ in the online-LOCAL model.*

These two theorems show that all LCL problems in rooted regular trees belong to one of the known complexity classes $O(\log^* n)$, $\Theta(\log n)$ and $n^{\Omega(1)}$ in all of the models we study. In what follows, we introduce the high-level ideas of the proofs of these theorems. For full proofs, we refer the reader to the full version.

5.1 Cycles and paths

We prove Theorem 4 by first showing that any LCL problem in cycles and paths has either locality $O(1)$ or $\Omega(n)$ in the online-LOCAL model. Next, we show that if a problem is solvable with locality $O(1)$ in the online-LOCAL model, then it is also solvable in locality $O(\log^* n)$ in the LOCAL model. These steps are described by the following two lemmas:

▶ **Lemma 7.** *Let Π be an LCL problem in paths or cycles (possibly with inputs), and let $\mathcal{A}$ be an online-LOCAL algorithm solving Π with locality $o(n)$. Then, there exists an online-LOCAL algorithm $\mathcal{A}'$ solving Π with locality $O(1)$.*

The high-level idea of the proof of Lemma 7 is to construct a large virtual graph P' such that when the original algorithm runs on the virtual graph P', the labeling produced by the algorithm is locally compatible with the labeling in the original graph P. We ensure this by applying a pumping-lemma-style argument on the LCL problem. The proof uses similar ideas as the ones presented by Chang and Pettie [17].

▶ **Lemma 8.** *Let Π be an LCL problem in paths or cycles (possibly with inputs), and let $\mathcal{A}$ be an online-LOCAL algorithm solving Π with locality $O(1)$. Then, there exists a LOCAL algorithm $\mathcal{A}'$ solving Π with locality $O(\log^* n)$.*

The high-level idea of the proof of Lemma 8 is to use the constant locality online-LOCAL algorithm to construct a canonical output labeling for each possible neighborhood of input labels. The fast LOCAL algorithm can then use these canonical labelings in disjoint neighborhoods of the real graph, and the construction of the canonical labelings ensures that the labeling also extends to the path segments between these neighborhoods.

The full proofs of these lemmas can be found in the full version of this paper. In order to prove Theorem 4, it is sufficient to combine these lemmas with the fact that the possible localities on paths and cycles in the LOCAL model are $O(1)$, $\Theta(\log^* n)$ and $\Theta(n)$ [18].

5.2 Rooted regular trees

We prove the equivalence of the LOCAL and the online-LOCAL models for LCL problems in rooted regular trees in two parts. We start out with Theorem 5 and show that if an LCL problem requires locality $n^{\Omega(1)}$ in the LOCAL model, then for every locality-$n^{o(1)}$ online-LOCAL algorithm we can construct an input instance which the algorithm must fail to solve. To prove Theorem 6, we show that a locality-$o(\log n)$ online-LOCAL algorithm for solving an LCL problem implies that there exists a locality-$O(\log^* n)$ LOCAL algorithm for solving that same problem. In the following, we outline the proofs of both theorems; the full proofs can be found in the full version of the paper. Before considering the full proof, we advise the reader to look at the example in the full version of this paper, where we show that the 2.5-coloring problem requires locality $\Omega(\sqrt{n})$ in the online-LOCAL model.

Proof outline of Theorem 5. Our proof is based on the fact that any LCL problem requiring locality $n^{\Omega(1)}$ in the LOCAL model has a specific structure. In particular, the problem can be decomposed into a sequence of *path-inflexible* labels and the corresponding sequence of more and more restricted problems [7]. Informally, a label is path-inflexible if two nodes having that label can exist only at specific distances apart from each other. For example, when 2-coloring a graph, two nodes having label 1 can exist only at even distances from each other. The problems in the path-inflexible decomposition are formed by removing the path-inflexible labels from the previous problem in the sequence until an empty problem is reached.

This decomposition of the problem into restricted problems with path-inflexible labels allows us to construct an input graph for any locality-$n^{o(1)}$ online-LOCAL algorithm. In particular, we force the algorithm to commit labels in disjoint fragments of the graph. Any label that the algorithm uses must be a path-inflexible label in some problem of the sequence of restricted problems. By combining two fragments containing labels that are path-inflexible in the same problem, we can ensure that the algorithm cannot solve that problem in the resulting graph. Hence the algorithm must use a label from a problem earlier in the sequence. At some point, the algorithm must use labels that are path-inflexible in the original problem. At that point, we can combine two fragments having path-inflexible labels in the original problem in such a way that no valid labeling for the original problem exists, and hence the algorithm must fail to solve the problem on the resulting graph.

