# Sinkless Orientation Made Simple

Alkida Balliu[*] Janne H. Korhonen[†] Fabian Kuhn[‡] Henrik Lievonen[§]
Dennis Olivetti[*] Shreyas Pai[§] Ami Paz[¶] Joel Rybicki[‖] Stefan Schmid[†]
Jan Studený[§] Jukka Suomela[§] Jara Uitto[§]

**Abstract**

The sinkless orientation problem plays a key role in understanding the foundations of distributed computing. The problem can be used to separate two fundamental models of distributed graph algorithms, LOCAL and SLOCAL: the locality of sinkless orientation is $\Omega(\log n)$ in the deterministic LOCAL model and $O(\log \log n)$ in the deterministic SLOCAL model. Both of these results are known by prior work, but here we give new simple, self-contained proofs for them.

## 1 Introduction

One of the fundamental challenges in the study of graph algorithms concerns the understanding of the *locality* of the considered graph problem: given a node in the middle of a large graph, *how far* do we need to see around that node to choose its output? For example, if we are interested in the graph coloring problem, how far do we need to see around a node before we can choose its color, so that the end result is a globally consistent coloring?

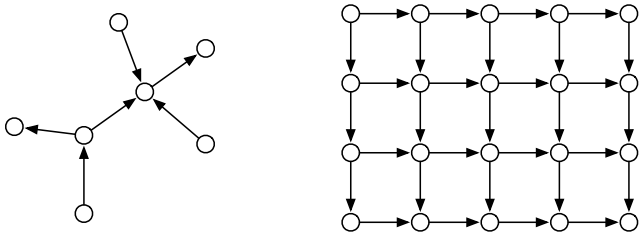
The notion of locality plays a particularly important role in characterizing the *distributed complexity* of graph problems [23, 24]—for example, problems that are local can be solved in a distributed setting with a small number of communication rounds. The past decade has seen a successful research program [1, 2, 9, 11, 14, 17, 18, 25] contributing to our systematic understanding of the fundamental interplay between locality, randomness, and the computational power of different models of distributed graph algorithms.

In this work, we give a new, simple proof for one of the key results in this area: the so-called sinkless orientation problem gives an exponential separation between the LOCAL and SLOCAL models of computing. The standard approach for proving this result relies on fairly heavy-weight machinery, whereas our new proof is elementary and self-contained.

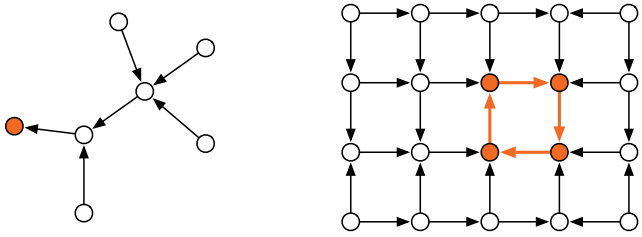
### 1.1 Sinkless Orientation.

In the *sinkless orientation* problem, we are given an undirected graph as input, and the task is to orient all edges so that all nodes of degree at least 3 have at least one outgoing edge (i.e., they are not sinks). Here are some examples of valid solutions:

[*]Gran Sasso Science Institute, Italy
[†]TU Berlin, Germany
[‡]University of Freiburg, Germany
[§]Aalto University, Finland
[¶]LISN, CNRS, France
[‖]IST Austria, Austria



A sinkless orientation always exists in any graph and it is easy to find given a global view of the input graph: Process each connected component separately. If the component is a tree, we can choose a leaf node $v$ and orient everything towards $v$. Otherwise there is a cycle $C$, and we can then orient $C$ in a consistent direction and orient all other edges towards $C$, breaking ties arbitrarily. The following figure illustrates both of these cases:



However, this simple algorithm is inherently global—the orientation of a given edge depends on information arbitrarily far from it. The key question that was first explicitly asked in 2016 [9] regards the locality of the sinkless orientation problem: Can one come up with a rule that always results in a sinkless orientation such that each edge is oriented based on the information that is within its radius-$T(n)$ neighborhood, where $n$ is the size of the input graph? How small can we make $T(n)$? Or is there some lower limit for $T(n)$ that we cannot cross? How are the answers affected by our model of computation? In this paper, we aim to give lower and upper bounds for $T(n)$ in the LOCAL and SLOCAL models of computation, and show that the sinkless orientation problem separates these models.

**1.2 LOCAL and SLOCAL Models.** In this work, we consider the LOCAL and SLOCAL models of (distributed) graph algorithms. For both models, the setting is as follows. We are given an input graph $G = (V, E)$ on $n$ nodes and the goal is to compute a sinkless orientation on $G$. Each node $v \in V$ has to produce a *local output*, which is in our case an orientation of all edges incident to $v$. The local output of $v$ is determined by an algorithm that has access to the information available in $G$ within distance $T(n)$ from $v$, where $T\colon \mathbb{N} \to \mathbb{N}$ is a function of the size of the input graph.

The key difference between the LOCAL and SLOCAL models is in the way nodes are processed (see Fig. 1):

**Deterministic LOCAL model:** Each node $v \in V$ chooses its local output *simultaneously in parallel* based on the information available within distance $T(n)$ from $v$. That is, each node $v$ maps its radius-$T(n)$ neighborhood in $G$ to an output value.

**Deterministic SLOCAL model:** The nodes are processed *sequentially* in some arbitrary order chosen by an adversary. When node $v \in V$ is processed, it chooses its internal state and output based on the information available within distance $T(n)$ in the graph. This information also includes the internal states of the nodes processed *before* node $v$.

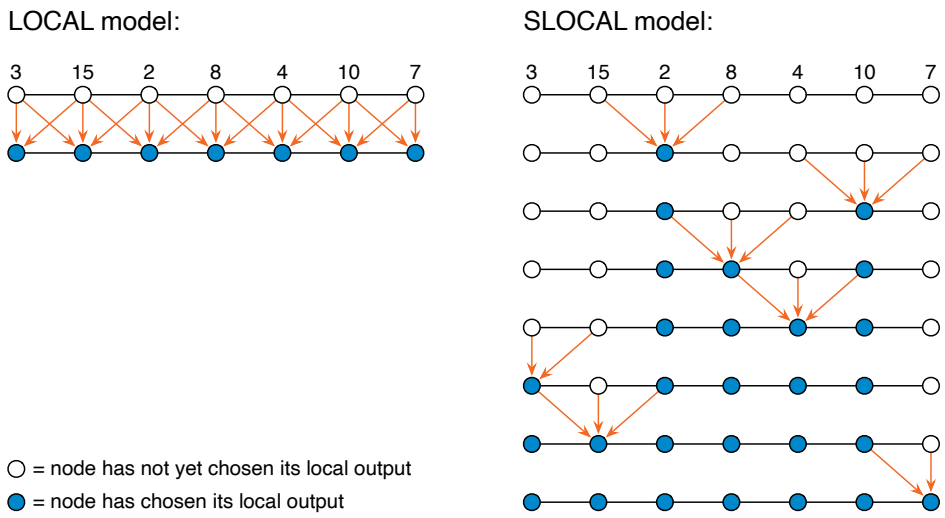



Figure 1: Information flow in the LOCAL and SLOCAL models; in this example we consider locality $T(n) = 1$, i.e., each node chooses its output based on its radius-1 neighborhoods. The input graph here is a path with 7 nodes; the nodes are labeled by the adversary with unique identifiers. In the LOCAL model, all nodes choose their final state simultaneously in parallel, while in the SLOCAL model, the nodes make their choices sequentially, in some order chosen by the adversary.

