Improved Local Computation of Edge Orientation

Reut Levi
Reichman University, Herzliya, Israel

Bar Rushkin
Reichman University, Herzliya, Israel

Abstract

In this paper, we study the problem of orienting the edges of a graph G so that every vertex has bounded out-degree in the *local computation algorithms (LCA)* model, as defined by Rubinfeld et al. (ICS 2011). More specifically, given a query $e \in E$ our algorithm returns the orientation of e such that with high constant probability (namely, at least 0.9) the out-degree of each vertex is bounded by r where r is a parameter. We provide such an upper bound for any $r = \Omega(arb(G) \cdot \log n)$ where $arb(G)$ denotes the arboricity of G (we note that such orientation exists only when $r = \Omega(arb(G))$). Our query complexity is $\tilde{O}(n \cdot arb(G)/r^2)$ in the worst case and $O(1)$ on expectation (over the vertices and the randomness of the algorithm). This generalizes the upper bound by Mitrović–Rubinfeld–Singhal (ESA 2024) that provided a similar upper bound only when $r = \Omega((arb(G)^2 \cdot n))^{1/3}$.

For $r = \Omega(arb(G) \cdot \log n)$, our algorithm also improves their weaker upper bound of $\tilde{O}(n/r)$ queries for the special case where the input graph is a tree (whose arboricity is 1).

Our algorithm assigns each vertex a *level* derived from locally sampled neighborhoods combined through a staggered multi-scale recursion, inspired by the recent arboricity-approximation framework of Dai–Ghaffari–Portmann (FOCS 2025). The main novelty of our approach is that it reconstructs levels consistently across the graph while simultaneously respecting the out-degree bound and maintaining locality of computation.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Local Algorithms, Sublinear-time Algorithms, Edge Orientation, Bounded Arboricity

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2026.75

Category RANDOM

Funding *Reut Levi*: The author was supported by the Israel Science Foundation under Grant 1867/20.
Bar Rushkin: The author was supported by the Israel Science Foundation under Grant 1867/20.

1 Introduction

Orienting graph edges while satisfying degree constraints has broad algorithmic applications. From a structural perspective, bounded out-degree orientations capture the sparsity of a graph via its degeneracy and arboricity, and play a key role in graph coloring and subgraph enumeration algorithms [4, 13].

Due to this fact, low-out-degree orientation has been extensively studied in the dynamic, distributed, and massively parallel settings (see e.g. [10, 17, 9, 7, 3]).

In many local settings, a standard approach is to simulate a distributed peeling-type process that, for a graph G of arboricity $\alpha = arb(G)$, yields an orientation with maximum out-degree at most $\alpha(2 + \epsilon)$ in $O(\log n)$ iterations.

Mitrović, Rubinfeld, and Singhal [14] initiated the study of computing bounded out-degree edge orientations in the LCA model. They showed that directly simulating the peeling approach via standard LCA transformations can require $\Omega(n)$ probes per query. They also proved a lower bound of $\Omega(\sqrt{n}/r)$ probes per query for producing an r-out-degree orientation, even when the input graph is a forest (i.e., when $\alpha = 1$).

On the algorithmic side, they gave sublinear-probe LCAs for forests and for general graphs. For general graphs, however, their approach requires $r = \Omega\big((\alpha^2 n)^{1/3}\big)$, and for this range they provide a randomized LCA that computes an r-orientation using $\tilde{O}(\max\{\alpha n/r^2, 1\})$ probes per query. Their approach is based on a degree-threshold decomposition together with randomized sampling of the immediate neighborhood of a vertex, rather than on recursively reconstructing a global peeling process.

As a result, their work leaves a gap both in probe complexity and in the range of parameters for which sublinear algorithms are known. In this paper, we address the latter gap, up to logarithmic factors, by providing an LCA with the same probe complexity for all $r = \Omega(\alpha \log n)$. Since r must be at least $\Omega(\alpha)$, this extends the best known upper bound essentially down to the smallest range currently compatible with the sampling requirements of the approach.