Proof outline of Theorem 6. Here, we show that a locality-$o(\log n)$ online-LOCAL algorithm solving an LCL problem implies that there exists a *certificate for* $O(\log^* n)$ *solvability* for that problem. It is known that the existence of such a certificate for a problem implies that there exists a locality-$O(\log^* n)$ LOCAL algorithm for solving the problem [7].

Informally, the certificate for $O(\log^* n)$ solvability for LCL problem Π with label set Γ and arity δ consists of a subset $\Gamma_{\mathcal{T}} = \{\gamma_1, \ldots, \gamma_t\}$ of labels Γ, and two sequences of correctly labeled complete δ-ary trees $\mathcal{T}^1$ and $\mathcal{T}^2$. The leaves of each tree in the sequence $\mathcal{T}^1$ (resp. $\mathcal{T}^2$) are labeled in the same way using only labels from set $\Gamma_{\mathcal{T}}$. For every label of set $\Gamma_{\mathcal{T}}$, there exists a tree in both of the sequences having a root labeled with that label.

We can use the online-LOCAL algorithm to construct such a certificate. We do this by constructing exponentially many deep complete δ-ary trees and using the algorithm to label nodes in the middle of those trees. We then glue these trees together in various ways. When the trees are glued together, we use the online-LOCAL algorithm to label the rest of the nodes to form one tree of the sequence. We repeat this procedure until all trees of both sequences have been constructed.

References

1 Susanne Albers and Sebastian Schraink. Tight bounds for online coloring of basic graph classes. *Algorithmica*, 83(1):337–360, 2021. doi:10.1007/s00453-020-00759-7.

2 Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 1132–1139. SIAM, 2012. doi:10.1137/1.9781611973099.89.

3 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proc. 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2018)*, pages 815–826, 2018. doi:10.1145/3188745.3188922.

4 Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. The distributed complexity of locally checkable problems on paths is decidable. In *Proc. 38th ACM Symposium on Principles of Distributed Computing (PODC 2019)*, pages 262–271. ACM Press, 2019. doi:10.1145/3293611.3331606.

5 Alkida Balliu, Sebastian Brandt, Yuval Efron, Juho Hirvonen, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Classification of distributed binary labeling problems. In *Proc. 34th International Symposium on Distributed Computing (DISC 2020)*, pages 17:1–17:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.DISC.2020.17.

6 Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. *Journal of the ACM*, 68(5), 2021. doi:10.1145/3461458.

7 Alkida Balliu, Sebastian Brandt, Dennis Olivetti, Jan Studený, Jukka Suomela, and Aleksandr Tereshchenko. Locally checkable problems in rooted trees. In *Proc. 40th ACM Symposium on Principles of Distributed Computing (PODC 2021)*, pages 263–272. ACM Press, 2021. doi:10.1145/3465084.3467934.

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

9 Leonid Barenboim and Tzalik Maimon. Fully dynamic graph algorithms inspired by distributed computing: Deterministic maximal matching and edge coloring in sublinear update-time. *ACM Journal of Experimental Algorithmics*, 24, 2019.

10 Dwight R. Bean. Effective coloration. *The Journal of Symbolic Logic*, 41(2):469–480, 1976. doi:10.2307/2272247.

11 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 1–20. SIAM, 2018. doi:10.1137/1.9781611975031.1.

12 Maria Paola Bianchi, Hans-Joachim Böckenhauer, Juraj Hromkovič, and Lucia Keller. Online coloring of bipartite graphs with and without advice. In *Computing and Combinatorics*, pages 519–530, 2012. doi:10.1007/978-3-642-32241-9_44.

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

14 Elisabet Burjons, Juraj Hromkovič, Xavier Muñoz, and Walter Unger. Online graph coloring with advice and randomized adversary. In *Proc. 42nd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2016)*, pages 229–240. Springer, 2016. doi:10.1007/978-3-662-49192-8_19.

15 Yi-Jun Chang. The complexity landscape of distributed locally checkable problems on trees. In *Proc. 34th International Symposium on Distributed Computing (DISC 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.DISC.2020.18.

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

17 Yi-Jun Chang and Seth Pettie. A Time Hierarchy Theorem for the LOCAL Model. *SIAM Journal on Computing*, 48(1):33–69, 2019. doi:10.1137/17M1157957.

18 Yi-Jun Chang, Jan Studený, and Jukka Suomela. Distributed graph problems through an automata-theoretic lens. In *Proc. 28th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2021)*, pages 31–49. Springer, 2021. doi:10.1007/978-3-030-79527-6_3.

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

20 Stefan Dobrev, Rastislav Královič, and Richard Královič. Independent set with advice: The impact of graph knowledge. In *Proc. 10th Workshop on Approximation and Online Algorithms (WAOA 2012)*. Springer, 2013. doi:10.1007/978-3-642-38016-7_2.

