Seeing Far vs. Seeing Wide: Volume Complexity of Local Graph Problems

Will Rosenbaum Max Planck Institute for Informatics will.rosenbaum@mpi-inf.mpg.de Jukka Suomela Aalto University jukka.suomela@aalto.fi

Abstract

Assume we have a graph problem that is *locally checkable but not locally solvable*—given a solution we can check that it is feasible by verifying all constant-radius neighborhoods, but to find a feasible solution each node needs to explore the input graph at least up to distance $\Omega(\log n)$ in order to produce its own part of the solution.

Such problems have been studied extensively in the recent years in the area of distributed computing, where the key complexity measure has been *distance*: how far does a node need to see in order to produce its own part of the solution. However, if we are interested in e.g. sublinear-time centralized algorithms, a much more appropriate complexity measure would be *volume*: how large a subgraph does a node need to see in order to produce its own part of the solution.

In this work we study locally checkable graph problems on bounded-degree graphs and we give a number of constructions that exhibit different tradeoffs between deterministic distance, randomized distance, deterministic volume, and randomized volume:

- If the deterministic distance is linear, it is also known that randomized distance is near-linear. We show that volume complexity is fundamentally different: there are problems with a linear deterministic volume but only logarithmic randomized volume.
- We prove a volume hierarchy theorem for randomized complexity: Among problems with (near) linear deterministic volume complexity, there are infinitely many distinct randomized volume complexity classes between $\Omega(\log n)$ and O(n). Moreover, this hierarchy persists even when restricting to problems whose randomized and deterministic distance complexities are $\Theta(\log n)$.
- Similar hierarchies exist for polynomial distance complexities: we show that for any $k, \ell \in \mathbf{N}$ with $k \leq \ell$, there are problems whose randomized and deterministic distance complexities are $\Theta(n^{1/\ell})$, randomized volume complexities are $\widetilde{\Theta}(n^{1/k})$, and whose deterministic volume complexities are $\widetilde{\Theta}(n)$.

Additionally, we consider connections between our volume model and massively parallel computation (MPC). We give a general simulation argument that any volume-efficient algorithm can be transformed into a space-efficient MPC algorithm.



Figure 1: Prior work: locally checkable graph problems (LCLs) classified based on their distance complexity [2, 3, 5, 9, 12, 13, 15, 20, 21, 23, 33, 37, 38, 44]. Blue dots represent examples of LCL problems with a known deterministic and randomized complexity, and the orange shading represents a region in which no LCL problems exist.

1 Introduction

Distance complexity. In message-passing models of distributed computing, *time* is intimately connected to *distance*: in T communication rounds, nodes can potentially learn some information that was originally within distance T from them, but not further. This idea is formalized in the *LOCAL model* [33, 41] of distributed computing, in which a distributed algorithm with a running time T is, in essence, a function that maps radius-T neighborhoods to local outputs. The key question in the theory of distributed computing can be stated as follows:

How far does an individual node need to see in order to produce its own part of the solution?

To give some simple examples, assume we have got a graph with n nodes and a maximum degree $\Delta = O(1)$, and all nodes are labeled with unique identifiers:

- Finding a proper vertex coloring with $\Delta + 1$ colors: Each node can pick its own color based on its radius- $O(\log^* n)$ neighborhood [15, 33, 37].
- Finding a proper vertex coloring with Δ colors: Each node needs to see up to distance $\Omega(\log \log n)$ in order to succeed with high probability and up to distance $\Omega(\log n)$ if we are using a deterministic algorithm [9, 13, 21, 39].

Graph coloring is an example of a *locally checkable labeling* (LCL) [38]. I.e., it is a graph problem in which we label nodes with labels from some finite set, and a solution is globally feasible if it looks feasible in a constant-radius neighborhood of each node. In the past several years our understanding of the distance complexity of LCLs has advanced rapidly [2–10, 12–14, 20, 21, 23, 24, 44], and it is now known that *all* LCL problems can be broadly classified in four classes, as shown in Figure 1. One of the key insights is that there are broad *gaps* between

the classes, and such gaps have immediate algorithmic applications: for example, if you can solve any LCL problem with $o(\log n)$ deterministic distance, it directly implies also a solution with $O(\log^* n)$ distance.

Volume complexity. While there has been a lot of progress on understanding how far each node must see in a graph to solve a given graph problem, this line of research has limited direct applicability beyond message-passing models of distributed computing. In many other settings—e.g., parallel algorithms and centralized sublinear-time algorithms—a key question is not how *far* do we need to explore the input graph, but how *many* nodes of the input graph we need to explore. One formalization of this idea is the (stateless) *local computation algorithms* (LCAs, a.k.a. centralized local algorithms or CentLOCAL) [45], where the key question is this:

How much of the input does an individual node need to see in order to produce its own part of the solution?

We will refer to this as the *volume* complexity of a graph problem. We will formalize the model of computing in Section 2, but in brief, the idea is this:

In time T each node can **adaptively** gather information about a connected component of size T around itself.

A bit more precisely, in each time step a node can choose to query any neighbor of a node that it has discovered previously. The query will reveal the unique identifier of the node, its degree, and its local input (if any). In randomized algorithms, each node has an independent stream of random bits that is part of its local input. Eventually, each node has to stop and produce its own part of the solution (e.g. its own color if we are solving graph coloring). While we assume that a node gathers a connected region, we point out that we can make this assumption without loss of generality for a broad range of graph problems [26].

Parnas and Ron [40] introduced a general framework that transforms algorithms in the LOCAL model to LCAs. In their framework, an algorithm with complexity f(n) yields an LCA with probe¹ complexity $\Delta^{\Theta(f(n))}$. Recently, Ghaffari and Uitto [22] asked if the $\Delta^{\Omega(f(n))}$ barrier inherent to Parnas and Ron's technique can be overcome by "sparsifying" the underlying LOCAL algorithm. They provide affirmative answers for several well-studied problems, such as maximal independent set, maximal matching, and approximating a minimum vertex cover. While there is a large body of work that introduces algorithms with a low volume complexity—see, e.g., [1, 11, 17–19, 22, 30–32, 34, 35, 40, 43, 45]—what is currently lacking is an understanding of the landscape of the volume complexity.

Connections to Massively Parallel Computation. Another motivation for studying volume complexity is its connection to massively parallel computation (MPC) frameworks, such as MapReduce [28]. In the MPC model, a system consists of M machines each with S local memory. An execution proceeds in synchronous rounds. In each round, each machine can communicate with all other machines—sending and receiving at most S bits in total—and perform arbitrary local computations. The goal is to perform a task while minimizing the space requirement S per machine as well as the number of communication rounds.

In the case where each machine represents a vertex in a network with maximum degree Δ , any algorithm with distance complexity T can be trivially simulated in the MPC model with space $S = \Delta^{O(T)}$ in T rounds. Using graph exponentiation [29], this runtime can be improved to $O(\log T)$ rounds. Recently, sparsification—i.e., exploiting volume efficient algorithms—has been applied to give strongly sub-linear space algorithms in the MPC model [22]. The volume model

¹The word "probe" is used in the LCA literature to refer to an atomic interaction with a data structure, whereas we use "query." The latter is standard terminology, e.g., in the literature on sublinear time graph algorithms and property testing.

we describe in Section 2 allows us to formalize a close connection between volume and the MPC model. Specifically, in Section 2.4 we show that any algorithm with volume complexity VOL can be simulated using space roughly $O(\text{VOL} + n^c)$ and O(VOL) rounds in the MPC model for any positive constant c. In some cases, the runtime can be improved to $O(\log \text{VOL})$.

1.1 Towards a Theory of Volume Complexity

In this work, we initiate the study of the volume complexity landscape of graph problems. As the study of LCL problems has proved instrumental in our understanding of distance complexity, we will follow the same idea here. Some of the key research questions include the following:

- What are possible deterministic and randomized volume complexities of LCL problems?
- Do we have the same four distinct classes of problems as what we saw in Figure 1, and similar gaps between the classes?
- For distance complexity, randomness is known to help exponentially for all problems of class C, while it is of limited use in class D and useless in classes A and B. Does a similar picture emerge for volume complexity?
- There are infinite families of distinct distance complexities in classes B and D (this is a distributed analogue of the time hierarchy theorem)—does it hold also for the volume complexity?
- How tightly can we connect the volume complexity of a problem with its computational complexity in other models of computing (e.g. time and message complexity in LOCAL and CONGEST models of distributed computing, and time complexity in various models of massively parallel computing)?

1.2 Preliminary Observations

Let us now make some preliminary observations on what we can say about the volume complexity of the four classes of LCL problems that are listed in Figure 1. We will summarize these results in Figure 2.

Class A. Volume complexity is at least as much as the distance complexity. In graphs of maximum degree $\Delta = O(1)$, volume complexity is at most exponential in distance complexity. A distance-*T* algorithm can be simulated if each node gathers a ball of volume $\Delta^{O(T)}$, and a volume-*T* algorithm can be simulated if each node gathers a ball of radius O(T). Hence it trivially follows that the following classes of LCL problems are equal:

- problems with distance complexity $\Theta(1)$,
- problems with volume complexity $\Theta(1)$.

Class B. Let us now look at the class of LCL problems that are solvable with distance between $\Omega(\log \log^* n)$ and $O(\log^* n)$. The trivial bounds for their volume complexity would be $\Omega(\log \log^* n)$ and $\Delta^{O(\log^* n)}$.

However, we can prove also a nontrivial upper bound. Any LCL problem in this class can be solved in two steps [13]:

- 1. find a distance-k coloring for a suitable constant k = O(1),
- 2. apply a constant-distance mapping to the colored graph.



Figure 2: Preliminary observations on the landscape of volume complexities. Blue dots are some examples of LCL problems for which the volume complexity is easy to establish.

It has already been known for decades that the first step can be solved in $O(\log^* n)$ distance[15]. However, recently Even et al. [17] introduced a graph coloring technique that makes it possible to solve the problem also in $O(\log^* n)$ volume. It follows that these classes of LCL problems are equal:

- problems with distance complexity between $\Omega(\log \log^* n)$ and $O(\log^* n)$,
- problems with volume complexity between $\Omega(\log \log^* n)$ and $O(\log^* n)$.

Moreover, the derandomization result by Chang et al. [13] can be used to show that randomness does not help in this region in either model (subject to some mild assumptions on the model of computing).

Classes C and D. Finally, we are left with the LCL problems that have deterministic distance between $\Omega(\log n)$ and O(n) and randomized distance between $\Omega(\log \log n)$ and O(n). Trivially, the volume complexity of any problem is bounded by O(n), and hence the following four classes of LCL problems are equal:

- problems with randomized distance complexity between $\Omega(\log \log n)$ and O(n),
- problems with deterministic distance complexity between $\Omega(\log n)$ and O(n),
- problems with randomized volume complexity between $\Omega(\log \log n)$ and O(n),
- problems with deterministic volume complexity between $\Omega(\log n)$ and O(n).

In the distance model, it is known that in this region randomness helps at most exponentially [13]. For example, if the randomized distance complexity is $O(\log \log n)$, then the deterministic distance complexity has to be $O(\log n)$. The same proof goes through verbatim for the volume model (under some technical assumptions on the model of computing), and hence we can conclude that e.g. randomized volume $O(\log \log n)$ implies deterministic volume $O(\log n)$.



Figure 3: An overview of our contributions. Each blue line represents one LCL problem; the left end of the line indicates the randomized and deterministic volume complexity, and the right end of the line indicates the randomized and deterministic distance complexity.

1.3 Our Contribution

While problems of classes A and B are well-understood both from the perspective of volume and distance, the volume complexity of problems in classes C and D is wide open—indeed, it is not even known if there are distinct classes C and D for volume complexity.

In this work, we start to chart problems of class D, i.e., "global" problems that require $\Omega(\log n)$ distance and hence also $\Omega(\log n)$ volume for both deterministic and randomized algorithms. This is a broad class of problems, with infinitely many distinct distance complexities [2, 3, 12].

We will show that in this region there are infinite families of LCL problems that exhibit different combinations of randomized volume, deterministic volume, randomized distance, and deterministic distance. The new complexities are summarized in Figure 3 and Table 1. We make the following observations:

- There are *infinitely many* LCLs with distinct randomized volume complexities between $\omega(\log n)$ and o(n).
- Randomness can help exponentially, even if the deterministic volume complexity is $\Theta(n)$. This is very different from distance complexities, in which, e.g. a linear deterministic distance implies near-linear randomized distance [23].
- There are LCL problems in which distance complexity *equals* randomized volume, and there are also LCL problems in which distance complexity is *logarithmic* in randomized volume. Hence distance and volume are genuinely distinct concepts in this region. Moreover, our constructions yield a volume hierarchy theorem for randomized algorithms: There are infinitely many distinct randomized volume complexity classes between $\Omega(\log n)$ and O(n), even when restricting attention to problems whose distance complexities are $\Theta(\log n)$.

2 Model and Preliminaries

We will now define the model of computing and the problem family that we study in this work. Here is a brief overview for a reader familiar with the LOCAL model [33, 41] of distributed computing and LCAs (local computation algorithms, a.k.a., centralized local algorithms) [17, 45]:

• Deterministic distance = round complexity in the deterministic LOCAL model.

Problem (section discussed)		R-DIST	D-DIST	R-VOL	D-VOL
LeafColoring	$(\S3)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(n)$
BalancedTree	$(\S 4)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(n)$	$\Theta(n)$
Hierarchical-THC(k)	$(\S5)$	$\Theta(n^{1/k})$	$\Theta(n^{1/k})$	$\widetilde{\Theta}(n^{1/k})$	$\widetilde{\Theta}(n)$
$Hybrid\operatorname{-THC}(k)$	$(\S 6)$	$\Theta(\log n)$	$\Theta(\log n)$	$\widetilde{\Theta}(n^{1/k})$	$\widetilde{\Theta}(n)$
$HH extsf{-}THC(k,\ell)$	$(\S 6.1)$	$\Theta(n^{1/\ell})$	$\Theta(n^{1/\ell})$	$\widetilde{\Theta}(n^{1/k})$	$\widetilde{\Theta}(n)$

Table 1: The new LCL problems constructed in this work. Here k and ℓ are natural numbers, $k \leq \ell$. We use Θ to suppress factors that are poly-logarithmic in the argument.

- Randomized distance = round complexity in the randomized LOCAL model (like deterministic distance, but each node has a private random string).
- Deterministic volume \approx probe complexity in the stateless deterministic LCA model.
- Randomized volume = like deterministic volume, but each node has a private random string.

Our goal here is to have a clean model that is as close to the standard LOCAL model as possible, but which captures the idea of paying for the volume that the algorithm explores. The deterministic volume model is very close to stateless deterministic LCAs—we restrict queries to a connected region, but for many graph problems this assumption does not matter [26]. However, the randomized volume model is somewhat different from randomized LCAs; one key difference is that randomized LCAs typically have direct access to *shared* randomness, while in our model each node has a *private* random string. That said, low randomized volume clearly implies that there exists also an efficient randomized LCA for solving the problem. We will discuss different flavors of randomness in more detail in Section 7.6.

2.1 Graphs

Our main object of study in this paper is distributed graph algorithms. In this context, an undirected graph G = (V, E) represents both a communication network and the (partial) input to a problem. We denote the number of nodes in G by n = |V|. For each node $v \in V$, we denote its degree by deg(v), and we assume that for some fixed constant $\Delta \in \mathbf{N}$, all nodes have degree at most Δ . In any input, we assume that each node $v \in V$ is given a unique identifier from the range $[n^{\alpha}]$ for some arbitrary fixed $\alpha \geq 1$. For any positive integer d and node $v \in V$, $N_v(d)$ denotes the d-radius neighborhood of v. That is, $N_v(d)$ is the induced subgraph of G containing all nodes $w \in V$ with dist $(v, w) \leq d$.

While we consider undirected graphs—where each edge serves as a bi-directional communication link—it is convenient to view each edge $\{v, w\} \in E$ as a pair of *ordered* edges (v, w)(from v to w) and (w, v) (from w to v). We assume that input graphs additionally specify a **port ordering**. For each vertex v and incident edge (v, w), there is an associated number $p(v, w) \in [\deg(v)]$ —the **port number** of (v, w)—such that p is a bijection between (ordered) edges incident to v and $[\deg(v)]$. Thus, on any input, we may speak unambiguously of v's ith neighbor, as the neighbor w satisfying p(v, w) = i (if any).

The input to a graph problem may additionally specify an input string for each node $v \in V$. An *input labeling* \mathcal{L} of a graph G specifies $O(\log n)$ -bit unique identifiers for each node, a port ordering, and any additional input required for the graph problem. We denote the input label of a particular node v by $\mathcal{L}(v)$. We also assume that n—the number of nodes in the graph—is provided as input to every algorithm.

2.2 Algorithms and Complexity

Each node $v \in V$ represents a single processor. Throughout an execution of an algorithm A initiated at a vertex $v \in V$, A maintains a set V_v of **visited nodes**, initialized to $V_v = \{v\}$. An execution proceeds in discrete **steps**, where in each step, A performs a single local **query** of the form query(w, j) where $w \in V_v$ and $j \in [\deg(w)]$ is a port number. In response, A receives

- the identity of the vertex u satisfying p(w, u) = j,
- the degree $\deg(u)$, and
- the entire input of u.

Additionally, v updates $V_v \leftarrow V_v \cup \{u\}$. Following the response to a query, A updates its local state, and determines its next query, or decides to produce output and halt. Given a graph G = (V, E), labeling \mathcal{L} of G, and vertex $v \in V$, we denote the output of A on (G, \mathcal{L}) initiated at v by $A(v, G, \mathcal{L})$. The set of outputs of A induces a new labeling \mathcal{L}' , where $\mathcal{L}'(v) = A(v, G, \mathcal{L})$.

We consider both deterministic and randomized algorithms. For randomized algorithms, random bits used by the algorithm are treated as part of the input at each node. Specifically, each node $v \in V$ has a random string $r_v : \mathbf{N} \to \{0, 1\}$, where each bit $r_v(i)$ is an i.i.d. 0–1 random variable with $\Pr(r_v(i) = 1) = \Pr(r_v(i) = 0) = 1/2$. Since we treat r_v as part of v's input, r_v is seen by every node that queries in v. For technical reasons, we assume that algorithms access the random strings r_v sequentially, and that for any algorithm A and any labeled graph (G, \mathcal{L}) there exists some finite bound b (which may depend on the input) such that with probability 1 - O(1/n) the execution of algorithm A on (G, \mathcal{L}) accesses at most b random bits.²

We are primarily interested in two complexity measures: distance and volume.

