Shared Randomness Helps with Local Distributed Problems

Alkida Balliu · Gran Sasso Science Institute
Mohsen Ghaffari · MIT
Fabian Kuhn · University of Freiburg
Augusto Modanese · Aalto University
Dennis Olivetti · Gran Sasso Science Institute
Mikaël Rabie · Université Paris Cité, CNRS, IRIF
Jukka Suomela · Aalto University
Jara Uitto · Aalto University

Abstract. By prior work, we have many wonderful results related to distributed graph algorithms for problems that can be defined with local constraints; the formal framework used in prior work is *locally checkable labeling problems* (LCLs), introduced by Naor and Stockmeyer in the 1990s. It is known, for example, that if we have a deterministic algorithm that solves an LCL in $o(\log n)$ rounds, we can speed it up to $O(\log^* n)$ rounds, and if we have a randomized algorithm that solves an LCL in $O(\log^* n)$ rounds, we can derandomize it for free.

It is also known that randomness helps with some LCL problems: there are LCL problems with randomized complexity $\Theta(\log \log n)$ and deterministic complexity $\Theta(\log n)$. However, so far there have not been any LCL problems in which the use of *shared randomness* has been necessary; in all prior algorithms it has been enough that the nodes have access to their own private sources of randomness.

Could it be the case that shared randomness never helps with LCLs? Could we have a general technique that takes any distributed graph algorithm for any LCL that uses shared randomness, and turns it into an equally fast algorithm where private randomness is enough?

In this work we show that the answer is *no.* We present an LCL problem Π such that the round complexity of Π is $\Omega(\sqrt{n})$ in the usual randomized LOCAL model (with private randomness), but if the nodes have access to a source of shared randomness, then the complexity drops to $O(\log n)$.

As corollaries, we also resolve several other open questions related to the landscape of distributed computing in the context of LCL problems. In particular, problem Π demonstrates that distributed *quantum* algorithms for LCL problems strictly benefit from a shared quantum state. Problem Π also gives a separation between *finitely dependent distributions* and *non-signaling distributions.*

1 Introduction

In this work we present a graph problem that is solely defined with local constraints, yet distributed algorithms for solving it benefit from shared randomness. More formally, we present a locally checkable labeling problem (LCL) Π such that any randomized distributed algorithm that solves Π in the usual LOCAL model of distributed computing requires $\Omega(\sqrt{n})$ communication rounds, but if we have access to shared randomness, then we can exponentially improve the round complexity, down to $O(\log n)$ rounds.

Context: LCL problems and the LOCAL model. LCL problems were originally introduced by Naor and Stockmeyer [35] in the 1990s, and in the recent years they have formed one of the cornerstones of the modern theory of distributed graph algorithms. An LCL problem is simply a graph problem that can be specified by giving a finite set of valid labeled neighborhoods. For example, the task of coloring vertices with 10 colors in graphs of maximum degree at most 20 is an LCL problem. (It can be defined by listing all possible radius-1 neighborhoods of degree at most 20, and by listing for each of them all valid 10-colorings.) Numerous problems that have been studied in distributed graph algorithms over the decades are LCL problems (at least when restricted to bounded-degree graphs); examples include maximal independent sets, maximal matching, various problems related to vertex and edge coloring, and various tasks related to orienting edges or partitioning of edges subject to local constraints. In fact even 3SAT can be interpreted as an LCL problem (with the bounded-degree assumption corresponding to the case in which each variable occurs in a bounded number of clauses).

While LCL problems are meaningful in any model of computing, they have been studied in particular from the perspective of distributed graph algorithms, and the most prominent model there is the LOCAL model of computing [33, 36]. In brief, an algorithm A with running time T in the LOCAL model is simply a function that maps radius-T neighborhoods to local outputs. That is, to apply A in a given graph G, each node v looks at all information in its radius-T neighborhood and uses A to determine its own local output (for example, the color of v if the task is to find a graph coloring). It turns out that we could also equivalently interpret G as a computer network, and then A can be interpreted as a distributed message-passing algorithm in which all nodes stop after T synchronous communication rounds. We will hence interchangeably refer to T as the running time, locality, or round complexity of A.

A comprehensive theory of LCL problems in the LOCAL model has been developed in the past 10 years. There are numerous theorems that apply to all LCL problems, or all LCL problems in some graph family, such as trees or grids [5–7, 9–11, 13–18, 21, 23, 24, 26, 38]. To give some flavor of the power of these results, here is one example: if there is a *randomized* LOCAL algorithm A that solves some LCL problem Π in $T(n) = o(\log \log n)$ rounds in n-node graphs, we can also construct a *deterministic* LOCAL algorithm A' that solves the same problem Π in $T'(n) = O(\log^* n)$ rounds [16]. That is, we can for free derandomize algorithms and speed them up.

In general, the relation between randomized and deterministic algorithms in the context of LCL problems is now well understood, and we also have a clear view of the landscape of all possible round complexities that we may have for LCL problems [39]. However, more care is needed here: what exactly do we mean by *randomized* LOCAL algorithms?

Question: shared vs. private randomness. Essentially all work on LCLs in the randomized LOCAL model assumes that each node has its own *private* source of randomness. More precisely, nodes are initially labeled independently and uniformly at random with strings of bits, and then a T-round algorithm can make use of all such bit strings within radius T.

However, there is another notion of randomized algorithms that has been studied for instance in the context of communication complexity: *shared* randomness (e.g., [1, 31]). That is, there is one global random bit string that all nodes can see. There are many contexts in which access to shared randomness helps [20, 34, 37], but does it help with any LCL problem?

Prior to this work, there was no evidence that shared randomness might help with LCL problems. On the contrary, all numerous LCL problems that we have encountered in prior work seem to be such that either (1) randomness does not help at all, or (2) randomness helps but private randomness is sufficient. There have even been systematic studies of infinite families of LCL problems [8, 18], as well as computer-assisted explorations of the space of LCL problems [40], yet there is no known candidate problem that might benefit from shared randomness. Intuitively, the key obstacle seems to be the combination that LCL problems are defined using local constraints and the set of input and output labels is finite. Shared randomness could be used to e.g. select a globally consistent random label from the set of finite output labels, but if that succeeds w.h.p. in arbitrarily large graphs, there also has to exist a deterministic choice that succeeds.

Hence, for all that we know, we might very well be living in a world in which the following conjecture is true: if an LCL problem Π can be solved in $T(n)$ rounds with the help of shared randomness, it can also be solved in $O(T(n))$ rounds with only private randomness.

Were this to be true, it would considerably simplify the landscape of models, as discussed further below. It would also give a helpful algorithm design tool: we could design algorithms that exploit shared randomness, and then for free turn them into genuine distributed algorithms that only use private randomness. Conversely, it would allow us to strengthen all existing lower bounds that hold for private randomness into lower bounds that extend all the way to shared randomness.

What we show in this work is that this result cannot be true. Indeed, conversion from shared to private randomness for some LCL problems may lead to an *exponential* increase in the round complexity.

Main contribution. In this work we present an LCL problem Π such that the round complexity of Π is $\Omega(\sqrt{n})$ in the usual randomized LOCAL model (with private randomness), but if the nodes have access to a source of shared randomness, then the complexity drops to $O(\log n)$. This is the first known LCL that separates these two models.

Our problem Π is an LCL exactly in the strict sense originally defined by Naor and Stockmeyer [35], and we do not exploit any promise on the graph family or input. Being promise-free is important, as the entire theory of LCL problems is fundamentally promise-free (for example, the known gap results would disappear if we can have arbitrary promises on the input structure), and hence also any interesting separations or counterexamples have to be promise-free.

We refer to Theorems 8.1 and 8.2 for the formal theorem statements of our lower bound and upper bound.

Corollary 1: distributed quantum computing. One of the major open questions at the intersection of distributed computing and quantum information theory is which distributed problems admit quantum advantage. A key model for studying this question is the quantum-LOCAL model, which is essentially what one gets if we imagine that nodes of the input graph are quantum computers and communication channels can be used to exchange qubits. It is known that there are some (artificial) graph problems that can be solved in constant time in quantum-LOCAL yet for which classical LOCAL requires linear time [32]. Nevertheless, it is still wide open whether there is any *LCL* problem that admits distributed quantum advantage; see, for instance, [2, 19].

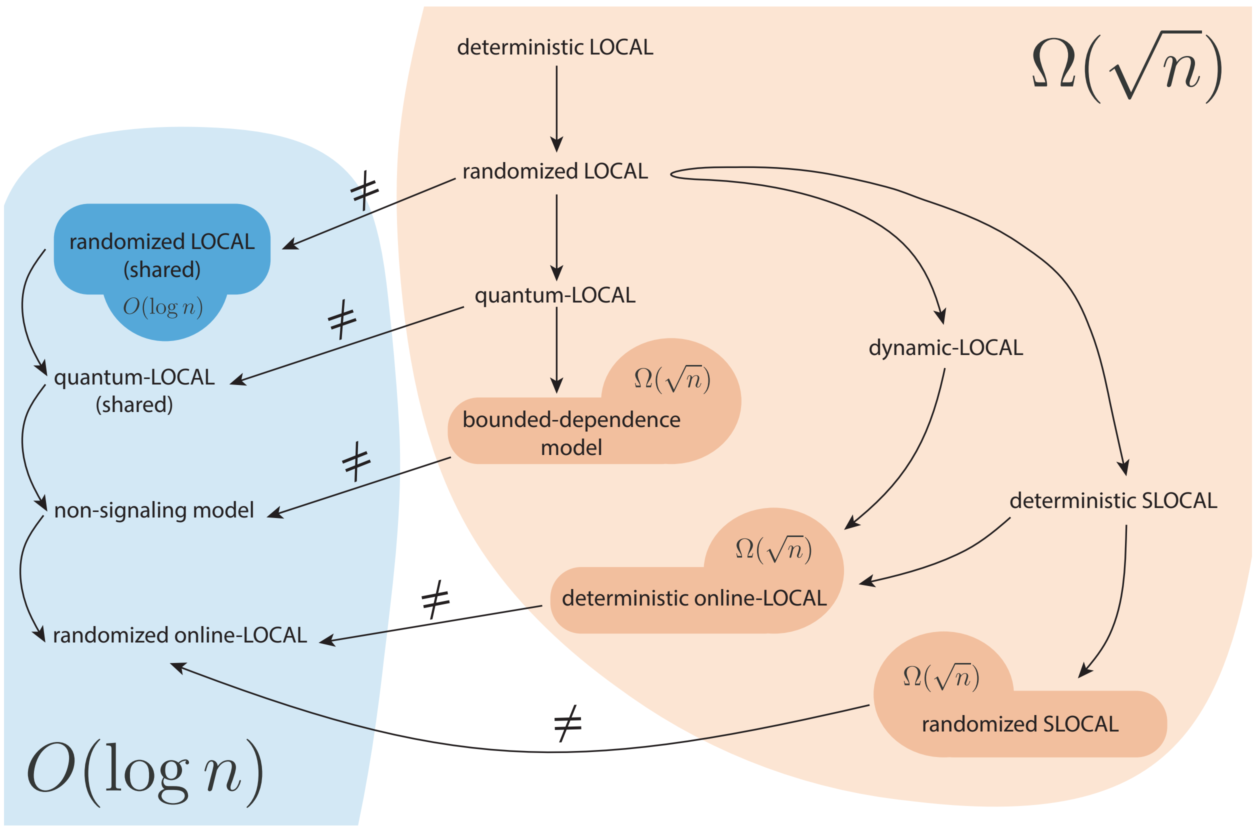

Figure 1: Landscape of models and the new separations between them. The general structure of the landscape is from [2]. In this work we present a new LCL problem Π that is easy in the blue-shaded region (models in which we have access to shared randomness), but hard in the red-shaded region (all other models), and we will get separations for all pairs of models that cross the cut. To prove these results, we give an upper bound in randomized LOCAL with shared randomness (Theorem 8.2), and all upper bounds in the blue region follow, and we give lower bounds in randomized SLOCAL (Theorem 9.1), deterministic online-LOCAL (Theorem 9.2), and the bounded-dependence model (Theorem 9.3), and all lower bounds in the red region follow.

So far, there have been two major variants of quantum-LOCAL that have been studied in the literature: quantum-LOCAL with a shared quantum state (i.e., nodes are configured in advance and share entangled qubits) and quantum-LOCAL without any shared quantum state [2, 4, 22]. It was not known whether either of these is (strictly) stronger than randomized LOCAL for any LCL problem. There are no known examples of LCL problems that would potentially benefit from the shared quantum state, and it seemed reasonable to conjecture that, even if quantum-LOCAL turns out to be is stronger than randomized LOCAL, the shared quantum state does not give it any additional power. Indeed, there were (unsuccessful) attempts at unifying the two variants of quantum-LOCAL.

One unexpected corollary of our work is that the two variants of quantum-LOCAL are indeed distinct, though for mundane reasons that have little to do with quantum physics. It simply happens to be the case that a shared quantum state gives the nodes access to shared randomness. As we show, the problem that we construct in this work is hard in quantum-LOCAL without shared quantum state, but it becomes easy in quantum-LOCAL with shared quantum state (since it is easy already in randomized LOCAL with shared randomness).

Hence, the entire question was wrong: shared quantum state does help, but for the wrong reasons. The present work highlights that the right question is whether shared quantum state provided further advantage beyond shared randomness.

Corollary 2: finitely dependent vs. non-signaling distributions. There is a line of research in mathematics that aims at capturing which problems admit *finitely-dependent distributions* [27–29, 41]; these are distributions over nodes such that their restriction to a set X of nodes is independent of their restriction to another set Y of nodes if the shortest-path distance between X and Y is greater than some constant. For example, the output distribution of any constant-time randomized LOCAL algorithm is a finitely-dependent distribution. A key question in this context has been whether finitely-dependent distributions are strictly stronger than constant-time randomized LOCAL algorithms, which indeed is the case [29]. A natural generalization of finitely-dependent distributions to arbitrary (not necessarily constant) distance is called a *bounded-dependence distribution* [2].