In both models, we assume that the number of nodes $n$ is known and that the nodes of the graph are labeled with unique identifiers from 1 to $\mathrm{poly}(n)$; this is particularly important for the LOCAL model so that one can distinguish nodes from each other.

The *locality of a graph problem* $\Pi$ is the smallest distance sufficient to solving $\Pi$ in the given model. That is, if $\Pi$ has locality $T(n)$, then there is an algorithm $\mathcal{A}$ such that

(1) the algorithm $\mathcal{A}$ uses only information available within distance $T(n)$ to compute the output of any node $v$, and

(2) the output of $\mathcal{A}$ is correct on any input graph, on any choice of unique identifiers, and, in the SLOCAL model, on any processing order of nodes.

**1.3 Is SLOCAL Any Stronger Than LOCAL?** It is natural to ask whether the locality of a given problem differs between these two models. It is straightforward to see that the locality in the SLOCAL model is always at most as large as in the LOCAL model: If we can solve problem $\Pi$ in the LOCAL model so that each node makes a choice based on its radius-$T(n)$ neighborhood, we can do the same in the SLOCAL model, as the algorithm can just ignore the internal states of nodes. However, this still leaves open the key question of whether the locality of a problem can be much smaller in the SLOCAL model than in the LOCAL model.

The answer may seem obvious. For example, consider the problem of coloring a path with 3 colors:

– In the SLOCAL model, we can solve this problem with locality $T(n) = 1$: each node can greedily pick a free color that is not yet used by any of its neighbors.

– In the LOCAL model, there is no constant-locality algorithm—this is the seminal result by Linial [23], and there exists a simple proof for it [22].

However, this problem only gives a slightly super-constant separation between the models: there is an algorithm with locality $T(n) = O(\log^* n)$ that solves the problem in the LOCAL model by making clever use of the unique identifiers [6, 13]; here $\log^*$ is the inverse of a power tower, i.e., a very slowly-growing function. More generally, for graphs with constant degree, any problem that can be solved in the SLOCAL model with locality $O(1)$ can be solved in the LOCAL model with locality $O(\log^* n)$ [18].

If we could always turn any SLOCAL algorithms into LOCAL algorithms with only $O(\log^* n)$ overhead in locality, this would be great news for the designers of distributed algorithms: it is often much easier to reason about sequential SLOCAL algorithms than about parallel LOCAL algorithms. However, sinkless orientation shows that this is not the case.

**1.4 Sinkless Orientation Separates LOCAL and SLOCAL.** Sinkless orientation can be used to prove a strong separation between deterministic LOCAL and deterministic SLOCAL. By prior work [9, 11, 16, 18], we know that:

THEOREM 1.1. *The locality of the sinkless orientation problem in the deterministic* LOCAL *model is* $\Omega(\log n)$.

THEOREM 1.2. *The locality of the sinkless orientation problem in the deterministic* SLOCAL *model is* $O(\log \log n)$.

Unfortunately, even though these results play a key role in understanding the landscape of models of distributed computing (see Section 4 for the broader context), there have not been simple proofs for either of these results. While the theorems are related to *deterministic* models, the prior proofs of Theorems 1.1 and 1.2 take a detour through *randomized* models, and apply fairly heavyweight machinery:

- The prior proof of Theorem 1.1 first shows that the locality is $\Omega(\log \log n)$ in the randomized LOCAL model [9]; this requires a careful analysis of how the local failure probability of a randomized algorithm behaves in the so-called *round elimination* technique. Then we can conclude that the locality is also $\Omega(\log \log n)$ in the deterministic LOCAL model. Finally, we can apply a general *gap result* to extend the lower bound to $\Omega(\log n)$ [11].
- The prior proof of Theorem 1.2 first constructs an algorithm with locality $O(\log \log n)$ in the randomized LOCAL model [16]; here one can use the so-called *shattering* technique, and argue that after the randomized shattering phase, which orients only some edges of the graph, the connected components of what remains to be processed are small enough so that even if one solves them deterministically, locality of $O(\log \log n)$ suffices. Then one can apply a generic derandomization result that enables the simulation of randomized LOCAL with deterministic SLOCAL [18], and the result follows.

**1.5 Contributions and Key Ideas.** We provide new short, elementary, and entirely self-contained proofs for Theorems 1.1 and 1.2.

**The lower bound.** To obtain Theorem 1.1, we prove a stronger lower bound: it turns out to be convenient to work in the *supported* version of the LOCAL model [26]. In the supported LOCAL model, there is a fixed *support graph* $G$ with a fixed assignment of unique identifiers that is known to all nodes in advance, and the input graph $H$ is a subgraph of $G$.

The fact that $G$ is globally known makes the supported model stronger than the usual LOCAL model (e.g. graph coloring with sufficiently many colors is trivial, as a proper vertex coloring of $G$ gives a proper vertex coloring of $H$). We show that the locality of the sinkless orientation problem is $\Omega(\log n)$ in the supported LOCAL model, which implies the same lower bound in the LOCAL model.

**The upper bound.** To prove Theorem 1.2, we introduce the *high-degree sinkless orientation problem*, in which we only care that nodes with high degree are not sinks. This problem is trivial to solve in the SLOCAL model. We then provide an SLOCAL algorithm which constructs a virtual graph on top of the actual graph and solves the high-degree sinkless orientation problem on the virtual graph. The algorithm then lowers the solution on the virtual graph to a solution for the ordinary sinkless orientation problem on the original graph.

## 2 Sinkless Orientation Has Locality $\Omega(\log n)$ in LOCAL

In this section, we show that the locality of the sinkless orientation problem in the deterministic LOCAL model is $\Omega(\log n)$. We in fact prove the lower bound in the stronger *supported* LOCAL *model*. In this variant of the LOCAL model, there is a globally known *support graph* $G = (V, E)$ with known assignment of unique identifiers, and the input is a subgraph $H$ of $G$. In an algorithm with locality $T(n)$, each node $v$ receives as input *the entire structure* of the support graph $G$, including all the unique identifiers, and information about which edges in its radius-$T(n)$ neighborhood in $G$, including the boundary edges, belong to $H$; we refer to edges of $H$ as input edges. In our case, we would like to find a sinkless orientation in the input graph $H$.