21 Yuhao Du and Hengjie Zhang. Improved algorithms for fully dynamic maximal independent set, 2018. arXiv:1804.08908.

22 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. In *Proc. 36th edition of the International Colloquium on Automata, Languages and Programming (ICALP 2009)*, pages 427–438. Springer, 2009. doi:10.1007/978-3-642-02927-1_36.

23 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: Haste makes waste! In *Proc. 48th Annual ACM Symposium on Theory of Computing (STOC 2016)*, pages 333–344, 2016. doi:10.1145/2897518.2897557.

24 Guy Even, Moti Medina, and Dana Ron. Deterministic stateless centralized local algorithms for bounded degree graphs. In *Proc. 22nd European Symposium on Algorithms (ESA 2014)*, pages 394–405. Springer, 2014. doi:10.1007/978-3-662-44777-2_33.

25 Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. On derandomizing local distributed algorithms. In *Proc. 59th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2018)*, pages 662–673. IEEE, 2018. doi:10.1109/FOCS.2018.00069.

26 Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the complexity of local distributed graph problems. In *Proc. 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017)*, pages 784–797. ACM Press, 2017. doi:10.1145/3055399.3055471.

27 Mika Göös, Juho Hirvonen, Reut Levi, Moti Medina, and Jukka Suomela. Non-local probes do not help with many graph problems. In *Proc. 30th International Symposium on Distributed Computing (DISC 2016)*. Springer, 2016. doi:10.1007/978-3-662-53426-7_15.

28 Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for maximal independent set and other problems, 2018. arXiv:1804.01823.

29 András Gyárfás and Jenő Lehel. On-line and first fit colorings of graphs. *Journal of Graph Theory*, 12(2):217–227, 1988. doi:h10.1002/jgt.3190120212.

30 Magnús M. Halldórsson, Kazuo Iwama, Shuichi Miyazaki, and Shiro Taketomi. Online independent sets. *Theoretical Computer Science*, 289(2):953–962, 2002. doi:10.1016/S0304-3975(01)00411-X.

31 Magnús M. Halldórsson. Parallel and on-line graph coloring. *Journal of Algorithms*, 23(2):265–280, 1997. doi:10.1006/jagm.1996.0836.

32 Magnús M. Halldórsson. Online coloring known graphs. *Electronic Journal of Combinatorics*, 7, 2000. doi:10.37236/1485.

33 Magnús M. Halldórsson and Mario Szegedy. Lower bounds for on-line graph coloring. *Theoretical Computer Science*, 130(1):163–174, 1994. doi:10.1016/0304-3975(94)90157-0.

34 Zoran Ivković and Errol L. Lloyd. Fully dynamic maintenance of vertex cover. In *Proc. 19th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 1993)*. Springer, 1994.

35 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC 1990)*, pages 352–358, 1990. doi:10.1145/100216.100262.

36 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *Journal of the ACM*, 63(2):1–44, 2016. doi:10.1145/2742012.

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

38 László Lovász, Michael Saks, and W.T. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics*, 75(1):319–325, 1989. doi:10.1016/0012-365X(89)90096-4.

39 Yishay Mansour, Aviad Rubinstein, Shai Vardi, and Ning Xie. Converting online algorithms to local computation algorithms. In *Proc. 39th International Colloquium on Automata, Languages and Programming (ICALP 2012)*, pages 653–664. Springer, 2012. doi:10.1007/978-3-642-31594-7_55.

40 Yishay Mansour and Shai Vardi. A local computation approximation scheme to maximum matching. In *Proc. 16th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2013) and 17th International Workshop on Randomization and Computation (RANDOM 2013)*, pages 260–273. Springer, 2013. doi:10.1007/978-3-642-40328-6_19.

41 Darya Melnyk, Jukka Suomela, and Neven Villani. Mending partial solutions with few changes. In *Proc. 25th International Conference on Principles of Distributed Systems (OPODIS 2022)*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

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

43 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Transactions on Algorithms*, 12(1), 2015. doi:10.1145/2700206.

44 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000. doi:10.1137/1.9780898719772.

45 Will Rosenbaum and Jukka Suomela. Seeing far vs. seeing wide: volume complexity of local graph problems. In *Proc. 39th ACM Symposium on Principles of Distributed Computing (PODC 2020)*, pages 89–98. ACM Press, 2020. doi:10.1145/3382734.3405721.

46 Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Proc. 2nd Symposium on Innovations in Computer Science (ICS 2011)*, pages 223–238, 2011.

47 Sundar Vishwanathan. Randomized online graph coloring. *Journal of Algorithms*, 13(4):657–669, 1992. doi:10.1016/0196-6774(92)90061-G.