Another closely related definition arises from quantum information theory and the study of distributed quantum advantage: *non-signaling distributions* [2, 4, 19, 22]. Informally, a family of output distributions is non-signaling with locality T if the distribution restricted to some set of nodes X does not change if we modify the input graph more than T hops away from X. A bounded-dependence distribution with locality T is non-signaling with locality $O(T)$, but the converse is not necessarily true.

Prior to this work, there were no known examples of LCL problems that admit a non-signaling distribution with locality T but do not admit a bounded-dependence distribution with locality $O(T)$. Indeed, it was again reasonable to conjecture that no such problem exists. Our construction gives a separation also between these two models.

Other corollaries. Our construction also gives an exponential separation between deterministic and randomized versions of the online-LOCAL model (see [2, 3]). Previously, there were no known examples of LCLs that separate these models. For our problem we can prove a lower bound in the deterministic online-LOCAL model, and the upper bound for randomized LOCAL with shared randomness directly works also in randomized online-LOCAL.

The big picture. All of our results are summarized in Fig. 1. All separations between the two regions are new, in the sense that there was previously no (promise-free) LCL that would separate these pairs of models. The results that lead to this landscape are:

- Theorem 8.2: an upper bound in randomized LOCAL with shared randomness,
- Theorem 9.1: a lower bound in randomized SLOCAL,
- Theorem 9.2: a lower bound in deterministic online-LOCAL,
- Theorem 9.3: a lower bound in bounded-dependence model.

However, to keep this work easy to follow also for those who are not interested in models beyond randomized LOCAL, we also prove the following result that is technically redundant, but serves as a warm-up for the other results:

- Theorem 8.1: a lower bound in randomized LOCAL without shared randomness.

Open questions. We conjecture that our problem Π also exhibits a doubly-exponential separation between the randomized LOCAL model and the *massively parallel computing* (MPC) model [30]. More precisely, we conjecture that our problem Π can be solved in $O(\log \log n)$ rounds in the MPC model, while it is known to require $\Omega(\sqrt{n})$ rounds in randomized LOCAL. Proving this is deferred for future work.

Our problem Π fundamentally exploits the existence of short cycles (in the sense that the problem is trivial in trees and interesting only in graphs with short cycles). A key open question is *whether shared randomness helps with any LCL in trees.* We have preliminary evidence suggesting that shared randomness never helps in rooted regular trees, but the case of general trees remains open.

2 Definitions

Labeled graphs. We start by defining the notion of labeled graph.

Definition 2.1 (Labeled graph)**.** Let $\mathcal{V}$ and $\mathcal{E}$ be sets of labels. A graph $G = (V, E)$ is called $(\mathcal{V}, \mathcal{E})$-*labeled* if:

- Each node $u \in V$ is assigned a label from $\mathcal{V}$;
- Each node-edge pair $(u, e) \in V \times E$, satisfying $u \in e$, is assigned a label from $\mathcal{E}$.

A node-edge pair (u, e) that satisfies $u \in e$ is also called *half-edge* incident to u.

Definition 2.2 (Labeled graph satisfying some constraints)**.** Let G be a graph, and let $\mathcal{C}$ be a set of constraints over the labels $\mathcal{V}$ and $\mathcal{E}$. The graph G satisfies $\mathcal{C}$ if and only if:

- G is $(\mathcal{V}, \mathcal{E})$-labeled, and
- the constraints of $\mathcal{C}$ are satisfied over all nodes of G.

Definition 2.3 (Locally checkable labeleling (LCL) problem)**.** A *locally checkable labeling* (LCL) problem Π is defined by a tuple $(\mathcal{V}_{\text{input}}, \mathcal{E}_{\text{input}}, \mathcal{V}_{\text{output}}, \mathcal{E}_{\text{output}}, \mathcal{C})$ where $\mathcal{C}$ is a set of constraints over $\mathcal{V}_{\text{output}}$ and $\mathcal{E}_{\text{output}}$. Given a $(\mathcal{V}_{\text{input}}, \mathcal{E}_{\text{input}})$-labeled graph G, one is asked to label G so that it satisfies $\mathcal{C}$.

We denote with $L_u(e)$ the label on the half-edge (u, e). Something that will be very useful throughout the paper is to define a way to denote the node that we can reach from a node u by following some specific chain of labels assigned to half-edges. Let $G = (V, E)$ be (Σ_V, Σ_E)-labeled. Let $L_1, L_2, \ldots, L_k$ be labels in Σ_E. We define a function $f(u, L_1, L_2, \ldots, L_k)$ that takes as input a node $u \in V$ and labels $L_1, \ldots, L_k$ in Σ_E, and returns the node v reachable from u by following the unique path whose edges are labeled with $L_1, \ldots, L_k$ (in this order); if there is no such path or it is not unique, then the value of f is undefined. More precisely, let $P = (v_1, v_2, \ldots, v_{k+1})$ be a path that starts at $v_1 = u$ and such that, for any edge $e = \{v_i, v_{i+1}\}$, the half-edge (v_i, e) is labeled with $L_{v_i}(e) = L_i$. Then

$$f(u, L_1, L_2, \ldots, L_k) = \begin{cases} v_{k+1}, & \text{if } P \text{ exists and is unique} \\ \perp, & \text{otherwise.} \end{cases}$$

The LOCAL model. In the LOCAL model of computing, we imagine that nodes of a graph $G = (V, E)$ are computers with access to unbounded computational resources (time and space). Each computer is assigned a unique identifier from the set $\{1, \ldots, |V|^c\}$, where $c \geq 1$ is some constant. The communication between computers is as defined by E. The computation proceeds in rounds where, in each round, each computer exchanges messages of unbounded size with their neighbors and performs some local computation (which we may perceive as instantaneous).

The above gives the deterministic variant of LOCAL. The randomized variant (with private randomness) is the same but where we also give each node access to an infinite string of random bits (that is guaranteed to be independent of the strings of other nodes in the network). In the variant with shared randomness, nodes are given simultaneous access to the *same* infinite string.

3 High-level ideas

The main ingredient of our work is an LCL problem Π with some desirable properties. On a high-level, this problem is promise-free, in the sense that it is defined on any graph. However, it is defined in such a way that there exists a family of graphs $\mathcal{G}$ that we call *hard instances*:

- If the graph G in which the algorithm solving Π is run is not in $\mathcal{G}$, the LCL is defined in such a way that the nodes of G can produce a locally checkable proof of this fact. Moreover, such a proof can be computed "fast".
- However, if $G \in \mathcal{G}$, the LCL forces the nodes to solve a problem that can be solved "fast" if they have access to shared random bits, whereas any algorithm working without shared randomness must be "slow".

Here "fast" and "slow" depend on the precise model considered. For example, in the case of the LOCAL model, the problem requires just $O(\log n)$ rounds with shared randomness, but $\Omega(\sqrt{n})$ rounds without it.

In the following, we start by providing an overview of the structure of hard instances. Then we explain what the problem Π is on a hard instance, and later we will explain how the problem is defined to be promise-free.

The hard instances. Consider the graph depicted in Figure 2. It is composed of a square grid, where on top of each column we place a tree-like structure. This kind of graph has a couple of useful properties:

- For any two nodes in the grid belonging to the same column, it holds that their distance is $O(\log n)$.
- As we will see, this structure can be certified. That is, it is possible to provide a constant-sized certificate to the nodes such that, if the certificate looks good everywhere, then the graph is indeed a hard instance. Conversely, if the graph is not a hard instance, then any assignment of the certificate leads to an error somewhere. We will make use of this fact later when making the problem promise-free.

The LCL problem Π^{hard} defined on hard instances. Consider the following problem Π^{hard}, defined on hard instances. Each node in the right-most column of the grid receives a bit as input. Then, all nodes of the grid must output a bit such that the two following constraints are satisfied:

C1: For each row of the grid, all nodes must output the same bit.

C2: There must exist at least one row such that the output bit is the same as the input bit of the right-most node of that row.

It is possible to present this problem as an LCL as follows:

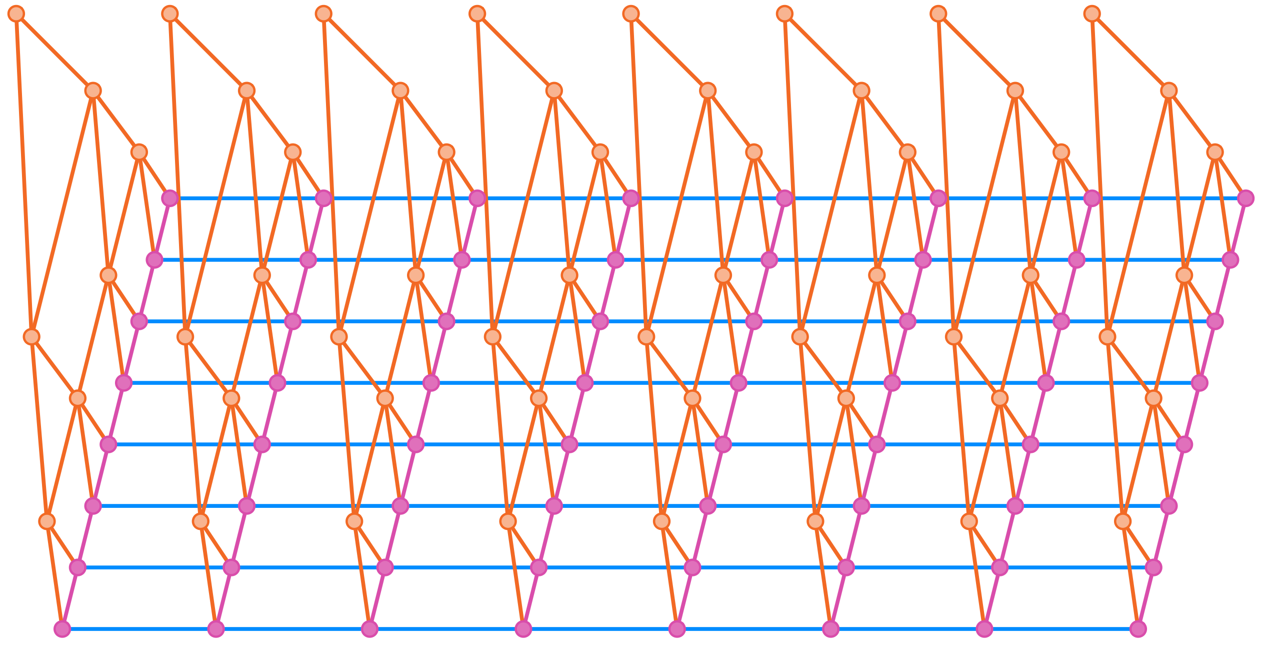

Figure 2: An example of a hard instance. The grid is composed of blue edges, purple edges, and purple nodes. Each connected component induced by orange nodes, orange edges, purple edges, and purple nodes connected to the orange edges is a tree-like structure.

- Assume that the grid has an input labeling encoding its orientation; that is, each node knows which of its neighbors is on its left, right, above, and below. Then constraint C1 can be encoded in the LCL by requiring every node to have the same output as its left and right neighbors.
- In order to enforce constraint C2, we use the tree-like structure on top of the last column. Say a grid node of the last column is *happy* if its output agrees with its input. Meanwhile an inner node of the tree is happy if at least one of their children is happy. All nodes of the tree output whether they are happy, and we require the root to be happy. These constraints are local, and in fact they can be encoded as an LCL.

An example of a valid output is shown in Figure 3.

Now, given such a problem, we can obtain a separation between shared and private randomness in the LOCAL model:

- If the nodes have access to shared randomness, then each node only needs $O(\log n)$ rounds to determine its vertical position y in the grid. Once it has figured out the value of y, the node simply outputs the yth shared random bit. In this way, all nodes in the same row output the same bit, and for each row with probability $1/2$ we have that this bit agrees with the input given to the node on the right-most column. Since there are $\Theta(\sqrt{n})$ rows, we get that there is at least one good row with high probability.
- Conversely, if only private randomness is allowed, nodes in the same row do not have any way to coordinate their output and communication across the whole row is expensive ($\Omega(\sqrt{n})$ rounds). The only alternative is for nodes in the same row to deterministically fix their output as a function of their vertical position. In this case we can adversarially pick the input to the rightmost node so that the row does not succeed. Since this cannot hold for every row (as otherwise the algorithm would not be solving Π), we get the lower bound of $\Omega(\sqrt{n})$ rounds.

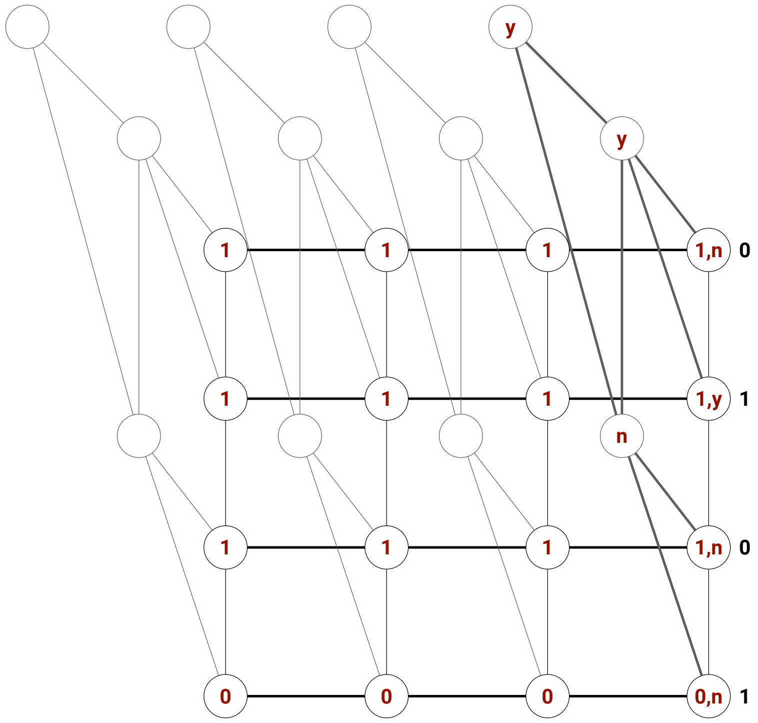

Figure 3: An example of a solution for Π^{hard}. Black bits represent the inputs of the nodes of the last column. The inputs of the other nodes do not affect the solution and are omitted. Labels in red represent the outputs, where the label y represents a happy node, and the label n represents an unhappy node. All nodes that are not labeled either y or n output y, which is omitted in the figure.