1.1 The problem of r-orientation and the peeling process

A natural *global* approach for achieving almost optimal out-degree is the *peeling process*. This process proceeds in iterations. In each iteration, i, all vertices whose *current* degree is at most $(2+\epsilon)\alpha$ are removed from the graphs. These vertices will be called the i-th layer. The process repeats until no vertices remain. Edges are then oriented from earlier to later layers, and arbitrarily within each layer, resulting in a maximum out-degree of $(2+\epsilon)\alpha$ after $O(\log n)$ phases. However, when simulated in the LCA model using the standard Parnas–Ron simulation argument, peeling requires $\Omega(n)$ probes per query, and is therefore not sublinear [15].

In a pair of papers studying the problem of approximating the arboricity of a graph, the authors nevertheless manage to locally simulate a peeling process. In particular, both the algorithm of Eden, Mossel, and Ron [6] and that of Dai, Ghaffari, and Portmann [5] employ a local *simulation of a peeling process*. However, in both cases the simulation does not reconstruct the order in which vertices are peeled. Moreover, the local views produced by the simulation do not correspond to any consistent global peeling sequence.

EMR [6]: pruning-based simulation

In this paper, Eden, Mossel, and Ron [6] introduced a local recursive procedure designed to estimate whether a global peeling process removes a vertex v by the i-th iteration. To keep query complexity manageable, the algorithm prunes branches of the recursion that are expected to be costly. While their analysis shows that this pruning does not significantly distort the simulation's outcome, the procedure has a key limitation for our purposes: it does not recover the actual peeling iteration, but only provides an upper bound. More critically, the local views generated by the simulation do not correspond to any coherent global peeling order. Specifically, on query v, the algorithm attempts to peel a sample consisting of a $\Theta(1/\alpha)$-fraction of the neighbors of v, while *pruning* those predicted to trigger deep recursion. This technique controls the expected recursion depth and yields an $O(\log^2 n)$-approximation using $\tilde{O}(n/\alpha)$ probes. However, the resulting simulation should be viewed only as an approximation of the ideal peeling process. In particular, the local

simulation used for one vertex may correspond to one approximate peeling process, while the simulation for another vertex may correspond to a different approximate peeling process. As a result, the algorithm does not recover the actual peeling iterations and does not define a globally consistent peeling order, which requires a different approach in our setting.

DGP [5]: staggered multi-copy simulation

In their recent work, Dai, Ghaffari, and Portmann [5] propose a procedure that estimates whether a vertex would be peeled in a global peeling process. Their method samples a set of vertices and, for each one, simulates a peeling process to determine if it gets removed – but crucially, it does not recover the iteration in which the vertex is peeled. To manage query complexity, they avoid pruning and instead run multiple simulations in parallel, each with a different pace. Their key insight is that if the graph has low arboricity then with high probability, one of these simulations will terminate quickly and return an answer. Moreover, they design a clever scheduling scheme that ensures the algorithm reaches this efficient simulation without incurring a significant overhead in queries.

However, this approach has a fundamental limitation for our purposes: each simulation uses independent randomness, which is essential for their analysis but prevents the results from aligning with any consistent global peeling order. In particular, the different simulations do not correspond to different executions of the same ideal peeling process, but rather to different approximate local simulations of peeling. Consequently, the depth of the first recursive exploration that terminates successfully should not be interpreted as the actual peeling level of the vertex.

To illustrate this, consider two adjacent vertices u and v, where u is peeled before v in the ideal peeling process. Due to the independent randomness used in the local simulations, it may happen that the recursive exploration rooted at u encounters several bad samples and only terminates after many recursive levels, while the exploration rooted at v quickly encounters a good sample and terminates much earlier. Thus, the local recursion depth may suggest that u belongs to a later layer than v, even though the ideal peeling order is the opposite. Since these local simulations are generated independently, there is no globally consistent peeling order that they collectively realize.

Moreover, the algorithm only determines whether a vertex is peeled – not when. This lack of global coherence motivates the need for a different strategy in our orientation setting.