By working in the supported LOCAL model, we can obtain a fairly simple proof of the $\Omega(\log n)$ lower bound result. As we will see below, the support graph and the fixed unique identifier assignment allows us to easily apply the round elimination technique in the presence of the unique identifiers to show a lower bound directly for *deterministic* algorithms – without the hurdle of going through randomized algorithms as in prior proofs. While our new proof could be presented without explicit reference to the supported LOCAL model, the notion of a support graph is a convenient way to obtain the properties that are needed, and the supported LOCAL model is already an established model that has been studied independently in the literature [14, 19, 26]; see Section 4 for further discussion.

**Roadmap.** For technical convenience, we prove the result in a stronger *bipartite* version of the supported LOCAL model. The lower bound in this setting then implies lower bounds for (non-bipartite) LOCAL and supported LOCAL models, by observing that algorithms from a weaker model can be translated to the stronger models with no overhead in locality.

The overall structure of our lower bound proof is as follows. We fix a bipartite 5-regular graph $G$ with girth $\Omega(\log n)$, and an assignment of unique identifiers on $G$. We then show that in bipartite supported LOCAL, any algorithm that solves sinkless orientation, even with the promise that the support graph is $G$, has locality $\Omega(\log n)$.

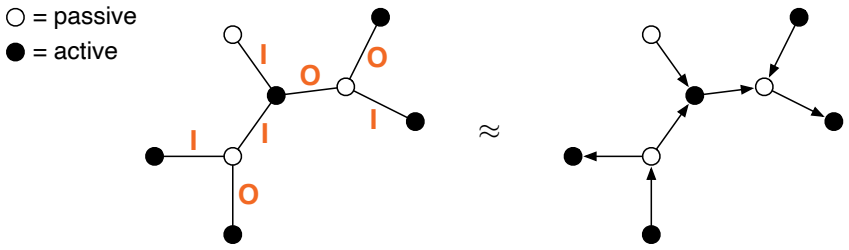
The proof has two main steps. First, we give a round elimination lemma showing that any sinkless orientation algorithm with locality $T$ on $H$ can be converted into an algorithm with locality $T - 1$, if $T$ is sufficiently less than the girth of $G$. By iterating this lemma, we can turn an algorithm with locality $T$ into an algorithm with locality 0. Second, we show that such trivial algorithms with locality 0 cannot exist, implying that any algorithm requires $\Omega(\log n)$ locality.

### 2.1 Setup

**Bipartite model.** In bipartite supported LOCAL, we are given a promise that the support graph $G$ is bipartite, and a 2-coloring is given to the nodes as an input; we refer to the two colors as black and white. In the bipartite model, we consider either the black or white nodes to be *active*, and the other color to be *passive*. All nodes of the graph run an algorithm as per the supported LOCAL model; upon termination of the algorithm, the active nodes produce an output, and the passive nodes output nothing. The outputs of the active nodes must form a globally valid solution; in particular, for the sinkless orientation problem,

the outputs of the active nodes already orient all edges, and neither active or passive nodes having degree at least 3 can be sinks.

**Sinkless orientation in bipartite model.** We encode the sinkless orientation problem in the bipartite supported LOCAL model as follows. Each active node outputs, for each incident input edge, one label from the alphabet $\Sigma = \{\mathsf{O}, \mathsf{I}\}$. The edge-output O indicates that the edge is outgoing from the active node, and the edge-output I indicates it is incoming to the active node:




An output is correct if

- for each *active* node of degree at least 3, there is at least one output O on an incident input edge, and
- for each *passive* node of degree at least 3, there is at least one output I on an incident input edge.

**Bipartite vs. non-bipartite.** One can immediately observe that any sinkless orientation algorithm for the supported LOCAL model can be translated to a sinkless orientation algorithm in the bipartite supported LOCAL model. Therefore, it follows that lower bounds for bipartite algorithms are also lower bounds for the standard models.

In more detail, consider a sinkless orientation algorithm $\mathcal{A}$ with locality $T$ in the (supported) LOCAL model, with some reasonable output encoding. To turn this into a bipartite (supported) LOCAL algorithm, one first runs algorithm $\mathcal{A}$ in the bipartite model—this requires no modifications, as computation in the bipartite model is done exactly as in the original. After the algorithm $\mathcal{A}$ has terminated,

(1) the passive nodes discard the output of $\mathcal{A}$ and output nothing, and

(2) the active nodes inspect the output of $\mathcal{A}$, and output I for each incident edge directed towards them, and O for each edge directed away from them in the output of $\mathcal{A}$.

Since $\mathcal{A}$ is a sinkless orientation algorithm, these outputs also guarantee that each passive node has one edge with output I incident to it.

**2.2 Step One: Round Elimination.** As the first step of the proof, we give a round elimination lemma for the sinkless orientation problem. In its basic form, a round elimination lemma shows that if there is an algorithm with locality $T > 0$ smaller than the lower bound, then we can use it to construct a new algorithm with locality $T - 1$ solving the same problem.

LEMMA 2.1. *Let $G$ be a fixed 5-regular bipartite graph with girth $g$, fixed unique identifiers, and a 2-coloring of the nodes. Let $0 < T < g/2$, and assume that there is an algorithm $\mathcal{A}_T$ that solves sinkless orientation with locality $T$ on graph $H$ supported by $G$. Then there is an algorithm $\mathcal{A}_{T-1}$ that solves sinkless orientation on $H$ with locality $T - 1$.*