Although the idea of the problem is simple, the main challenge is making the problem promise-free. That is, we also have to account for all the cases where the input graph is not as in Figure 2. Next we provide an overview of this process.

3.1 A tree-like structure

A useful property that our problem satisfies is the following: nodes that belong to the same column of the grid should either be able to see the whole column by inspecting their $O(\log n)$-radius neighborhood, or the nodes can *prove that there is some error* within distance $O(\log n)$. In Section 4, we describe *tree-like structures* and how they give us exactly this property. This kind of structure has already been used in [12]; in fact some useful properties that we exploit here have already been proved in [12].

Definition of the tree-like structure. Informally, a tree-like structure is a perfect binary tree in which nodes at the same depth are also connected via a path. Such a structure can be certified by assigning a label to each node-edge pair. An example of this structure as well as an assignment of a certificate for it are depicted in Figure 4. For example, the certificate ensures that all leaves are at the same depth by requiring that, starting from a node not having any incident edge labeled Ch_L or Ch_R (i.e., it does not have any children), and following the edge labeled R, we must reach a node that also does not have any children.

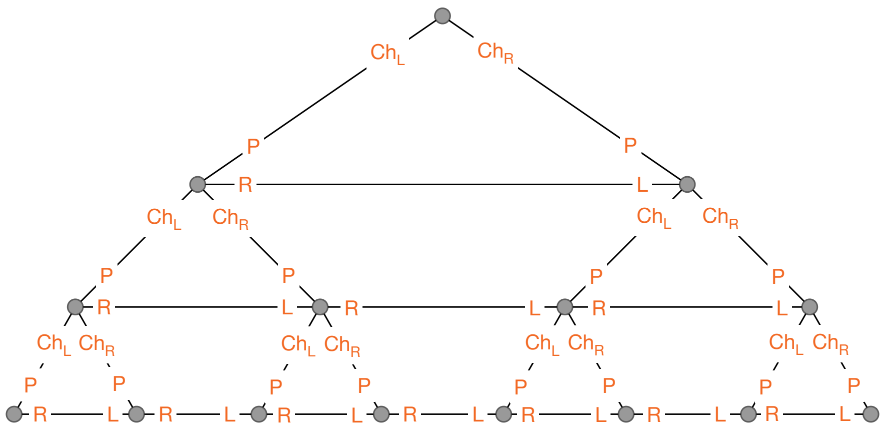

Figure 4: An example of a properly labeled tree-like structure. The labels L, R, P, Ch_L, and Ch_R, stand, respectively, for left, right, parent, left child, and right child.

Local certification of tree-like structures. In Section 4, we start by describing tree-like structures from a local perspective. In particular, we define a set of input half-edge labels $\mathcal{E}^{\mathsf{tree}}$ of constant size and a set of constraints $\mathcal{C}^{\mathsf{tree}}$ over constant distance that satisfy the following:

- For all tree-like structures G, there exists an assignment of labels of $\mathcal{E}^{\mathsf{tree}}$ to the half-edges of the graph such that the constraints of $\mathcal{C}^{\mathsf{tree}}$ are satisfied on all nodes of G.
- Let G be a graph where half-edges are labeled with labels from $\mathcal{E}^{\mathsf{tree}}$ and such that the constraints of $\mathcal{C}^{\mathsf{tree}}$ are satisfied on all nodes of G. Then, G is a tree-like structure.

In other words, we show that there exists a locally checkable proof of constant size for the fact that the graph is tree-like.

The tree-like structure as an LCL. Building on the previously defined locally checkable proof, we define an LCL problem Π^{badTree} that satisfies the following:

- For all tree-like graphs G, there exists an input for Π^{badTree} that can be assigned to G such that the only valid solution for Π^{badTree} is the one assigning $\perp$ to all nodes of G.
- Let G be a graph where half-edges are labeled with labels from $\mathcal{E}^{\mathsf{tree}}$ such that the constraints of $\mathcal{C}^{\mathsf{tree}}$ are not satisfied on at least one node of G. Then there exists a solution for Π^{badTree} where all nodes produce an output different from $\perp$. Moreover, such a solution can be computed in $O(\log n)$ rounds in the LOCAL model.

In other words, we define an LCL problem where the inputs are from $\mathcal{E}^{\mathsf{tree}}$ and where the nodes have two options: they can either prove that the graph is not tree-like, or they can do nothing (by outputting $\perp$). This problem is defined in such a way that an output different from $\perp$ can be used only on structures that are not tree-like (or on tree-like structures whose input labels are incorrect) whereas, if an output different from $\perp$ can be used, then this can be done relatively fast ($O(\log n)$ rounds in the LOCAL model).

How we will use Π^{badTree}. We now describe how the problem Π^{badTree} is used when defining our main LCL problem Π. The problem Π is defined in such a way that all nodes receive an input indicating whether they are part of a grid, or whether they are part of a tree-like structure. Then Π is defined so that:

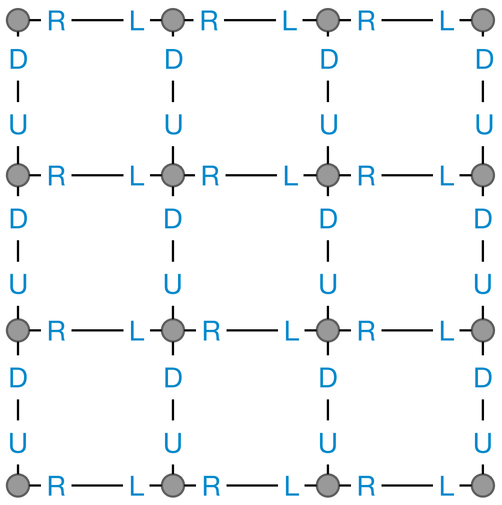

Figure 5: An example of a properly labeled grid structure. The labels L, R, U, and D stand, respectively, for left, right, up, and down.

- Nodes can *mark* areas of the graph that do not look like valid hard instances. In particular, nodes having an input indicating that they are part of a tree-like structure are considered marked if they solve Π^{badTree} by giving an output different from $\perp$.
- We prove that nodes can efficiently mark bad parts of the graph such that the remaining connected components *almost* look like hard instances. Although these remaining parts do not exactly look like hard instances (we provide more details about this later), there is an efficient algorithm for solving them.

3.2 A grid structure

As already mentioned, our hard instances are grids in which we connect a tree-like structure on top of each column. In Section 5, we describe a structure that we call *grid structure*. A similar structure has already been used in [12] and in fact some useful properties follow directly from what was proved in that paper.

Definition of the grid structure. Informally, a grid structure denotes a two-dimensional grid that does not wrap around (i.e., it is not a torus). Unlike the case of tree-like structures, we cannot achieve full certification of grids; in particular our scheme for certifying grids is defective and also allows one to label invalid grid structures (and in particular tori) in a way that no node sees any error.

Figure 5 illustrates a grid structure and its corresponding certificate. An example of a constraint that needs to be checked is that, if a node has R (i.e., “right”) on an incident half-edge, then the corresponding half-edge must be labeled L (i.e., “left”).

Local certification of grid structures. In Section 5, we describe grid structures from a local perspective. In particular, we define a set of input half-edge labels $\mathcal{E}^{\mathsf{grid}}$ of constant size as well as a set of constraints $\mathcal{C}^{\mathsf{grid}}$ over constant distance that satisfy the following:

- For any grid structure G, there exists an assignment of labels of $\mathcal{E}^{\mathsf{grid}}$ to the half-edges of the graph such that the constraints of $\mathcal{C}^{\mathsf{grid}}$ are satisfied on all nodes of G.

- Let G be a graph where half-edges are labeled with labels from $\mathcal{E}^{\mathsf{grid}}$ and such that the constraints of $\mathcal{C}^{\mathsf{grid}}$ are satisfied on all nodes of G. Moreover, suppose that there exists at least one node that has no incident half-edge labeled D (or U) and that there exists at least one node that has no incident half-edge labeled L (or R). Then G is a grid structure.

In other words, if we assume that all nodes satisfy the constraints of $\mathcal{C}^{\mathsf{grid}}$ and we additionally assume that there is at least one node satisfying some additional constraints (which essentially guarantee that there is some "corner" of the grid, thus preventing the graph from being a torus), then the graph is indeed a grid.

Enforcing the dimensions of the grid. As previously discussed, when defining our main problem Π, we allow nodes to *mark* areas of the graph that do not look like valid hard instances. Recall we cannot guarantee that unmarked areas are exactly hard instances. Nevertheless, we are able to ensure that the unmarked parts of the graphs are grids (with properly attached tree-like structures) that are at least as tall as they are large. We call such grids *vertical.* We later discuss how this property is sufficient to obtain an efficient algorithm that has access to shared randomness.

On a high-level, in order to enforce a grid to be vertical, we define a set of input node labels $\mathcal{V}^{\mathsf{vGrid}}$, and a set of constraints $\mathcal{C}^{\mathsf{vGrid}}$, that satisfy the following.

- For any vertical grid structure G, there exists an assignment of labels of $\mathcal{E}^{\mathsf{grid}}$ to the half-edges of G and an assignment of labels of $\mathcal{V}^{\mathsf{vGrid}}$ to the nodes of G, such that the constraints of $\mathcal{C}^{\mathsf{grid}}$ and $\mathcal{C}^{\mathsf{vGrid}}$ are satisfied on all nodes of G.

- Suppose G is a graph where half-edges are labeled with labels from $\mathcal{E}^{\mathsf{grid}}$, nodes are labeled with labels from $\mathcal{V}^{\mathsf{vGrid}}$, such that the constraints of $\mathcal{C}^{\mathsf{grid}}$ and $\mathcal{C}^{\mathsf{vGrid}}$ are satisfied on all nodes of G. Then, G is a vertical grid structure.

We note that, while the second point provides the intuition of what we will do, the statement, as is, is false. We will provide more details in Section 5.

How we will use vertical grids. Consider a hard instance in which the grid is more large than tall, and in particular consider the extreme case in which the grid is just a path of linear length. In this case, in the case of shared randomness, if we apply the LOCAL algorithm for solving Π^{hard} described at the beginning of the section, we would have a success probability of just $1/2$, since there is only a single row in the grid. In order to guarantee a large-enough success probability, the problem Π will be defined such that unmarked regions are *vertical* grid structures (possibly of small size). This will guarantee the following.

- If the grid has height less than $\log n$, then its width is also less than $\log n$, which implies that nodes can see the whole grid in just $O(\log n)$ rounds, and solve the problem Π^{hard} by brute force.

- If the grid has height strictly larger than $\log n$, then the success probability will be at least $1 - 1/2^{\log n} = 1 - 1/n$, and hence the algorithm will succeed with high probability.

By combining the above, we will get that $O(\log n)$ rounds will be an upper bound on the runtime for succeeding in solving Π^{hard} with high probability when using shared randomness.

3.3 Family of hard instances

In Section 6, we formally define the family $\mathcal{G}$ of hard instances. These graphs are similar to the ones informally explained at the beginning of this section (see Figure 2 for an example), with the only difference that grids do not need to be squares, but they only need to satisfy that their height is at least as large as their width. Then, we define an LCL Π^{badGraph} satisfying the following.

- There are two possible types of output, and different nodes could give outputs of different type.
- One possible output is the empty output, and if a graph G is in $\mathcal{G}$, the problem Π^{badGraph} is defined such that all nodes must produce the empty output.
- The other possible output is a proof for the fact that $G \notin \mathcal{G}$. More in detail, if the graph is not in the family, nodes can spend $O(\log n)$ time in the LOCAL model to produce a proof of this fact, such that, the subgraph G' induced by nodes producing an empty output satisfies that each connected component of G' is a graph in $\mathcal{G}$.

Informally, our main problem Π will be defined such that all nodes need to solve Π^{badGraph}, and then, on the subgraph induced by nodes producing an empty output for Π^{badGraph}, nodes need to solve the problem Π^{hard} that we informally defined on hard instances at the beginning of this section. The definition of Π will satisfy that, if we consider some graph $G \in \mathcal{G}$, the only possible way to solve Π is by producing an empty solution for Π^{badGraph}, implying that nodes must then solve Π^{hard} on the whole graph. On the other hand, if a graph G is not in $\mathcal{G}$, nodes can quickly (i.e., in $O(\log n)$ rounds in the LOCAL model) mark bad parts of the graph, and then solve Π^{hard} on the connected components that are part of $\mathcal{G}$. In other words, the problem Π^{badGraph} is the one allowing us to remove the promise in the definition of Π^{hard}.

High-level ideas behind the definition of Π^{badGraph}. Observe that, by how hard-instances are constructed, nodes can either be exclusively part of a tree-like structure, or they can belong at the same time to the grid and to some tree-like structure. However, edges can be of three types: they can be exclusively part of the grid, exclusively part of a tree-like structure, or be part of both. In the problem Π^{badGraph}, nodes and edges are input labeled to indicate to which structure(s) they belong to. Then, the possible outputs for Π^{badGraph} are the following.