Definition 2.1. Let A be an algorithm, G = (V, E) a graph, \mathcal{L} a labeling of G, and $v \in V$ a node. Then the *distance cost* of A on (G, \mathcal{L}) initiated from v is

$$DIST(A, G, \mathcal{L}, v) = \max \left\{ dist(v, w) \, | \, w \in V_v \right\},\$$

where V_v is the set of nodes visited by the execution when A terminates. Let \mathcal{G}_n denote the family of labeled graphs on at most n nodes with maximum degree at most Δ . The distance cost of A on graphs of n nodes is defined by

$$DIST_n(A) = \sup \{ DIST(A, G, \mathcal{L}, v) \mid (G, \mathcal{L}) \in \mathcal{G}_n, G = (V, E), v \in V \}.$$

Definition 2.2. Let A be an algorithm, G = (V, E) a graph, \mathcal{L} a labeling of G, and $v \in V$ a node. Then the **volume cost** of A on (G, \mathcal{L}) initiated from v is

$$\operatorname{VOL}(A, G, \mathcal{L}, v) = |V_v|,$$

where V_v is the set of nodes visited by the execution when A terminates. Let \mathcal{G}_n denote the family of labeled graphs on at most n nodes. The volume cost of A on graphs of n nodes is defined by

$$\operatorname{VOL}_n(A) = \sup \{ \operatorname{VOL}(A, G, \mathcal{L}, v) \mid (G, \mathcal{L}) \in \mathcal{G}_n, G = (V, E), v \in V \}.$$

Remark 2.3. The distance cost of an algorithm in our model is closely related to the well-known LOCAL model of computation [33, 41]. In the LOCAL model, in *T* rounds each node can query all of its nodes within distance *T*. Thus, on input (G, \mathcal{L}) , an algorithm *A* can be implemented in *T* rounds in the LOCAL model if and only if it $DIST(A, G, \mathcal{L}, v) \leq T$ for all $v \in V$.

²With this assumption the derandomization result by Chang et al. [13, Theorem 3] holds also in the volume model. This seems to be a very mild assumption, and it should be automatically satisfied for most "natural" models of computation, e.g., probabilistic Turing machines. However, in standard message passing models no computational assumptions are made about individual processors. We suspect that for LCL problems, our restriction on how randomness is used is essentially without loss of generality. See the discussion in Section 7.6.

Definition 2.4. Let Π be a graph problem—that is, a family of triples $(G, \mathcal{I}, \mathcal{O})$, where \mathcal{I} and \mathcal{O} are input and output labelings (respectively) of G. We say that a *deterministic* algorithm A solves Π if for every allowable input \mathcal{I} the output $\mathcal{O} = (\mathcal{L}', G)$ formed by taking $\mathcal{L}'(v) = A(v, G, \mathcal{L})$ satisfies $(G, \mathcal{I}, \mathcal{O}) \in \Pi$. A randomized algorithm A solves Π if for all inputs \mathcal{I}

$$\Pr((G, \mathcal{I}, \mathcal{O}) \in \Pi) = 1 - O(1/n)$$

where the probability is taken over the (joint) randomness of all nodes, and n is the number of nodes in G.

Given a problem Π , the *complexity* of the problem Π is the infimum over all algorithms A computing Π of the cost of A. We denote the deterministic distance, randomized distance, deterministic volume, and randomized volume complexities of Π by

$$D$$
-DIST (Π) , R -DIST (Π) , D -VOL (Π) , R -VOL (Π)

respectively.

2.3 Comparing Distance and Volume

Here we give an elementary relationship between distance and volume complexities.

Lemma 2.5. Let Π be a problem defined on the family of graphs of maximum degree at most Δ . Then we have

$$\text{R-DIST}(\Pi) \le \text{R-VOL}(\Pi) \le \Delta^{\text{R-DIST}(\Pi)} + 1$$
(1)

and

$$D-DIST(\Pi) \le D-VOL(\Pi) \le \Delta^{D-DIST(\Pi)} + 1.$$
 (2)

Proof. For the first inequalities in Equations (1) and (2), suppose A is an algorithm that solves Π on G = (V, E) with labeling \mathcal{L} using volume m. For any $v \in V$, let $V_v \subseteq V$ denote the subset of nodes queried by an execution of A initiated from v, so that $|V_v| \leq m$. Since the subgraph of G induced by V_v is connected, we have dist $(v, w) \leq m$ for all $w \in V_v$, hence DIST $(A, G, \mathcal{L}, v) \leq m$.

For the second inequalities, suppose A solves Π using distance at most m, and let $N_v(m)$ denote the *m*-neighborhood of (v) (i.e., $N_v(m) = \{w \in V | \operatorname{dist}(v, w) \leq m\}$). Since G has maximum degree at most Δ , we have $|N_v(m)| \leq \Delta^m + 1$. Since $V_v \subseteq N_v(m)$, we have $\operatorname{VOL}(A, G, \mathcal{L}, v) = |V_v| \leq \Delta^m + 1$, which gives the desired result. \Box

2.4 Comparing Volume and MPC

In the MPC model [28], there are M machines each with S memory. An execution proceeds in synchronous rounds of all-to-all communication, with each node sending and receiving at most S bits per round. For simplicity, consider the case where each machine stores the adjacency list of a single vertex in G so that M = n (and $S \gg \Delta$). Here we describe how an arbitrary algorithm with volume cost VOL can be simulated efficiently in the MPC model.

Lemma 2.6. Suppose an algorithm A for Π has volume cost VOL when executed on a (labeled) graph G. Then for any number c > 0, there exists a (randomized) algorithm in the MPC model solving Π in O(VOL) rounds using space $S = O(\text{VOL} + n^c + \Delta)$ per node.

We only sketch the proof of Lemma 2.6. We show that for $S = O(n^c + \Delta)$, we can simulate each node $v \in V$ performing a single step (i.e., query and response) of A in O(1) rounds. The lemma follows by having each machine store the component queried by the vertex it represents in the simulation of A. Without loss of generality, assume the machines are labeled $1, 2, \ldots, n$. Each query is of the form (v, w, i), interpreted as, "node v queries for the i^{th} neighbor of node w." We refer to v as the **source** of the query, and w the **destination**. The difficulty of the simulation arises because a single node w could be the destination of many queries from different sources. Thus, queries cannot simply be sent directly from source to destination. However, all queries with a single destination must have at most Δ unique responses, corresponding to the (at most) Δ neighbors of the destination. The crux of our argument is showing how to identify the set of unique queries, and route the responses back to their sources in O(1) rounds.

The basic idea is the following:

- 1. Sort the set of queries made in a single step $(v_1, w_1, i_1), (v_2, w_2, i_2), \ldots, (v_n, w_n, i_n)$ with $w_j \leq w_{j+1}$ for all j, breaking ties first by i_j , then by v_j . This sorting can be performed in O(1) rounds using memory $O(n^c)$ per machine (with high probability) by applying an algorithm of Goodrich, Sitchinava, and Zhang [25]. After sorting, all queries of the form (\cdot, w_i, i_j) will be stored in consecutive machines.
- 2. If $(w_j, i_j) \neq (w_{j+1}, i_{j+1})$, machine j sends its query to (the machine hosting) w_j , and w_j sends its response to machine j in the following round. Since only a single request of the form (\cdot, w_j, i_j) is sent to w_j, w_j receives/sends at most Δ requests in total.
- 3. Machines receiving responses propagate the responses backwards (to smaller j's) in O(1) rounds. Specifically, let t be the round in which a machine j receives the response to (v_j, w_j, i_j) directly from w_j . Then in round t + 1, j sends the response to nodes $j 1, j 2, \ldots, j n^c$. In round t + 2, each machine j' with $j n^c \leq j' \leq j$ and $(w_{j'}, i_{j'}) = (w_j, i_j)$ sends the responses to nodes $j' n^c, j' 2n^c, \ldots, j' n^{2c}$. This continues for O(1/c) = O(1) rounds, at which point every machine j stores the response to the query (v_j, w_j, i_j) . During each of these rounds, each node sends and receives at most n^c messages.
- 4. Machine j sends the response to the query (v_j, w_j, i_j) to v_j . Each node sends and receives a single message.

Lemma 2.6 follows directly from analyzing the simulation described above. We note that the same argument also applies to simulating LCAs in the MPC model. We discuss further connections between the volume model and MPC in Section 7.

2.5 LCLs

In this paper, we are primarily interested in the study of locally checkable labeling problems (LCLs) [38]. Suppose $\Pi = \{(G, \mathcal{I}, \mathcal{O})\}$ is a graph problem such that the sets of possible input and output labels are finite. Informally, Π is an LCL if a global output \mathcal{O} is valid if and only if \mathcal{O} is valid on a bounded radius neighborhood of every node in the network. Since we consider families of graphs such that maximum degree Δ is bounded, every LCL has a finite description: it is enough to enumerate every possible input labeling of every *c*-radius neighborhood of a node, together with the list of valid output labelings for each input-labeled neighborhood. Familiar examples of LCLs include *k*-coloring (for fixed *k*), maximal independent set, and maximal matching.

Definition 2.7. Fix a positive integer Δ and let \mathcal{G}^{Δ} denote the family of graphs with maximum degree at most Δ . Let L_{in} and L_{out} be finite sets of input and output labels, respectively. Suppose

$$\Pi \subseteq \left\{ (G, \mathcal{I}, \mathcal{O}) \mid G = (V, E) \in \mathcal{G}^{\Delta}, \mathcal{I} : V \to L_{\text{in}}, \mathcal{O} : V \to L_{\text{out}} \right\}$$

is a graph problem. We call Π a *locally checkable labeling problem* or *LCL* if there exists an absolute constant c such that $(G, \mathcal{I}, \mathcal{O}) \in \Pi$ if and only if for every $v \in V$,

$$(N_v(c), \mathcal{I}|_{N_v(c)}, \mathcal{O}|_{N_v(c)}) \in \Pi.$$

Here $N_v(c)$ denotes the distance c neighborhood of v, and for a subgraph H of G, $\mathcal{I}|_H$ and $\mathcal{O}|_H$ denote the restrictions of \mathcal{I} and \mathcal{O} (respectively) to H.

2.6 Lower Bounds via Communication Complexity

Here, we briefly review a technique (introduced in [16]) of applying lower bounds from communication complexity to yield query lower bounds. The basic concepts for the technique are the notions of embedding of a function and query cost.

Definition 2.8. For $N \in \mathbf{N}$, let $f : \{0,1\}^N \times \{0,1\}^N \to \{0,1\}$ be a Boolean function. Let \mathcal{G}_n denote the set of (labeled) graphs on n vertices. Suppose $\mathcal{E} : \{0,1\}^N \times \{0,1\}^N \to \mathcal{G}_n$, and let $g : \mathcal{G}_n \to \{0,1\}$. We say that the pair (\mathcal{E},g) is an *embedding* of f if for all $x, y \in \{0,1\}^N$, $f(x,y) = g(\mathcal{E}(x,y))$.

Suppose two parties, Alice and Bob, hold private inputs x and y respectively, and wish to compute f(x, y). Given an embedding (\mathcal{E}, g) of f as above, any algorithm A that computes g on \mathcal{G}_n gives rise to a two-party communication protocol that Alice and Bob can use to compute f. Alice an Bob individually simulate an execution of A on $\mathcal{E}(x, y)$. Whenever A queries $\mathcal{E}(x, y)$, Alice and Bob exchange sufficient information about their private inputs x and y to simulate the response to A's query to $\mathcal{E}(x, y)$. If the responses to all such queries can be computed by Alice and Bob with little communication, then we may infer a lower bound on the number of queries needed by A to compute g. Indeed, the number of queries needed to compute g is at least the communication complexity of f divided by the maximum number of bits Alice and Bob must exchange in order to answer a query.

Definition 2.9. Let $q : \mathcal{G}_n \to \{0,1\}^*$ be a query and (\mathcal{E},g) an embedding of f. We say that q has *communication cost* at most B and write $\text{cost}_{\mathcal{E}}(q) \leq B$ if there exists a (zero-error) two-party communication protocol Π_q such that for all $x, y \in \{0,1\}^N$ we have $\Pi_q(x,y) = q(\mathcal{E}(x,y))$ and $|\Pi_q(x,y)| \leq B$.

The main result of [16] shows that given an embedding (\mathcal{E}, g) of f, the query complexity of g is bounded from below by the communication complexity of f divided by the communication cost of simulating each query.

Theorem 2.10 ([16]). Let Q be a set of allowable queries, $f : \{0,1\}^N \times \{0,1\}^N \to \{0,1\}$, and (\mathcal{E},g) an embedding of f. Suppose that each query $q \in Q$ satisfies $\operatorname{cost}_{\mathcal{E}}(q) \leq B$, and A is an algorithm that computes g using T queries (in expectation) from Q. Then $T = \Omega(R(f)/B)$, where R(f) is the (randomized) communication complexity of f.

In Section 4, we apply Theorem 2.10 using an embedding of the disjointness function, disj, defined as follows:

$$\operatorname{disj}(x,y) = \begin{cases} 1 & \text{if } \sum_{i=1}^{N} x_i y_i = 0\\ 0 & \text{otherwise.} \end{cases}$$

We apply the following fundamental result of Kalyanasundaram and Schnitger on the communication complexity of disj.

Theorem 2.11 ([27, 42]). The randomized communication complexity of the disjointness function is $R(\text{disj}) = \Omega(N)$. This result holds even if x and y are promised to satisfy $\sum_{i=1}^{N} x_i y_i \in \{0, 1\}$.

2.7 Tail Bounds

In our analysis of randomized algorithms, we will employ the following standard Chernoff bounds. See, e.g., [36] for derivations.

Lemma 2.12 ([36], Theorems 4.4 and 4.5). Suppose Y_1, Y_2, \ldots, Y_m are independent random variables with $\Pr(Y_i = 1) = p_i$ and $\Pr(Y_i = 0) = 1 - p_i$. Let $Y = \sum_{i=1}^m$ and $\mu = \mathbb{E}(Y) = \sum_{i=1}^m p_i$. Then for any δ with $0 < \delta < 1$ we have

$$\Pr(Y \ge (1+\delta)\mu) \le e^{-\mu\delta^2/3} \tag{3}$$

and

$$\Pr(Y \le (1 - \delta)\mu) \le e^{-\mu\delta^2/2}.$$
(4)

We will also require tail bounds for the *negative binomial distribution*, defined as follows. For any positive integer k and $p \in (0, 1]$, let $Y_1, Y_2, ...$ be a sequence of independent Bernoulli random variables with parameter p (i.e., $\Pr(Y_i = 1) = p$ and $\Pr(Y_i = 0) = 1 - p$ for all i). Then the random variable

$$N = \inf\left\{m \left|\sum_{i=1}^{m} Y_i \ge k\right.\right\}$$

is distributed according to the *negative binomial distribution* $\mathcal{N}(k, p)$. (For completeness, we use the convention that $\inf \emptyset = 0$.)

Notice that for $N \sim \mathcal{N}(k, p)$, we have

$$\Pr(N > m) = \Pr\left(\sum_{i=1}^{m} Y_i < k\right).$$

Setting $m = c \cdot k/p$ for any c > 1, the sum on the right has expected value $\mu = c \cdot p \cdot m = c \cdot k$. Taking $Y = \sum_{i=1}^{m} Y_i$, we then obtain N > m if and only if $Y < (1 - \delta)\mu$ for $\delta = (c - 1)/c$. Applying the Chernoff bound (4) to bound the right side of the expression above gives the following result.

Lemma 2.13. Suppose $N \sim \mathcal{N}(k, p)$. Then

$$\Pr(N > c \cdot k/p) \le e^{-k(c-1)^2/2c}$$

3 Leaf Coloring

In this section, we describe an LCL problem, LeafColoring, whose randomized distance, deterministic distance, and randomized volume complexities are $O(\log n)$, but whose deterministic volume complexity is $\Omega(n)$.

Before defining LeafColoring formally, we describe a "promise" version of the problem that restricts the possible input graphs. Specifically, consider the promise that all input graphs G = (V, E) are binary trees in which every node has either 0 or 2 children, and all edges are directed from parent to child. Moreover, each internal node (i.e., node with 2 children) has a pre-specified right and left child. Each node $v \in V$ is assigned an input color $\chi_{in}(v) \in \{R, B\}$ (R for red, B for blue). The LeafColoring problem requires each node v to output a color $\chi_{out}(v) \in \{R, B\}$ such that (1) if v is a leaf, $\chi_{out}(v) = \chi_{in}(v)$, and (2) if v is internal, it outputs the same color as one of its children.

In the non-promise version of LeafColoring, the input may be an arbitrary graph (with maximum degree at most Δ). In order to mimic the promise problem described above, each node receives as input a "tree labeling" (defined below) that assigns a parent, right child, and left child to each node. Using this assignment, each node can locally check that its own and its neighbors input labelings are locally consistent with a binary tree structure as described in the promise version of the problem. We observe (Observation 3.8) that the set of locally consistent nodes and edges form a binary sub-pseudo-forest of G (i.e., a subgraph in which each connected component contains at most a single cycle). The generic LeafColoring problem then requires each leaf in this pseudo-forest to output its input color, while each internal node outputs the same color as one of its children.

Definition 3.1. Let G = (V, E) be a graph of maximum degree at most Δ , and $\mathcal{P} = [\Delta] \cup \{\bot\}$. A *(binary) tree labeling* consists of the following for each $v \in V$:

• a *parent*, $P(v) \in \mathcal{P}$,

- a *left child*, $LC(v) \in \mathcal{P}$,
- a *right child*, $\operatorname{RC}(v) \in \mathcal{P}$.

A colored tree labeling additionally specifies for each $v \in V$

• a *color*, $\chi_{in}(v) \in \{R, B\}$.

We refer to R as **red** and B as **blue** in our depictions of colored tree labelings. For a fixed node v, we call the labeling of v **well-formed** if the non- \perp ports P(v), LC(v), and RC(v) are pair-wise distinct. For example, we have $P(v) \neq LC(v)$, unless both are \perp .

Notation 3.2. While the labels P, LC, and RC are formally elements of \mathcal{P} , it will be convenient to associate, for example, P(v) with the node adjacent to v via the edge whose port label is P(v). In particular, this convention allows us to compose labels; for example, P(LC(v)) is the parent of v's left child.

Remark 3.3. In what follows, we assume without loss of generality that in all tree labelings, the labels of all nodes are well-formed in the sense of Definition 3.1. Indeed, an arbitrary labeling \mathcal{L} can be transformed to a well-formed instance in the following manner: If v is not well-formed, it sets $P(v), LC(v), RC(v) = \bot$; if v is well-formed, but, e.g., P(v) is not, then v sets $P(v) = \bot$. This preprocessing can be performed using $\Delta = O(1)$ queries.

Definition 3.4. Let G = (V, E) be a graph and \mathcal{L} a well-formed tree labeling of G. We say that a node $v \in V$ is:

- *internal* if
 - 1. $LC(v) \neq \bot$ and P(LC(v)) = v,
 - 2. $\operatorname{RC}(v) \neq \bot$ and $\operatorname{P}(\operatorname{RC}(v)) = v$;
- a *leaf* if
 - 1. $LC(v) = RC(v) = \bot$
 - 2. P(v) is internal.