1.2 Our result

We present a new local computation algorithm for r-orientation in general graphs. Given a query access to an input graph $G = (V, E)$ of arboricity α and a parameter $r = \Omega(\alpha \log n)$, the algorithm answers each edge-orientation query using $O\left(\frac{\alpha n \log^2 n}{r^2}\right)$ probes, and with high constant probability produces an orientation of maximum out-degree r. In particular, we prove the following theorem.

▶ **Theorem 1** (LCA for bounded out-degree orientation). *Let $G = (V, E)$ be an n-vertex graph of arboricity $\alpha = arb(G)$, given through adjacency-list probe access. Let r be a parameter satisfying $r \geq C\alpha \log n$ for a sufficiently large universal constant C. Then there exists a randomized LCA which, on query an edge $\{u, v\} \in E$, returns an orientation of this edge such that, with high constant probability over the random tape, all answers are consistent with a single orientation of G whose maximum out-degree is at most r. The probe complexity per edge query is $O\left(\frac{\alpha n \log^2 n}{r^2}\right)$.*

To compare the parameter ranges, consider the case $\alpha = O(1)$. The previous upper bound of Mitrović, Rubinfeld, and Singhal [14] achieves probe complexity $\tilde{O}(n\alpha/r^2)$, but only for $r = \Omega(n^{1/3})$. Our result achieves the same asymptotic upper bound for every $r = \Omega(\log n)$. In particular, for $r = n^{1/4}$, which is outside the range of the previous general-graph algorithm, our LCA uses $O(n^{1/2} \log^2 n)$ probes per query and produces an orientation of maximum out-degree $n^{1/4}$. Moreover, in this regime the lower bound of Mitrović, Rubinfeld, and Singhal [14] is already $\Omega(n^{1/4})$, even for trees, which are in some sense the simplest constant-arboricity graphs. Thus, the improvement in the range of r is meaningful even in parameter regimes where the probe complexity remains strongly sublinear.

1.3 Description of our algorithm

At a high level, our LCA consists of three components. The first assigns each vertex a level by locally simulating a peeling process. The second answers an edge query $\{u, v\}$ by comparing the levels of u and v, orienting the edge from the lower-level endpoint to the higher-level endpoint (breaking ties according to vertex IDs). The third component implements a single local simulation step of the peeling process.

We begin with the intuition behind the local simulation. Consider the ideal peeling process in which, at every iteration, all vertices of degree at most $\Theta(\alpha)$ are removed. A vertex belongs to layer t if it is removed during the t-th iteration. Observe that if one could recursively follow only neighbors belonging to strictly lower layers, then any recursive exploration rooted at a vertex in layer t would have depth at most t.

However, the algorithm does not know the layers in advance. Instead, for each vertex v and iteration t, the algorithm generates random samples of the neighbors of v, where each neighbor is included independently with probability $\Theta(\log n/r)$. Intuitively, if many neighbors of v belong to lower layers, then with constant probability, a sampled set contains only such neighbors. In this case, recursively exploring the sampled neighbors simulates progress toward lower peeling layers.

A single local simulation step proceeds as follows. Initially, the algorithm queries the degree of v. If the degree is at most $\Theta(\alpha)$, the the first peeling layer. Otherwise, the algorithm recursively explores sampled neighbors while decreasing the iteration counter. The recursion terminates if the iteration counter reaches zero, if a low-degree vertex is encountered, or if the total number of probes exceeds the target budget $\tilde{O}(n\alpha/r^2)$.

One difficulty is that a random sample may contain neighbors that do not belong to lower layers, causing the recursive exploration to continue for much longer than the ideal peeling depth. As a mental experiment, we first assume that this does not happen, and explain why the query complexity can be bounded under this assumption. We then explain how this assumption can be removed using the staggered scheduling technique of Dai, Ghaffari, and Portmann [5]. Finally, we explain why introducing the structure of iterations is necessary in order to obtain a globally coherent solution.

Step 1: Assuming one good sample per vertex

To provide intuition for the query bound, let us first consider an idealized setting in which every vertex v is associated with a single *good* sample, namely a sample containing only neighbors that belong to strictly lower peeling layers than v. For the moment, we ignore the staggered scheduling and assume that the algorithm recursively explores only this single sample.