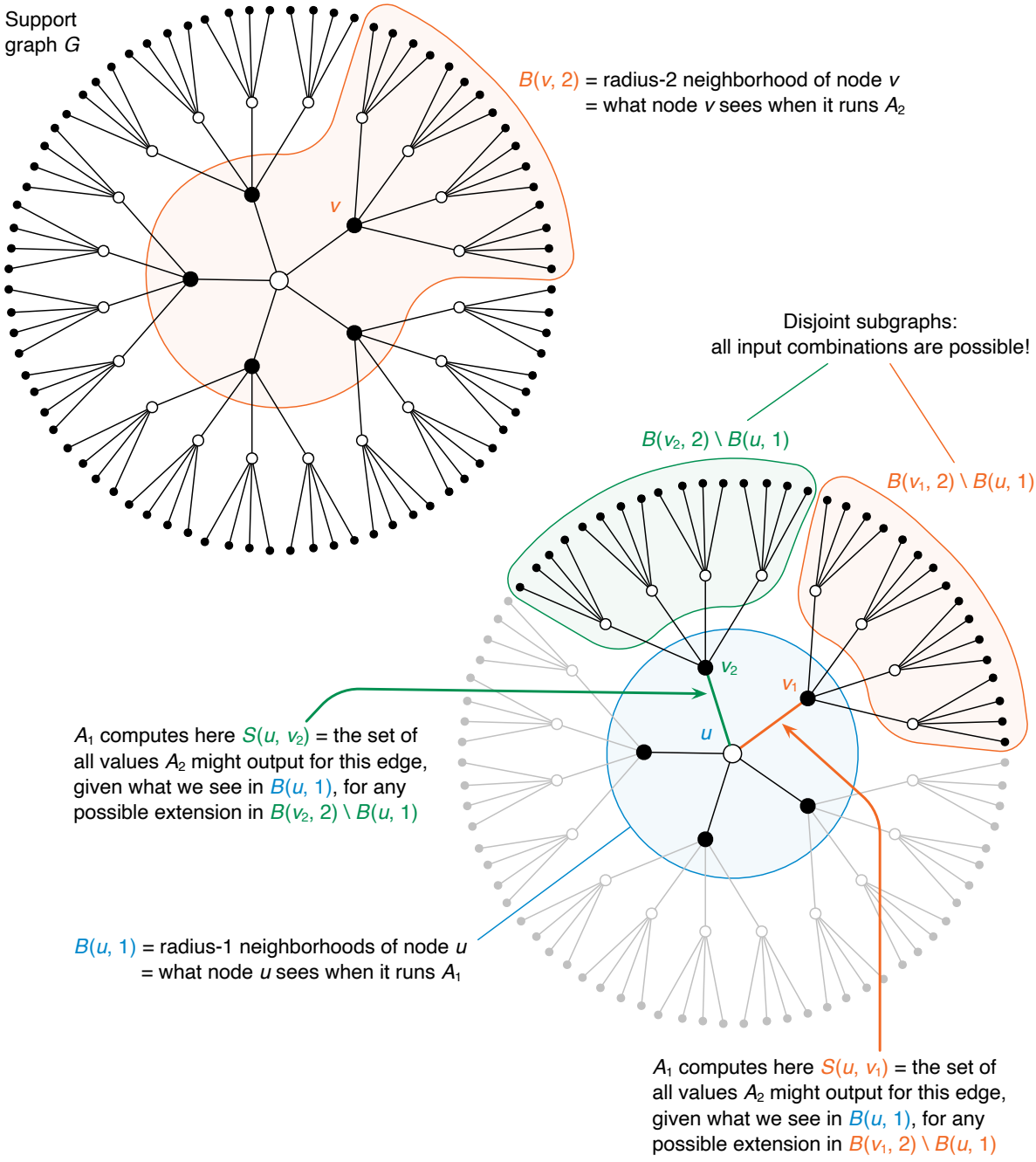



Figure 2: The proof of Lemma 2.1 (round elimination). Here we have illustrated the case of $T = 2$, i.e., we are given an algorithm $\mathcal{A}_2$ that solves sinkless orientation with locality 2 so that black nodes are active, and we construct a new algorithm $\mathcal{A}_1$ that solves the same problem with locality 1 so that white nodes are active.

*Proof.* Let us assume without loss of generality that black nodes are active in $\mathcal{A}_T$. For any non-negative integer $t$ and node $v \in V$, we use $B(v,t)$ to denote the nodes within distance $t$ from the node $v$ in the support graph $G$; see Fig. 2 for illustrations. We construct an algorithm $\mathcal{A}_{T-1}$ where white nodes are active. In the algorithm $\mathcal{A}_{T-1}$, each white node $u \in V$ performs the following steps:

(1) Node $u$ gathers the inputs in its $(T-1)$-radius neighborhood $B(u, T-1)$.

(2) For each neighbor $v$ of $u$, the node $u$ enumerates all possible input graphs $H'$ on $B(v,T)$ which are compatible with the actual input graph $H$ on $B(u, T-1)$. For each such $H'$, $u$ simulates $\mathcal{A}_T$ to compute what $v$ would output on the edge $\{u,v\}$ under input $H'$. Let $S(u,v)$ denote the set of all possible outputs obtained for the edge $\{u,v\}$ this way.

(3) If $S(u,v) = \{\mathsf{I}\}$, then node $u$ outputs $\mathsf{O}$ on the edge $\{u,v\}$, and otherwise $u$ outputs $\mathsf{I}$ on it.

We now prove that $\mathcal{A}_{T-1}$ produces a valid solution for the sinkless orientation problem.

Consider a white node $u$, and its neighbors $v_1, v_2, \ldots, v_k$ in $H$. Since $T < g/2$, we have $B(v_i, T) \cap B(v_j, T) = B(u, T-1)$ for all $i \neq j$, and thus the inputs in $B(v_i, T) \setminus B(u, T-1)$ do not affect the output of $v_j$ in $\mathcal{A}_T$ for any $j \neq i$. Thus, any combination of labels $\mathsf{L}_{v_1} \in S(u, v_1), \ldots, \mathsf{L}_{v_k} \in S(u, v_k)$ may occur as an output: for any such labels $\mathsf{L}_{v_1}, \ldots, \mathsf{L}_{v_k}$, there is some input graph[1] such that $v_i$ in $\mathcal{A}_T$ outputs the label $\mathsf{L}_i$ for the edge $\{u, v_i\}$.

Let $u$ be a white node of degree at least 3 in $H$ with neighbors $N(u)$ in $H$. By the above argument, for any choice of one $\mathsf{L}_v \in S(u,v)$ for each neighbor $v \in N(u)$ of $u$, there is an input graph on which $\mathcal{A}_T$ outputs $\mathsf{L}_v$ for the edge $\{u,v\}$. If each $S(u,v)$ contains $\mathsf{O}$, there would be an input graph on which $\mathcal{A}_T$ outputs $\mathsf{O}$ for all incident input edges of $u$, implying that $\mathcal{A}_T$ is not a correct sinkless orientation algorithm. Hence, at least one neighbor $v'$ of $u$ satisfies $S(u,v') = \{\mathsf{I}\}$, and in $\mathcal{A}_{T-1}$ where $u$ is active, $u$ outputs $\mathsf{O}$ on the edge $\{u,v'\}$.

On the other hand, consider black node $v$ of degree at least 3 with neighbors $N(v)$ in $H$. On the actual input $H$ node $v$ in $\mathcal{A}_T$ will output $\mathsf{O}$ on an incident edge $\{v,u\}$, for some $u \in N(v)$. In $\mathcal{A}_{T-1}$, the node $u$ will consider the input $H$ on $B(v,T)$ (among other inputs), so we have $\mathsf{O} \in S(u,v)$. Thus, in $\mathcal{A}_{T-1}$ the node $u$ will output $\mathsf{I}$ on $\{v,u\}$, and $v$ has an incident edge labeled $\mathsf{I}$ as desired. □

**2.3 Step Two: There Exists No Algorithm with Locality 0.** As the second step of the proof, we show that the sinkless orientation problem does not admit trivial algorithms, i.e., there are no algorithms with locality 0. Recall that in a locality-0 algorithm in the supported LOCAL model, a node $v$ knows the entire structure of the support graph $G$, as well as which of the edges incident to $v$ belong to the input graph $H$. Unlike in locality-0 LOCAL algorithms, node $v$ also knows the identifiers of the other endpoints of its incident edges.