- If on a node the constraints of $\mathcal{C}^{\mathsf{vGrid}}$ or the constraints of $\mathcal{C}^{\mathsf{tree}}$ are not locally satisfied, or the input labeling of Π^{badGraph} is such that there is some local error in how the two structures are connected, then the node can output an error.
- Consider the subgraph obtained by excluding edges labeled as horizontal edges (i.e., L and R) of the grid. Each connected component is either a valid grid column with a properly-attached tree-like structure on top, or not. In the latter case, we also include the scenario in which some node in the connected component gave error in the previous step. In each connected component, nodes need to solve Π^{badTree}, and thus we get the following two cases:
  - The connected component looks good (i.e., it contains no nodes that output error), and the only solution for Π^{badTree} is the one giving $\perp$ (i.e., an empty output) on all nodes;
  - The connected component does not look good, and nodes can (efficiently) solve Π^{badTree} such that no node uses the output $\perp$.

An example of an input labeling for Π^{badGraph} that forces all nodes to output $\perp$ is depicted in Figure 6.

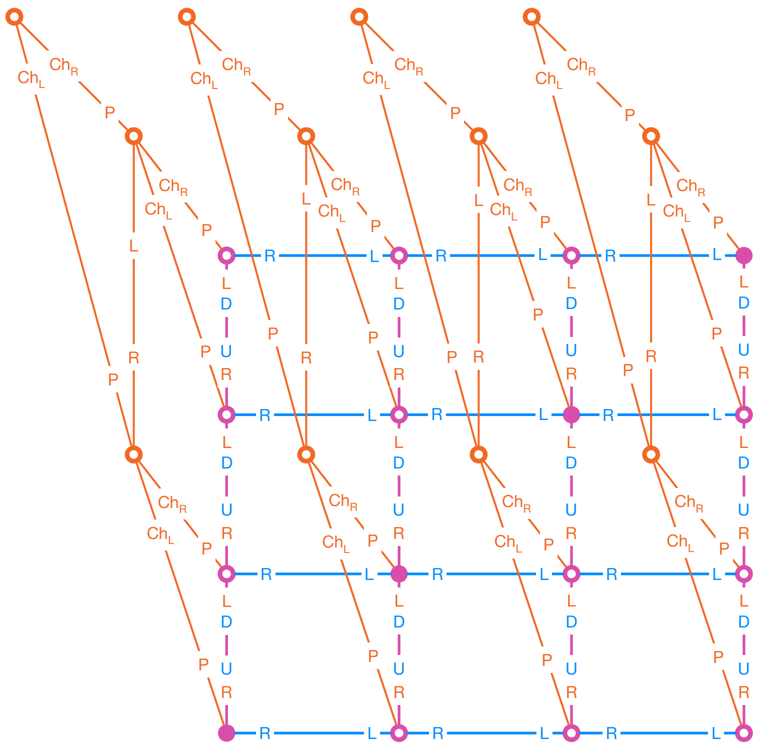

Figure 6: An example of a properly input-labeled hard instance. Orange nodes are labeled (treeNode), purple empty nodes are labeled (treeNode, gridNode, 0), purple full nodes are labeled (treeNode, gridNode, 1), orange half-edges with label ℓ are labeled (treeEdge, ℓ), blue half-edges with label ℓ are labeled (gridEdge, ℓ), purple half-edges labeled ℓ in blue and ℓ' in orange are labeled ((gridEdge, ℓ), (treeEdge, ℓ'))

.

Properties of Π^{badGraph}. In Section 6, we will prove that the following properties are satisfied by our construction:

- For any $G \in \mathcal{G}$, i.e., for any hard instance, it is possible to assign an input labeling on G, such that the only valid output for Π^{badGraph} is the one where each node outputs $\perp$ (i.e., an empty output).
- For any graph G, there exists a solution for Π^{badGraph}, which can be computed in $O(\log n)$ rounds in the LOCAL model, such that the subgraph induced by nodes that output $\perp$ satisfies that each connected component is a graph in $\mathcal{G}$.

Note that, as discussed earlier, these two properties are exactly the ones that we need in order to make Π^{hard} promise-free.

3.4 Problem Π and its complexity in the LOCAL model

We already discussed the high-level idea behind the definition of Π, and in Section 7 we define the LCL problem Π more formally. We now give a bit more details about Π. Each node u receives a pair of inputs. One input is exactly the input for Π^{badGraph}, and the other input is a single bit b_u. The problem Π is defined such that each node needs to solve Π^{badGraph}, and then, on the subgraph induced by nodes producing an empty output for Π^{badGraph}, each node u needs to solve the problem

Π^{hard}, where the input of node u for Π^{hard} is b_u. While Π^{hard} was explained on square grids of size $\Theta(\sqrt{n}) \times \Theta(\sqrt{n})$, as we already discussed, hard instances can be a bit different from square grids, which makes it harder to prove an upper bound for the problem Π. In Section 8 we will provide lower and upper bounds for the problem Π in different models of computation.

The complexity of Π in the LOCAL model with shared randomness. When we discussed vertical grids, we also provided the high-level idea of the upper bound for solving Π^{hard} with shared randomness in the LOCAL model on hard instances, which we now recap and explain how it is used to solve Π in any graph. At first, the nodes spend $O(\log n)$ rounds to mark the "bad" parts of the graph. After that, each remaining connected component is a hard instance. Then, the problem Π^{hard} is solved by brute force on components of diameter $O(\log n)$, while shared randomness is used to solve components of larger diameter.

The complexity of Π in the LOCAL model with private randomness. In order to prove a lower bound of $\Omega(\sqrt{n})$ in the LOCAL model for private randomness, in Section 8, we essentially show that any algorithm that is too fast cannot deviate too much from being deterministic, or in other words, for each row of a hard instance, the algorithm must fix the output bits almost deterministically (i.e., the output must always be the same, with high probability). Then, we fix the input of the last column in an adversarial way, as a function of the almost-deterministic outputs of the algorithm, and we prove that in such case the failure probability of the algorithm is too large.

3.5 Complexity of Π in other models

In Section 9 we extend the $\Omega(\sqrt{n})$ lower bound to several other models that are far more powerful than LOCAL:

- **SLOCAL with private randomness (Section 9.1).** In the SLOCAL model [25], nodes are revealed in a sequential, potentially adversarial order. Each time a node is revealed, it sees all the states of previously revealed nodes that are at most T hops away from it (in particular also their outputs) and is asked to commit to an output. This model is clearly more powerful than LOCAL since the sequential processing of nodes gives us symmetry breaking essentially for free.
- **Deterministic online-LOCAL model (Section 9.2).** The deterministic online-LOCAL model [3] is similar to SLOCAL in that the nodes are also revealed following some sequential order. Nevertheless, it is potentially more powerful than SLOCAL because it is able to maintain *global* knowledge of what has been revealed thus far. We show the lower bound for the deterministic variant of online-LOCAL. A randomized variant of online-LOCAL also exists [2], but clearly it has access to shared randomness by definition, and so the upper bound from LOCAL extends there.
- **Bounded-dependence model (Section 9.3).** The bounded-dependence model encompasses all algorithms satisfying the property that the output of any node is independent of outputs that are more than T hops away from it. This includes, for instance, all quantum-LOCAL algorithms without a pre-shared quantum state. The bounded-dependence model has been shown to be less powerful than the randomized version of online-LOCAL [2], but its relation to deterministic online-LOCAL is still unclear.

Throughout this work we assume that the nodes have (implicitly or explicitly) some knowledge on the total number of nodes n. In Appendix A we show that this is a necessary assumption.

4 A tree-like structure

In this section, we first formally define what is a tree-like structure, then we provide some local constraints that, if satisfied on all nodes, guarantee that the graph is tree-like, and finally we define an LCL problem allowing nodes to quickly prove that the graph is not tree-like.

Definition 4.1 (Tree-like structure)**.** We say that a graph G is a *tree-like structure* of height ℓ if it is possible to assign coordinates (l_u, k_u) to each node $u \in G$, where

- $0 \leq l_u < \ell$ denotes the depth of u in the tree, and
- $0 \leq k_u < 2^{l_u}$ denotes the position of u (according to some order) in layer l_u,

such that there is an edge connecting two nodes $u, v \in G$ with coordinates (l_u, k_u) and (l_v, k_v) if and only if:

- $l_u = l_v$ and $|k_u - k_v| = 1$, or
- $l_v = l_u - 1$ and $k_v = \lfloor \frac{k_u}{2} \rfloor$, or
- $l_u = l_v - 1$ and $k_u = \lfloor \frac{k_v}{2} \rfloor$.

4.1 Local checkability of a tree-like structure

We now define sets of labels $\mathcal{V}^{\mathsf{tree}}$ and $\mathcal{E}^{\mathsf{tree}}$, and a set of local constraints over such labels, satisfying that a graph G can be $(\mathcal{V}^{\mathsf{tree}}, \mathcal{E}^{\mathsf{tree}})$-labeled satisfying the constraints if and only if G is a tree-like structure. The tree-like structure that we use in this paper is the same exact one used in [12], and hence we now report here the constraints as defined in [12].

First of all, nodes are not labeled at all, or equivalently, all nodes have the same label $\perp$. Hence, we define $\mathcal{V}^{\mathsf{tree}} = \{\perp\}$. Then, the possible half-edge labels are $\mathcal{E}^{\mathsf{tree}} = \{\mathsf{L}, \mathsf{R}, \mathsf{P}, \mathsf{Ch_L}, \mathsf{Ch_R}\}$ (the labels stand for "left", "right", "parent", "left child", and "right child", respectively). The set of local constraints $\mathcal{C}^{\mathsf{tree}}$ is defined as follows.

The constraints $\mathcal{C}^{\mathsf{tree}}$ of [12]

1. For any two edges e, e' incident to a node u, it must hold that $L_u(e) \neq L_u(e')$;
2. For each edge $e = \{u, v\}$, if $L_u(e) = \mathsf{L}$, then $L_v(e) = \mathsf{R}$, and vice versa;
3. For each edge $e = \{u, v\}$, if $L_u(e) = \mathsf{P}$, then $L_v(e) \in \{\mathsf{Ch_L}, \mathsf{Ch_R}\}$, and vice versa;
4. If a node u has an incident edge $e = \{u, v\}$ with label $L_u(e) = \mathsf{P}$ such that $L_v(e) = \mathsf{Ch_L}$, then $f(u, \mathsf{P}, \mathsf{Ch_R}, \mathsf{L}) = u$;
5. If a node u has an incident edge $e = \{u, v\}$ with label $L_u(e) = \mathsf{P}$ such that $L_v(e) = \mathsf{Ch_R}$, if u has an incident edge labeled R, then $f(u, \mathsf{P}, \mathsf{R}, \mathsf{Ch_L}, \mathsf{L}) = u$.
6. If a node has an incident half-edge labeled $\mathsf{Ch_L}$, then it must also have an incident half-edge labeled $\mathsf{Ch_R}$, and vice versa;
7. A node does not have an incident half-edge labeled P if and only if it has no incident half-edges labeled L or R;
8. If a node u does not have an incident edge e with label $L_u(e) \in \{\mathsf{Ch_L}, \mathsf{Ch_R}\}$, then neither do nodes $f(u, \mathsf{L})$ and $f(u, \mathsf{R})$ (if they exist);

9. If a node u has an incident edge $e = \{u, v\}$ with label $L_u(e) = \mathsf{P}$ such that $L_v(e) = \mathsf{Ch_R}$ (resp. $L_v(e) = \mathsf{Ch_L}$), then u has an incident edge labeled R (resp. L) if and only if $f(u, \mathsf{P})$ has an incident edge labeled R (resp. L).

An example of valid tree-like structure labeled according to $\mathcal{C}^{\mathsf{tree}}$ is depicted in Figure 4.

In [12], it has been shown that a graph G can be $(\mathcal{V}^{\mathsf{tree}}, \mathcal{E}^{\mathsf{tree}})$-labeled satisfying $\mathcal{C}^{\mathsf{tree}}$ if and only if it is a tree-like structure. This result is captured by the following two lemmas.

Lemma 4.2 ([12]). *Let G be a graph that is labeled with labels in $\mathcal{E}^{\mathsf{tree}}$ such that $\mathcal{C}^{\mathsf{tree}}$ is satisfied for all nodes in G. Then, G is a tree-like structure.*

Lemma 4.3 ([12]). *Each tree-like structure graph G can be labeled with labels in $\mathcal{E}^{\mathsf{tree}}$ such that the constraints $\mathcal{C}^{\mathsf{tree}}$ are satisfied at all nodes in G.*

4.2 Proving that a graph is not tree-like

We now define an LCL problem Π^{badTree}, already informally introduced in Section 3.1. We omitted some details there, that we now present. While we described tree-like structures as separate entities, we will use such structures in connection to another one, i.e., a grid. For this reason, we want to allow nodes to not only prove that the tree-like structure is invalid, but also to produce a proof in the case in which the tree-like structure is valid but wrongly connected to the rest of the graph. In order to capture this, in the definition of Π^{badTree}, nodes receive some input, and in particular nodes could be marked, indicating that there is some issue unrelated to the tree-like structure per se. More in detail, the high-level idea behind the definition of Π^{badTree} is the following:

- Nodes receive an input indicating whether they are marked or not;
- There are two possible types of output, either nodes give an empty output, or nodes produce a proof that the graph is not tree-like or that it contains at least one marked node;
- If the graph is tree-like and does not contain any marked node, the only valid solution is the one where all nodes produce an empty output.
- If the graph is not tree-like, or it contains at least one marked node, nodes can spend $O(\log n)$ time in the LOCAL model to produce a proof of this fact.

More formally, the problem Π^{badTree} is defined on $(\{0, 1\}, \mathcal{E}^{\mathsf{tree}})$-labeled graphs, that is, it is assumed that every half-edge has an input label in $\mathcal{E}^{\mathsf{tree}}$, and that each node is either labeled 0 or 1, where nodes labeled 1 are denoted as *marked* nodes. The output labels of Π^{badTree} are $\mathcal{V}^{\mathsf{badTree}} = \{\mathsf{Error}, \bot\} \cup \{(\mathsf{pointer}, p) \mid p \in \{\mathsf{L}, \mathsf{R}, \mathsf{P}, \mathsf{Ch_R}\}\}$. The constraints $\mathcal{C}^{\mathsf{badTree}}$ of Π^{badTree} are defined as follows.