A natural way to analyze the query complexity would be to bound the size of the recursive exploration tree rooted at every vertex. Instead, we reverse the perspective and ask the following question: for a fixed vertex u, how much does u contribute to the recursive trees rooted at other vertices?

Since samples are good, every recursive call moves to a strictly lower peeling layer. Thus every root-to-node trajectory in the recursive exploration corresponds to a directed path in the graph, where edges are directed from higher layers to lower layers. By the peeling property, every vertex has out-degree at most 3α in this directed graph.

Fix a vertex u. Every time u appears in the recursive tree of another vertex, there must exist a sampled directed path leading to u. The expected number of sampled directed paths of length j ending at u is at most $(3\alpha)^j p^j = (3\alpha p)^j$, because there are at most $(3\alpha)^j$ directed paths of length j leading to u, and each such path survives in the recursive exploration with probability p^j. Therefore, the expected total contribution of u to all recursive trees is at most $1 + \sum_{j\geq 1}(3\alpha p)^j$.

Since $p = \Theta(\log n/r)$ and $r = x\alpha$, we have $3\alpha p = \tilde{O}(1/x)$. For sufficiently large x, this geometric series is bounded by $1 + \tilde{O}(1/x)$.

Summing over all vertices, the expected total additional recursive work is therefore $\tilde{O}(n/x)$. Consequently, by Markov's inequality, only a small number of vertices can have recursive explorations whose size exceeds the probe budget. These vertices are the overflow vertices. Since the overflow threshold is set to $\tilde{O}(n\alpha/r^2)$, the expected total work bound implies that, with constant probability, the number of overflow vertices is at most $O(r)$, and the constants are chosen so that it is at most $r/2$.

Step 2: Why staggered scheduling is needed

The discussion above assumed that every sample is good. In reality, however, a random sample is good only with constant probability. Thus, in order to find a good sample, the algorithm performs $O(\log n)$ independent sampling trials for every vertex.

A naive implementation would recursively explore all these trials, increasing the query complexity by an additional logarithmic factor *at every* recursive level. Instead, following Dai, Ghaffari, and Portmann [5], we evaluate these trials according to a staggered schedule.

The key idea is that trial i is slowed down by a factor 2^i, namely it is advanced only once every 2^i scheduling steps. Since each trial is good with constant probability, the index of the first good trial is geometrically distributed. Consequently, although there are $O(\log n)$ possible trials, the expected amount of work invested before reaching the first good one remains bounded by a convergent geometric sum. Intuitively, the algorithm tries many possible recursive explorations, while paying essentially only for the first successful one.

As a result, the accounting argument from the idealized setting still applies, up to a constant-factor overhead in expectation.

Step 3: Why iterations are needed

The discussion above explains why the recursive explorations remain small in expectation. We now turn to the correctness argument, namely why the resulting levels induce an orientation in which every vertex has at most r neighbors whose level is greater than or equal to its own.

The key point is that the algorithm should mimic the ideal peeling process. Suppose that a vertex v still has many neighbors that were not peeled after t iterations of the ideal process. Then we would like to conclude that v itself should also survive iteration $t + 1$. Intuitively, this follows because every sample of v is likely to contain at least one neighbor that survived iteration t, preventing the recursive exploration from terminating too early.

For this argument to hold, however, the randomness used to determine whether a neighbor survives iteration t must be independent of the randomness used when evaluating v in iteration $t+1$. Otherwise, the events become circularly dependent: the randomness used to decide whether u survives could simultaneously influence whether v observes u, destroying the inductive peeling argument.

For this reason, the algorithm uses different random samples for different iterations. Intuitively, iteration t determines which vertices survive after t rounds of peeling, while iteration $t+1$ uses fresh randomness to test whether a vertex still has many neighbors that survived iteration t. This separation between iterations is what allows us to argue inductively that vertices with many surviving neighbors cannot themselves be peeled too early.