LEMMA 2.2. *Let $G$ be a fixed 5-regular bipartite graph with fixed unique identifiers and a 2-coloring of the nodes. The locality of sinkless orientation in bipartite supported* LOCAL *on support graph $G$ is greater than 0.*

*Proof.* Assume for contradiction that there is an algorithm $\mathcal{A}_0$ with locality 0 for the sinkless orientation problem. Without loss of generality, we may assume that black nodes are active in $\mathcal{A}_0$. We mark each edge $e$ of $G$ with the set of all outputs that $\mathcal{A}_0$ can produce for $e$ when $e$ is part of the input. For any

[1] Note that this step is where we require the existence of the support graph: in the LOCAL model, it might be the case that there exists a combination of labels $\mathsf{L}_{v_i}$ and $\mathsf{L}_{v_j}$ such that nodes $v_i$ and $v_j$ output these labels only when $B(v_i,T) \setminus B(u,T-1)$ and $B(v_j,T) \setminus B(u,T-1)$, respectively, contain a node with some specific unique identifier. This means that there does not exist any concrete input graph such that algorithm $\mathcal{A}_T$ outputs labels $\mathsf{L}_{v_i}$ and $\mathsf{L}_{v_j}$ for edges $\{u,v_i\}$ and $\{u,v_j\}$. The support graph $G$ fixes this problem by ensuring that the identifiers are unique across all possible input graphs $H'$ compatible with the real input graph $H$.

black node $v$, there must be at least three edges marked with either $\{\mathsf{O}\}$ or $\{\mathsf{O},\mathsf{I}\}$. Otherwise at least three edges of $v$ are marked with $\{\mathsf{I}\}$, which implies that there exists some input where $v$ has exactly three incident input edges and it would output $\mathsf{I}$ on all of them, contradicting the assumption that $\mathcal{A}_0$ produces a correct output for the sinkless orientation problem in the bipartite model.

Since every edge is incident to exactly one black node, at most a fraction of $2/5$ of the edges are marked with $\{\mathsf{I}\}$. Hence, there is a white node $u$ such that $u$ is incident to at least three edges $\{u, v_1\}$, $\{u, v_2\}$ and $\{u, v_3\}$ marked with either $\{\mathsf{O}\}$ or $\{\mathsf{O},\mathsf{I}\}$. Now consider an input where these three edges are the only input edges incident to $u$. Since the output of each node $v_i$ depends only on its incident input edges, we can select for each $v_i$ an input where $v_i$ outputs $\mathsf{O}$ for edge $\{u, v_i\}$. Moreover, since $\mathcal{A}_0$ is an algorithm with locality 0 and black nodes $v_1$, $v_2$ and $v_3$ are not neighbors, we can do this for all of them simultaneously. Thus, there exists an input where $\mathcal{A}_0$ outputs $\mathsf{O}$ on all incident input edges of the passive node $u$, a contradiction. □

### 2.4 Putting Things Together

THEOREM 2.1. *The locality of the sinkless orientation problem in the deterministic supported* LOCAL *model is* $T(n) = \Omega(\log n)$.

*Proof.* Let $G$ be a bipartite 5-regular graph with girth $g = \Omega(\log n)$. Observe that we can obtain one e.g. by taking the bipartite double cover of any 5-regular graph of girth $\Omega(\log n)$, which are known to exist (see e.g. [7, Ch. 3]).

Assume that there is a supported LOCAL algorithm $\mathcal{A}_T$ that solves sinkless orientation with locality $T < g/2$ on support graph $G$. This implies that there is a bipartite supported LOCAL algorithm for sinkless orientation on $G$ running in time $T$. By repeated application of Lemma 2.1, there is a sequence of bipartite supported LOCAL algorithms $\mathcal{A}_T, \mathcal{A}_{T-1}, \ldots, \mathcal{A}_1, \mathcal{A}_0$, where algorithm $\mathcal{A}_i$ solves sinkless orientation with locality $i$.

In particular, $\mathcal{A}_0$ solves sinkless orientation with locality 0. By Lemma 2.2, this is impossible, so algorithm $\mathcal{A}_T$ cannot exist. □

The above result readily implies Theorem 1.1.

THEOREM 1.1. *The locality of the sinkless orientation problem in the deterministic* LOCAL *model is* $\Omega(\log n)$.

*Proof.* Assume that there exists a LOCAL algorithm $\mathcal{A}$ solving sinkless orientation with locality $T(n) = o(\log n)$. This implies that there exists a supported LOCAL algorithm with the same locality solving sinkless orientation, as the supported LOCAL algorithm can simulate the LOCAL algorithm $\mathcal{A}$ on the real input graph $H$ by ignoring its knowledge of the support. This is impossible by Theorem 2.1, and hence the locality of sinkless orientation must be $\Omega(\log n)$ also in the LOCAL model. □

## 3 Sinkless Orientation Has Locality $O(\log \log n)$ in SLOCAL

We now show that the locality of the sinkless orientation problem in the deterministic SLOCAL model is $O(\log \log n)$. Together with the lower bound result of the previous section, this separates the deterministic LOCAL and SLOCAL models.

**Roadmap.** As the first step, we consider a variant of the sinkless orientation problem called *high-degree sinkless orientation*, where only nodes with degree $\Omega(\log n)$ are required not to be sinks. We show that this problem can be solved with a simple greedy algorithm that processes edges one at a time, and this

algorithm can be implemented in the SLOCAL model with locality $O(1)$. As the second step, we show how to reduce the general sinkless orientation problem to the high-degree problem.

As the high-level idea, we compute a clustering of the nodes using a maximal distance-$k$ independent set, where all nodes in the independent set are separated by distance at least $k = \Theta(\log \log n)$, and solve high-degree sinkless orientation on the graph formed by the clusters and edges between the clusters. We can then orient edges inside each cluster independently without creating sinks; high-degree clusters will already have one outgoing edge oriented away from the cluster, and low-degree clusters contain either a node of degree at most two or a cycle. Moreover, this idea can be implemented as an SLOCAL algorithm with locality $O(\log \log n)$.

**3.1 Step One: High-Degree Sinkless Orientation.** *The high-degree sinkless orientation problem* is a variation of the sinkless orientation problem in which we only care that nodes with degree of at least $\lfloor \log_2 n \rfloor + 1$ are not sinks; we call such nodes *high-degree nodes*. For technical purposes, we will assume in this section that the input graph $G = (V, E)$ is a multigraph.

**Greedy algorithm.** We describe a greedy algorithm $\mathcal{A}$ for solving high-degree sinkless orientation that orients the edges of the input multigraph $G = (V, E)$ one at a time. During the execution of $\mathcal{A}$, we say that a node $v \in V$ is *satisfied* if it either is not a high-degree node or at least one incident edge has been oriented away from $v$; otherwise, $v$ is *unsatisfied*. Algorithm $\mathcal{A}$ processes each edge $e = \{u, v\}$ using the following rules:

(1) If either $u$ or $v$ is already satisfied, the algorithm orients the edge towards the satisfied node, breaking ties arbitrarily.