A node is **consistent** if it is internal or a leaf. A node that is neither internal nor a leaf is **inconsistent**.

Definition 3.5. The problem LeafColoring consists of the following:

Input: a colored tree labeling \mathcal{L}

Output: for each $v \in V$, a color $\chi_{out} \in \{R, B\}$

Validity: for each $v \in V$ we have

- $\chi_{\text{out}}(v) = \chi_{\text{in}}(v)$ if v is a leaf or inconsistent,
- $\chi_{\text{out}}(v) \in \{\chi_{\text{out}}(\text{LC}(v)), \chi_{\text{out}}(\text{RC}(v))\}$ if v is internal.

Lemma 3.6. LeafColoring is an LCL.

Proof. It follows directly from Definition 3.1 that a node being internal, leaf, or inconsistent is locally checkable. Thus, both validity conditions in Definition 3.5 are locally checkable as well. \Box



Figure 4: An input (left) and valid output (right) for an instance of LeafColoring. On the left, the consistent nodes have black borders, while the inconsistent nodes have gray borders. The subgraph consisting of black (directed) edges is the graph G_T described in Observation 3.8. The node colors on the left indicate input colors, while the colors on the right are output colors. For example, the input label of v is given by $\chi_{in}(v) = B$, P(v) = 4, RC(v) = 1, LC(v) = 3. In the output (right), black edges indicate that the parent and child in G_T output the same color. Validity of the output follows because all leaves and inconsistent nodes output their input color, while all internal nodes output the color of some child in G_T .

Theorem 3.7. The complexity of LeafColoring is

 $\begin{aligned} & \text{R-DIST}(\mathsf{LeafColoring}) = \Theta(\log n), \\ & \text{D-DIST}(\mathsf{LeafColoring}) = \Theta(\log n), \\ & \text{R-VOL}(\mathsf{LeafColoring}) = \Theta(\log n), \\ & \text{D-VOL}(\mathsf{LeafColoring}) = \Theta(n). \end{aligned}$

Before proving Theorem 3.7 in detail, we provide a high level overview of the proof. The upper bounds on R-DIST and D-DIST following from the observation that all internal nodes are within distance $O(\log n)$ of a leaf node. Thus, in $O(\log n)$ distance, each internal node finds its *nearest* leaf (breaking ties by choosing the left-most leaf at minimal distance) and outputs the color of that leaf. In Section 3.1, we show that this deterministic process correctly solves LeafColoring with distance complexity $O(\log n)$. This upper bound is tight, as in a balanced binary tree, the root has distance $\Omega(\log n)$ from its closest leaf. Thus, in order to distinguish the cases where all leaves are red vs blue, the root must query a node at distance $\Omega(\log n)$. (Note that if all leaves have input color, say, red, then the root must output red in any legal solution.) See Section 3.3 for details.

The idea of the R-VOL upper bound is that a "downward" random walk in a binary tree in which every internal node has two children will reach a leaf after $O(\log n)$ steps with high probability. To simulate such a random walk, each node in our volume-efficient algorithm chooses a single child at random. An execution of the algorithm from v follows the path of chosen children until a leaf is found, and v outputs the color of this child. Since all nodes along this path reach the same leaf, they all output the same color. The only complication that may arise is if v encounters a (necessarily unique) cycle, in which case the path of chosen children returns to v. In this case, v follows the edge to its child not chosen in the first step, and continues until a leaf is encountered. This second path is guaranteed to be cycle-free. Details are given in Section 3.2.

Finally, the argument for the lower bound on D-VOL is as follows. Given any deterministic algorithm A purporting to solve LeafColoring in using $q \ll n$ queries, we can adaptively construct a binary tree G rooted at v with $n \leq 3q$ such that the execution A initiated from v never queries a leaf of G. By giving each leaf the input color that is the opposite of v's output, we conclude that some node in G must output incorrectly in this instance. See Section 3.3 for details.

3.1 Distance Upper Bounds

Observation 3.8. Let G = (V, E) be a graph and \mathcal{L} a tree labeling of G. Define the directed graph $G_T = (V_T, E_T)$ by

 $V_T = \{ v \in V \mid v \text{ is internal or a leaf} \}$

and

$$E_T = \{(u, v) \in V_T \times V_T \mid u \text{ is internal and } u = P(v)\}$$

That is, G_T is the subgraph of internal nodes and leaves in G where we consider only edges directed from internal parents to children. Then every node in G_T has out-degree 0 or 2, and in-degree 0 or 1. In particular, this implies that G_T is a (directed) pseudo-forest, and each connected component of G_T contains at most one (directed) cycle. Moreover, all internal nodes have two descendants in G_T , and $v \in V$ is a leaf in the sense of Definition 3.1 if and only if v is a leaf in G_T .

Lemma 3.9. Let G = (V, E) and be a graph and \mathcal{L} a tree labeling of G. Suppose $v_0 \in V$ is an internal node. Then there exists a path $P = (v_0, v_1, \ldots, v_\ell)$ in G_T with $\ell \leq \log n$ such that v_ℓ is a leaf and for all $i \in [\ell], v_{i-1} = P(v_i)$.

Proof. Fix v_0 to be an internal node in V, and take G_T as in Observation 3.8. By Observation 3.8, v_0 has at least one child v_1 such that the (directed) edge (v_0, v_1) is not contained in any cycle in G_T . Thus, the set of descendants of v_1 forms a (directed) binary tree rooted at v_1 .

For each $r \in \mathbf{N}$, $r \geq 1$ define $B(r) \subseteq V_T$ to be the set of nodes containing v_0 and all descendants of v_1 (in G_T) up to distance r - 1 (from v_1). Observe that if B(r) contains only internal nodes, then

$$|B(r)| = 1 + \sum_{i=0}^{r-1} 2^i = 2^r.$$

In particular, if $r \ge \log n$, then B(r) must contain a non-internal node, w. By Observation 3.8, w is a leaf, which gives the desired result.

We are now ready to prove the claims of Theorem 3.7.

Proposition 3.10. Let G = (V, E) be a graph on n nodes, \mathcal{L} a colored tree labeling, and $v \in V$. Then there exists a deterministic algorithm A that solves LeafColoring on G with $\text{DIST}(A, G, \mathcal{L}, v) = O(\log n)$. Thus

$$D$$
-DIST(LeafColoring), R-DIST(LeafColoring) = $O(\log n)$.

Proof. The algorithm solving LeafColoring works as follows. In O(1) rounds, v determines if it is internal, a leaf, or inconsistent. If it is not internal, v outputs $\chi_{out}(v) = \chi_{in}(v)$. If v is internal, in $r = O(\log n)$ rounds, v queries its distance $\log n$ neighborhood, $N_v(r)$, and for each $w \in N_v(r)$, v determines if w is internal, leaf, or inconsistent. From this information, v computes

 $d = \min \{ d(v, u) \mid u \text{ is a descendant of } v \text{ and } u \text{ is a leaf} \}.$

For any leaf w that is a descendant of v at distance d, we associate the path

$$(v = w_d, w_{d-1}, \dots, w_1, w_0 = w)$$

from v to w with the sequence $P_w \in {\text{LC}, \text{RC}}^d$ where the i^{th} term in the sequence indicates if w_{d-i} is the left or right child of w_{d-i+1} . The node v then takes w_0 to be its "left-most" descendant leaf at distance d and outputs $\chi_{\text{out}}(v) = \chi_{\text{in}}(w_0)$. That is w_0 is v's distance d leaf descendant that minimizes the associated sequence P_{w_0} with respect to the lexicographic ordering. Let $(v = w_d, w_{d-1}, \ldots, w_1, w_0)$ denote the path from v to w_0 described above. We claim that for all $i = 0, 1, \ldots, d$, $\chi_{out}(w_i) = \chi_{in}(w_0)$. In particular, this implies that $\chi_{out}(v) = \chi_{out}(w_{d-1})$, hence the second validity condition in Definition 3.5 is satisfied. We prove the claim by induction on d.

For base case d = 0, w_0 is a leaf hence $\chi_{out}(w_0) = \chi_{in}(w_0)$ by the algorithm description. For the inductive step, suppose the lemma holds for all internal nodes having a descendant leaf at distance less than d. Suppose v is a node who's nearest descendant leaf is at distance d. Let w_0 be the left-most such leaf, and let $(v = w_d, w_{d-1}, \ldots, w_0)$ be the path from v to w_0 as above. Observe that w_{d-1} 's nearest descendant leaf is at distance d-1, and w_0 is also the left-most such leaf for w_{d-1} . Therefore, $\chi_{out}(w_{d-1}) = \chi_{in}(w_0)$, as required.

3.2 Randomized Volume Upper Bound

Proposition 3.11. Let G = (V, E) be a graph on *n* nodes. Then there exists a randomized algorithm that solves LeafColoring on *G* in $O(\log n)$ volume. Thus, R-VOL(LeafColoring) = $O(\log n)$.

To prove Proposition 3.11 consider the algorithm, RWtoLeaf (v, \perp) (Algorithm 1). If v is a leaf or inconsistent, it outputs $\chi_{in}(v)$. Otherwise, if v is internal, RWtoLeaf performs a (directed) random walk towards v's descendants in G_T . When the random walk is currently at a node w, w's private randomness is used to determine the next step of the random walk. This ensures that all walks visiting w choose the same next step of the walk, hence all such walks will reach the same leaf.

The only complication arises if G_T contains cycles (in which case, each connected component of G_T contains at most one cycle by Observation 3.8). In this case, the random walk may return to the initial node v_0 . If the walk returns to v_0 , the algorithm steps towards the previously unexplored child of v_0 . Since G_T contains at most one directed cycle, the branch below v_0 's second child is cycle-free, thus guaranteeing that the walk eventually reaches a leaf.

Remark 3.12. To simplify the presentation, we give an algorithm where the runtime (number of queries) is random, and may be linear in n. We will show that the runtime is $O(\log n)$ with high probability. In order to get a worst-case runtime of $O(\log n)$, an execution can be truncated after $O(\log n)$ steps—as n is known to each node—with the node producing arbitrary output.

Algorithm 1 RWtoLeaf (v, v_0) . Random walk step from v towards leaves. v_0 is the starting node of the walk, and v is the current location of the walk.

```
1: if v is a leaf or inconsistent then
2:
      return \chi_{in}(v)
3: end if
4: if v revisits v_0 then
      b \leftarrow 1 - r_v(0)
5:
6: else
      b \leftarrow r_v(0)
7:
8: end if
9: if b = 0 then
      return RWtoLeaf(LC(v), v_0)
10:
11: else
      return RWtoLeaf(RC(v), v_0)
12:
13: end if
```

Proof of Proposition 3.11. Consider the algorithm where each node $v \in V$ outputs $\chi_{out}(v) \leftarrow RWtoLeaf(v, v)$. If v is a leaf or inconsistent, then Line 2 ensures that the first validity condition of Definition 3.5 is satisfied.

Now consider the case where v is internal. Let $\pi_v = (v = v_0, v_1, v_2, ...)$ denote the sequence of nodes visited by the random walk in the invocation of RWtoLeaf(v, v). Thus π_v is a directed path in G_T . Suppose there exist indices i < j with $v_i = v_j$, so that π_v contains a cycle. By Observation 3.8, G_T contains at most one cycle C, and all non-cycle edges are directed away from C in G_T . Therefore, it must be the case that i = 0, so that the condition of Line 4 was satisfied when RWtoLeaf (v_j, v_0) was called. Thus, $v_{j+1} \neq v_1$, and v_{j+1} is not contained in any cycle in G_T . Accordingly, define

$$\pi'_{v} = \begin{cases} \pi_{v} & \text{if } \pi \text{ is cycle-free} \\ (v_{0}, v_{j+1}, v_{j+2}, \ldots) & \text{otherwise} \end{cases}$$

Since v_{j+1} is not contained in any cycle, π'_v is a finite sequence, and by the description of RWtoLeaf, π'_v terminates at a leaf v_ℓ . Let $w = \pi'_v(1)$ be the second node in the path π'_v . Then a straightforward induction argument (on ℓ) shows that $\pi'_v = v \circ \pi'_w$. In particular, π'_v and π'_w terminate at the same leaf v_ℓ so that $\chi_{out}(v) = \chi_{out}(w) = \chi_{in}(v_\ell)$. Thus the second validity condition in Definition 3.5 is satisfied, as desired.

It remains to bound the number of queries made by an invocation of RWtoLeaf. Since checking if a node is internal, a leaf, or inconsistent can be done with O(1) queries, each recursive call to RWtoLeaf can be performed with O(1) queries. Thus, the total number of queries used by RWtoLeaf (v, v_0) is $O(|\pi_v|)$, where $|\pi_v|$ denotes the length of π_v . We will show that with high probability for all $v \in V$, $|\pi_v| = O(\log n)$, whence the desired result follows.

First consider the case where an internal node v is not contained in any cycle. Let n_v denote the number of nodes reachable from v in G_T . Since v is not contained in any cycle, $n_v = 1 + n_u + n_w$ where u and w are v's (distinct) children. Therefore, n_u or n_w is at most $n_v/2$. For any edge (w, w') in the path π'_v , we call the edge good if $n_{w'} \leq n_w/2$. Since $n_v \leq n$, there cannot be more than $\log n$ good edges in π'_v . Moreover, by the selection in Line 7, each edge in π'_v is good independently with probability at least 1/2.

Claim. $\Pr(|\pi'_v| \ge 16 \log n) \le 1/n^3$

Proof of Claim. For i = 1, 2, ..., let Z_i be the indicator random variable for the event that the i^{th} step of the random walk crosses a good edge. The Z_i are independent, and we have $\Pr(Z_i = 1) \ge 1/2$ for all i. In fact, $\Pr(Z_i = 1) \in \{1/2, 1\}$. We define a coupled sequence of random variables Y_i as follows. For $i \le |\pi'_v|$ we set

$$Y_i = \begin{cases} Z_i & \text{if } \Pr(Z_i = 1) = 1/2\\ \text{independent Bernoulli r.v. with } p = 1/2 & \text{if } \Pr(Z_i = 1) = 1. \end{cases}$$

For $i > |\pi'_v|$, Y_i is an independent Bernoulli random variable with p = 1/2. Thus, the sequence Y_1, Y_2, \ldots is a sequence of independent Bernoulli random variables with p = 1/2 (coupled to the sequence Z_1, Z_2, \ldots).

Since every good edge (v, w) satisfies $n_w \leq \frac{1}{2}n_v$, π'_v can contain at most $\log n$ good edges. Therefore, we have

$$\sum_{i=1}^{|\pi'_v|} Y_i \le \sum_{i=1}^{|\pi'_v|} Z_i \le \log n,$$

where the first equality holds because $Y_i \leq Z_i$ (by construction) for $i \leq |\pi'_v|$. Now define the random variable N by

$$N = \inf \left\{ m \left| \sum_{i=1}^{m} Y_i \ge \log n \right\} \right\}.$$

Thus $N \sim \mathcal{N}(\log n, 1/2)$ has a negative binomial distribution. Thus, by Lemma 2.13, $\Pr(N > 16 \log n) \leq e^{-7^2(\log n)/2 \cdot 8} < n^{-3}$. The claim follows by observing that $|\pi'_v| > 16 \log n$ implies that $N > 16 \log n$.

Applying a simple union bound, the claim shows that all v not contained in some cycle in G_T will output after $O(\log n)$ queries with high probability. If v is contained in a cycle C, we consider two cases separately. If $|C| \leq 16 \log n$, then the random walk will leave the cycle C after at most $16 \log n$ steps (if it returns to the initial node). On the other hand, if $|C| > 16 \log n$, essentially the same argument as given in the proof of the claim shows that the random walk started at v will leave the cycle after at most $16 \log n$ steps with probability at least $1 - 1/n^3$. Combining these observations with the conclusion of the claim, we obtain that for all $v \in V$

$$\Pr(\left|\pi'_{v}\right| \ge 32\log n) \le 2/n^{3}.$$

Taking the union bound over all v, we find that all nodes output after $O(\log n)$ queries with probability at least $1 - O(1/n^2)$, which gives the desired result.

3.3 Lower Bounds

Proposition 3.13. There exists a graph G on n nodes and a probability distribution on colored tree labelings of G such that for any (randomized) algorithm A whose distance complexity is less than $\log n - 1$, the probability that A solves LeafColoring is at most 1/2.

Proof. Let G be a complete (rooted) binary tree of depth k, so that $n = 2^{k+1} - 1$. Consider the port ordering where the parent of each (non-root) node has port 1, and the children of each (non-leaf) node have ports 2 and 3. Suppose the node identities are 1 through n, where the root has ID 1, its left and right children are 2 and 3, and so on. Finally, fix \mathcal{L} to be the tree labeling where the root has LC(v) = 1, RC(v) = 2, all (non-root) nodes have P(v) = 1, and all internal, non-root nodes have LC(v) = 2, RC(v) = 3. That is, \mathcal{L} is the tree labeling consistent with the tree structure of G. Finally, consider the distribution D over input colorings where all internal nodes have $\chi_{in}(v) = R$, while all leaves have the same color χ_0 chosen to be R or B each with probability 1/2.

Since every leaf v in G has $\chi_{in}(v) = \chi_0$, the first validity condition of Definition 3.5 stipulates that $\chi_{out}(v) = \chi_0$. A simple induction argument on the height of a node (i.e., the distance from the node to a leaf) shows that the unique solution to LeafColoring is for all $v \in V$ to output $\chi_{out}(v) = \chi_0$.

Suppose A is any deterministic algorithm whose distance complexity is at most k - 1. Then an execution of A initiated at the root r of G will not query any leaf. Therefore, $\Pr_D(\chi_{out}(r) = \chi_0) = 1/2$. By the conclusion of the preceding paragraph, the probability that A solves LeafColoring is therefore at most 1/2. By Yao's minimax principle, the no randomized algorithm with distance complexity at most k - 1 solves LeafColoring with probability better than 1/2 as well.

Proposition 3.14. For any deterministic algorithm A, there exists a graph G = (V, E) on n nodes, a colored tree labeling \mathcal{L} on G such that if A uses fewer than n/3 queries, then A fails to solve LeafColoring. Thus, D-VOL(LeafColoring) = $\Omega(n)$.

Proof. Suppose A uses fewer than n/3 queries on all graphs G on n nodes. We define a process \mathcal{P} that interacts with A and constructs a graph G_A and a labeling \mathcal{L} such that A does not solve LeafColoring on G_A . The basic idea is that \mathcal{P} constructs a binary tree G_A such that A never queries a leaf of the tree. The leaves of G_A are then given input colors that disagree with A's output.