This should be contrasted with the approach of Dai, Ghaffari, and Portmann [5], where independent recursive simulations are used only to decide whether a vertex is peeled. Since those simulations are not meant to define a common level assignment, they may use fresh randomness across recursive calls. In our setting, however, the randomness associated with each vertex and each iteration is fixed globally and reused across all recursive calls and all edge queries. This is crucial for ensuring that the local computations are consistent with one global level assignment.

1.4 Why the DGP batching technique does not remove the $\log n$ Gap

Dai, Ghaffari, and Portmann [5] strengthened the staggered scheduling framework by introducing *batches* of parallel recursive simulations. Each batch contains many independent recursive calls, and a vertex is considered peeled once a majority of the recursive calls in some batch terminate. The role of these batches is to amplify correctness in the high-arboricity case: if a vertex belongs to a dense core, then with high probability most recursive calls do not terminate, preventing the vertex from being mistakenly declared peelable. This amplification allows them to obtain a constant-factor approximation.

In our setting, however, this batching mechanism does not seem useful. First, the arboricity bound is already given, and thus we do not need to distinguish between low- and high-arboricity graphs. More importantly, the logarithmic gap between r and α does not stem from the probability amplification aspect of the recursion, but rather from the need to detect surviving neighbors with high probability in every iteration.

Specifically, to ensure the out-degree bound, if a vertex has many neighbors that survived previous iterations, then its samples in the next iteration must hit one of these surviving neighbors with high probability. This requires $p = \Omega(\log n/r)$. At the same time, the query complexity analysis requires the branching factor to remain bounded, namely that αp is bounded away from one. Together, these conditions imply $r = \Omega(\alpha \log n)$. Since the batching mechanism amplifies correctness but does not reduce the sampling probability p, it does not seem to improve the parameter range in our orientation application.

1.5 Related Work

Local Computation Algorithms (LCAs) answer *local* queries about a global solution using a sublinear number of input probes per query, and the answers over all queries are consistent with one global solution [16, 1].

By now, a wide range of results have been established within the LCA model, including maximal independent set [16], graph coloring [8], approximate maximum matching [12], and minor-free graph partitioning [11]. Edge orientations [14] are among the more recent additions to this growing body of work. In particular Mitrović, Rubinfeld, and Singhal [14] obtained the following results.

- *Lower bound.* Any (possibly randomized) LCA that outputs an r-orientation with success probability at least 0.9 must make $\Omega(n^{1/2}/r)$ probes in the worst case, even on forests ($\alpha = 1$) and even when allowed adjacency-list, adjacency-matrix, or degree queries.
- *Upper bounds for general graphs.* When $r \geq 10(\alpha^2 n)^{1/3}$, they obtain probe complexity $\tilde{O}(\max\{\alpha n/r^2, 1\})$.
- *General forests.* When the graph is a forest they obtain an upper bound of $\tilde{O}(n/r)$ queries for any r.
- *Bounded-degree forests.* For forests of maximum degree Δ they provide an LCA with probe complexity $\Delta\, n^{\,1-\log_\Delta r+o(1)}$.

2 Preliminaries

Let $G = (V, E)$ be a simple graph on $n = |V|$ vertices. For $S \subseteq V$ we denote by $E(S)$ the set of edges with both endpoints in S, and by $G[S]$ the induced subgraph on S. The degree of a vertex v is denoted $d(v)$.

▶ **Definition 2** (Arboricity). *The* arboricity *of G, denoted α, is the smallest integer k for which E can be partitioned into k forests. Equivalently,*

$$\alpha = \max_{S \subseteq V,\, |S| \geq 2} \left\lceil \frac{|E(S)|}{|S| - 1} \right\rceil .$$

Graph access model. We assume adjacency-list probe access to the input graph $G = (V, E)$ where $V = [n]$. Specifically, the probe $\texttt{neighbor}(v, i)$ returns the i-th neighbor or $\perp$ if it has less than i neighbors. We also use degree probes, where $\texttt{degree}(v)$ returns the degree of v.

The complexity of the LCA is mainly measured by the number of probes (query complexity), running time per query, and the amount of randomness used.