(2) Otherwise, both $u$ and $v$ are unsatisfied high-degree nodes. The algorithm orients the edge towards the node which has fewer adjacent edges already processed, breaking ties arbitrarily.

LEMMA 3.1. *Algorithm $\mathcal{A}$ produces a valid solution to high-degree sinkless orientation.*

*Proof.* At each step of the execution of $\mathcal{A}$, consider the connected components formed by the edges that have been processed by Rule (2) up to current step. We want to show that the following invariants hold:

(a) Each connected component formed in this way is a tree with each edge oriented towards the root. Moreover, all unsatisfied nodes are roots of the trees.

(b) If an unsatisfied node $v \in V$ has $b$ edges oriented towards $v$, then the current connected component has at least $2^b$ nodes. In other words, node $v$ is a root of a tree with at least $2^b$ nodes.

These suffice to prove the theorem, as any unsatisfied node after the termination of the algorithm would need to be part of a component containing at least $2^{\lfloor \log_2 n \rfloor + 1} > n$ nodes, and therefore the component needs to be larger than the whole graph, a contradiction.

The invariants trivially hold before any edges have been processed, as each node has 0 edges directed towards them and thus each node is the root of its own tree. The invariants also trivially remain true after any step where we process an edge $e = \{u, v\}$, where $u$ or $v$ is satisfied, as we invoke Rule (1) instead of Rule (2).

Consider now the case where the algorithm processes an edge $e = \{u, v\}$ with both $u$ and $v$ unsatisfied, and let the number of processed edges incident to $u$ and $v$ be $b_u$ and $b_v$, respectively. Without loss of generality, we may assume that $b_v \leq b_u$ and that $\mathcal{A}$ orients $e$ towards $v$. Node $u$ is now satisfied, and node $v$ has indegree $b_v + 1$. Since Invariant (b) held before this step, we have that the new connected component containing $v$ now has size at least $2^{b_v} + 2^{b_u} \geq 2^{b_v} + 2^{b_v} = 2^{b_v+1}$, implying that the invariant

continues to hold. Moreover, node $u$ is no longer the root of its component but has an edge oriented towards the new root $v$, and therefore Invariant (a) holds. □

**3.2 Step Two: Sinkless Orientation on General Graphs.** We start by describing our algorithm for sinkless orientation on general graphs in three steps. Each one of these steps can be implemented in the SLOCAL model with locality $O(\log \log n)$, assuming that the output from previous steps is available at the nodes. We defer the proof that these steps can be combined into a single-step SLOCAL algorithm with locality $O(\log \log n)$ until the end of the section. In the following, let $T = \lceil \log_2(\lfloor \log_2 n \rfloor + 1) \rceil = O(\log \log n)$.

**Clustering.** Let $G = (V, E)$ be the input graph. A set $I \subseteq V$ is said to be a distance-$k$ independent set of $G$ if any two nodes in $I$ are at least distance $k$ apart. We construct a *clustering* of the input graph $G$ by computing a maximal distance-$(2T + 2)$ independent set $I$ and assigning each node to the cluster of the closest independent set node $v \in I$, breaking ties arbitrarily. For node $v \in I$, we denote by $C_v$ the cluster corresponding to $v$. We say that an edge $e \in E$ is an *inter-cluster* edge if its endpoints are in different clusters, and an *intra-cluster* edge otherwise.

We note that the radii of the clusters are bounded: All nodes in the radius-$T$ neighborhood of node $v \in I$ belong to cluster $C_v$. On the other hand, for every node $u$ in $C_v$, the distance between $u$ and the cluster center $v$ is at most $2T + 1$.

We now define a virtual *cluster graph* with a node for each cluster $C_v$ for $v \in I$, and adding an edge between $C_v$ and $C_u$ for every edge of the original graph that connects a node in $C_v$ to $C_u$, preserving duplicate edges. That is, the cluster graph can be a multigraph.

Finally, we note that this clustering can be done with locality $O(T)$ by a simple greedy SLOCAL algorithm: When the algorithm processes node $v$, it checks whether there are any other nodes belonging to set $I$ in the radius-$(2T + 1)$ neighborhood of $v$. If there are, then node $v$ belongs to the cluster of the nearest such node, and otherwise the algorithm adds node $v$ to set $I$.

**Orienting inter-cluster edges.** As the next step, we compute a high-degree orientation of the cluster graph, using the greedy algorithm of Section 3.1. Since there is a one-to-one correspondence between the edges of the cluster graph and edges between the clusters in the input graph $G$, this naturally induces an orientation of inter-cluster edges in $G$. We observe that under this partial orientation, any cluster $C_v$ with degree at least $\lfloor \log_2 n \rfloor + 1$ has at least one edge oriented away from $C_v$, as the number of nodes in the cluster graph is at most $n$ and thus $C_v$ counts as a high-degree node.

Again, this step can be implemented in the SLOCAL model with locality $O(T)$: When the algorithm processes a node that is adjacent to an unprocessed inter-cluster edge, it can fully see both of the clusters in its radius-$O(T)$ neighborhood, including the directions of inter-cluster edges that have been previously processed. The algorithm can then orient all adjacent inter-cluster edges one-by-one using the greedy algorithm of Section 3.1.

**Orienting intra-cluster edges.** Finally, we show that given the orientation of inter-cluster edges as above, the intra-cluster edges can be oriented without creating any sinks of degree 3 or higher. We have two cases to consider:

- *High-degree* clusters with at least $\lfloor \log_2 n \rfloor + 1$ inter-cluster edges have at least one outgoing inter-cluster edge.

- *Low-degree* clusters with less than $\lfloor \log_2 n \rfloor + 1$ inter-cluster edges may have all inter-cluster edges directed towards the cluster.

We show that in both cases, it is possible to compute an orientation of intra-cluster edges based on the internal structure of the cluster and the orientation of the boundary edges so that no node of degree at least 3 is a sink.

For a high-degree cluster $C$, we know that there is a node $v \in C$ with an outgoing inter-cluster edge. In this case, picking an arbitrary spanning tree for $C$, orienting its edges towards $v$, and orienting remaining edges arbitrarily clearly suffices.

For a low-degree cluster $C$, we first observe that $C$ cannot be locally tree-like while having only nodes with degree at least 3:

LEMMA 3.2. *A low-degree cluster $C_v$ contains either a cycle or a node with degree 1 or 2.*

*Proof.* Assume for contradiction that the cluster does not contain a cycle and that the degree of every node is at least 3. Recall that all nodes within distance $T$ of the cluster center $v$ are contained in $C_v$, and thus there are at least

$$3 \cdot 2^{T-1} > 2^T = 2^{\lceil \log_2(\lfloor \log_2 n \rfloor + 1) \rceil} \geq \lfloor \log_2 n \rfloor + 1$$

nodes in $C_v$ at distance $T$ from $v$. Moreover, it follows that there are at least this many edges on the boundary of the cluster, and thus the cluster is adjacent to at least $\lfloor \log_2 n \rfloor + 1$ inter-cluster edges, a contradiction. □

If the cluster contains a cycle, then we can orient that cycle in a consistent manner, and orient the rest of the edges towards the cycle. Otherwise the cluster contains a node with degree 1 or 2, in which case we orient all edges towards that node.