 \mathcal{P} constructs a sequence of labeled binary trees G_0, G_1, G_2, \ldots where G_t is the tree constructed after A's t^{th} query. Initially G_0 is the graph consisting of a single node v_0 with ID 0 and two ports,

1 and 2. The label of v_0 is $P(v_0) = \bot$, $LC(v_0) = 1$, $RC(v_0) = 2$. Suppose \mathcal{P} has constructed G_{t-1} , and A's t^{th} query asks for the neighbor of v from port $i \in [3]$. If i = 1, \mathcal{P} returns v's parent, and we set $G_t = G_{t-1}$. If i = 2 or 3, \mathcal{P} forms G_t by adding a node w to G_{t-1} together with an edge $\{v, w\}$. The ordered edge (w, v) gets assigned port 1, while w has two "unassigned" ports 2 and 3. The label of w is P(w) = 1, LC(w) = 2, RC(w) = 3, and w's input color is $\chi_{\text{in}}(w) = R$.

It is straightforward to verify (by induction on t) that at each step, G_t is a subgraph of a binary tree G'_t on at most 3t nodes: take G'_t to be the tree formed by appending a leaf to each unassigned port in G_t . If A halts after T queries and outputs $\chi_{out}(v_0) = C$, define $G_A = G'_T$, and complete the tree labeling of G_A by assigning P(w) = 1, $LC(w) = \bot$, $RC(w) = \bot$ for all new leaves. Finally, for each leaf w in G_A , set $\chi_{in}(w) = \chi_1 \neq \chi_0$ (the color not output by v_0). Since all leaves have $\chi_{in}(w) = \chi_1$, validity of LeafColoring requires that all nodes in G_A output χ_1 . However, $\chi_{out}(v_0) = \chi_0 \neq \chi_1$, so that A does not solve LeafColoring on G_A .

4 Balanced Tree Labeling

Here, we introduce an LCL called BalancedTree. The input labeling, which we call a "balanced tree labeling" extends a tree labeling (Definition 3.1) by additionally specifying "lateral edges" between nodes. We define a locally checkable notion of "compatibility" (formalized in Definition 4.2) such that the subgraph G_T of consistent nodes (in the underlying tree labeling) admits a balanced tree labeling in which all nodes are compatible if and only if G_T is a balanced (complete) binary tree and G contains certain additional edges between nodes at each fixed depth in G_T .

To solve BalancedTree, each node v outputs a pair $(\beta(v), p(v))$, where $\beta(v)$ is a label in the set $\{B, U\}$ (for Balanced, Un-balanced) and $p(v) \in \mathcal{P}$ is a port number. The interpretation is that if every vertex w in the sub-tree of G_T rooted at v is compatible, then v should output (B, P(v)). If v is incompatible, it outputs (U, \bot) . Finally, if v is compatible, but some descendant of v is incompatible, then v outputs (U, p), where $p \in \{\operatorname{RC}(v), \operatorname{LC}(v)\}$ is a port number corresponding to the first hop on a path to an incompatible node below v. Thus, a valid output has the following global interpretation: Starting from any vertex, following the path of port numbers (edges) output each subsequent node terminates either at the root of a balanced binary tree, or at an incompatible node.

Definition 4.1. Let G = (V, E) be a graph of maximum degree at most Δ . A *balanced tree labeling* consists of a tree labeling (Definition 3.1) together with the following labels for each node $v \in V$:

- a left neighbor $LN(v) \in \mathcal{P}$,
- a *right neighbor* $\operatorname{RN}(v) \in \mathcal{P}$.

Definition 4.2. Let G = (V, E) be a graph and \mathcal{L} a balanced tree labeling on G. Suppose v is consistent in the sense of Definition 3.4. We say that \mathcal{L} is *compatible* at a node v if the following conditions hold:

- type-preserving: If v is internal (respectively a leaf), then RN(v) and LN(v) are internal (respectively leaves) or \perp .
- agreement: If $LN(v) \neq \bot$ then RN(LN(v)) = v; if $RN(v) \neq \bot$ then LN(RN(v)) = v.
- siblings: If $LC(v), RC(v) \neq \bot$ (i.e., v is internal) then RN(LC(v)) = RC(v) and LN(RC(v)) = LC(v).
- persistence: If v is internal and $w = \text{RN}(v) \neq \bot$, then w is internal and RN(RC(v)) = LN(LC(w)). Symmetrically, if $u = \text{LN}(v) \neq \bot$ then u is internal and LN(LC(v)) = RN(RC(v)).

• *leaves*: If v is a leaf then $LN(v) \neq \bot \implies LN(v)$ is a leaf and $RN(v) \neq \bot \implies RN(v)$ is a leaf.

The labeling \mathcal{L} is *globally compatible* if every consistent vertex v is compatible.

Definition 4.3. The problem BalancedTree consists of the following:

Input: a balanced tree labeling \mathcal{L}

Output: for each $v \in V$, a pair $(\beta(v), p(v)) \in \{B, U\} \times \mathcal{P}$

Validity: for each consistent $v \in V$ we have

- 1. if v is not compatible, then v outputs (U, \bot)
- 2. if v is a compatible leaf then v outputs (B, P(v))
- 3. if v is compatible and internal then
 - (a) if LC(v) and RC(v) output (B, P(LC(v))) and (B, P(RC(v))), respectively, then v outputs (B, P(v))
 - (b) if LC(v) (resp. RC(v)) outputs (U, \cdot) , then v outputs (U, LC(v)) (resp. (U, RC(v)))

Lemma 4.4. BalancedTree is an LCL.

Proof. As noted before, checking if a node is internal, a leaf, or inconsistent can be done locally. Also, it is clear that all of the conditions for compatibility (Definition 4.2) are locally checkable. Thus, the validity conditions of BalancedTree are also locally checkable. \Box

Theorem 4.5. The complexity of BalancedTree is

 $\begin{aligned} & \text{R-DIST}(\mathsf{BalancedTree}) = \Theta(\log n), \\ & \text{D-DIST}(\mathsf{BalancedTree}) = \Theta(\log n), \\ & \text{R-VOL}(\mathsf{BalancedTree}) = \Theta(n), \\ & \text{D-VOL}(\mathsf{BalancedTree}) = \Theta(n). \end{aligned}$

The proof of Theorem 4.5 is as follows. In Section 4.1, we show that in a valid output for BalancedTree, any consistent internal node is either the root of a balanced binary tree, or it has an inconsistent descendant within distance $\log n$. Thus, each node can determine its correct output by examining its $O(\log n)$ radius neighborhood. The upper bound is tight, as a node may need to see up to distance $\Omega(\log n)$ in order to see its nearest incompatible or inconsistent node.

For the volume lower bounds, consider a balanced binary tree with lateral edges such that there exists a globally compatible labeling, and let \mathcal{L} be such a labeling (see Figure 5). By modifying the input of a single pair of sibling leaves, we can form an input labeling \mathcal{L}' which is not globally compatible. The validity conditions of BalancedTree imply that the root of the tree must be able to distinguish \mathcal{L} from \mathcal{L}' to produce its output. Therefore, to solve BalancedTree, the root must query a large fraction of the leaves—i.e., $\Omega(n)$ nodes. We formalize this argument using the communication complexity framework of Eden and Rosenbaum [16] in Section 4.2.

4.1 Structure of Valid Outputs

Lemma 4.6. Suppose G = (V, E) is a graph, \mathcal{L} a balanced tree labeling of G, and $v \in V$ is consistent. Then either the sub-(pseudo)tree of G_T rooted at v is a balanced binary tree (i.e., all leaves below v are at the same distance from v), or there exists a descendant w of v with $\operatorname{dist}(v, w) < \log n$ such that w is incompatible.



Figure 5: An instance of BalancedTree constructed in the proof of Proposition 4.9. The diagonal edges are directed from parent to child, as specified by the underlying binary tree labeling. The horizontal edges are left and right neighbors. The dashed edges between leaves are included in G. Given strings $a, b \in \{0, 1\}^N$, the labeling of the leaves are determined as follows: we have $\text{RN}(u_i) = \text{LN}(w_i) = \perp$ if $a_i = b_i = 1$; otherwise $\text{RN}(u_i) = 5$ and $\text{LN}(w_i) = 4$. The labeling constructed in this way is globally compatible if and only if disj(a, b) = 1. In this case, the unique valid output is to label each node (U, p).

Proof. Let H denote the sub-(pseudo)tree of G_T rooted at v. Suppose H is not a balanced binary tree. We will show that there is an incompatible w in H within $\log n$ distance from v. To this end for $d \in \mathbf{N}$, let $D_v(d)$ denote the set of descendants w of v in H such that there exists a path from v to w of length d. That is, $D_v(0) = \{v\}$, and for each $d \ge 1$, $D_v(d)$ consists of the children of nodes in $D_v(d-1)$.³ Let H_d be the (induced) subgraph of G with vertex set $D_v(0) \cup D_v(1) \cup \cdots \cup D_v(d)$.

- Claim. Suppose every vertex v in H_d is compatible. Then $D_v(d)$ is laterally connected. That is, for every u, w in $D_v(d)$, there exists a path connecting u and w consisting only of edges $\{x, y\}$ such that y = RN(x) (and symmetrically x = LN(y)).
- Proof of claim. We argue by induction on d. The case d = 0 is trivial as $D_v(0)$ consists of a single vertex. Now suppose the claim holds for d-1, and take $u, w \in D_v(d-1)$. By the inductive hypothesis, there exists a path $v_0, v_1, \ldots v_\ell$ with $P(u) = v_1$ and $P(w)v_\ell$, and without loss of generality (by possibly exchanging the roles of u and w) we have $v_i = \text{RN}(v_{i-1})$ for $i = 1, 2, \ldots, \ell$. For each i, let $u_i = \text{LC}(v_i)$ and $w_i = \text{RC}(v_i)$. By the siblings property of compatibility, we have $w_i = \text{RN}(v_i)$, and by persistence, $v_{i+1} = \text{RN}(w_i)$. Therefore, the sequence $v_0, w_0, v_1, \ldots, v_\ell, w_\ell$ forms a path. Since $u \in \{v_0, w_0\}$ and $w \in \{v_\ell, w_\ell\}$, the claim follows.

Using the claim, we will show that H contains an incompatible node w. Since H is assumed not to be balanced, there exist leaves u and u' at distances d and d' (respectively) from v with d' > d. In particular, take u to be the nearest leaf to v, and u'' be u''s (unique) ancestor in $D_v(d)$. By the claim, there exists a path v_0, v_1, \ldots, v_ℓ between u and u'' in $D_v(d)$ such that for each i we have $v_i = \text{RN}(v_{i-1})$. Without loss of generality, assume $u = v_0$ and $u'' = v_\ell$. Since v_0 is a leaf and v_ℓ is internal, there exists some $i \in [\ell]$ such that v_{i-1} is a leaf, and v_i is internal or inconsistent. However, this implies that $w = v_{i-1}$ is incompatible. Moreover, dist(v, w) = dist(v, u) = d which is at most log n (as u was chosen to be the nearest leaf to v), which gives the desired result. \Box

³Since G_T is a pseudo-forest, and hence, could contain cycles, it may be that the same vertex is contained in $D_v(d_i)$ for different values of d_1, d_2, \ldots Since each node has at most a single parent, there is still a unique path from v to w of each length d_i .

Lemma 4.7. Suppose G = (V, E) is a graph and \mathcal{L} a globally compatible labeling. Then in every valid solution to BalancedTree, every consistent node v outputs (B, P(v)). Conversely, if v has a descendant w in G_T that is incompatible, then v outputs (U, \cdot) in any valid solution to BalancedTree.

Proof. First consider the case where \mathcal{L} is globally compatible. We argue by induction on the height of v that v outputs (B, P(v)). For the base case, the height of v is 0, hence v is a leaf. Then by Condition 2 of validity v outputs (B, P(v)). For the inductive step, suppose all nodes at height h - 1 output (B, P(v)), and v is at height h. In particular, the children of v both output (B, v). Therefore, v outputs (B, P(v)) by Condition 3(a) of validity. This gives the first conclusion of the lemma.

Now suppose v has a descendant w in G_T that is incompatible. Let $v = u_\ell, u_{\ell-1}, \ldots, u_0 = w$ be the path from v to w in G_T . We argue by induction on i that u_i outputs (U, \cdot) . The base case i = 0 follows from Condition 1 of validity. The inductive step follows from Condition 3(b) of validity: since u_i has a child that outputs (U, \cdot) (namely, u_{i-1}), u_i must output (U, \cdot) as well. \Box

Proposition 4.8. There exists an algorithm A with the following property. Let G = (V, E) be a graph on n nodes and \mathcal{L} a balanced tree labeling. Then A solves BalancedTree on (G, \mathcal{L}) and for all $v \in V$, DIST $(A, G, \mathcal{L}, v) = O(\log n)$. Therefore,

D-DIST(BalancedTree), R-DIST(BalancedTree) = $O(\log n)$.

Proof. Consider the following algorithm, A. Starting from a vertex v, A searches the O(1) neighborhood to determine if v is internal, a leaf, or inconsistent. If v is inconsistent, it outputs (B, \perp) . If v is a leaf, it outputs (B, P(v)) if all of the conditions of compatibility (Definition 4.2) are satisfied, and (U, \perp) otherwise. Finally, if v is internal, A queries v's log n+O(1) neighborhood in order to find its nearest leaf, which is at distance $d \leq \log n$. If v sees a descendant within distance d that is incompatible, then v outputs (U, p) where p is the port towards the nearest such in compatible node, breaking ties by choosing the left-most descendant. Otherwise, v outputs (B, P(v)).

Conditions 1 and 2 in the validity of Definition 4.3 are trivially satisfied for all node. Consider the case where v is internal and compatible. By Lemma 4.6, if the subtree H of G_T rooted at vis not balanced, then there is a (closest, leftmost) incompatible node w in H at distance at most log n. Thus, in this case v outputs (U, p(v)) where p(v) is the port towards w. Similarly, the child v' = p(v) will output (U, p(v')) so that condition 3(b) of validity is also satisfied. Finally, if H is balanced and all nodes in H are compatible, then v will output (B, P(v)), as will all other nodes w in H. Thus condition 3(a) of validity is also satisfied.

4.2 Volume Lower Bounds

Proposition 4.9. Any (randomized) algorithm A that solves BalancedTree with probability bounded away from 1/2 requires $\Omega(n)$ queries in expectation. Thus

$$R$$
-VOL(BalancedTree), D -VOL(BalancedTree) = $\Omega(n)$.

Proof. For any $k \in \mathbb{N}$ form the graph G by starting with the complete binary tree of depth k. Assign IDs, port numbers, and labels as in the proof of Proposition 3.13, so that the root has ID 1, its left child has ID 2, its right child has ID 3, and so on. In particular, the nodes at depth d have IDs $2^d, 2^d + 1, \ldots, 2^{d+1} - 1$. For each d, add lateral edges between nodes with IDs $2^d + i - 1$ and $2^d + i$ for all $i = 1, 2, \ldots, 2^d - 1$, and assign port numbers so that $2^d + i's$ port 4 leads to $2^d + i - 1$, and port 5 leads to $2^d + i - 1$ (for $1 \le i \le 2^d - 1$). Finally, for all nodes v at depths $d \le k - 1$, assign labels LN(v) and RN(v) to be consistent with the lateral edges described above. Thus, a balanced tree labeling \mathcal{L} has been determined at all nodes except the leaves of G. Note that \mathcal{L} is constructed such that all nodes at depth $d \leq k - 2$ are compatible. See Figure 5 for an illustration.

Let $N = 2^{k-1} (= \Omega(n))$. We complete the labeling \mathcal{L} to be an embedding of the disjointness function disj: $\{0,1\}^N \times \{0,1\}^N \to \{0,1\}$, as follows. Given $i \in [N]$, let v_i be the i^{th} left-most node at depth k-1 in G, and let u_i and w_i be its left and right child, respectively. Thus u_1 is the left-most leaf in G, w_1 its right sibling, and so on. For $i \leq N-1$, assign $\text{RN}(w_i) = v_{i+1}$ and $\text{LN}(v_{i+1}) = w_i$, and take $\text{LN}(v_1) = \text{RN}(w_N) = \bot$. Finally, given any $a, b \in \{0,1\}^N$, we complete the balanced tree labeling \mathcal{L} as follows:

$$RN(u_i) = LN(w_i) = \bot$$
 if $a_i = b_i = 1$

$$RN(u_i) = w_i, LN(w_i) = u_i$$
 otherwise.

For the labeling \mathcal{L} constructed as above, it is straightforward to verify that all nodes satisfy all conditions of compatibility with one possible exception: v_i fails to satisfy the siblings condition if and only if $a_i = b_i = 1$. That is, \mathcal{L} is globally compatible if and only if $\operatorname{disj}(a, b) = 1$. Thus, by Lemma 4.7, for any solution to BalancedTree on input \mathcal{L} , the root outputs (B, \bot) if and only if $\operatorname{disj}(a, b) = 1$.

Fix v to be the root of G, and consider an execution of any algorithm A solving BalancedTree from v. We will apply Theorem 2.10. The observation that v outputs (B, \bot) if and only if $\operatorname{disj}(a,b) = 1$ shows that our construction of $\mathcal{E} : (a,b) \mapsto \mathcal{L}$ and $g(G,\mathcal{L}) = 1$ if and only if $\mathcal{L}(v) = (B, \bot)$ gives an embedding of disj in the sense of Definition 2.8. Moreover, all labels in \mathcal{L} are independent of a and b except for the leaves, and for each i, the labels of u_i and w_i depend only on the values of a_i and b_i . Therefore, all queries to (G,\mathcal{L}) have communication cost 0, except the queries of the form $\operatorname{query}(v_i, \operatorname{LC}(v_i))$ and $\operatorname{query}(v_i, \operatorname{RC}(v_i))$. The latter queries can be answered by exchanging a_i and b_i , hence the communication cost of such queries is 2. Therefore, by Theorems 2.10 and 2.11, the expected query complexity of any algorithm A that solves BalancedTree with probability bounded away from 1/2 is $\Omega(N) = \Omega(n)$, as desired. \Box

4.3 Proof of Theorem 4.5

Proof of Theorem 4.5. By Proposition 4.8 give the upper bounds on D-DIST and R-DIST. The corresponding lower bound of $\Omega(\log n)$ follows by analyzing the same construction used in the proof of Proposition 4.9. Starting from the root v of G, any algorithm that queries nodes only up distance $d \leq k - 1$ can be simulated by Alice and Bob without communication. Thus, such an algorithm cannot solve disjointness (hence BalancedTree) with probability bounded away from 1/2.

Finally, the lower bounds of Proposition 4.9 are tight, as all LCLs trivially have R-VOL, D-VOL = O(n).

5 Hierarchical $2\frac{1}{2}$ Coloring

In this section, we describe a variant of the family of "hierarchical $2\frac{1}{2}$ coloring problems" introduced by Chang and Pettie [12]. Like the original problem, our variant, Hierarchical-THC(k), has randomized and deterministic distance complexities $\Theta(n^{1/k})$. We will show that the problem has randomized volume complexity $O(n^{1/k} \log^{O(k)}(n))$, and deterministic volume complexity $\Omega(n/\log n)$.