▶ **Definition 3** (Local Computation Algorithm (LCA) - [16]). *Let $\mathcal{P}$ be a computational problem where, given an input x of size n, the goal is to compute a valid solution $y \in F(x)$. A randomized algorithm $\mathcal{A}$ is a* ***(t, s, δ)-local computation algorithm*** *for $\mathcal{P}$ if, when given probe access to input x and random tape r (the random tape r is fixed once and reused across all queries):*

1. ***Query Handling:*** *For any query q_i (representing a specific part of the solution), $\mathcal{A}(q_i, x, r)$ returns a partial answer y_{q_i}.*
2. ***Sublinear Complexity:*** *The probe complexity per query is sublinear in n.*
3. ***Consistency:*** *The algorithm is* consistent *with a single global valid solution $y \in F(x)$. That is, for any sequence of queries $q_1, q_2, \ldots, q_m$, the algorithm produces an output sequence $y_{q_1}, y_{q_2}, \ldots, y_{q_m}$ that matches the values in y at those locations.*
4. ***Correctness:*** *For all x, with probability at least $2/3$ (over the random tape r) [1], the algorithm answers all queries consistently with a valid solution.*

2.1 r-Orientation in the LCA model

An LCA for the problem of r-orientation receives queries of the form "what is the orientation of edge $\{u, v\}$?" and must answer with either $u \to v$ or $v \to u$, using a number of probes that is sublinear in n, while ensuring that all such answers are consistent with a single global orientation whose maximum out-degree is at most r.

[1] Alternatively, one may require high success probability, namely, $1 - 1/n^c$, for any constant c (without changing the asymptotic complexity of the algorithm).

2.2 A Structural Ordering for α-Arboricity Graphs

Our analysis relies on a fixed acyclic orientation as described next.

▶ **Definition 4** (Ideal Peeling Layers). *Let $G = (V, E)$ be a graph, and let $G_0 = G$. For every integer $k \geq 1$, define recursively*

$$L_k^* := \{ v \in V(G_{k-1}) \mid d_{G_{k-1}}(v) \leq 3\alpha \},$$

and let

$$G_k := G\Big[V \setminus \bigcup_{i=1}^{k} L_i^*\Big]$$

be the subgraph induced by the vertices that remain after removing the first k layers.

For each vertex $v \in V$, define $\ell(v)$ to be the unique index k such that $v \in L_k^*$.

This peeling-based partition originates in the work of Barenboim and Elkin [2].

▶ **Lemma 5** (Structural Depth Bound). *The ideal peeling process terminates in $L = O(\log n)$ levels.*

Proof. Consider any subgraph $H \subseteq G$. Since G has arboricity α, $|E(H)| \leq \alpha|V(H)|$. By the Handshaking Lemma, $\sum_{v \in V(H)} d_H(v) \leq 2\alpha|V(H)|$. Let $S \subseteq V(H)$ be the set of vertices with $d_H(v) > 3\alpha$. Then $3\alpha|S| < \sum d_H(v) \leq 2\alpha|V(H)|$, implying $|S| < \frac{2}{3}|V(H)|$. Thus, each level removes at least $1/3$ of the remaining vertices, leading to $O(\log n)$ levels. ◀

▶ **Definition 6** (Good sample). *Fix a vertex v and a sampled neighbor set $S \subseteq N(v)$. We say that S is* good *(with respect to the peeling layers) if every vertex $u \in S$ belongs to a strictly lower level than v, i.e., $\ell(u) < \ell(v)$ for all $u \in S$.*

The structural levels are introduced purely for the purpose of analysis and are never accessed by the algorithm. Assume that each vertex initially possesses a good sample. Starting from a vertex v, we consider a recursive process that follows sampled (good) neighbors until it reaches a terminal vertex, defined as a vertex of degree at most 3α. Lemma 5 implies that the recursion depth of this process is bounded by $O(\log n)$. The depth of this recursion allows us to bound the number of iterations of our algorithm as we shall see next.