The constraints $\mathcal{C}^{\mathsf{badTree}}$

1. A node can always output $\bot$.
2. A node u can output Error only if u does not satisfy $\mathcal{C}^{\mathsf{tree}}$ or if u is marked.
3. Let u be a node outputting $(\mathsf{pointer}, p)$. It is required that node u has an incident half-edge labeled p. Let v be the node $f(u, p)$, that is, the node reachable from u by following its half-edge labeled p. Then, either the output of v is Error, or the output of

v is $(\mathsf{pointer}, p')$. In the latter case, it must hold that $f(v,p) \neq u$, and the values of p and p' must satisfy the following:

(a) If $p \in \{\mathsf{L}, \mathsf{R}\}$ then $p' = p$;

(b) If $p = \mathsf{P}$ then $p' \in \{\mathsf{P}, \mathsf{L}, \mathsf{R}\}$;

(c) If $p = \mathsf{Ch_R}$ then $p' \in \{\mathsf{Ch_R}, \mathsf{L}, \mathsf{R}\}$.

We now prove that, on properly labeled tree-like structures not containing any marked nodes, the only solution for Π^{badTree} is the one where all nodes output $\perp$.

Lemma 4.4. *Let G be a $(\{0,1\}, \mathcal{E}^{\mathsf{tree}})$-labeled graph where $\mathcal{C}^{\mathsf{tree}}$ is satisfied on all nodes and all nodes are not marked. Then, the only valid solution for Π^{badTree} is the one assigning $\perp$ to all nodes.*

Proof. Constraint 1 ensures that a solution where all nodes output $\perp$ is indeed possible, and constraint 2 ensures that there is no node in G outputting the label Error. In the following, we prove that no node can output a pointer label, implying the claim.

For a contradiction, assume that there is a node u outputting $(\mathsf{pointer}, p)$, for some $p \in \{\mathsf{L}, \mathsf{R}, \mathsf{P}, \mathsf{Ch_R}\}$. Since no node outputs Error, then $v := f(u,p)$ must output $(\mathsf{pointer}, p')$ satisfying that $f(v,p') \neq u$. The same reasoning applies recursively on v. Hence, there must exist a path $(v_1, v_2, \ldots, v_k)$ where $v_1 = u$, $v_2 = v$, and each node v_i outputs $(\mathsf{pointer}, p_i)$, for some value p_i. Since no node outputs Error, then $v_k = v_i$ for some $i \leq k-2$, forming a cycle. In the following, we prove that the constraints of Π^{badTree} ensure that cycles are not possible.

A necessary condition to properly label $(v_i, \ldots, v_k)$ with pointers is to use both labels P and $\mathsf{Ch_R}$, since any cycle on these nodes must start on some layer of the tree-like structure, then change layer, and eventually go back to the starting layer. However, the sequence of pointers allowed by constraint 3 must satisfy the regular expression $(\mathsf{P}^*|\mathsf{Ch_R}^*)(\mathsf{L}^*|\mathsf{R}^*)$, which contradicts the fact that P and $\mathsf{Ch_R}$ are present at the same time. ☐

In [12], it has been shown that, if the graph is not a valid tree-like structure, or if there is at least one marked node, then nodes can efficiently produce a proof of this fact.

Lemma 4.5 (Lemma 6.10 in the arXiv version of [12], rephrased)**.** *Let G be a $(\{0,1\}, \mathcal{E}^{\mathsf{tree}})$-labeled graph where either $\mathcal{C}^{\mathsf{tree}}$ is not satisfied on at least one node, or there is at least one marked node. Then, there exists a solution for Π^{badTree} where all nodes produce an output different from $\perp$. Moreover, such a solution can be computed in $O(\log n)$ rounds in the LOCAL model.*

5 Grid structure

In this section, we first formally define what is a grid structure, then we provide some local constraints that, if satisfied on all nodes, guarantee that the graph is a grid (or something that locally looks like a grid everywhere), and finally we define some constraints that enforce a grid to be vertical.

Definition 5.1 (Grid structure)**.** We say that a graph G is a *grid stricture* of size $h \times w$ $(h, w > 0)$ if it is possible to assign coordinates (x_u, y_u) to each node $u \in G$, where $0 \leq x_u < w$, $0 \leq y_u < h$, such that there is an edge connecting two nodes $u, v \in G$ with coordinates (x_u, y_u) and (x_v, y_v) if and only if:

- $x_u = x_v$ and $|y_u - y_v| = 1$, or
- $y_u = y_v$ and $|x_u - x_v| = 1$.

5.1 Local checkability of a grid structure

We now define sets of labels $\mathcal{V}^{\mathsf{grid}}$ and $\mathcal{E}^{\mathsf{grid}}$, and a set of local constraints over such labels, satisfying that a graph G can be $(\mathcal{V}^{\mathsf{grid}}, \mathcal{E}^{\mathsf{grid}})$-labeled satisfying the constraints if and only of G is a grid structure (with some exceptions that we will see later). The grid structure that we use in this paper is the same exact one used in [12], and hence we now report here the constraints as defined in [12].

First of all, nodes are not labeled at all, or equivalently, all nodes have the same label $\perp$. Hence, we define $\mathcal{V}^{\mathsf{grid}} = \{\perp\}$. Then, the possible half-edge labels are $\mathcal{E}^{\mathsf{grid}} = \{\mathsf{U}, \mathsf{D}, \mathsf{L}, \mathsf{R}\}$ (the labels stand for "up", "down", "left", and "right", respectively). The set of local constraints $\mathcal{C}^{\mathsf{grid}}$ is defined as follows.

The constraints $\mathcal{C}^{\mathsf{grid}}$ of [12]

1. For any two edges e, e' incident to a node u, it must hold that $L_u(e) \neq L_u(e')$.
2. For each edge $e = \{u, v\}$, if $L_u(e) = \mathsf{L}$, then $L_v(e) = \mathsf{R}$, and vice versa.
3. For each edge $e = \{u, v\}$, if $L_u(e) = \mathsf{U}$, then $L_v(e) = \mathsf{D}$, and vice versa.
4. If a node u has two incident edges labeled with R and U respectively, then it must hold that $f(u, \mathsf{R}, \mathsf{U}, \mathsf{L}, \mathsf{D}) = u$.
5. If $f(u, \mathsf{R})$ exists, then u has an incident edge labeled with D (resp. U) if and only if $f(u, \mathsf{R})$ has an incident edge labeled with D (resp. U).
6. If $f(u, \mathsf{U})$ exists, then u has an incident edge labeled with L (resp. R) if and only if $f(u, \mathsf{U})$ has an incident edge labeled with L (resp. R).

An example of valid grid structure labeled according to $\mathcal{C}^{\mathsf{grid}}$ is depicted in Figure 5.

In [12], it has been shown that a graph G can be $(\mathcal{V}^{\mathsf{grid}}, \mathcal{E}^{\mathsf{grid}})$-labeled satisfying $\mathcal{C}^{\mathsf{grid}}$ if and only if it is a grid structure, with some exceptions. This result is captured by the following two lemmas.

Lemma 5.2 ([12])**.** *Let G be a graph that is $(\mathcal{V}^{\mathsf{grid}}, \mathcal{E}^{\mathsf{grid}})$-labeled such that $\mathcal{C}^{\mathsf{grid}}$ is satisfied for all nodes in G. Moreover, assume that there exists at least one node that has no incident half-edge labeled* D *(or* U*), and that there exists at least one node that has no incident half-edge labeled* L *(or* R*). Then, G is a grid structure.*

Lemma 5.3 ([12])**.** *Each grid structure graph G can be $(\mathcal{V}^{\mathsf{grid}}, \mathcal{E}^{\mathsf{grid}})$-labeled such that the constraints $\mathcal{C}^{\mathsf{grid}}$ are satisfied at all nodes in G.*

5.2 Additional constraints on the grid structure

We now formally define what is a vertical grid structure.

Definition 5.4 (Vertical grid structure)**.** A $(\mathcal{V}^{\mathsf{grid}}, \mathcal{E}^{\mathsf{grid}})$-labeled graph G is a *vertical grid structure* if and only if:

- The graph G is a grid structure and $\mathcal{C}^{\mathsf{grid}}$ is satisfied on all nodes;
- Let w (resp. h) be the length of the paths induced by half-edges labeled L or R (resp. U or D). Then, $h \geq w$.

We augment the $(\mathcal{V}^{\mathsf{grid}}, \mathcal{E}^{\mathsf{grid}})$-labeling, and we define additional constraints, in order to obtain the local checkability of a vertical grid structure. The labeling for vertical grids is obtained by only changing the labels of the nodes. We define $\mathcal{V}^{\mathsf{vGrid}}$ as $\{0,1\}$. The additional constraints are the following.

The constraints to be added to $\mathcal{C}^{\mathsf{grid}}$ to obtain $\mathcal{C}^{\mathsf{vGrid}}$

1. If a node u is labeled 1, then $f(u, \mathsf{U}, \mathsf{R})$ and $f(u, \mathsf{D}, \mathsf{L})$ are also labeled 1, if they exist.
2. If a node u is labeled 1 and $f(u, \mathsf{D}) = \perp$, then $f(u, \mathsf{L}) = \perp$.
3. If a node u is labeled 1 and $f(u, \mathsf{U}) = \perp$, then $f(u, \mathsf{R}) = \perp$.

We call $\mathcal{C}^{\mathsf{vGrid}}$ the constraints obtained by adding the above additional constraints to $\mathcal{C}^{\mathsf{grid}}$. Note that, by labeling all nodes with 0, the additional constraints in $\mathcal{C}^{\mathsf{vGrid}}$ are trivially satisfied. Hence, we cannot claim that if $\mathcal{C}^{\mathsf{vGrid}}$ is satisfied then the graph is a vertical grid. However, we now prove that a graph G can be $(\mathcal{V}^{\mathsf{vGrid}}, \mathcal{E}^{\mathsf{grid}})$-labeled *such that at least one node is labeled* 1 and satisfying $\mathcal{C}^{\mathsf{vGrid}}$ if and only if it is a vertical grid structure.

Lemma 5.5. *Let G be a graph that is $(\mathcal{V}^{\mathsf{vGrid}}, \mathcal{E}^{\mathsf{grid}})$-labeled such that $\mathcal{C}^{\mathsf{vGrid}}$ is satisfied on all nodes in G. Moreover, assume that there exists at least one node that has no incident half-edge labeled* D *(or* U*), that there exists at least one node that has no incident half-edge labeled* L *(or* R*), and that there exists at least one node labeled* 1*. Then, G is a vertical grid structure.*

Proof. By Lemma 5.2, the graph is a grid structure, and hence it has some size $h \times w$. We prove that $h \geq w$. By assumption, there is at least one node labeled 1, and let (x, y) be its coordinates. By the additional constraint 1 of $\mathcal{C}^{\mathsf{vGrid}}$, for any integer i, each node with coordinates $(x+i, y+i)$, if it exists, is also labeled 1. By constraint 2, and by the fact that G is a grid structure, there exists some i such that the node $(x+i, y+i) = (0, y-x)$ exists, where $y - x \geq 0$. Similarly, by constraint 3, and by the fact that G is a grid structure, there exists some i such that the node $(x+i, y+i) = (w-1, y+w-1-x)$ exists, where $y - x + w - 1 \leq h - 1$. By combining the two inequalities, we obtain $w \leq h$, as required. □

Lemma 5.6. *Each vertical grid structure graph G can be $(\mathcal{V}^{\mathsf{vGrid}}, \mathcal{E}^{\mathsf{grid}})$-labeled such that, at least one node is labeled* 1*, and the constraints $\mathcal{C}^{\mathsf{vGrid}}$ are satisfied at all nodes in G.*

Proof. We label the nodes as follows. Initially, all nodes are labeled 0. Then, we start from the node with coordinates $(0,0)$, i.e., the node not having any edge labeled D or L, and we iteratively do the following.

- Label the current node u with 1.
- Move to $f(u, \mathsf{R}, \mathsf{U})$ if it exists, otherwise stop.

Since the grid is vertical, this process necessarily ends on some node u that does not have any edge labeled R, ensuring that the constraints are satisfied on all nodes. □

6 Graph family $\mathcal{G}$

In this section, we first formally define the family $\mathcal{G}$ of hard instances, and then we define an LCL problem Π^{badGraph} that allows nodes to prove that some parts of the graph do not look like valid hard instances.

Definition 6.1 (The graph family $\mathcal{G}$)**.** A graph G is in $\mathcal{G}$ if and only if G can be constructed by the following process. Take an $h \times w$ grid structure H, where $h \geq w$, $h = 2^{\ell}$ for some integer $\ell \geq 0$, and $w > 0$. Take w many tree-like structures T_i, where $i \in \{0, 1, \ldots, w-1\}$, of height ℓ. Put each tree-like structure T_i on top of the ith column of the grid structure H, that is, identify the node $u \in T_i$ with coordinates $(\ell - 1, k_u)$ with the node $v \in H$ with coordinates (i, k_u).

An example of a graph in the family $\mathcal{G}$ is depicted in Figure 2.

6.1 LCL problem Π^{badGraph}

We now define a problem Π^{badGraph} that, informally, satisfies the following.

- There are two possible types of output, either nodes give an empty output, or nodes produce a proof that the graph is not in the family.
- If the graph is in the family, the only valid solution is the one where all nodes produce an empty output.
- If the graph is not in the family, nodes can spend $O(\log n)$ time in the LOCAL model to produce a proof of this fact. Moreover, the output can be constructed such that the subgraph induced by nodes producing an empty output satisfies that each connected component is a graph in the family.