Like the problems LeafColoring and BalancedTree, the input labels Hierarchical-THC(k) induce a pseudo-forest structure on (a subgraph of) the input graph G. In the case of Hierarchical-THC(k), the individual pseudo-trees have the following hierarchical structure: At level 1 of the hierarchy connected components consist of directed paths and cycles. At a level $\ell > 1$, each node is the parent of the "root" of a level $\ell - 1$ component. To solve Hierarchical-THC(k), each node v produces an output color $\chi_{out}(v) \in \{R, B, D, X\}$. Nodes outputting D are said to **decline**, and



Figure 6: An example of the hierarchical forest induced by a tree labeling. Edges are oriented from parent to child. Left-diagonal and vertical edges indicate that a node is a left child, while right-diagonal edges indicate right children. The number in each node gives the node's level. The shaded region indicates a subtree H_2 up to hierarchy level 2. The level 2 root in H_2 is indicated by a diamond-shaped node. The double struck node in H_2 (at level exactly 2) comprise a level 2 backbone, C_2 . The lower left node in C_2 is a level 2 leaf.

nodes outputting X are said to be *exempt*. Nodes are only allowed to be exempt under certain locally checkable conditions, described below. Upon removal of exempt nodes, the nodes in each connected component at each level of the hierarchy are required to output R, B, or D unanimously, with each "leaf" outputting its input color or D if $\ell < k$.

Definition 5.1. Let \mathcal{L} be a (colored) tree labeling. Let G' be the subgraph of G consisting of edges $\{u, v\}$ where u = P(v) and v = LC(u) or RC(u). The *level* of a node v, denoted level(v), is defined inductively as follows: If $RC(v) = \bot$, then level(v) = 1. Otherwise, level(v) =1 + level(RC(v)). The *hierarchical forest to level* k, denoted $G_k = (V_k, E_k)$, is the sub-(pseudo)-forest of G' consisting of edges $\{u, v\}$ with level(u), $level(v) \leq k$ satisfying one of the following properties:

- v = P(u), u = LC(v), and level(v) = level(u), or
- v = P(u), u = RC(v), and level(v) = level(u) + 1.

Observation 5.2. The hierarchical forest to level k, G_k , is locally computable in the sense that each node v can determine level(v), and which of its incident edges are in G_k by examining its O(k)-radius neighborhood. Moreover, we assume without loss of generality that every non- \perp label P(v), LC(v), and RC(v) corresponds to an edge in G_k . That is, for example, we have $\{v, P(v)\} \in E_k$.

Definition 5.3. Suppose v is a vertex with $\text{level}(u) = \ell$. Then we call v a *level* ℓ *root* if $P(u) = \bot$ or u = RC(P(u)) (and hence $\text{level}(P(u)) = \ell + 1$). We call u a *level* ℓ *leaf* if $\text{LC}(v) = \bot$.

Observation 5.4. Since each node in G_k has at most one parent, G_k is a pseudo-forest. Moreover, it has the following structure: For every $\ell \leq k$, each connected component of G_k consisting of nodes v with $\text{level}(v) = \ell$ is a path or cycle, and every (directed) edge is of the form (v, LC(v)). If $\ell = 1$, then for all such v we have $\text{RC}(v) = \bot$. If $1 < \ell \leq k$, then each RC(v) is the level $\ell - 1$ root of a (directed) subtree of G_k . Finally, if $\ell > k$, then v is an isolated vertex in G_k . **Definition 5.5.** For any fixed constant $k \in \mathbf{N}$, the problem Hierarchical-THC(k) consists of the following:

Input: a colored tree labeling \mathcal{L}

Output: for each $v \in V$, a color $\chi_{out}(v) \in \{R, B, D, X\}$

Validity: for each $v \in V$, and $\ell = \text{level}(v)$

- 1. if $\ell > k$, then $\chi_{out}(v) = X$
- 2. if v is a level ℓ leaf then $\chi_{out}(v) \in {\chi_{in}(v), D, X}$
- 3. if $\ell = 1$ then
 - (a) $\chi_{\text{out}}(v) \in \{R, B, D\}$, and
 - (b) if v is not a level 1 leaf, then $\chi_{out}(v) = \chi_{out}(LC(v))$
- 4. if $1 < \ell < k$ and v is not a level ℓ leaf then either
 - (a) $\chi_{\text{out}}(v) = \chi_{\text{out}}(\text{LC}(v)) \in \{R, B, D\},\$
 - (b) $\chi_{\text{out}}(v) = X$ and $\chi_{\text{out}}(\text{RC}(v)) \in \{R, B, X\}$, or
 - (c) $\chi_{\text{out}}(v) \in \{\chi_{\text{in}}(v), D\}$ and $\chi_{\text{out}}(\text{LC}(v)) = X$
- 5. if $\ell = k$ then $\chi_{out}(v) \in \{R, B, X\}$ and
 - (a) if $\chi_{out}(v) = X$ then $\chi_{out}(RC(v)) \in \{R, B, X\}$, and
 - (b) if v is not a level ℓ leaf and $\chi_{out}(v) \neq X$, then either
 - $\chi_{\text{out}}(\text{LC}(v)) \neq X$ and $\chi_{\text{out}}(v) = \chi_{\text{out}}(\text{LC}(v))$, or
 - $\chi_{\text{out}}(\text{LC}(v)) = X$ and $\chi_{\text{out}}(v) = \chi_{\text{in}}(v)$

The following observation gives some intuition about valid outputs of Hierarchical-THC(k).

Observation 5.6. Consider a valid output for Hierarchical-THC. Then Conditions 2 and 3 imply that each connected component of level 1 vertices in G_k is unanimously colored either D or $\chi_{in}(u)$ where u is the (unique) level 1 leaf in the connected component. Similarly, Conditions 2 and 4 characterize valid colorings of each connected component of G_k at levels ℓ satisfying $1 < \ell < k$, although components are no longer required to output unanimous colors. Instead, Condition 4(b) allows nodes v to "choose" to output X if RC(v) outputs a color in $\{R, B, X\}$. However, Conditions 4(a) and 4(c) require that nodes that are not allowed to choose X must either output $\chi_{out}(LC(v))$, or $\chi_{in}(v)$ (if $\chi_{out}(LC(v)) = X$). Finally, Condition 5 restricts valid outputs at level k. By Condition 5(a) a level k node v is only allowed to output X if $\chi_{out}(RC(v)) \in \{R, B\}$. Meanwhile, Conditions 2 and 5(b) stipulate that on a path between X's at level k, all nodes output $\chi_{in}(u)$, where u is the parent of the left vertex outputting X.

Remark 5.7. Our problem Hierarchical-THC differs from the version of hierarchical $2\frac{1}{2}$ coloring described by Chang and Pettie [12] in two respects. First, we require that connected components of non-exempt vertices are unanimously colored R, B, or D, whereas in [12], such components must either be unanimously colored D or properly colored by R in B (i.e., an R node's non-exempt neighbors must output B). By using unanimous (rather than proper) colorings in all cases, our version of Hierarchical-THC allows us to impose more restrictions on valid outputs by designating input colors of nodes. This is helpful, for example, in the proof of Proposition 5.20 (the deterministic volume lower bound), where the main claim in the proof relies on unanimous coloring of components. The second difference between Hierarchical-THC and that of [12] is that in the latter problem, a node v with $\chi_{out}(\text{RC}(v)) \neq D$ is required to output X, whereas our Conditions 4(b) and 5(a) merely allow v to output X if $\text{RC}(v) \neq D$. Our relaxation of the exemption conditions does not affect the distance complexity of the problem, however our modification seems necessary in order for the problem to have small volume complexity.



Figure 7: Input (left) and valid output (right) for Hierarchical-THC(3).

The following lemma is clear from previous discussion.

Lemma 5.8. For every fixed constant $k \in \mathbf{N}$, Hierarchical-THC(k) is an LCL.

We now state the main result of this section.

Theorem 5.9. For each fixed positive integer k, the complexity of Hierarchical-THC(k) satisfies

$$\begin{split} & \text{R-DIST}(\mathsf{Hierarchical-THC}(k)) = \Theta(n^{1/k}), \\ & \text{D-DIST}(\mathsf{Hierarchical-THC}(k)) = \Theta(n^{1/k}), \\ & \text{R-VOL}(\mathsf{Hierarchical-THC}(k)) = O(n^{1/k}\log^{O(k)}(n)) \\ & \text{D-VOL}(\mathsf{Hierarchical-THC}(k)) = \Omega(n/k\log n). \end{split}$$

5.1 Shallow and Light Components

Before proving the claims of Theorem 5.9, we provide some preliminary results on the structure of G_k for any colored tree labeling \mathcal{L} . For the remainder of the section, fix some tree labeling \mathcal{L} , positive integer k, and let G_k be the hierarchical forest to level k.

Definition 5.10. For $\ell \in \mathbf{N}$ with $\ell \leq k$, let C_{ℓ} be a maximal connected component of G_k consisting of nodes v at level ℓ . We say that C_{ℓ} is **shallow** if $|C_{\ell}| \leq 2n^{1/k}$. Otherwise, if $|C_{\ell}| > 2n^{1/k}$, we say that C is **deep**.

Let H_{ℓ} be a connected component of G_k consisting of C_{ℓ} and all of descendants of nodes $v \in C_{\ell}$ (at all levels $\ell, \ell - 1, ..., 1$). We call H_{ℓ} light if $|H_{\ell}| \leq n^{\ell/k}$. Otherwise H_{ℓ} is said to be **heavy**. Similarly, if v is the level ℓ root of H_{ℓ} , we call v light (resp. heavy) if H_{ℓ} is light (resp. heavy).

Lemma 5.11. Let C_{ℓ} and let H_{ℓ} be as in Definition 5.10 with $2 \leq \ell \leq k$, and suppose H_{ℓ} is light. Then at most $n^{1/k}$ nodes in C_{ℓ} have heavy right children.

Proof. Let $W \subseteq C_{\ell}$ be the nodes with heavy right children, and let $m = |H_{\ell}|$. By the assumption that H_{ℓ} is light, we have $m \leq n^{\ell/k}$. On the other hand, we have $|W| \cdot n^{(\ell-1)/k} \leq m$, as each $w \in W$ has a heavy right child at level $\ell - 1$. Combining the two previous inequalities gives $|W| \cdot n^{(\ell-1)/k} \leq n^{\ell/k}$, which gives $|W| \leq n^{1/k}$, as desired.

Lemma 5.11 implies the following dichotomy for light components, H_{ℓ} : Either C_{ℓ} is shallow, or every subset U of C_{ℓ} of size at least $2n^{1/k}$ has the property that at least half of the nodes $v \in U$ have light right children. In the case where C_{ℓ} is shallow, the nodes $v \in C_{\ell}$ can be validly colored according to Definition 5.5 by exploring all of C_{ℓ} using distance and volume $O(n^{1/k})$. Indeed, for any $\ell \in [k]$, it suffices for each $v \in C_{\ell}$ to output $\chi_{in}(u_0)$, where u_0 is either the (unique) leaf in C_{ℓ} (in the case C_{ℓ} is a path), or u_0 is the vertex with minimal ID (in the case when C_{ℓ} is a cycle).

On the other hand, if C_{ℓ} is deep (and H_{ℓ} is light, hence we must have $\ell \geq 2$), then every node $v \in C_{\ell}$ has a descendant $u \in C_{\ell}$ and ancestor $w \in C_{\ell}$, with $\operatorname{dist}(u, w) \leq n^{1/k}$, such that uis a leaf or u has a light right child, u', and w is a root or w has a light right child, w'. In the case where $u' = \operatorname{RC}(u)$ is light, let $H_{\ell-1}$ be the sub-component of H_{ℓ} rooted at u'. Then working recursively, we will show that $H_{\ell-1}$ can be validly colored using distance $O(n^{1/k})$ such that u' outputs a color $\chi_{\operatorname{out}}(u') \in \{R, B\}$. Therefore, u satisfies Condition 4(b) or the implication of 5(a) of validity, so that $\chi_{\operatorname{out}}(u) = X$ satisfies validity. Similarly, if $w' = \operatorname{RC}(w)$ is light, wcan output X. Choosing u and w to be the closest descendant and ancestor of v in C_{ℓ} with these properties, v can then output $\chi_{\operatorname{out}}(v) = \chi_{\operatorname{in}}(\operatorname{P}(u))$ —as will all other nodes between u and w—so that v satisfies Condition 4(a/c) or 5(b). We formalize this procedure in Algorithm 2. The analysis and matching lower bound appear in Section 5.2.

The (deterministic) recursive approach to coloring nodes v in deep components C_{ℓ} gives an $O(n^{1/k})$ distance protocol. However, the volume of the protocol may still be large because all nodes between u and w are recursively checked for solvability with $\chi_{out}(u') \in \{R, B\}$. In order to solve Hierarchical-THC in a volume-efficient manner, our next procedure samples a small fraction of candidates u to try to (validly) color $\chi_{out}(u) = X$. By choosing each candidate in C_{ℓ} with probability $p = \Theta((\log n)/n^{1/k})$, the number of such candidates in any $2n^{1/k}$ radius neighborhood of v is $O(\log n)$. If H_{ℓ} is light, with high probability at least one of the candidates will correctly output $\chi_{out}(u) = X$, thus allowing v to output the $\chi_{in}(\mathbf{P}(u))$.⁴ Each node $v \in C_{\ell}$ must visit at most $2n^{1/k}$ nodes in C_{ℓ} , and an inductive argument shows that each recursive call to a sampled u incurs an additional volume of $O(n^{1/k} \log^{O(\ell)} n)$. The argument is formalized in Section 5.3.

For the deterministic volume lower bound, our argument essentially shows that if a deterministic algorithm A has the property that many executions of A on input G never query a leaf of G, then A cannot solve Hierarchical-THC(k) on G. We formalize this idea in Section 5.4. Given any deterministic algorithm A purporting to solve Hierarchical-THC(k) with volume complexity $o(n/(k \log n))$, we describe a procedure \mathcal{P} that produces a labeled graph G with n vertices for which A produces an incorrect output.

5.2 Distance Bounds

Proposition 5.12. There exists a deterministic algorithm A such that for every graph G on n nodes, tree labeling \mathcal{L} , and positive integer k, A solves Hierarchical-THC(k) with distance complexity $O(kn^{1/k})$.

Proof. Consider the algorithm A where each node v at level $\ell \leq k$ outputs

 $\chi_{\text{out}}(v) = \text{RecursiveHTHC}(v, \ell),$

⁴Note that sampling each candidate u must be done using u's private randomness to ensure that all nodes v visiting u agree on whether or not u is sampled.

Algorithm 2 RecursiveHTHC (v, ℓ)

1: $C \leftarrow$ the level ℓ component of G_k containing v2: if $|C| \le 2n^{1/k}$ then $u_0 \leftarrow$ leaf in C if C is a path and node with minimal ID otherwise 3: return $\chi_{in}(u_0)$ 4: 5: else if $\ell = 1$ then return D6: 7: else if RecursiveHTHC(RC(v), $\ell - 1$) $\in \{R, B, X\}$ then return X8: 9: end if 10: $u, w \leftarrow v$ 11: for i = 0 to $2n^{1/k}$ do if RecursiveHTHC(RC(u)) = D and u not a level ℓ leaf then 12:{no exempt left descendant found yet} 13: $u \leftarrow LC(u)$ 14:end if if RecursiveHTHC(RC(w)) = D and w not a level ℓ root then 15: $w \leftarrow P(w)$ {no exempt ancestor found yet} 16:end if 17:18: end for 19: **if** u = v **then** $\{\chi_{\text{out}}(\text{RC}(u)) \neq D\}$ 20: return X21: end if 22: if dist $(u, w) \leq 2n^{1/k}$ then if $\operatorname{RecursiveHTHC}(\operatorname{RC}(u)) \in \{R, B, X\}$ then 23: $\{\chi_{\text{out}}(u) = X\}$ 24:return $\chi_{in}(\mathbf{P}(u))$ else 25:return $\chi_{in}(u)$ {u is a leaf and $\chi_{out}(u) = \chi_{in}(u)$ } 26:27:end if 28: **else** return D29:30: end if

and outputs X if $\ell > k$. We first argue that the output of A satisfies the validity conditions of Definition 5.5. Fix a vertex v at level ℓ . If $\ell > k$, then $\chi_{out}(v) = X$ as prescribed by condition 1. Now consider the case $\ell = 1$, and let C be as in Line 1. By the provisions in Lines 2 and 5, all C unanimously output D or $\chi_{in}(u_0)$. In particular, Conditions 2 and 3 are satisfied.

Consider $1 \leq \ell < k$, and let H be the connected component of G_h consisting of C together with all descendants of C.

Claim. The output of each $v \in C$ is valid. Moreover, if H is light, then $\chi_{out}(v) \in \{R, B, X\}$.

Proof of claim. We argue by induction on ℓ . The base case $\ell = 1$ is handled by the previous paragraph, and the observation that for $\ell = 1$, H = C being light implies C is shallow. Thus the condition of Line 2 is satisfied and all nodes in C output $\chi_{in}(u_0) \in \{R, B\}$.

For the inductive step, suppose the claim holds at level $\ell - 1$. If C is shallow, then all nodes $v \in C$ output $\chi_{in}(u_0) \in \{R, B\}$, so Conditions 2 and 4(a)/5(b) of validity are satisfied. Further, the level ℓ root (if any) outputs a color in $\{R, B\}$, so the conclusion of claim follows when C is shallow. Now suppose C is not shallow. Fix $v \in C$. If RecursiveHTHC(RC(v), $\ell - 1$) $\in \{R, B, X\}$, then $\chi_{out}(v) = X$ and v satisfies validity condition 4(b)/5(a). Otherwise let u and w be v's descendant and ancestor (respectively) in C during the execution of Line 22. If the condition dist $(u, w) \leq 2n^{1/k}$ in Line 22 is satisfied, then all nodes $v' \in C$ on the path from w to u (possibly excluding w or u themselves) will store the same values for w and u when Line 22 is executed. Therefore, all such v' will unanimously output $\chi_{in}(P(u))$ or $\chi_{in}(u)$ (both in $\{R, B\}$) according to Line 24 or 26. Thus all nodes between w and u in Csatisfy validity conditions 2/4(a)/4(c) or 5(b). Moreover, if v is a root level ℓ root, then v = w and $\chi_{out}(v) = \{R, B, X\}$ as claimed.