As the radius of each cluster is bounded by $2T + 1$, every node can see the whole cluster it belongs to within its radius-$O(T)$ neighborhood. Therefore the algorithm can orient the intra-cluster edges in a consistent manner with locality $O(T)$.

**Composability of SLOCAL algorithms.** To conclude the description of our algorithm, we need to show that one can compose a multiple-step SLOCAL algorithm into a one-step SLOCAL algorithm. This is a well-known result [17], but we include a short proof for completeness.

LEMMA 3.3. *Let $\mathcal{A}$ and $\mathcal{B}$ be* SLOCAL *algorithms with localities $T_{\mathcal{A}}$ and $T_{\mathcal{B}}$, respectively, and let $\mathcal{B}$ depend on the output of $\mathcal{A}$. Then there exists an* SLOCAL *algorithm $\mathcal{B} \circ \mathcal{A}$ with locality $T_{\mathcal{A}} + 2T_{\mathcal{B}}$ that solves the same problem as $\mathcal{B}$ without dependency on the output of $\mathcal{A}$.*

*Proof.* To construct algorithm $\mathcal{B} \circ \mathcal{A}$, we use the fact that algorithm $\mathcal{A}$ works for *any* order of nodes, not just the one that the adversary uses for $\mathcal{B} \circ \mathcal{A}$. In particular, when the adversary shows a node $v$ to $\mathcal{B} \circ \mathcal{A}$, algorithm $\mathcal{B} \circ \mathcal{A}$ can simulate $\mathcal{A}$ for all nodes in the radius-$T_{\mathcal{B}}$ neighborhood of $v$ in an arbitrary order. Once it has done this, algorithm $\mathcal{B} \circ \mathcal{A}$ can use $\mathcal{B}$ to compute the output for node $v$.

The challenge here is that when simulating $\mathcal{A}$ for node $u$ in the radius-$T_{\mathcal{B}}$ neighborhood of $v$, algorithm $\mathcal{B} \circ \mathcal{A}$ needs to store the output of $\mathcal{A}$ for $u$ for later use, but it can only select the output state for node $v$. To circumvent this problem, we allow the output of $\mathcal{A}$ for node $u$ to be stored at any node in the radius-$T_{\mathcal{B}}$ neighborhood of $u$. In particular, its output can be stored at node $v$. This gives algorithm $\mathcal{B} \circ \mathcal{A}$ its locality: When processing node $v$, the algorithm needs to run $\mathcal{A}$ for all nodes in the radius-$T_{\mathcal{B}}$ neighborhood of $v$. Each of those nodes needs to see its radius-$T_{\mathcal{A}}$ neighborhood, including the previous outputs, which may be stored at distance $T_{\mathcal{B}}$ from the node in question. Hence algorithm $\mathcal{B} \circ \mathcal{A}$ has locality $T_{\mathcal{A}} + 2T_{\mathcal{B}}$. □

THEOREM 1.2. *The locality of the sinkless orientation problem in the deterministic* SLOCAL *model is* $O(\log \log n)$.

*Proof.* We can apply Lemma 3.3 twice to combine the three-step SLOCAL algorithm we described above into a one-step SLOCAL algorithm. As each of the three steps has locality $O(T)$, the final algorithm has locality $O(T) = O(\log \log n)$, completing the proof. ∎

## 4 Discussion and Broader Context

In this work, we have presented simple, self-contained proofs of Theorems 1.1 and 1.2, which show that the locality of the sinkless orientation problem in the LOCAL model is exponentially larger than in the SLOCAL model. We now conclude the paper by briefly discussing the broader context and the role of the sinkless orientation problem and the SLOCAL model in understanding the foundations of distributed computing.

**Complexity of distributed sinkless orientation.** While we presented a lower bound in the LOCAL model and an upper bound in the SLOCAL model, we note that locality of sinkless orientation is fully understood in these models:

- In deterministic LOCAL model, sinkless orientation has locality $\Theta(\log n)$ [9, 11, 16].
- In randomized LOCAL model, sinkless orientation has locality $\Theta(\log \log n)$ [9, 16].
- In deterministic SLOCAL model, sinkless orientation has locality $\Theta(\log \log n)$ [16, 18].
- In randomized SLOCAL model, sinkless orientation has locality $\Theta(\log \log \log n)$ [16, 18].

**The role of sinkless orientation in understanding the Lovász Local Lemma.** The sinkless orientation problem was introduced in [9] with the purpose of understanding the locality of the constructive Lovász Local Lemma problem in the distributed setting.

Lovász Local Lemma (LLL) is a classic result in probability theory that can be used to show the existence of various combinatorial objects. For example, one can use LLL to prove that a sinkless orientation exists in any graph [9].

In the distributed setting, the key question is the locality of *constructive, algorithmic Lovász Local Lemma*: given a problem where LLL guarantees the *existence* of a solution, what can we say about the locality of *finding* such a solution (e.g. in the LOCAL or SLOCAL model)? For many interesting problems one can prove that a solution exists by using LLL, and hence a generic way to solve LLL in the distributed setting gives a distributed algorithm for all these problems. Since LLL can be used to find a sinkless orientation, any *lower bound* on the locality of sinkless orientation implies also a lower bound on the locality of general LLL algorithms.

**The role of sinkless orientation in understanding splitting problems.** Sinkless orientation can be seen as the most relaxed version of the *degree splitting* problem, for which two variants exist, *directed* and *undirected*. The directed variant asks for an orientation of the edges such that each node has roughly the same number of incoming and outgoing edges. The undirected variant asks for a coloring of the edges with red and blue such that each node has roughly the same number of red and blue incident edges. Observe that on bipartite two-colored graphs these two problems are equivalent. It is known [16] that efficient algorithms for degree splitting allow us to obtain efficient algorithms for e.g. edge coloring, and hence understanding the easiest splitting variant (sinkless orientation) may give insights for understanding the more general case.

**The role of sinkless orientation in understanding round elimination.** In order to prove the $\Omega(\log \log n)$ rounds lower bound for the sinkless orientation problem, authors of [9] used the so-called *round elimination* technique. Since then, this technique has been better understood, and sinkless orientation played a key role in developing this technique, which has been since then used to show lower bounds for many fundamental problems [3–5, 8, 12]. On a high level, the standard way of applying this technique works as follows:

(1) First, prove a lower bound for deterministic algorithms in a weaker setting, where nodes do not have IDs, using a strategy similar to what we do in Section 2.2.

(2) Then, lift this lower bound to a stronger setting, where nodes have no IDs but randomization is allowed. This step is quite non-trivial, since it requires one to track how the failure probability evolves when making the algorithm one round faster.