Input. Each node is either marked to be both a grid node and a tree node, or only a tree node. Moreover, if it is a grid and tree node, it also receives an input label from $\mathcal{V}^{\mathsf{vGrid}}$. That is, the possible inputs are $\mathcal{V}^{\mathsf{badGraph}} = \{(\mathsf{treeNode})\} \cup \{(\mathsf{treeNode}, \mathsf{gridNode}, \ell) \mid \ell \in \mathcal{V}^{\mathsf{vGrid}}\}$. Each half-edge is either marked to be a grid edge, or a tree edge, or both. Moreover, grid half-edges also receive an input label from $\mathcal{E}^{\mathsf{grid}}$, and tree half-edges also receive an input label from $\mathcal{E}^{\mathsf{tree}}$. Additionally, the input satisfies that all grid half-edges that receive the input U (resp. D) are also marked as tree edges labeled R (resp. L). Finally, all grid half-edges that receive the input L or R are not marked as tree half-edges. Hence, more formally, the possible half-edge labels are $\mathcal{E}^{\mathsf{badGraph}} = \{(\mathsf{treeEdge}, \ell) \mid \ell \in \mathcal{E}^{\mathsf{tree}}\} \cup \{(\mathsf{gridEdge}, \ell) \mid \ell \in \{\mathsf{L}, \mathsf{R}\}\} \cup \{((\mathsf{gridEdge}, \mathsf{D}), (\mathsf{treeEdge}, \mathsf{L}))\} \cup \{((\mathsf{gridEdge}, \mathsf{U}), (\mathsf{treeEdge}, \mathsf{R}))\}$. An example of a "good" input labeling (i.e., that will force all nodes to produce an empty output) for Π^{badGraph} is depicted in Figure 6.

Output. On a high level, there are two possible outputs.

- Nodes can give an empty output. This is always an option, making the problem trivial.
- However, we allow nodes to prove that something is broken in their column, where a column of a node u is defined as the connected component containing u in the subgraph obtained by ignoring grid edges labeled L or R. A column is considered broken if edges are not consistently labeled, the tree-like structure is not valid, or there is some grid node in the column not satisfying the grid constraints.

We will prove that, if a graph belongs to $\mathcal{G}$, the only valid output is the empty output, while if a graph does not belong to $\mathcal{G}$, then there exists a valid output, computable in $O(\log n)$ rounds in the LOCAL model, where the connected components of the subgraph induced by nodes producing an empty output satisfies that each connected component is a graph in $\mathcal{G}$. More formally, the possible outputs are the following.

- $\perp$: used by the nodes to produce an empty output.

- Error: used by the nodes to indicate that the edges are not consistently labeled (for example, if one half-edge of an edge is labeled as $\mathsf{treeEdge}$, the other half-edge of the same edge must also be labeled $\mathsf{treeEdge}$, or if a node is not marked as a grid node but it has incident grid edges).
- $\mathsf{TreeError}$: used by the nodes to indicate that the constraints $\mathcal{C}^{\mathsf{tree}}$ are not satisfied.
- $\mathsf{GridError}$: used by the nodes to indicate that the constraints $\mathcal{C}^{\mathsf{vGrid}}$ are not satisfied.
- $(\mathsf{ColumnError}, \ell)$, where $\ell \in \mathcal{V}^{\mathsf{badTree}}$: used by the nodes to indicate that the tree-like structure they belong to (i.e., their column in the grid) contains some error, which can be Error, $\mathsf{GridError}$, or $\mathsf{TreeError}$. For technical reasons, we identify the label $(\mathsf{ColumnError}, \bot)$ as the same label as $\bot$.
- $\mathsf{VertError}$: used by the nodes to indicate that no node in the column they belong to has label $(\mathsf{treeNode}, \mathsf{gridNode}, 1)$, i.e., no node in the column has input 1 from $\mathcal{V}^{\mathsf{vGrid}}$, meaning that the proof that the grid is vertical is missing.

Constraints. We now define the constraints of the problem Π^{badGraph}.

We first define a function t that, given the labeling of a half-edge label, extracts its *type*.

Definition 6.2 (Type of half-edge labels and types of edges)**.** The type of a half-edge label is defined as the result of applying the function t defined as follows: $t((\mathsf{treeEdge}, \ell)) = \{\mathsf{treeEdge}\}$, $t((\mathsf{gridEdge}, \ell)) = \{\mathsf{gridEdge}\}$, $t(((\mathsf{treeEdge}, \ell), (\mathsf{gridEdge}, \ell'))) = \{\mathsf{treeEdge}, \mathsf{gridEdge}\}$.

Moreover, the type of an edge $e = \{u, v\}$ is defined as $\{\}$ if $t(L_u(e)) \neq t(L_v(e))$, and as $t(L_u(e))$ otherwise.

Then, we define a function val_T (resp. val_G) that takes a label containing type $\mathsf{treeEdge}$ (resp. $\mathsf{gridEdge}$) and returns the label associated with it.

Definition 6.3 (Tree value of a half-edge label)**.** The tree value of a half-edge label is defined as the result of applying the function val_T defined as follows:

$$\begin{aligned} \mathsf{val}_T((\mathsf{treeEdge}, \ell)) &= \ell, \\ \mathsf{val}_T(((\mathsf{treeEdge}, \ell), (\mathsf{gridEdge}, \ell'))) &= \ell, \end{aligned}$$

and it is undefined otherwise.

Definition 6.4 (Grid value of a half-edge label)**.** The grid value of a half-edge label is defined as the result of applying the function val_G defined as follows:

$$\begin{aligned} \mathsf{val}_G((\mathsf{gridEdge}, \ell)) &= \ell, \\ \mathsf{val}_G(((\mathsf{treeEdge}, \ell), (\mathsf{gridEdge}, \ell'))) &= \ell', \end{aligned}$$

and it is undefined otherwise.

We are now ready to define the constraints $\mathcal{C}^{\mathsf{badGraph}}$ of Π^{badGraph}.

The constraints $\mathcal{C}^{\mathsf{badGraph}}$

1. The output $\bot$ is always allowed.

2. A node u can output Error if it has an incident edge $e = \{u, v\}$ satisfying that $t(L_u(e)) \neq t(L_v(e))$, or if its input is (treeNode) and it has at least one incident half-edge with a type containing gridEdge.

3. Consider the graph G' induced by edges that have a type containing treeEdge. Consider the labeling of the half-edges of G' obtained by mapping each half-edge (u, e) (satisfying $u \in e$) into $\mathsf{val}_T(L_u(e))$. A node can output TreeError if, in G', the constraints $\mathcal{C}^{\mathsf{tree}}$ are not satisfied.

4. Consider the graph G' induced by edges that have a type containing gridEdge. Consider the labeling of the half-edges of G' obtained by mapping each half-edge (u, e) (satisfying $u \in e$) into $\mathsf{val}_G(L_u(e))$. A node can output GridError if its input is not (treeNode) and, in G', the constraints $\mathcal{C}^{\mathsf{grid}}$ are not satisfied.

5. Consider the graph G' induced by edges that have a type containing treeEdge. Consider the following labeling of G':

   - The input labeling of the half-edges of G' is obtained by mapping each half-edge (u, e) (satisfying $u \in e$) into $\mathsf{val}_T(L_u(e))$.
   - The output labeling of the nodes of G' is obtained by mapping each node u labeled $(\mathsf{ColumnError}, \ell)$ into ℓ, each node u labeled Error, TreeError, or GridError into Error, and each other node into $\perp$.
   - The input labeling of the nodes is obtained by labeling 1 nodes that have an output in $\{\mathsf{Error}, \mathsf{TreeError}, \mathsf{GridError}\}$, and 0 all other nodes.

   Then, on G', the constraints of $\mathcal{C}^{\mathsf{badTree}}$ must be satisfied.

6. Consider the graph G' induced by edges that have a type containing treeEdge. If a node has output VertError, then all its neighbors in G' must also have VertError as output. Moreover, if a node has output VertError, then it must not have input $(\mathsf{treeNode}, \mathsf{gridNode}, 1)$.

Properties of the problem. We now prove some useful properties about the problem Π^{badGraph}.

Lemma 6.5. *Let G be a graph in $\mathcal{G}$. There exists a $(\mathcal{V}^{\mathsf{badGraph}}, \mathcal{E}^{\mathsf{badGraph}})$-labeling of G satisfying that the only valid output labeling for Π^{badGraph} is the one assigning $\perp$ to all nodes.*

Proof. Recall that any $G \in \mathcal{G}$ is obtained by starting from a vertical grid structure and then connecting a tree-like structure on top of each column. For each node u that is part of the grid structure, let $\ell_g(u)$ be the label of node u obtained by Lemma 5.6. For each half-edge (u, e) that is part of the grid structure, let $\ell_g(u, e)$ be the label of the half-edge (u, e) obtained by Lemma 5.6. For each half-edge that is part of the tree-like structure, let $\ell_t(u, e)$ be the label of the half-edge (u, e) given by Lemma 4.3. For each node u of G, if it is only part of a tree-like structure, we assign the label (treeNode), while if u is part of both a tree-like structure and the grid, we assign the label $(\mathsf{treeNode}, \mathsf{gridNode}, \ell_g(u))$. Then, to each half-edge (u, e) that is only part of the tree-like structure we assign the label $(\mathsf{treeEdge}, \ell_t(u, e))$, to each half-edge (u, e) that is only part of the grid structure we assign the label $(\mathsf{gridEdge}, \ell_g(u, e))$, while to all other half-edges we assign the label $((\mathsf{gridEdge}, \ell_g(u, e)), (\mathsf{treeEdge}, \ell_t(u, e)))$.

By construction of the input labeling, the outputs Error, $\mathsf{TreeError}$, and $\mathsf{GridError}$ are not allowed. Moreover, since by Lemma 4.3 at least one node is labeled $(\mathsf{treeNode}, \mathsf{gridNode}, 1)$, the output $\mathsf{VertError}$ is not allowed on that node, and by a propagation argument no node belonging to the same tree-like structure can use the output $\mathsf{VertError}$.

We thus get that the only possible outputs are $\perp$ or $(\mathsf{ColumnError}, \ell)$ for some ℓ. We prove that ℓ must be $\perp$, implying the claim. The fact that no node is allowed to output Error, $\mathsf{TreeError}$, $\mathsf{GridError}$, or $\mathsf{VertError}$, implies, by constraint 5 of $\mathcal{C}^{\mathsf{badGraph}}$, that the input labeling for Π^{badTree} satisfies that all nodes receive 0 as input. Since the tree-like structure is valid, by Lemma 4.4, the only way to satisfy $\mathcal{C}^{\mathsf{badTree}}$ is for all nodes to output $\perp$ as output for Π^{badTree}. $\square$

Lemma 6.6. *For any $(\mathcal{V}^{\mathsf{badGraph}}, \mathcal{E}^{\mathsf{badGraph}})$-labeled graph G, there exists a solution for Π^{badGraph} satisfying the following.*

- *The solution can be computed in $O(\log n)$ rounds in the* LOCAL *model.*
- *The graph induced by nodes that output $\perp$ satisfies that each connected component is a graph in $\mathcal{G}$.*

Proof. We present an algorithm that satisfies the requirements of the lemma. At first, each node u spends $O(1)$ rounds to check whether there is some local inconsistency in the structure of the graph. In particular, each node u first checks whether constraint 2 applies, and in that case it outputs Error. Then, it checks whether constraint 3 applies, and in that case it outputs $\mathsf{TreeError}$. Then, it checks whether constraint 4 applies, and in that case it outputs $\mathsf{GridError}$.

Then, nodes construct the input for Π^{badTree} by marking themselves 1 if they have output Error, $\mathsf{TreeError}$, or $\mathsf{GridError}$, and 0 otherwise. In each connected component obtained by ignoring grid edges labeled L or R, by Lemma 4.5, nodes can then spend $O(\log n)$ rounds to produce a solution of Π^{badTree} such that, if the tree-like structure they belong to is invalid or some node is marked, then no node of the connected component outputs $\perp$, while if the tree-like structure they belong to is valid and no node is marked, then all nodes output $\perp$. A node that obtained output ℓ outputs $(\mathsf{ColumnError}, \ell)$. Recall that $(\mathsf{ColumnError}, \perp) = \perp$. Note that this output satisfies the requirements of constraint 5.

Finally, each node u that output $\perp$ in the previous step, can spend $O(\log n)$ rounds to gather the whole tree-like structure it belongs to (since a valid tree-like structure has diameter $O(\log n)$). If u does not see any node labeled $(\mathsf{treeNode}, \mathsf{gridNode}, 1)$, it changes its output to $\mathsf{VertError}$. Since this operation is done consistently by all nodes, constraint 6 is satisfied.

We obtained an algorithm that produces a correct output for Π^{badGraph} in $O(\log n)$ rounds. We now prove that the output satisfies the second requirement of the lemma. Summarizing the above, we get that if a node u outputs $\perp$ then:

- the grid constraints are locally satisfied;
- the tree-like structure containing u is valid;
- by a propagation argument, the tree-like structure is correctly connected to the whole column of the grid;
- all the nodes in such a column also output $\perp$;
- the column of the grid contains at least one node labeled 1.

Hence, the subgraph G' induced by nodes that output $\perp$ is composed of whole columns of the grid with tree-like structures correctly attached. Moreover, if two such columns are connected, then the additional constraints of $\mathcal{C}^{\mathsf{vGrid}}$ are also satisfied.

Consider a connected component G'' in G'. Since the tree-like structures are valid and properly connected to the columns, no column can *wrap around*, i.e., for each column there is at least one grid node not having any half-edge labeled D and at least one grid node not having any half-edge labeled U. We now prove that, in the connected component G'', there is at least one node not having any incident half-edge labeled L (i.e., the grid does not wrap around horizontally). Consider an arbitrary column c of G'', and let u be an arbitrary node labeled 1 in c (such a node, by assumption, exists), and let i be the vertical coordinate of u in c. Consider the column c' obtained by starting from u and moving left for i steps. By the definition of $\mathcal{C}^{\mathsf{vGrid}}$, the column c' must satisfy that the node with vertical coordinate 0 has input 1, which, again by the definition of $\mathcal{C}^{\mathsf{vGrid}}$, implies that the node does not have any half-edge labeled L, as desired.