Finally consider the case where $\operatorname{dist}(u, w) > 2n^{1/k}$. By Lemma 5.11 and the inductive hypothesis, this case can only occur if H is heavy. In particular, this case can only occur for $\ell < k$. If v is a level ℓ leaf, then v = u and either v outputs X (at Line 8) or D (at Line 29). Either way, validity condition 2 is satisfied. If v is not a leaf, then if v outputs X at Line 8, v satisfies validity condition 4(b). Otherwise, v outputs D in Line 29. Similarly, $\operatorname{LC}(v)$ will output D or X (according to if RecursiveHTHC(RC(LC(v)), $\ell - 1$) $\in \{R, B, X\}$), because $\operatorname{LC}(v)$ and v agree on whether the condition $\operatorname{dist}(u, w) > 2n^{1/k}$ in Line 2 is satisfied (even though v and $\operatorname{LC}(v)$ may store different values of u and w). Accordingly, validity condition 4(a) or 4(c) is satisfied. Thus the claim holds for v at level ℓ , as desired.

By the claim, the output of A satisfies validity (where again, we observe that if $\ell = k$, then H is light. All that remains is to bound the distance complexity of A. To this end, a straightforward induction argument (on ℓ) shows that RecursiveHTHC(v, ℓ) queries nodes at distance at most $O(\ell \cdot n^{1/k})$ from v. For $\ell = 1$, this is immediate, as v can determine if $|C| \leq 2n^{1/k}$ using $O(n^{1/k})$ queries. For $\ell > 1$, the same applies. Further, RecursiveHTHC(v, ℓ) queries $O(n^{1/k})$ nodes in C (at level ℓ) in the loop starting at Line 11, and each such query makes a single call to RecursiveHTHC($v, \ell - 1$). Thus, applying the inductive hypothesis, the distance complexity of RecursiveHTHC(v, ℓ) is $O(n^{1/k}) + O((\ell - 1)n^{1/k})$, which gives the desired result.

Proposition 5.13. Any (randomized) algorithm A that solves Hierarchical-THC with probability bounded away from 1/2 has distance complexity $\Omega(n^{1/k})$.

We omit a proof of Proposition 5.13, as the argument is essentially the same as the proof lower bound proof for Chang and Pettie's variant of hierarchical $2\frac{1}{2}$ coloring (Theorem 2.3 in [12]). We note that the instance achieving the lower bound is a "balanced" instance of Hierarchical-THC(k), where every "backbone"—i.e., maximal connected component of G_k consisting of nodes at the same level—has size $\Theta(n^{1/k})$.

5.3 Randomized Volume Upper Bound

Proposition 5.14. There exists a randomized algorithm A' that for every graph G on n nodes and tree labeling \mathcal{L} , A' solves Hierarchical-THC with high probability using volume $O(n^{1/k} \log^{O(k)}(n))$.

The algorithm A' achieving the conclusion of Proposition 5.14 is a slight modification of RecursiveHTHC. In A', only a small fraction of the recursive calls to RecursiveHTHC are made. Specifically, each node v uses its private randomness to independently become a **way-point** with probability $p = (c \log n)/n^{1/k}$ (for some constant c to be chosen later). In A', a recursive call to RecursiveHTHC(v', ℓ) (in Line 7, 12, 15, or 23) is made if and only if v' is a way-point. In particular, if v is not a way-point, then Line 7 evaluates to false; if u (resp. w) is not a way-point, then Line 12 (resp. Line 15) evaluates to true.

The proof of correctness of A' is essentially the same as in the proof of Proposition 5.12, at least in cases where the distribution of way-points is sufficiently well-behaved. The potential complication in the analysis arises in deep components C (i.e., C with $|C| \ge 2n^{1/k}$), because light components can be validly colored deterministically using $O(|C|) = O(n^{1/k})$ volume. For deep components C, the choice of $p = (c \log n)/n^{1/k}$ ensures that in any segment of C of length $O(n^{1/k})$ there are $O(\log n)$ way-points in the segment with high probability (Lemma 5.16). Thus, we bound the number of recursive calls to RecursiveHTHC made by A'. On the other hand, if C is contained in a light component H, then by Lemma 5.11 any segment of length at least $2n^{1/k}$ will have at least 1/2-fraction of its nodes being the parents of light right children. The choice of p allows us to infer that some such "light parent" u is a way-point (Lemma 5.18). We then argue by induction that u will output X, hence C can be validly colored without any node outputting D.

Definition 5.15. Fix ℓ with $1 < \ell \leq k$ and let C be a maximal connected component of G_k consisting of nodes at level ℓ . A **short segment** is a path or cycle $S \subseteq C$ such that $|S| \leq 4n^{1/k}$. We say that S is **crowded** if it contains more than $8c \log n$ way-points.

Lemma 5.16. Suppose each $v \in V$ is chosen to be a way-point independently with probability $p = (c \log n)/n^{1/k}$. Then

 $\Pr(G_k \text{ contains a crowded segment}) \leq O(1/n).$

Proof. First observe that if G_k contains a crowded short segment, then it contains a crowded maximal short segment (i.e., a short segment that is not a subset of any other short segment). By associating each maximal short segment with its midpoint (in the case of a path), or node with lowest ID (in the case of a cycle), there are at most n maximal short segments in G_h .

Consider some fixed maximal short segment S. For $i \leq 4n^{1/k}$, let Y_i be an indicator random variable for the event that the *i*th node in S is a way-point if $i \leq |S|$, and Y_i is an independent Bernoulli random variable with probability p otherwise. Then

$$\Pr(S \text{ is crowded}) = \Pr\left(\sum_{i=1}^{|S|} Y_i \ge 8c \log n\right) \le \Pr\left(\sum_{i=1}^m Y_i \ge 8c \log n\right).$$

Since the Y_i are iid Bernoulli random variables, we can apply the Chernoff bound Lemma 2.12 to $Y = \sum_{i=1}^{m} Y_i$. Note that $\mu = E(Y) = 4c \log n$. Therefore, Lemma 2.12 gives

$$\Pr(Y \ge 8c \log n) = \Pr(Y \ge 2\mu) \le e^{-\mu/3} = e^{-(4/3)c \log n} = n^{-4c/3}.$$

For any $c \ge 3/2$, this final expression is at most $1/n^2$. Thus taking the union bound over all maximal short segments, we find that

$$\begin{aligned} \Pr(G_h \text{ contains a crowded segment}) &\leq \sum_{\substack{S \text{ maximal, short}}} \Pr(S \text{ is crowded}) \\ &\leq n \cdot \frac{1}{n^2} = \frac{1}{n} \end{aligned}$$

which gives the desired result.

Definition 5.17. Let ℓ , C, H, and v be as in Lemma 5.18. We call $u \in C$ a *light way-point* if u is a way-point and RC(u) is light.

Lemma 5.18. Fix ℓ with $1 < \ell \leq k$ and let C be a connected component of G_k consisting of nodes at level ℓ , and let H be the subgraph of G_k consisting of nodes in C together with all of their descendants. Suppose C is deep (i.e., $|C| > 2n^{1/k}$ and H is light (i.e., $|H| \leq n^{\ell/k}$. Let $v \in C$, and suppose every node $v' \in C$ is chosen to be a way-point independently with probability $(c \log n)/n^{1/k}$. Then with probability $1 - O(1/n^2)$ there exists a descendant $u \in C$ of v and ancestor $w \in C$ of v such that $dist(w, u) \leq 2n^{1/k}$ and

- 1. u is either a light waypoint or a level ℓ leaf, and
- 2. w is either a light waypoint or a level ℓ root.

Proof. We consider the case where C is a path. The case where C is a cycle can be handled similarly. Let u_0 be the leaf in C and let u_0, u_1, \ldots be defined by taking $u_{i+1} = P(u_i)$. For $j = 0, 1, \ldots, |C| - 2n^{1/k}$, let S_j be the segment of C containing $u_j, u_{j+1}, \ldots, u_{j+2n^{1/k}}$. By Lemma 5.11, at least $n^{1/k}$ nodes $u_i \in S_j$ have light right children. Therefore

$$\Pr(\text{no } u_i \in S_j \text{ is a light waypoint}) \le \prod_{i=1}^{n^{1/k}} (1-p) = (1-p)^{n^{1/k}} \le e^{-p \cdot n^{1/k}} = e^{-c \log n} = n^{-c}.$$

Taking any $c \ge 3$ and applying the union bound over all j, we have that every S_j contains a light waypoint with probability $1 - O(1/n^2)$. In particular, this implies that with probability at $1 - O(1/n^2)$, the maximum distance between consecutive light way-points is at most $2n^{1/k}$, which gives the desired result.

Corollary 5.19. Suppose every $v \in V$ is chosen to be a waypoint independently with probability $p = (c \log n)/n^{1/k}$. Then with probability 1 - O(1/n) the following holds: for every ℓ with $1 < \ell \leq k$ and every v such that $v \in C \subseteq H$ where H is light, then v has a descendant u and ancestor w both in C with $dist(w, u) \leq 2n^{1/k}$ such that u is either a level ℓ leaf or a light waypoint and w is either a level ℓ root or a light waypoint.

Proof of Proposition 5.14. Suppose way-points are chosen in such a way that the conclusions of Lemma 5.16 and Corollary 5.19 hold. Note that such events occur with probability 1 - O(1/n). We claim that in this case, the modification A' of RecursiveHTHC as described in the paragraph following Proposition 5.14 gives a valid solution to Hierarchical-THC(k) using volume $O(n^{1/k} \log^{O(k)} n)$.

The idea is to argue inductively that for each level ℓ and light H_{ℓ} , H_{ℓ} is validly colored with the level ℓ root of H_{ℓ} outputting $\chi_{out}(v) \neq D$. The base case $\ell = 1$ is straightforward, as H_1 being light implies $C_1 = H_1$ is shallow. Therefore, all nodes in H_1 output $\chi_{in}(u)$ where u is the leaf of C_1 . For the inductive step suppose the claim is true for $\ell - 1$, and consider $v \in C_{\ell}$. If v is a light waypoint, then by induction $\chi_{out}(\operatorname{RC}(v)) \neq D$, so that v outputs X in Line 20. Since all light way-points in C_{ℓ} output X and the conclusion of Corollary 5.19 guarantees that every connected component of $C_{\ell} \setminus \{\text{exempt nodes}\}$ is of size at most $2n^{1/k}$. Thus, each component is unanimously colored by R or B in Line 24. This in turn gives a valid coloring where the root of H_{ℓ} does not output D, as desired.

Finally, the upper bound on volume follows from Lemma 5.16. By the algorithm description, an execution initiated at $v \in C_{\ell}$ queries $O(n^{1/k})$ nodes in C_{ℓ} . Further, conclusion of the lemma implies that v only recursively calls RecursiveHTHC on $O(\log n)$ nodes in C_{ℓ} . Thus, a straightforward induction argument shows that the total number of queries is $O(n^{1/k} \log^{\ell} n)$, which gives the desired result.

5.4 Deterministic Volume Lower Bound

Proposition 5.20. Any deterministic algorithm A solving Hierarchical-THC(k) requires volume $\Omega(n/k \log n)$.

Proof. Suppose A is any deterministic algorithm with volume complexity at most m purportedly solving Hierarchical-THC(k). We assume $k \ge 2$, and describe a process \mathcal{P} that produces a graph G on $n = O(k \cdot m \log m)$ vertices and labeling \mathcal{L} on G such that \mathcal{P} does not solve Hierarchical-THC(k) on G. We begin by observing the following claim, whose (omitted) proof is straightforward.

Claim. Let A be any deterministic algorithm that solves Hierarchical-THC(k), and let H_v be the subgraph of some input graph G queried by an instance of A initiated at a vertex v in G. Suppose H_v has the property that every descendant (in G_h) of v in H_v has input color B (resp. R). Then the output of A must satisfy $\chi_{out}(v) \neq R$ (resp. $\chi_{out}(v) \neq B$).

The process \mathcal{P} constructs graphs in phases $k, k - 1, \ldots, 1$. In phase i, \mathcal{P} constructs a graph G_i by simulating executions of A initiated at nodes at level i. Each phase consists of $O(\log m)$ subphases, and in subphase \mathcal{P} simulates an execution of A initiated at a single vertex. We illustrate an example iteration of \mathcal{P} with some algorithm A in Figure 8.

The procedure \mathcal{P} maintains the following invariants. Let G_t denote the labeled tree constructed after \mathcal{P} simulates the t^{th} query in its simulations. Every node v in G_t has degree 2 or 3 (with some neighbors possibly not yet assigned), with P(v) = 1, LC(v) = 2. If v has degree 3 (i.e., is at level $\ell > 1$) then RC(v) = 3. G_t is a tree with at most k levels. \mathcal{P} maintains level(v) for each vertex v in G_t which will correspond to v's final level in the completed graph G. In particular, if level(v) = $\ell > 1$ then level(RC(v)) = $\ell - 1$. Finally, \mathcal{P} assigns IDs to newly added nodes serially so that the j^{th} node created has ID j.

 \mathcal{P} begins phase k, subphase 1 by simulating A from a vertex v_B with level $(v_B) = k$ and ID 1, and $\chi_{in}(v_B) = B$. During step t (i.e., when A makes its t^{th} query), if A queries a new node (necessarily a neighbor of some u in G_{t-1}) \mathcal{P} forms G_t by adding a corresponding node u_t to G_{t-1} whose ID and label maintain the invariants described above, and sets $\chi_{in}(u_t) = B$. At the end of subphase 1, every node u queried from the execution of A initiated at v_B has $\chi_{in}(u) = B$. Thus, by the claim, we must have $\chi_{out}(v_B) \neq R$. Moreover, by Condition 5 of validity (Definition 5.5), the output must satisfy $\chi_{out}(v_B) \neq D$, so that $\chi_{out}(v_B) \in \{B, X\}$. If $\chi_{out}(v_B) = X$, \mathcal{P} ends Phase k, and sets $v_{k-1} = \operatorname{RC}(v_B)$.

If $\chi_{out}(v_B) = B$, \mathcal{P} continues to subphase 2 as follows. \mathcal{P} simulates A from a new vertex $v_R \notin G_t$ —and not yet connected to G_t —exactly as in subphase 1, except that all nodes u created in this subphase have $\chi_{in}(u) = R$. When the execution of A initiated at v_R terminates, G_t consists of two connected components: a blue component (containing v_B) and a red component (containing v_R). As before, $\chi_{out}(v_R) \in \{R, X\}$, otherwise A does not solve Hierarchical-THC(k) on (some completion of) G_t . If $\chi_{out}(v_R) = X$, \mathcal{P} sets $v_{k-1} = \text{RC}(v_R)$ and ends Phase k. Otherwise (if $\chi_{out}(v_R) = R$), \mathcal{P} transforms G_t by connecting the blue and red components as follows. Let u_B be the highest ancestor of v_B in the blue component, and let w_R be the left-most descendant of v_R in the red component. In particular, $\text{level}(u_B) = \text{level}(w_R) = k$. \mathcal{P} includes an edge between u_B and w_R and sets $w_R = P(u_B)$, $u_B = \text{LC}(w_R)$. Thus, in G_t , v_B becomes a left descendant of v_R .

Consider the path $w_0 = v_R, u_1, \ldots, w_b = v_B$ from v_R to v_B . By construction, we have level $(w_i) = k$ for all i, and $b \leq 2m$ (since the red and blue components each have size at most m). Since $\chi_{out}(v_B) = B$ and $\chi_{out}(v_R) = R$, in any completion of G_t , there must either exist some w_i satisfying $\chi_{out}(w_i) = X$, or adjacent nodes w_{i-1}, w_i such that $\chi_{out}(w_{i-1}) = R$ and $\chi_{out}(w_i) = B$. Since the latter case violates validity Condition 5(b), some output of a correct algorithm Ashould satisfy the former condition. The procedure \mathcal{P} attempts to find such a vertex w_i by using binary search on the path (v_R, \ldots, v_B) . More formally, let u_1 be the midpoint of the path from v_R to v_B . In subphase 3, \mathcal{P} simulates an execution of A from u_1 . As before, every time a new node (not already in G_t) is queried, \mathcal{P} creates a new node u (with the next value of ID) and forms G_{t+1} by appending the node to G_t . The input color of u is chosen to be the same as the input color of the node from which u was queried. That is, if u is a child/parent of u', then $\chi_{in}(u) = \chi_{in}(u')$. Thus, at every step G_t consists of blue and red components connected by a single edge.

Consider the graph G_t at the end of the execution of A initiated at u_1 . As before, we must have $\chi_{out}(u_1) \in \{R, B, X\}$. (Note that u_1 may not satisfy the hypothesis of the claim above, as u_1 may query nodes of both input colors). If $\chi_{out}(u_1) = X$, \mathcal{P} ends phase k and sets $v_{k-1} = \operatorname{RC}(u_1)$. If $\chi_{out}(u_1) = R$ (resp. B), \mathcal{P} sets u_2 to be the midpoint of u_1 and v_B (resp. v_R) and repeats the procedure of the previous paragraph starting from u_2 . Continuing in this way \mathcal{P} produces a sequence u_1, u_2, \ldots, u_r of nodes in the path from v_R to v_B until either $\chi_{out}(u_r) = X$, or adjacent nodes in the path output R and B, respectively, thus contradicting validity condition 5(b). Since \mathcal{P} uses binary search to determine u_{i+1} from the previous simulations, phase k requires



Figure 8: Illustration of the process \mathcal{P} interacting with a deterministic algorithm A purportedly solving Hierarchical-THC(3). The top figure shows the partial instance constructed at the end of the second subphase of phase k = 3, assuming the nodes v_B and v_R output B and R respectively. The second image shows the region explored by the node u_1 . Assuming u_1 outputs X, the third image gives the result of the first sub-phase of phase 2, showing the region explored by v_2 . Finally, if v_2 outputs X, the fourth image shows the region explored by v_1 . Since v_1 sees only red vertices in the subtree below it, and v_1 's parent, v_2 , outputs X, v_2 must output R. However, by construction, v_1 does not query a leaf. By appending a blue leaf to the region explored by v_1 , we \mathcal{P} forces A to produce an incorrect output on the instance.

 $O(m \log m)$ queries (i.e., $|G_t| = O(m \log m)$ at the end of phase k).