Our algorithm proceeds in $\Theta(\log n)$ iterations. In order to describe it we introduce several objects and definitions that will be used throughout the algorithm and its analysis.

Notation and Setup

For every vertex $v \in V$ and every iteration t, we use a different set of sampled neighbors. These samples are fixed implicitly by the algorithm's randomness and are used to guide the recursive exploration of the neighborhood of v at different paces during the level computation. We define these sets next.

2.2.1 Per-Iteration Samples

In this subsection, we describe how the samples associated with each vertex and iteration are generated. The parameters introduced here will be used throughout the paper.

Fix an arboricity α, an orientation target $r = x\alpha$ where $x = \Omega(\log n)$, and a bound on the number of iterations $T \stackrel{\text{def}}{=} \Theta(\log n)$.

For every vertex $v \in V$ and every iteration $t \in [T]$, the algorithm associates with v a collection of $\ell \stackrel{\text{def}}{=} \Theta(\log n)$ independent random neighbor samples

$$S_{v,1}^{(t)}, S_{v,2}^{(t)}, \ldots, S_{v,\ell}^{(t)} \subseteq N(v).$$

Each sample $S_{v,i}^{(t)}$ is generated by including every neighbor $u \in N(v)$ independently with probability $p \stackrel{\text{def}}{=} c \cdot \frac{\log n}{r}$, where c is a constant whose precise choice will be dictated by the inequalities appearing later in the proof.

Implementation details

Our analysis focuses on query complexity. By Lemma 2.3 of Dai, Ghaffari, and Portmann [5], the samples $S_{v,1}^{(t)}, \ldots, S_{v,\ell}^{(t)}$ can be generated in expected time proportional to their size. Consequently, the running time of our algorithm matches its query complexity up to constant factors.

The generation occurs according to the following schedule.

2.2.2 Per-Iteration Staggered Scheduling

Fix an iteration $t \in [T]$. We next describe the execution of iteration t of the algorithm which is essentially a staggered generation of the samples defined above.

Iteration t proceeds in steps $y = 1, 2, \ldots$ Let $i(y)$ denote the (unique) index of the bit that flips from 0 to 1 when incrementing $y - 1$ to y in binary. Equivalently, $i(y)$ is the index of the least significant 1 bit in the binary representation of y.

Scheduling rule

At time step y, the algorithm advances the generation of the sample $S_{v,i}^{(t)}$, where $i = i(y)$. Once the sample $S_{v,i}^{(t)}$ has been fully generated, the algorithm recursively processes its vertices using the same scheduling rule with the number of iterations decreased by 1. Thus, we obtain the following.

Execution frequency

The generation of $S_{v,i}^{(t)}$ and its recursion advance exactly at steps y satisfying

$$y \equiv 2^{i-1} \pmod{2^i}.$$

Therefore, over the first y steps, this advancement occurs $\Theta\left(\frac{y}{2^i}\right)$ times. Since each advancement accounts for 2^i underlying steps, the total processing time is obtained by multiplying the number of advancements by 2^i.

3 The algorithm for r-Orientation

Our LCA is organized into three procedures: Orient answers an edge query using the two endpoint levels; ComputeLevel tries iterations $t = 1, \ldots, T$ and returns the first successful one; and ComputeLevelIter implements a single iteration.

Algorithm 1 Orient(u, v): LCA for edge orientation.

Require: An undirected edge $\{u, v\}$
$a \leftarrow$ ComputeLevel(u)
$b \leftarrow$ ComputeLevel(v)
if $a < b$ **then**
    return $u \to v$
else if $b < a$ **then**
    return $v \to u$
else
    return orientation by vertex IDs
end if

The procedure ComputeLevel(v) determines the smallest iteration t in which v can be resolved within the probe budget by invoking ComputeLevelIter$(v, t, \cdot)$ for increasing t.

Algorithm 2 ComputeLevel(v).