Recall that, when nodes of G'' checked whether the constraints of $\mathcal{C}^{\mathsf{vGrid}}$ were satisfied, they did it on G. However, it is easy to see that the constraints $\mathcal{C}^{\mathsf{vGrid}}$ satisfy a special property: if a node u satisfies $\mathcal{C}^{\mathsf{vGrid}}$, and an entire column is removed from the left or the right of u, then the constraints are still satisfied on u. This implies that, even if we restrict to G'', the constraints $\mathcal{C}^{\mathsf{vGrid}}$ are still satisfied on all nodes of G''. Hence, we now have all the requirements for applying Lemma 5.5 and proving that the nodes of G'' labeled as grid nodes form a vertical grid structure, which implies our claim. □

7 LCL problem Π

On a high level, the problem Π will be defined in such a way that it requires nodes to solve Π^{badGraph}, and then, additionally, nodes that output $\perp$ for Π^{badGraph} are required to solve an additional problem. Such a problem requires non-local coordination.

Input. The set of input labels of the nodes is $\mathcal{V}^{\Pi} = \mathcal{V}^{\mathsf{badGraph}} \times \{0, 1\}$, that is, nodes receive an input of the LCL Π^{badGraph} and an additional bit. The set of input labels of half-edges is $\mathcal{E}^{\Pi} = \mathcal{E}^{\mathsf{badGraph}}$.

Output. The set of output labels of the nodes contains the following labels.

- All labels in $\mathcal{V}^{\mathsf{badGraph}} \setminus \{\perp\}$. A valid solution for Π will be to solve Π^{badGraph} without using the label $\perp$. In this case, nothing additional will be required.
- All labels in $\{0, 1\} \times \{\mathsf{yes}, \mathsf{no}\}$. These outputs will be used by grid nodes.
- The labels yes and no, which will be used by nodes that are not in the grid.

Constraints. We now define the node constraints $\mathcal{C}^{\Pi}$ of Π.

The constraints $\mathcal{C}^{\Pi}$

1. Consider the output labeling given by mapping all labels not in $\mathcal{V}^{\mathsf{badGraph}}$ to $\perp$. The constraints of Π^{badGraph} must be satisfied.
2. If a node is labeled $(\mathsf{treeNode}, \mathsf{gridNode}, \ell)$ for some ℓ and its output is not in $\mathcal{V}^{\mathsf{badGraph}}$, then its output must be in $\{0, 1\} \times \{\mathsf{yes}, \mathsf{no}\}$.
3. Let u be a node with output $(b, x) \in \{0, 1\} \times \{\mathsf{yes}, \mathsf{no}\}$. Let v be $f(u, (\mathsf{gridEdge}, \mathsf{R}))$. If v exists, the output of v must be either a label in $\mathcal{V}^{\mathsf{badGraph}}$ or (b, x') for some x'.
4. Let u be a node with input (ℓ, b_{in}) for some ℓ, output $(b_{\mathrm{out}}, x) \in \{0, 1\} \times \{\mathsf{yes}, \mathsf{no}\}$, and such that $f(u, (\mathsf{gridEdge}, \mathsf{R})) = \perp$ (i.e., u does not have a right neighbor in the grid). It must hold that $x = \mathsf{yes}$ if and only if $b_{\mathrm{in}} = b_{\mathrm{out}}$.

5. If a node is labeled $(\mathsf{treeNode})$ and its output is not in $\mathcal{V}^{\mathsf{badGraph}}$, then its output must be in $\{\mathsf{yes}, \mathsf{no}\}$.

6. Let u be a node with output $x_u \in \{\mathsf{yes}, \mathsf{no}\}$, i.e., it is a node that does not belong to the grid, but it belongs to a tree-like structure and did not give an output from $\mathcal{V}^{\mathsf{badGraph}}$. Let v be the node $f(u, (\mathsf{tree}, \mathsf{Ch_L}))$, and let z be the node $f(u, (\mathsf{tree}, \mathsf{Ch_R}))$. It must hold that the output of v is either $x_v \in \{\mathsf{yes}, \mathsf{no}\}$, or $(b_v, x_v) \in \{0, 1\} \times \{\mathsf{yes}, \mathsf{no}\}$. Similarly, the output of z is either $x_z \in \{\mathsf{yes}, \mathsf{no}\}$, or $(b_z, x_z) \in \{0, 1\} \times \{\mathsf{yes}, \mathsf{no}\}$. It must hold that $x_u = \mathsf{yes}$ if and only if at least one of x_v or x_z is yes.

7. Let u be a node with output $x_u \in \{\mathsf{yes}, \mathsf{no}\}$ and such that $f(u, (\mathsf{tree}, \mathsf{P})) = \perp$. Then, x_u must be yes.

An example of valid output is shown in Figure 3.

Properties of Π. We now characterize what are the valid solutions for Π, when we consider hard instances.

Lemma 7.1. *Let $G \in \mathcal{G}$. It is possible to label G such that the only valid solutions for Π satisfy the following.*

- *For each row of the grid structure, all nodes in that row output the same bit.*
- *There exists at least one row satisfying that the output bit given by the nodes is the same as the one provided as input to the right-most node in the row.*

Proof. By Lemma 6.5, there exists a $(\mathcal{V}^{\mathsf{badGraph}}, \mathcal{E}^{\mathsf{badGraph}})$-labeling of G where the only output labeling satisfying $\mathcal{C}^{\mathsf{badGraph}}$ is the one assigning $\perp$ to all nodes. Consider this labeling as input for the problem Π. Nodes cannot use any output from $\mathcal{V}^{\mathsf{badGraph}} \setminus \{\perp\}$.

By constraint 2 of $\mathcal{C}^{\Pi}$, each node of the grid must output a pair $\{0, 1\} \times \{\mathsf{yes}, \mathsf{no}\}$, and by constraint 3 of $\mathcal{C}^{\Pi}$ for each row of the grid it must hold that the first element of the pairs given by the nodes must be the same, i.e., all 0 or all 1, which implies the first property of the lemma.

By constraint 4, a grid node of the right-most column is allowed to output (b, yes) for some b only if b matches the bit received as input. Then, by constraints 5 and 6, a node of a tree-like structure is allowed to output yes only if at least one of its children outputs also yes. Since, by constraint 7, the root of the tree-like structure in the right-most column must output yes, we get that at least one node of the right-most column outputs (b, yes) for some b, which implies the second property of the lemma. $\square$

8 Complexity in the LOCAL model

In this section we will show that our problem Π is indeed hard in the randomized LOCAL model with private randomness, but easy with shared randomness.

8.1 Complexity with private randomness

Theorem 8.1. *In the LOCAL model with private randomness, solving the problem Π with success probability at least $1 - 1/n$ requires $\Omega(\sqrt{n})$ rounds.*

Proof. We consider the subfamily of graphs in $\mathcal{G}$ satisfying that the dimensions w and h of the grid satisfy $w = h$, and we apply Lemma 7.1.

Assume for a contradiction that there exists an algorithm $\mathcal{A}$ that solves Π in $o(\sqrt{n})$ rounds.

Let N be such that, for any $n \geq N$, the time complexity of $\mathcal{A}$ in graphs of size n is at most $w/3$. Consider a row, its left-most node u and its right-most node v. By the assumption on the runtime of the algorithm, their outputs are independent. Let p_u (resp. p_v) be the probability that u (resp. v) outputs 0. It must hold that $p_u(1 - p_v) < 1/n$ and that $(1 - p_u)p_v < 1/n$, since otherwise, the algorithm would produce a row that does not have the same bit everywhere (which contradicts Lemma 7.1) with too large probability. This implies that either both p_u and p_v are at most $2/n$ or that they are both at least $1 - 2/n$.

We now restrict to instances where, for each row with left-most node u and right-most node v, the input of v is 0 if $p_u \leq 2/n$, and 1 otherwise. On these instances, the success probability of the algorithm is upper bounded by the probability that at least one left-most node picks its least probable output, which, since the number of rows is upper bounded by $O(\sqrt{n})$, happens with probability at most $c/\sqrt{n}$ for some constant c. This probability, for large enough n, is strictly smaller than $1 - 1/n$, implying that the algorithm fails with too large probability. ☐

8.2 Complexity with shared randomness

Theorem 8.2. *In the* LOCAL *model with shared randomness, the problem* Π *can be solved in* $O(\log n)$ *rounds, with success probability* $1 - 1/n^c$ *for any constant* c.

Proof. By Lemma 6.6, nodes can spend $O(\log n)$ rounds to produce a solution for Π^{badGraph} where the graph induced by nodes that output $\perp$ satisfies that each connected component is a graph in $\mathcal{G}$. Observe that such labeling satisfies constraint 1 of $\mathcal{C}^{\Pi}$.

Nodes that have an output different from $\perp$ immediately terminate. Then, each node u does the following. Node u spends $O(\log \log n)$ rounds to check if the height of the grid structure is at most $c \log n$. In this case, the width of the grid is also guaranteed to be at most $c \log n$. This implies that nodes are in a small connected component that is a valid graph of the family. In this case, each node u can spend $O(\log n)$ rounds to know the bit b_v given as input to the right-most node v in the row of u. In this case, let $b_u = b_v$.

Otherwise, i.e., the height is strictly larger than $c \log n$, if u is a grid-node, it spends $O(\log n)$ rounds to compute its position i in its column. Then, u sets b_u as the ith shared random bit. Note that this implies that nodes in the same row pick the same random bit.

Then, if u is not in the last column, it outputs (b_u, yes), while if u is in the last column it outputs (b_u, x_u), where $x_u = \mathsf{yes}$ if x_u is equal to the bit given as input to u, and $x_u = \mathsf{no}$ otherwise. Finally, nodes in the tree-like structures output yes if at least one of their children has yes, and no otherwise.

By construction of the output, all constraints 2–6 are clearly satisfied. If the grid structure has height at most $c \log n$, then all nodes in the trees output yes and constraint 7 is also satisfied. If the grid structure has height strictly larger than $c \log n$, since for each right-most node v it holds that the randomly picked bit is the same as its input bit with probability $1/2$, constraint 7 is satisfied with probability at least $1 - 1/2^{c \log n} = 1 - 1/n^c$, as required. ☐

9 Complexity in other models

In this section we explore the complexity of problem Π in other models, beyond the usual LOCAL model.

9.1 SLOCAL model with private randomness

In the SLOCAL model [25], nodes are processed sequentially according to an ordering $\sigma = v_1, \dots, v_n$. The order σ is controlled by an adversary; that is, an SLOCAL algorithm must work for any such sequence σ. Let T be given as a function of n. When processing node v_i, an algorithm with time complexity T has access to the neighborhood at distance T of v_i, including whatever information the nodes there may have in their memory (and thus also their output). In the randomized version of the model (with private randomness), the algorithm is given access to a (read-once) infinite sequence of random bits.

Theorem 9.1. *In the* SLOCAL *model with private randomness, solving the problem* Π *requires time* $\Omega(\sqrt{n})$.

Proof. We consider the subfamily of graphs in $\mathcal{G}$ with grid dimensions w and h where $w = h$, and we apply Lemma 7.1.

Assume towards a contradiction that there exists an algorithm $\mathcal{A}$ that solves Π in $o(\sqrt{n})$ rounds. The difference compared to the proof of Theorem 8.1 is that we need to provide a suitable ordering under which the nodes are processed. Having done so, the proof remains the same.

Let N be such that, for any $n \geq N$, the time complexity T of $\mathcal{A}$ in graphs of size n is at most $w/3$. Consider the nodes in the left- and right-most columns of the grid. Observe that, if we select any pair of nodes in the two columns, their T-radius neighborhoods do not intersect.

Let us now fix the processing order such that all the nodes of the left-most column come first, followed by the nodes in the right-most column (from top to bottom), and finally by the remaining nodes in arbitrary order.

Consider a row where its left-most node is u and its right-most one v. By construction, the outputs of u and v must be independent. Let p_u (resp., p_v) be the probability that u (resp., v) outputs 0. Observe that $p_u(1 - p_v) < 1/n$ and $(1 - p_u)p_v < 1/n$; otherwise, the algorithm would produce a row that does not have the same bit everywhere (contradicting Lemma 7.1) with too large probability. This implies that either both p_u and p_v are at most $2/n$ or that they are both at least $1 - 2/n$.

We now restrict to instances where, for each row with left-most node u and right-most node v, the input of v is 0 if $p_u \leq 2/n$, and 1 otherwise. On these instances, the success probability of the algorithm is upper-bounded by the probability that at least one left-most node picks its least probable output. Since the number of rows is $O(\sqrt{n})$, by the union bound this happens with probability at most $c/\sqrt{n}$ for some constant c. For large enough n, this is strictly smaller than $1 - 1/n$, implying that the algorithm fails with too large probability. $\square$

9.2 Deterministic online-LOCAL model

The online-LOCAL model [3] is slightly more powerful than SLOCAL. Again we have an (adversarial) sequence $\sigma = v_1, \dots, v_n$ in which the nodes are processed. Upon node v_i being revealed, the algorithm is given knowledge of all nodes (including their inputs) and their connections in the T-radius neighborhood of v_i (in addition to everything the algorithm has seen so far).

Here we restrict ourselves to the deterministic variant of online-LOCAL.

Theorem 9.2. *In the deterministic online-*LOCAL *model, solving the problem* Π *requires* $\Omega(\sqrt{n})$ *rounds.*

Proof. As in the proof of Theorem 9.1, we consider the subfamily of graphs in $\mathcal{G}$ with grid dimensions w and h where $w = h$, and we apply Lemma 7.1. Assume towards a contradiction that there exists an algorithm $\mathcal{A}$ that solves Π in $o(\sqrt{n})$ rounds. Let N be such that, for any $n \geq N$, the time complexity of $\mathcal{A}$ in graphs of size n is at most $w/3$.