If \mathcal{P} has not found outputs violating validity at the end of phase k, then \mathcal{P} stores a node v_{k-1} such that $v_{k-1} = \operatorname{RC}(\operatorname{P}(v_k))$, $\operatorname{level}(v_{k-1}) = k - 1$, and $\chi_{\operatorname{out}}(\operatorname{P}(v_{k-1})) = X$. By validity condition 5(a), A must satisfy $\chi_{\operatorname{out}}(v_{k-1}) \neq D$. Without loss of generality, assume that v_{k-1} is in the red component of G_t so that $\chi_{\operatorname{in}}(v_{k-1}) = R$. (If $\chi_{\operatorname{in}}(v_{k-1}) = B$, interchange the roles of R and B in the following discussion.) \mathcal{P} begins the first subphase of phase k - 1 by simulating an execution of A initiated at v_{k-1} , where again the input color of each new node in G_t is consistent with the red or blue component to which the new node is appended. Since this implies that all descendants of v_{k-1} have input color R, v_{k-1} should output in $\{R, X\}$. Again, if $\chi_{\operatorname{out}}(v_{k-1}) = X$, \mathcal{P} sets $v_{k-2} = \operatorname{RC}(v_{k-1})$ and ends phase k - 1. Otherwise, if $\chi_{\operatorname{out}}(v_{k-1}) = R$, then \mathcal{P} forms a new vertex v'_{k-1} with input color B with level $(v_{k-1}) = k - 1$. \mathcal{P} then simulates an execution of A from v'_{k-1} in a new blue component disconnected from the component of G_t constructed so far, just as it did for v_B . \mathcal{P} additionally maintains the invariant that all new nodes in this component are at level at most k - 1 by setting labels so that v'_{k-1} is the left-most descendant of all its ancestors of v'_{k-1}

Since v'_{k-1} is at level k-1, the claim implies that correct output can satisfy $\chi_{out}(v'_{k-1}) \in \{B, D, X\}$. Again if $\chi_{out}(v'_{k-1}) = X$, \mathcal{P} sets $v_{k-2} = \operatorname{RC}(v_{k-1})$ and ends phase k-1. Otherwise, if $\chi_{out}(v'_{k-1}) \in \{B, D\}$, the component of G_t containing v'_{k-1} is attached to the component containing v_{k-1} so that v'_{k-1} becomes a left descendant of v_{k-1} . Since $\chi_{out}(v_{k-1}) = R$ and $\chi_{out}(v'_{k-1}) \in \{B, D\}$, validity conditions 4 now imply that there is some node v''_{k-1} on the path between v_{k-1} and v'_{k-1} such that $\chi_{out}(v'_{k-1}) = X$. \mathcal{P} finds such a node v''_{k-1} using binary search (simulating at most log m more executions of A), and sets $v_{k-2} = \operatorname{RC}(v''_{k-1})$.

Continuing in this way, \mathcal{P} either finds a node violating one of the validity conditions, or it constructs a sequence $v''_k, v''_{k-1}, \ldots, v''_2$ such that level $(v''_i) = i$ and $\chi_{out}(v''_i) = X$. Now consider the graph G_t at the beginning of phase 1, and let $v_1 = \operatorname{RC}(v''_2)$. Assume without loss of generality that v_1 lies in a red component, so that $\chi_{in}(v_1) = R$, and similarly for all of v_1 's descendants of G_t constructed so far. \mathcal{P} simulates v_1 , and appends to G_t accordingly, maintaining that all of v_1 's descendants input colors are R. As before, the claim implies that $\chi_{out}(v_1) \neq B$, and validity condition 3(a) implies that $\chi_{out}(v_1) \neq X$. Further, validity condition 4(b) (applied to $P(v_1) = v''_2$) implies that $\chi_{out}(v_1) \neq D$. Thus, we must have $\chi_{out}(v_1) = R$. Now \mathcal{P} appends to G_t a single leaf v'_1 that is a left descendant of v_1 and setting $\chi_{in}(v_1) = B$. By validity conditions 2 and 3(a), $\chi_{out}(v_1) \in \{B, D\}$. However, this together with $\chi_{out}(v_1) = R$ implies that some node v''_1 on the path from v_1 to v'_1 violates validity condition 3(b).

When the simulation of v_1 is completed, \mathcal{P} has simulated $O(k \cdot m \log m)$ nodes, hence $|G_t| = O(k \cdot m \log m)$. To complete G, \mathcal{P} appends nodes to each "unassigned" port that are consisted with the level of each node containing the unassigned port (with arbitrary input colors). Since there are at most $2 |G_t|$ unassigned ports, and a minimal tree with ℓ levels contains $O(\ell)$ nodes, the final graph G satisfies $n = |G| = O(k^2 m \log m)$. Therefore for constant k, we must have $m = \Omega(n/\log n)$, which gives the desired result.

6 Hybrid Balanced $2\frac{1}{2}$ Coloring

Here, we describe a family of LCLs, Hybrid-THC(k), that are hybrids of BalancedTree(k) introduced in Section 4 and Hierarchical-THC(k) described in Section 5. Like Hierarchical-THC(k), each node has an associated level $\ell \in [k + 1]$, which in the case of Hybrid-THC(k) is explicitly given to each node as part of its input label. Each connected component (of G_T induced by a tree labeling) at level $\ell = 1$ corresponds to an instance of BalancedTree, which may either be solved (with all nodes outputting B or U) or declined (with all nodes outputting D). Nodes at levels $\ell \geq 2$ induce an instance of Hierarchical-THC(k), except that validity conditions 4(b) and 5(a) are modified for level $\ell = 2$. Specifically, a level 2 node v is allowed to output X if and only if RC(v) (at level 1) outputs a value in $\{B, U\}$ —i.e., if the instance of BalancedTree below v is solved.

The hybrid LCL Hybrid-THC(k) described above is easier than Hierarchical-THC(k) in terms of *distance* complexity because the basic problem BalancedTree can be solved in each level 1 component with distance complexity $O(\log n)$ (Proposition 4.8). Since nodes at level $\ell \ge 2$ are distance $\ell - 1$ from (the root of) such a component, every node at any level $\ell \ge 2$ can simply output X, knowing that every level 1 sub-instance can be solved.

In terms of *volume* complexity, Hybrid-THC(k) is no easier than Hierarchical-THC(k) because there is no volume-efficient method of solving each level 1 sub-instance (Proposition 4.9). The same algorithmic technique used to solve Hierarchical-THC(k) in randomized volume $O(n^{1/k} \log^{O(k)} n)$ can also be applied to solve Hybrid-THC(k) with the same volume bound. As one would expect, the deterministic volume complexity of Hybrid-THC(k) is (nearly) linear.

Definition 6.1. For any fixed constant $k \in \mathbf{N}$, the problem Hybrid-THC(k) consists of the following:

Input: a colored balanced tree labeling together with a number $level(v) \in [k+1]$ for each $v \in V$

Output: for each $v \in V$ a string encoding either a pair $(\beta(v), p(v)) \in \{B, U\} \times \mathcal{P}$ or a symbol in $\{R, B, D, X\}$

Validity: depending on the input label $\ell = \text{level}(v)$, the output must satisfy

- $\ell = 1$: the output is valid in the subgraph of all nodes w satisfying level(w) = 1 in the sense of Definition 4.1, or $\chi_{\text{out}}(v) = D$ and all level 1 neighbors of v (in G_T) also output D
- $\ell = 2$: the output satisfies conditions 2 and 4 of Definition 5.5, except that 4(b) is replaced by " $\chi_{out}(v) = X$ and $\chi_{out}(RC(v)) \in \{B, U\}$ "
- $\ell > 2$: the output is valid in the sense of Definition 5.5

The following lemma is clear from previous discussion.

Lemma 6.2. For every fixed positive integer k, Hybrid-THC(k) is an LCL.

We state the main theorem of this section below. As noted at the beginning of the section, the proofs of the various claims in Theorem 6.3 are analogous to the results appearing in Sections 4 and 5. Details are left to the reader.

Theorem 6.3. For each fixed positive integer k, the complexity of Hybrid-THC(k) satisfies

$$\begin{split} & \text{R-DIST}(\mathsf{Hybrid-THC}(k)) = \Theta(\log n), \\ & \text{D-DIST}(\mathsf{Hybrid-THC}(k)) = \Theta(\log n), \\ & \text{R-VOL}(\mathsf{Hybrid-THC}(k)) = \widetilde{\Theta}(n^{1/k}), \\ & \text{D-VOL}(\mathsf{Hybrid-THC}(k)) = \widetilde{\Theta}(n). \end{split}$$

6.1 More Complexity Classes

In this final technical section, we describe a family of LCLs—called *hierarchical-or-hybrid* $2\frac{1}{2}$ coloring (HH-THC) with (randomized and deterministic) volume complexity $\Theta(n^{1/\ell})$, randomized volume complexity $\widetilde{\Theta}(n^{1/k})$, and deterministic volume complexity $\widetilde{\Theta}(n)$ for any $k, \ell \in \mathbb{N}$ with $k \leq \ell$. The idea of HH-THC is that each node v receives an input label for Hierarchical-THC(ℓ) or Hybrid-THC(k) together with a single bit b_v . The nodes with input bit $b_v = 0$ should solve Hierarchical-THC(ℓ), while the nodes with input bit $b_v = 1$ should solve Hybrid-THC(k).

Definition 6.4. For any fixed constants $k, \ell \in \mathbb{N}$, $k \leq \ell$, the problem HH-THC (k, ℓ) consists of the following:

- Input: a colored balanced tree labeling together with a number $level(v) \in [k+1]$ and bit $b_v \in \{0,1\}$ for each $v \in V$
- *Output:* for each $v \in V$ a string encoding either a pair $(\beta(v), p(v)) \in \{B, U\} \times \mathcal{P}$ or a symbol in $\{R, B, D, X\}$
- Validity: if G_0 and G_1 are the induced subgraphs of G consists of nodes with inputs $b_v = 0$ and $b_v = 1$ respectively, then
 - the output labeling in G_0 is a valid output for Hierarchical-THC(ℓ) (Definition 5.5) (with the input level level(v) ignored)
 - the output labeling in G_1 is a valid output for Hybrid-THC(k) (Definition 6.1)

Since determining membership in G_0 or G_1 can be done locally, HH-THC (k, ℓ) is an LCL. To analyze, say, the randomized distance complexity of HH-THC (k, ℓ) , observe that each node v can solve HH-THC (k, ℓ) using distance at most the maximum of $O(\text{R-DIST}(\text{Hierarchical-THC}(\ell)))$ and O(R-DIST(Hybrid-THC(k))). Thus,

 $R\text{-}DIST(\mathsf{HH}\text{-}\mathsf{THC}(k,\ell)) = O(\max\{\text{R}\text{-}DIST(\mathsf{Hierarchical}\text{-}\mathsf{THC}(\ell)), \text{R}\text{-}DIST(\mathsf{Hybrid}\text{-}\mathsf{THC}(k))\})$ $= O(n^{1/\ell}).$

Arguing similarly for the other complexity measures, we obtain the following result.

Theorem 6.5. For all positive integers k, ℓ with $k \leq \ell$ the complexity of HH-THC (k, ℓ) satisfies

$$\begin{aligned} & \text{R-DIST}(\mathsf{HH-THC}(k,\ell)) = \Theta(n^{1/\ell}), \\ & \text{D-DIST}(\mathsf{HH-THC}(k,\ell)) = \Theta(n^{1/\ell}), \\ & \text{R-VOL}(\mathsf{HH-THC}(k,\ell)) = \widetilde{\Theta}(n^{1/k}), \\ & \text{D-VOL}(\mathsf{HH-THC}(k,\ell)) = \widetilde{\Theta}(n). \end{aligned}$$

7 Discussion and Open Questions

7.1 Denser and Deterministic Volume Hierarchies

In the preceding sections, our results establish a randomized volume hierarchy for LCL problems: for every $k \in \mathbf{N}$ there exists an LCL whose randomized volume complexity is $\tilde{\Theta}(n^{1/k})$. Moreover, this hierarchy persists even when restricting to problems whose randomized and deterministic distance complexities are both $\Theta(\log n)$. However, we do not establish an analogous hierarchy for *deterministic* volume complexity.

Question 7.1. Does there exist an LCL Π whose deterministic volume complexity is in $\omega(\log^* n) \cap o(n)$?

We conjecture a negative answer to Question 7.1. The conjecture holds in the relaxed setting where an algorithm is given not n in advance and unique IDs are not required to be polynomial in n. The argument is as follows. Let A be any deterministic algorithm, and let G = (V, E) be a (large) instance of an LCL II such that VOL $(A, G, \mathcal{L}, v) \ll n$. For fixed $v \in V$, let H be the induced (labeled) subgraph of G consisting of nodes that are either queried by A in an execution initiated from v, or neighbors of nodes queried by A. An instance of Ainitiated at v in H will query precisely the same nodes and give the same output as it did in G. Since II is an LCL, the solution found by A on H is also a valid solution to II. But VOL $(A, H, \mathcal{L}', v) = \text{VOL}(A, G, \mathcal{L}, v) \ge |H|/(\Delta - 1)$. Thus, A queries a constant fraction of H. If D-DIST(II) = $\Omega(\log^* n)$, then we can construct an infinite family of graphs H as above such that any deterministic algorithm requires linear volume on H, thus implying that D-VOL(II) = $\Omega(n)$. Recall from Section 1.2 that the gap between randomized and deterministic volume is at most exponential, and hence finding any LCL whose *randomized* volume complexity is in $\omega(\log^* n) \cap o(\log n)$ would immediately imply a positive answer to Question 7.1. Thus, our conjecture also implies that no such LCLs exist.

We believe that the complexity classes for R-VOL described in this paper—namely problems with complexities roughly $n^{1/k}$ for all $k \in \mathbf{N}$, and $\log n$ —are far from exhaustive.

Question 7.2. For what values of $\alpha \in [0, 1]$ is there an LCL problem Π with R-VOL $(\Pi) = \widetilde{\Theta}(n^{\alpha})$? Are there problems with complexities in $\omega(\log n) \cap (\bigcap_{k \in \mathbb{N}} O(n^{1/k}))$?

We conjecture that problems Π with R-VOL $(\Pi) = \widetilde{\Theta}(n^{\alpha})$ exist for all α in a dense subset of [0,1]. That is, for every $\alpha' \in [0,1]$ and $\varepsilon > 0$, there exists an $\alpha \in [0,1]$ with $|\alpha' - \alpha| < \varepsilon$ and an LCL Π with R-VOL $(\Pi) = \widetilde{\Theta}(n^{\alpha})$. We believe that LCL constructions similar to those described in [2] will yield more R-VOL complexity classes.

7.2 Graph Shattering

The LCLs described in preceding sections all have deterministic and randomized distance complexities $\Omega(\log n)$. However, an interesting class of LCLs have randomized and deterministic distance complexities between $\Omega(\log \log n)$ and $O(\log n)$. A canonical example of such a problem is *sinkless orientation* (SO), whose randomized distance complexity is $\Theta(\log \log n)$, and whose deterministic distance complexity is $\Theta(\log n)$.

Question 7.3. What are D-VOL(SO) and R-VOL(SO)?

We note that a negative answer to Question 7.1 would also settle Question 7.3. Indeed, if D-VOL(SO) = $\Omega(n)$, then R-VOL(SO) = $\Omega(\log n)$, as randomness helps at most exponentially [13], but at the same time we also have R-VOL(SO) = $O(\log n)$, since we can simulate $O(\log \log n)$ -distance algorithms with $O(\log n)$ volume. Conversely, R-VOL(SO) = $o(\log n)$ would imply D-VOL(SO) = o(n), thus giving a positive answer to Question 7.1.

7.3 Complexity Classes with Restricted Bandwidth

The query model we consider, together with the associated volume complexities, can be viewed as a refinement of the LOCAL model in distributed computing, where an algorithm incurs a cost for each query made to a node. Like round complexity in the LOCAL model, volume complexities are purely combinatorial. The CONGEST model [41] is a refinement of the LOCAL model where the complexity measure is communication based: in each round, each node can send at most B (typically $O(\log n)$) bits to each of its neighbors. We believe it is interesting to compare the relative power of the query model and the CONGEST model. We observe that fairly naive bounds on the relative complexities in the query and CONGEST models are the best possible.

Observation 7.4. Suppose a problem Π can be solved in T rounds in the deterministic CONGEST model. Then D-VOL(Π) = $\Delta^{O(T)}$. This follows immediately from Lemma 2.5, as T rounds in the CONGEST model can trivially be simulated in T rounds in the LOCAL model. The same relationship holds for randomized algorithms: if Π can be solved in T rounds of CONGEST with randomness, then R-VOL(Π) = $\Delta^{O(T)}$. Moreover, these bounds are tight. BalancedTree can be solved in $O(\log n)$ rounds of CONGEST by each inconsistent/incompatible node v announcing this defect to its neighbors in O(1) rounds. Then after $O(\log n)$ rounds of flooding (each node simply rebroadcasts an "inconsistent" message heard from any neighbor), every unbalanced node will witness an inconsistency, thus allowing it to (correctly) output. By the $\Omega(n)$ query lower bound of $\Omega(n)$ posited in Proposition 4.9, the bounds R-VOL(Π), D-VOL(Π) = $\Delta^{O(T)}$ are the best possible even when restricting attention to LCL problems.

Observation 7.5. Suppose Π is a problem with deterministic (resp. randomized) volume complexity D, such that the input of each node is of size $O(\log n)$ (including random bits in the randomized case). Then Π can be solved in $\Delta^{O(D)}$ rounds in the deterministic (resp. randomized) CONGEST model as follows. Using a simple flooding procedure, each node can learn its distance D neighborhood in $\Delta^{O(D)}$ rounds of CONGEST. Each node then simulates the query based algorithm on its D-neighborhood. Example 7.6 shows that the bound $\Delta^{O(D)}$ is the best possible, though the example is *not* an LCL.

Example 7.6. Let G = (V, E) be a graph on $n = 2(2^{k+1} - 1)$ nodes constructed as follows. G consists of two balanced binary trees of depth k rooted at nodes u and v respectively, together with an edge between u and v. All internal nodes have a (specified) left and right child. Let $u_1, u_2, \ldots, u_{2^k}$ and $v_1, v_2, \ldots, v_{2^k}$ be the leaves below u and v, respectively, where u_1 is u's left-most descendant, u_{2^k} is u's right-most descendant, and similarly for v_1, \ldots, v_{2^k} . Each v_i has an input bit b_i . To solve the problem Π , each u_i must output b_i (initially stored in v_i). This problem can easily be solved in $O(\log n)$ queries. However, in the CONGEST model, Π requires $\Omega(n/B)$ rounds. Indeed, to solve Π , the entire vector $b_1b_2\cdots b_{2^k}$ must be transmitted across the single edge $\{u, v\}$, which requires $\Omega(2^k/B) = \Omega(n/B)$ rounds.

While the problem Π in Example 7.6 shows that in general the CONGEST round complexity can be exponentially larger than the volume complexity, Π is not an LCL. It is not clear to us if this exponential gap is achievable for an LCL problem.

Question 7.7. What is the largest possible gap between volume complexity and CONGEST round complexity for an LCL? Are there LCLs with (deterministic) volume complexity D that require $\Delta^{\Omega(D)}$ rounds in the CONGEST model?

7.4 Pushing, Pulling, and MPC

In our volume model (and LCA models), an algorithm interacts with the graph by "pulling" information. That is, the output of a node v is determined by a sequence of (adaptively chosen) read operations; there is no direct interaction between executions instantiated from different nodes. One could consider a "push" model, in which an algorithm can send messages to other visited nodes in the network. An execution of an algorithm in such a model could proceed in three phases as follows:

- 1. Each node adaptively queries its local neighborhood using at most Q_{pull} queries (as in our volume model).
- 2. Each node sends messages to at most Q_{push} nodes queried in phase 1.
- 3. Nodes output based on the result of the queries made in phase 1 and messages received in phase 2.