(3) Finally, convert the obtained randomized lower bound into a stronger deterministic lower bound for the LOCAL model, by using non-trivial techniques typically used to prove *gap results* in the LOCAL model.

One of our contributions is to simplify this three step process by showing how to directly handle unique identifiers in a round elimination proof. A similar concept for handling unique IDs, called the ID graph technique, was independently discovered in [10].

**The role of SLOCAL.** The SLOCAL model has played a key role in understanding the LOCAL model itself. One of the major challenges that we encounter in the LOCAL model is the fact that all nodes act in parallel, and they have to decide their output at the same time. The SLOCAL model abstracts away this issue, since in this model nodes are processed sequentially. Hence, developing algorithms for the SLOCAL model may be much easier than developing algorithms for the LOCAL model. Combining this with the fact that, by paying some overhead, we have black box ways to convert SLOCAL algorithms to LOCAL ones [18], this gives us an easier way to design LOCAL algorithms. Moreover, SLOCAL played an important role for understanding the role of randomness in the LOCAL model. In fact, it has been shown that any randomized LOCAL algorithm can be derandomized by paying an $O(\text{poly} \log n)$ overhead [18]. This result has been shown by providing an SLOCAL algorithm as an intermediate step.

**The role of supported models in distributed computing.** Our new simple proof of the $\Omega(\log n)$ lower bound for the locality of sinkless orientation in the LOCAL model also holds in the stronger supported LOCAL model. While originally introduced in the context of *software-defined networking* [26], the supported LOCAL model and its bandwidth-bound cousin, the supported CONGEST model, have subsequently played a key role in understanding the power of *preprocessing* in distributed computing [14, 15, 19, 20].

The underlying idea of these supported models is that the globally known support graph $G$ can be preprocessed in advance (or locally at each node) to facilitate the efficient solution of graph problems on possible input subgraphs of $G$. While a priori such preprocessing may sound powerful, it turns out that for many problems, the additional information provided by the support graph does not help much compared to standard, non-supported model, particularly in the supported CONGEST model [15, 19, 20].

In the case of the supported LOCAL model, it is known that symmetry-breaking problems with locality $O(\log^* n)$ in the standard LOCAL model have locality $O(1)$ in the supported LOCAL model [26] and that there are problems that have $\Theta(n)$ locality in the supported LOCAL model [14]. Beyond these results, much less is known about complexity of symmetry-breaking problems in the supported LOCAL model. In particular, Foerster et al. [14] raised the question whether there exist so-called locally checkable

problems with "intermediate" complexity between $\Omega(\log n)$ and $O(\operatorname{poly}\log n)$ in the supported LOCAL model. Our new lower bound for sinkless orientations in the supported LOCAL model affirmatively answers this question.

**Acknowledgements**

We thank the anonymous reviewers for their helpful feedback on previous versions of this work. Parts of this work appeared in DISC 2021 as a brief announcement [21]. This work was supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 805223 ScaleML), the Academy of Finland (grant agreement No 333837), the Austrian Science Fund (FWF) and netIDEE (grant agreement No P 33775-N), and the Austrian Science Fund (FWF) project DELTA (grant agreement No I 5025-N).

**References**

[1] 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)*, 2018. doi:10.4230/LIPIcs.DISC.2018.9.

[2] 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)*, 2018. doi:10.1145/3188745.3188860.

[3] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2019)*, 2019. doi:10.1109/FOCS.2019.00037.

[4] Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Deterministic $\Delta$-coloring plays hide-and-seek. In *Proc. 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)*. ACM, 2022.

[5] Alkida Balliu, Sebastian Brandt, and Dennis Olivetti. Distributed lower bounds for ruling sets. *SIAM Journal on Computing*, 51(1):70–115, 2022. doi:10.1137/20M1381770.

[6] Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Morgan & Claypool Publishers, 2013. doi:10.2200/S00520ED1V01Y201307DCT011.

[7] Béla Bollobás. *Extremal graph theory*. Courier Corporation, 2004.

[8] Sebastian Brandt. An Automatic Speedup Theorem for Distributed Problems. In *Proc. 38th ACM Symposium on Principles of Distributed Computing (PODC 2019)*, 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)*, 2016. doi:10.1145/2897518.2897570.

[10] Sebastian Brandt, Yi-Jun Chang, Jan Grebík, Christoph Grunau, Václav Rozhoň, and Zoltán Vidnyánszky. Local Problems on Trees from the Perspectives of Distributed Algorithms, Finitary Factors, and Descriptive Combinatorics. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, 2022. doi:10.4230/LIPIcs.ITCS.2022.29.

[11] 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)*, 2016. doi:10.1109/FOCS.2016.72.

[12] Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. Distributed edge coloring and a special case of the constructive Lovász local lemma. *ACM Transactions on Algorithms (TALG)*, 16(1):1–51, 2019. doi:10.1145/3365004.

[13] 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.

[14] Klaus-Tycho Foerster, Juho Hirvonen, Stefan Schmid, and Jukka Suomela. On the Power of Preprocessing in Decentralized Network Optimization. In *Proc. IEEE Conference on Computer Communications (INFOCOM 2019)*, 2019. doi:10.1109/INFOCOM.2019.8737382.

[15] Klaus-Tycho Foerster, Janne H. Korhonen, Joel Rybicki, and Stefan Schmid. Does preprocessing help under congestion? In *Proc. 38nd ACM Symposium on Principles of Distributed Computing, (PODC 2019)*, pages 259–261, 2019. doi:10.1145/3293611.3331581.

[16] 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.

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

[18] 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)*, 2018. doi:10.1109/FOCS.2018.00069.

[19] Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In *Proc. 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2021)*, pages 1166–1179. ACM, 2021. doi:10.1145/3406325.3451081.

[20] Janne H. Korhonen and Joel Rybicki. Deterministic Subgraph Detection in Broadcast CONGEST. In *Proc. 21st International Conference on Principles of Distributed Systems (OPODIS 2017)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 4:1–4:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.OPODIS.2017.4.

[21] Janne H. Korhonen, Ami Paz, Joel Rybicki, Stefan Schmid, and Jukka Suomela. Brief announcement: Sinkless orientation is hard also in the supported LOCAL model. In *Proc. 35th International Symposium on Distributed Computing (DISC 2021)*, volume 209 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 58:1–58:4. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.DISC.2021.58.

[22] Juhana Laurinharju and Jukka Suomela. Brief announcement: Linial's lower bound made easy. In *Proc. 33rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2014)*, pages 377–378. ACM Press, 2014. doi:10.1145/2611462.2611505.

[23] Nathan Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1): 193–201, 1992. doi:10.1137/0221015.

[24] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000. doi:10.1137/1.9780898719772.

[25] Václav Rozhoň and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2020)*, pages 350–363, 2020. doi:10.1145/3357713.3384298.

[26] Stefan Schmid and Jukka Suomela. Exploiting locality in distributed SDN control. In *Proc. ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN 2013)*, pages 121–126. ACM Press, 2013. doi:10.1145/2491185.2491198.