Consider the processing order where all nodes in the left-most column come first in the sequence, followed by all other nodes in arbitrary order. As the algorithm is deterministic, it must produce some output on each node, in particular without any knowledge of the input to the right-most column. Hence we can easily pick an adversarial input: For each node u on the left-most column, letting o_u be its output and v the corresponding right-most node in the same row as u, we set $1 - o_u$ as the input to v. Since the algorithm cannot modify its outputs, it does not produce a valid solution to Π. $\square$

9.3 Bounded-dependence model

The bounded-dependence model [2] (with locality T) encompasses any algorithm where the output distributions of nodes that are at distance more than T from one other are independent. Equivalently, given two graphs G_1 and G_2 (with inputs) with the same number of nodes, if any subgraph H_1 of G_1 is isomorphic to a subgraph H_2 of G_2, then the algorithm must have identical marginal distributions on H_1 and H_2.

Theorem 9.3. *In the bounded-dependence model, solving the problem Π requires locality $\Omega(\sqrt{n})$.*

Proof. The proof of Theorem 8.1 works directly here since, for each row, the output distributions of the left- and right-most nodes are independent (as their neighborhoods do not intersect). $\square$

Acknowledgements

This work was started in the *Research Workshop on Distributed Algorithms* (RW-DIST 2024) in L'Aquila, Italy. We would like to thank all workshop participants for discussions, and in particular Henrik Lievonen and Amirreza Akbari for helping us with this research project. This work has been partially funded by MUR (Italy) Department of Excellence 2023–2027, the PNRR MIUR research project GAMING "Graph Algorithms and MinINg for Green agents" (PE0000013, CUP D13C24000430001). Augusto Modanese is supported by the Helsinki Institute for Information Technology (HIIT).

References

[1] Jayadev Acharya, Clément L. Canonne, and Himanshu Tyagi. "Communication-Constrained Inference and the Role of Shared Randomness." In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 30–39.

[2] Amirreza Akbari, Xavier Coiteux-Roy, Francesco D'Amore, François Le Gall, Henrik Lievonen, Darya Melnyk, Augusto Modanese, Shreyas Pai, Marc-Olivier Renou, Václav Rozhon, and Jukka Suomela. "Online Locality Meets Distributed Quantum Computing." In: *CoRR* abs/2403.01903 (2024). DOI: 10.48550/ARXIV.2403.01903.

[3] Amirreza Akbari, Navid Eslami, Henrik Lievonen, Darya Melnyk, Joona Särkijärvi, and Jukka Suomela. “Locality in Online, Dynamic, Sequential, and Distributed Graph Algorithms.” In: *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany.* Ed. by Kousha Etessami, Uriel Feige, and Gabriele Puppis. Vol. 261. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 10:1–10:20. DOI: `10.4230/LIPICS.ICALP.2023.10`.

[4] Heger Arfaoui and Pierre Fraigniaud. “What can be computed without communications?” In: *SIGACT News* 45.3 (2014), pp. 82–104. DOI: `10.1145/2670418.2670440`.

[5] Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. “The Distributed Complexity of Locally Checkable Problems on Paths is Decidable.” In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019.* Ed. by Peter Robinson and Faith Ellen. ACM, 2019, pp. 262–271. DOI: `10.1145/3293611.3331606`.

[6] Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Jan Studený, and Jukka Suomela. “Efficient Classification of Locally Checkable Problems in Regular Trees.” In: *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*. Ed. by Christian Scheideler. Vol. 246. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 8:1–8:19. DOI: `10.4230/LIPICS.DISC.2022.8`.

[7] Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Jan Studený, Jukka Suomela, and Aleksandr Tereshchenko. “Locally checkable problems in rooted trees.” In: *Distributed Computing* 36.3 (2023), pp. 277–311. DOI: `10.1007/S00446-022-00435-9`.

[8] Alkida Balliu, Sebastian Brandt, Yuval Efron, Juho Hirvonen, Yannic Maus, Dennis Olivetti, and Jukka Suomela. “Classification of Distributed Binary Labeling Problems.” In: *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*. Ed. by Hagit Attiya. Vol. 179. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 17:1–17:17. DOI: `10.4230/LIPICS.DISC.2020.17`.

[9] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. “Lower Bounds for Maximal Matchings and Maximal Independent Sets.” In: *Journal of the ACM* 68.5 (2021), 39:1–39:30. DOI: `10.1145/3461458`.

[10] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. “Almost global problems in the LOCAL model.” In: *Distributed Computing* 34.4 (2021), pp. 259–281. DOI: `10.1007/S00446-020-00375-2`.

[11] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. “How much does randomness help with locally checkable problems?” In: *PODC ’20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020.* Ed. by Yuval Emek and Christian Cachin. ACM, 2020, pp. 299–308. DOI: `10.1145/3382734.3405715`.

[12] Alkida Balliu, Keren Censor-Hillel, Yannic Maus, Dennis Olivetti, and Jukka Suomela. “Locally Checkable Labelings with Small Messages.” In: *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*. Ed. by Seth Gilbert. Vol. 209. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 8:1–8:18. DOI: `10.4230/LIPICS.DISC.2021.8`.

[13] Alkida Balliu, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Dennis Olivetti, and Jukka Suomela. “New classes of distributed time complexity.” In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*. Ed. by Ilias Diakonikolas, David Kempe, and Monika Henzinger. ACM, 2018, pp. 1307–1318. DOI: 10.1145/3188745.3188860.

[14] Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. “A lower bound for the distributed Lovász local lemma.” In: *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*. Ed. by Daniel Wichs and Yishay Mansour. ACM, 2016, pp. 479–488. DOI: 10.1145/2897518.2897570.

[15] Sebastian Brandt, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Patric R. J. Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela, and Przemyslaw Uznanski. “LCL Problems on Grids.” In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*. Ed. by Elad Michael Schiller and Alexander A. Schwarzmann. ACM, 2017, pp. 101–110. DOI: 10.1145/3087801.3087833.

[16] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. “An Exponential Separation between Randomized and Deterministic Complexity in the LOCAL Model.” In: *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*. Ed. by Irit Dinur. IEEE Computer Society, 2016, pp. 615–624. DOI: 10.1109/FOCS.2016.72.

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

[18] Yi-Jun Chang, Jan Studený, and Jukka Suomela. “Distributed graph problems through an automata-theoretic lens.” In: *Theoretical Computer Science* 951 (2023), p. 113710. DOI: 10.1016/J.TCS.2023.113710.

[19] Xavier Coiteux-Roy, Francesco D’Amore, Rishikesh Gajjala, Fabian Kuhn, François Le Gall, Henrik Lievonen, Augusto Modanese, Marc-Olivier Renou, Gustav Schmid, and Jukka Suomela. “No Distributed Quantum Advantage for Approximate Graph Coloring.” In: *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*. Ed. by Bojan Mohar, Igor Shinkar, and Ryan O’Donnell. ACM, 2024, pp. 1901–1910. DOI: 10.1145/3618260.3649679.

[20] Pierluigi Crescenzi, Pierre Fraigniaud, and Ami Paz. “Trade-Offs in Distributed Interactive Proofs.” In: *33rd International Symposium on Distributed Computing, DISC 2019, October 14-18, 2019, Budapest, Hungary*. Ed. by Jukka Suomela. Vol. 146. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 13:1–13:17. DOI: 10.4230/LIPICS.DISC.2019.13.

[21] Manuela Fischer and Mohsen Ghaffari. “Sublogarithmic Distributed Algorithms for Lovász Local Lemma, and the Complexity Hierarchy.” In: *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*. Ed. by Andréa W. Richa. Vol. 91. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 18:1–18:16. DOI: 10.4230/LIPICS.DISC.2017.18.

[22] Cyril Gavoille, Adrian Kosowski, and Marcin Markiewicz. “What Can Be Observed Locally?” In: *Distributed Computing, 23rd International Symposium, DISC 2009, Elche, Spain, September 23-25, 2009. Proceedings*. Ed. by Idit Keidar. Vol. 5805. Lecture Notes in Computer Science. Springer, 2009, pp. 243–257. DOI: 10.1007/978-3-642-04355-0_26.

[23] Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. "On Derandomizing Local Distributed Algorithms." In: *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*. Ed. by Mikkel Thorup. IEEE Computer Society, 2018, pp. 662–673. DOI: 10.1109/FOCS.2018.00069.

[24] Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, Yannic Maus, Jukka Suomela, and Jara Uitto. "Improved distributed degree splitting and edge coloring." In: *Distributed Computing* 33.3-4 (2020), pp. 293–310. DOI: 10.1007/S00446-018-00346-8.

[25] Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. "On the complexity of local distributed graph problems." In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. Ed. by Hamed Hatami, Pierre McKenzie, and Valerie King. ACM, 2017, pp. 784–797. DOI: 10.1145/3055399.3055471.

[26] Mohsen Ghaffari and Hsin-Hao Su. "Distributed Degree Splitting, Edge Coloring, and Orientations." In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. Ed. by Philip N. Klein. SIAM, 2017, pp. 2505–2523. DOI: 10.1137/1.9781611974782.166.

[27] Alexander E. Holroyd. "Symmetrization for finitely dependent colouring." In: *CoRR* abs/2305.13980 (2023). DOI: 10.48550/arXiv.2305.13980.

[28] Alexander E. Holroyd, Tom Hutchcroft, and Avi Levy. "Finitely dependent cycle coloring." In: *Electronic Communications in Probability* 23 (2018). DOI: 10.1214/18-ecp118.

[29] Alexander E. Holroyd and Thomas M. Liggett. "Finitely Dependent Coloring." In: *Forum of Mathematics, Pi* 4, e9 (2016). DOI: 10.1017/fmp.2016.7.

[30] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. "A Model of Computation for MapReduce." In: *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*. Ed. by Moses Charikar. SIAM, 2010, pp. 938–948. DOI: 10.1137/1.9781611973075.76.

[31] Gowtham R. Kurri, Vinod M. Prabhakaran, and Anand D. Sarwate. "Coordination Through Shared Randomness." In: *IEEE Transactions on Information Theory* 67.8 (2021), pp. 4948–4974. DOI: 10.1109/TIT.2021.3091604.

[32] François Le Gall, Harumichi Nishimura, and Ansis Rosmanis. "Quantum Advantage for the LOCAL Model in Distributed Computing." In: *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*. Ed. by Rolf Niedermeier and Christophe Paul. Vol. 126. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 49:1–49:14. DOI: 10.4230/LIPICS.STACS.2019.49.

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

[34] Pedro Montealegre, Diego Ramírez-Romero, and Ivan Rapaport. "Shared vs Private Randomness in Distributed Interactive Proofs." In: *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*. Ed. by Yixin Cao, Siu-Wing Cheng, and Minming Li. Vol. 181. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 51:1–51:13. DOI: 10.4230/LIPICS.ISAAC.2020.51.

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

[36] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, 2000.

[37] Anup Rao and Amir Yehudayoff. *Communication Complexity and Applications*. Cambridge University Press, 2020. ISBN: 978-1-108-49798-5.

[38] Václav Rozhon and Mohsen Ghaffari. "Polylogarithmic-time deterministic network decomposition and distributed derandomization." In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*. Ed. by Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy. ACM, 2020, pp. 350–363. DOI: `10.1145/3357713.3384298`.

[39] Jukka Suomela. "Landscape of Locality (Invited Talk)." In: *17th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2020, June 22-24, 2020, Tórshavn, Faroe Islands*. Ed. by Susanne Albers. Vol. 162. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 2:1–2:1. DOI: `10.4230/LIPICS.SWAT.2020.2`.

[40] Aleksandr Tereshchenko. "Automated classification of distributed graph problems." Master's thesis. Aalto University, 2021. URL: `https://urn.fi/URN:NBN:fi:aalto-202105236941`.

[41] Ádám Timár. "Finitely dependent random colorings of bounded degree graphs." In: *CoRR* abs/2402.17068 (2024). DOI: `10.48550/arXiv.2402.17068`.

A Monte Carlo algorithms require some knowledge of n

In this appendix, we show that any truly (i.e., non-deterministic) Monte Carlo algorithm (regardless of whether it uses shared or private randomness) for any component-wise checkable problem requires some form of knowledge of the number of nodes n, regardless of its locality. Note component-wise checkable problems are a much broader class than LCLs.

Definition A.1 (Component-wise checkability)**.** A labeling problem Π is *component-wise checkable* if the following holds: For every labeled graph G and every connected component H of G, G satisfies Π if and only if its labeled subgraphs H and $G - H$ (seen as two separate graphs) both satisfy Π.

An example of a problem that does not qualify as such is non-component-wise leader election, that is, the labeled graph contains exactly one node marked as the leader. (Of course, component-wise leader election is a component-wise checkable problem.)

Definition A.2 (Monte Carlo algorithm)**.** A randomized distributed algorithm $\mathcal{A}$ is said to be a *Monte Carlo* algorithm if, for any constant $c > 0$, there is $n_0 > 0$ such that, if $\mathcal{A}$ is ran on a graph G with $n \geq n_0$ nodes, then the success probability of $\mathcal{A}$ is at least $1 - n^{-c}$. The algorithm $\mathcal{A}$ is *error-free* if its success probability is 1.

In particular, error-free algorithms can be trivially derandomized.

Theorem A.3. *Let Π be component-wise checkable, and let $\mathcal{A}$ be a Monte Carlo algorithm that solves Π (with any locality) and is given no knowledge of n whatsoever. Then $\mathcal{A}$ is error-free.*

Proof. Suppose there is a graph G for which $\mathcal{A}$ is not error-free, that is, $\mathcal{A}$ fails on G with probability $p > 0$. Let $c > 0$ be fixed, and let n_0 be as in Definition A.2. Consider the graph $G' = G \cup H$ where H is disconnected from G and contains $N > 1/p^{1/c}$ nodes. Since $\mathcal{A}$ is given no knowledge of the number of nodes of G', its distribution of outputs on G as a component of G' is identical to the distribution obtained when running $\mathcal{A}$ on G alone. (Failing on the component G might be correlated with failing on H, but this is immaterial.) In particular the failure probability of $\mathcal{A}$ on G' is at least $p > 1/N^c$, and thus its success probability is strictly smaller than $1 - N^{-c}$, contradicting Definition A.2. It follows that $p = 0$. □