Thus, such a model would allow for some limited interaction between nodes. Equivalently, this model can be viewed as a variant of the volume model where an execution of an algorithm is granted read-write access to (some subset of) nodes it visits, rather than read-only access. One could further generalize this model by allowing multiple iteration of the phases above, or limiting the size of messages sent in phase 2.

It is clear that an algorithm as above can be simulated in Q_{pull} rounds of LOCAL computation. The pull/push model is trivially at least a strong as our volume model as well, as our volume model is the special case with $Q_{\text{push}} = 0$. However, it is not clear how much more computational power is afforded by allowing algorithms to push messages. The following example shows that pushing can give an exponential improvement in the complexity of a problem. **Example 7.8.** Consider the problem BalancedTree. By Theorem 4.5, R-VOL(BalancedTree) = $\Omega(n)$. However, BalancedTree can be solved in the pull/push model using $O(\log n)$ queries. In phase 1, each node queries its O(1) distance neighborhood to determine if it is compatible in the sense of Definition 4.2. If so, v waits until phase 3. Otherwise, v queries its $\log n$ nearest ancestors (or until a root is encountered). In phase 2, an incompatible node sends its local view (after phase 1) to all of its ancestors. In phase 3, incompatible nodes output (U, \perp) . A compatible node receiving an "incompatible" message from a descendant outputs in accordance with the validity condition of Definition 4.3 (choosing the direction of the left-most nearest incompatible descendant). All other nodes output (B, P(v)). By Lemma 4.6, all unbalanced internal nodes will receive an incompatible message from a descendant, so it is straightforward to verify that this procedure solves BalancedTree with $Q_{push}, Q_{pull} = O(\log n)$. This upper bound is tight, as R-DIST(BalancedTree) = $\Omega(\log n)$.

Example 7.8 shows that allowing algorithms read-write access to queried nodes can give an exponential improvement in volume complexity. Moreover, the example gives an LCL whose *deterministic* push-pull query complexity is $\Theta(\log n)$. We are unaware of any LCLs with deterministic volume complexity $\omega(\log^* n)$ and o(n), and indeed we suspect that no such LCLs exist. Thus, it may be the case that randomness plays a much smaller role in the push-pull volume model than it does in the pull-only volume model.

Question 7.9. What does the volume complexity landscape look like in the push-pull volume model? How is it different from the pull-only volume model?

One reason we believe the push-pull model may be interesting, is that it captures an aspect of interactivity of message passing models (i.e., LOCAL and CONGEST), but restricts the type of information that can be conveyed from one node to another. Thus it may be easier to reason about algorithms in the push-pull model than, say, CONGEST algorithms. In particular, we believe that efficient push-pull algorithms may be valuable in designing volume efficient protocols in the MPC model. For example, the $O(\log n)$ distance algorithm we describe for BalancedTree can be simulated using O(n) space and $O(\log \log n)$ rounds in the MPC model. However, a sparsified version of the push-pull algorithm described in Example 7.8 solves BalancedTree using $O(n^c)$ space and $O(\log \log n)$ rounds for any positive constant c. (The $O(\log \log n)$ run-time is achieved by using graph exponentiation to propagate "incompatible" messages upward in the network. To achieve the same effect with $O(n^c)$ space, note that each node needs only to see its leftmost nearest incompatible neighbor. Thus, in each step of the graph exponentiation procedure, it suffices to propagate a single "incompatible" message upward from each node, requiring only $O(\log n)$ space per step.)

Question 7.10. What algorithms using volume VOL in the push-pull model can be simulated using $O(\text{VOL} + n^c + \Delta)$ space in the MPC model? Does the analogue of Lemma 2.6 hold in the push-model?

7.5 Space and Time Efficient Algorithms in MPC

While Lemma 2.6 shows that volume-efficient algorithms can be simulated in a space-efficient manner, the stated round complexity may be large. For example, consider a problem Π with distance and volume complexities R-DIST(Π) = $O(\log n)$ and R-VOL(Π) = $O(n^c)$. Then an MPC simulation of an $O(\log n)$ round LOCAL algorithm using graph exponentiation gives an O(n)-space, $O(\log \log n)$ -round algorithm, while Lemma 2.6 gives $O(n^c)$ -space and $O(n^c)$ rounds. Thus the improved volume efficiency may come at the cost of a doubly-exponential increase in run-time! For an arbitrary algorithm using volume VOL, it does not seem likely that we can achieve, say, O(VOL) space and $O(\log \text{VOL})$ rounds by using graph exponentiation (or something similar). One reason for this potential inefficiency of simulating volume-efficient algorithms in

the MPC model is that in the volume model, queries are adaptive. That is, executions do not know in advance which nodes will be queried. Thus graph exponentiation cannot be used naively without the ability to "guess" which nodes will actually be queried in a given execution. Yet, for some cases—such as our $O(\log n)$ -volume algorithm for LeafColoring—graph exponentiation can be used: Algorithm 1 can easily be simulated in $O(n^c)$ space and $O(\log \log n)$ volume.

Question 7.11. What subclass of algorithms A with (randomized) volume complexity VOL can be simulated in $O(\text{VOL} + n^c + \Delta)$ space and $O(\log \text{VOL})$ rounds in the MPC model? What about in the push-pull model?

7.6 Randomness

In this final section, we pose some questions related to randomness in the volume model. Throughout the paper, we assume that randomness is provided via a private random string $r_v \colon \mathbf{N} \to \{0, 1\}$ for each vertex $v \in V$. When an algorithm A executed from a node v queries another node w, A has access to r_w . We call this model the **private randomness** model. In our model, we assumed that an algorithm A accesses random strings sequentially, and that the number of bits accessed by A is bounded with high probability—some assumption of this flavor is needed in the proof of the derandomization result by Chang et al. [13, Theorem 3]. We believe that this assumption is not necessary, at least for LCLs.

Question 7.12. Suppose Π is an LCL and A an algorithm solving Π in R(n) rounds in the private randomness model. Is there always an algorithm A' and a function $f: \mathbb{N} \to \mathbb{N}$ that solves Π in O(R(n)) rounds with high probability using f(n) random bits per node? If so, how slowly can f(n) grow?

In addition to the amount of randomness used per node, we think it is interesting to consider other random models. We describe three below, in decreasing power of computation.

public randomness: There is a single random string $r: \mathbf{N} \to \{0, 1\}$ that is seen by every node.

private randomness: Each node v has an independent random string $r_v \colon \mathbf{N} \to \{0, 1\}$. When v is queried, r_v is given to the process querying v.

secret randomness: Each node v has an independent random string r_v , but r_v is known only to v—algorithms querying v do not have access to r_v .

It is straightforward to show that any private random protocol can be simulated in the public random model, and that any secret random protocol can be simulated in the private randomness model. However, it is not clear if there is strict separation in the computational power of these models for LCLs.

Question 7.13. Are there strict separations between the public, private, and secret randomness models for LCLs?

We suspect that the public and private randomness models are essentially the same for LCL problems. On the other hand, in all of the randomized algorithms described in this paper, randomized coordination (i.e., non-secret randomness) seems essential. Since secret randomness does not allow for coordinated random choices between nodes, it is not clear that secret randomness should give much power over deterministic computation.

To see an example where secret randomness does help, consider the promise version of LeafColoring (Section 3) where all leaves are promised to have the same input color. To solve the promise problem, it is enough for all internal nodes to query a single leaf. Using secret randomness, each internal node can perform a "downward" random walk, and all internal nodes will visit some leaf with high probability after $O(\log n)$ queries. (The analysis is identical to the proof of Proposition 3.11.) Nonetheless, we do not know of any non-promise LCL problem for which there is a gap between secret randomness and deterministic volume complexities.

Acknowledgments

We thank Alkida Balliu, Faith Ellen, Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, Dennis Olivetti, and Jara Uitto for numerous discussions related to the volume complexity, in particular related to the preliminary observations in Section 1.2.

References

- Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient Local Computation Algorithms. In Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012), pages 1132–1139, 2012.
- [2] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. Almost global problems in the LOCAL model. In Proc. 32nd International Symposium on Distributed Computing (DISC 2018), Leibniz International Proceedings in Informatics (LIPIcs), pages 9:1–9:16. Schloss DagstuhlLeibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.DISC.2018.9.
- [3] Alkida Balliu, Juho Hirvonen, Janne H Korhonen, Tuomo Lempiäinen, Dennis Olivetti, and Jukka Suomela. New classes of distributed time complexity. In Proc. 50th ACM Symposium on Theory of Computing (STOC 2018), pages 1307–1318. ACM Press, 2018. doi:10.1145/3188745.3188860.
- [4] Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. The Distributed Complexity of Locally Checkable Problems on Paths is Decidable. In Proc. 38th ACM Symposium on Principles of Distributed Computing (PODC 2019), pages 262–271. ACM Press, 2019. doi:10.1145/3293611.3331606.
- [5] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. How much does randomness help with locally checkable problems?, 2019. URL http://arxiv.org/abs/1902. 06803.
- [6] Alkida Balliu, Juho Hirvonen, Christoph Lenzen, Dennis Olivetti, and Jukka Suomela. Locality of not-so-weak coloring. In Proc. 26th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2019), volume 11639 of LNCS, pages 37–51. Springer, 2019. doi:10.1007/978-3-030-24922-9_3.
- [7] Alkida Balliu, Juho Hirvonen, Dennis Olivetti, and Jukka Suomela. Hardness of Minimal Symmetry Breaking in Distributed Computing. In Proc. 38th ACM Symposium on Principles of Distributed Computing (PODC 2019), pages 369–378. ACM Press, 2019. doi:10.1145/3293611.3331605.
- [8] Sebastian Brandt. An Automatic Speedup Theorem for Distributed Problems. In Proc. 38th ACM Symposium on Principles of Distributed Computing (PODC 2019), pages 379–388. ACM Press, 2019. doi:10.1145/3293611.3331611.
- [9] Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A lower bound for the distributed Lovász local lemma. In Proc. 48th ACM Symposium on Theory of Computing (STOC 2016), pages 479–488. ACM Press, 2016. doi:10.1145/2897518.2897570.
- [10] Sebastian Brandt, Juho Hirvonen, Janne H Korhonen, Tuomo Lempiäinen, Patric R J Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela, and Przemysław Uznański. LCL problems on grids. In Proc. 36th ACM Symposium on Principles of Distributed Computing (PODC 2017), pages 101–110. ACM Press, 2017. doi:10.1145/3087801.3087833.

- [11] Andrea Campagna, Alan Guo, and Ronitt Rubinfeld. Local Reconstructors and Tolerant Testers for Connectivity and Diameter. In *Proc. APPROX/RANDOM 2013*, volume 8096 of *LNCS*, pages 411–424. Springer, 2013. doi:10.1007/978-3-642-40328-6_29.
- [12] Yi-Jun Chang and Seth Pettie. A Time Hierarchy Theorem for the LOCAL Model. SIAM Journal on Computing, 48(1):33–69, 2019. doi:10.1137/17M1157957.
- [13] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An Exponential Separation between Randomized and Deterministic Complexity in the LOCAL Model. In Proc. 57th IEEE Symposium on Foundations of Computer Science (FOCS 2016), pages 615–624. IEEE, 2016. doi:10.1109/FOCS.2016.72.
- [14] Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. The Complexity of Distributed Edge Coloring with Small Palettes. In Proc. 29th ACM-SIAM Symposium on Discrete Algorithms (SODA 2018), pages 2633–2652. Society for Industrial and Applied Mathematics, 2018. doi:10.1137/1.9781611975031.168.
- [15] Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986. doi:10.1016/S0019-9958(86)80023-7.
- [16] Talya Eden and Will Rosenbaum. Lower Bounds for Approximating Graph Parameters via Communication Complexity. In Proc. APPROX/RANDOM 2018, volume 116 of Leibniz International Proceedings in Informatics (LIPIcs), pages 11:1–11:18. Schloss Dagstuhl– Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.11.
- [17] Guy Even, Moti Medina, and Dana Ron. Deterministic Stateless Centralized Local Algorithms for Bounded Degree Graphs. In Proc. 22th Annual European Symposium on Algorithms (ESA 2014), pages 394–405. Springer, 2014. doi:10.1007/978-3-662-44777-2_33.
- [18] Guy Even, Moti Medina, and Dana Ron. Best of two local models: Centralized local and distributed local algorithms. *Information and Computation*, 262:69–89, 2018. doi:10.1016/j.ic.2018.07.001.
- [19] Uriel Feige, Yishay Mansour, and Robert Schapire. Learning and inference in the presence of corrupted inputs. In Proc. 28th Conference on Learning Theory (COLT 2015), Proceedings of Machine Learning Research, pages 637–657, 2015.
- [20] Manuela Fischer and Mohsen Ghaffari. Sublogarithmic Distributed Algorithms for Lovász Local Lemma, and the Complexity Hierarchy. In Proc. 31st International Symposium on Distributed Computing (DISC 2017), pages 18:1–18:16, 2017. doi:10.4230/LIPIcs.DISC.2017.18.
- [21] Mohsen Ghaffari and Hsin-Hao Su. Distributed Degree Splitting, Edge Coloring, and Orientations. In Proc. 28th ACM-SIAM Symposium on Discrete Algorithms (SODA 2017), pages 2505–2523. Society for Industrial and Applied Mathematics, 2017. doi:10.1137/1.9781611974782.166.
- [22] Mohsen Ghaffari and Jara Uitto. Sparsifying Distributed Algorithms with Ramifications in Massively Parallel Computation and Centralized Local Computation. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 1636–1653. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2019. doi:10.1137/1.9781611975482.99.
- [23] Mohsen Ghaffari, David G Harris, and Fabian Kuhn. On Derandomizing Local Distributed Algorithms. In Proc. 59th IEEE Symposium on Foundations of Computer Science (FOCS 2018), pages 662–673, 2018. doi:10.1109/FOCS.2018.00069. URL http://arxiv.org/abs/ 1711.02194.

- [24] Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, and Yannic Maus. Improved Distributed Δ-Coloring. In Proc. 37th ACM Symposium on Principles of Distributed Computing (PODC 2018), pages 427–436. ACM, 2018. doi:10.1145/3212734.3212764.
- [25] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, Searching, and Simulation in the MapReduce Framework. In Proc. 22nd International Symposium on Algorithms and Computation (ISAAC 2011), volume 7074 of LNCS, pages 374–383. Springer, 2011. doi:10.1007/978-3-642-25591-5_39.
- [26] Mika Göös, Juho Hirvonen, Reut Levi, Moti Medina, and Jukka Suomela. Non-local Probes Do Not Help with Many Graph Problems. In Proc. 30th International Symposium on Distributed Computing (DISC 2016), volume 9888 of Lecture Notes in Computer Science, pages 201–214. Springer, 2016. doi:10.1007/978-3-662-53426-7_15.
- [27] Bala Kalyanasundaram and Georg Schintger. The Probabilistic Communication Complexity of Set Intersection. SIAM Journal on Discrete Mathematics, 5(4):545–557, 1992. doi:10.1137/0405044.
- [28] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010), pages 938–948. SIAM, Philadelphia, PA, 2010. doi:10.1137/1.9781611973075.76.
- [29] Christoph Lenzen and Roger Wattenhofer. Brief announcement: Exponential speed-up of local algorithms using non-local communication. In Proc. 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2010), pages 295–296. ACM Press, 2010. doi:10.1145/1835698.1835772.
- [30] Reut Levi, Dana Ron, and Ronitt Rubinfeld. Local Algorithms for Sparse Spanning Graphs. In Proc. APPROX/RANDOM 2014, volume 28 of Leibniz International Proceedings in Informatics (LIPIcs), pages 826–842. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.826.
- [31] Reut Levi, Guy Moshkovitz, Dana Ron, Ronitt Rubinfeld, and Asaf Shapira. Constructing near spanning trees with few local inspections. *Random Structures & Algorithms*, 50(2): 183–200, 2017. doi:10.1002/rsa.20652.
- [32] Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Local Computation Algorithms for Graphs of Non-constant Degrees. *Algorithmica*, 77(4):971–994, 2017. doi:10.1007/s00453-016-0126-y.
- [33] Nathan Linial. Locality in Distributed Graph Algorithms. SIAM Journal on Computing, 21 (1):193–201, 1992. doi:10.1137/0221015.
- [34] Yishay Mansour and Shai Vardi. A Local Computation Approximation Scheme to Maximum Matching. In Proc. APPROX/RANDOM 2013, volume 8096 of LNCS, pages 260–273. Springer, 2013. doi:10.1007/978-3-642-40328-6_19.
- [35] Yishay Mansour, Aviad Rubinstein, Shai Vardi, and Ning Xie. Converting Online Algorithms to Local Computation Algorithms. In Proc. 39th International Colloquium on Automata, Languages, and Programming (ICALP 2012), volume 7391 of LNCS, pages 653–664. Springer, 2012. doi:10.1007/978-3-642-31594-7_55.
- [36] Michael Mitzenmacher and Eli Upfal. Probability and Computing. Cambridge University Press, Cambridge, 2005. doi:10.1017/CBO9780511813603.
- [37] Moni Naor. A lower bound on probabilistic algorithms for distributive ring coloring. SIAM Journal on Discrete Mathematics, 4(3):409–412, 1991. doi:10.1137/0404036.

- [38] Moni Naor and Larry Stockmeyer. What Can be Computed Locally? SIAM Journal on Computing, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.
- [39] Alessandro Panconesi and Aravind Srinivasan. The local nature of Δ -coloring and its algorithmic applications. *Combinatorica*, 15(2):255–280, 1995. doi:10.1007/BF01200759.
- [40] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3): 183–196, 2007. doi:10.1016/j.tcs.2007.04.040.
- [41] David Peleg. Distributed Computing: A Locality-Sensitive Approach. Society for Industrial and Applied Mathematics, 2000. doi:10.1137/1.9780898719772.
- [42] A.A. Razborov. On the distributional complexity of disjointness. Theoretical Computer Science, 106(2):385–390, 1992. doi:10.1016/0304-3975(92)90260-M.
- [43] Omer Reingold and Shai Vardi. New techniques and tighter bounds for local computation algorithms. Journal of Computer and System Sciences, 82(7):1180–1200, 2016. doi:10.1016/j.jcss.2016.05.007.
- [44] Václav Rozho and Mohsen Ghaffari. Polylogarithmic-Time Deterministic Network Decomposition and Distributed Derandomization. In Proc. 52nd Annual ACM Symposium on Theory of Computing (STOC 2020), 2020. URL http://arxiv.org/abs/1907.10937.
- [45] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast Local Computation Algorithms. In Proc. 2nd Symposium on Innovations in Computer Science (ICS 2011), pages 223–238, 2011.