Require: Vertex v
Ensure: $\ell(v) \in \{1, \ldots, L\} \cup \{\infty\}$
if $d(v) \leq 3\alpha$ **then**
    return 1
end if
for $t \leftarrow 1$ **to** T **do**
    $q \leftarrow 0$ ▷ local probes used in iteration t
    if ComputeLevelIter$(v, t, q) =$ **true then**
        return t
    end if
end for
return ∞

At iteration t, ComputeLevelIter explores sampled neighbors of v using the staggered schedule, and returns **true** iff it certifies that the sampled recursive exploration starting from v terminates without exceeding the local probe budget $B \stackrel{\text{def}}{=} \Theta(\alpha n \log n / r^2)$. More concretely, a sampled set is considered successful if all recursive calls made on vertices in that set return **true**. If a recursive call returns **false**, then this particular sampled set is discarded; however, the entire iteration does not necessarily fail, since the staggered schedule may still find another sampled set whose recursive calls all succeed. The iteration fails only if no sampled set succeeds before the budget is exceeded.

3.1 Analysis

In the analysis of our algorithm we use the following notion of processing time.

▶ **Definition 7** (Processing-time). *For a vertex v and an iteration $t \geq 1$, define* $\mathrm{Proc}_t(v)$ *recursively as*

$$\mathrm{Proc}_t(v) := \begin{cases} 1, & \text{if } d(v) \leq 3\alpha, \\ 1 + \min\limits_{i \in [\ell]} \left\{ 2^i \cdot \left(|S_{v,i}^{(t)}| + \sum\limits_{u \in S_{v,i}^{(t)}} \mathrm{Proc}_{t-1}(u) \right) \right\}, & \text{if } d(v) > 3\alpha. \end{cases}$$

and set $\mathrm{Proc}_0(v) := 1$ *for all* v.

Algorithm 3 ComputeLevelIter(v, t, q).

Require: Vertex v, iteration t, global counter q
Ensure: **true** iff iteration t succeeds for v

$q \leftarrow q + 1$
if $q > B$ **then**
    return false
end if
if $d(v) \leq 3\alpha$ **then**
    return true
end if
if $t = 0$ **then**
    return false
end if
Initialize the recursive processing of $S_{v,1}^{(t)}, \ldots, S_{v,\ell}^{(t)}$.
for each scheduling step $y = 1, 2, \ldots$ while $q \leq B$ **do**
    $i \leftarrow i(y)$
    Continue processing the vertices of $S_{v,i}^{(t)}$.
    For the next unprocessed vertex $u \in S_{v,i}^{(t)}$, invoke ComputeLevelIter$(u, t-1, q)$.
    if the recursive call returns **false then**
        Discard the sample $S_{v,i}^{(t)}$.
    else if all vertices of $S_{v,i}^{(t)}$ have been processed **then**
        return true
    end if
end for
return false

The factor 2^i appears because sample $S_{v,i}^{(t)}$ is advanced only once every 2^i scheduling steps. Thus, if the evaluation of this sample requires Q units of local work, then the global schedule spends $O(2^i Q)$ steps before this evaluation completes. This is why the processing time takes a minimum over terms of the form $2^i(\cdot)$.

By construction, $\mathrm{Proc}_t(v)$ upper bounds the number of probes used to evaluate iteration t rooted at v under the staggered schedule. For a fixed sample index i, recursively processing its sampled neighbors costs $\sum_{u \in S_{v,i}^{(t)}} \mathrm{Proc}_{t-1}(u)$, and the execution frequency contributes a multiplicative $\Theta(2^i)$ factor, giving the term $2^i(\cdot)$. Since the schedule runs all indices in parallel and stops when the first completes, we take the minimum over $i \in [\ell]$.

By Algorithm 2, the level of a vertex v is defined as follows.

▶ **Definition 8** (Level). *The level of a vertex v, $level(v)$ is defined to be the minimum $t \in T$ such that the processing time of* ComputeLevelIter$(v, t, \cdot)$ *is less than B, or ∞ if such t does not exist. In the latter case we call v and* overflow vertex.

3.1.1 Total expected work per iteration

In this subsection, we bound the expected total work per iteration, summed over all vertices, when excluding from the count the initial degree query performed for each vertex.

▷ Claim 9. For every $t \leq T$, $\mathbb{E}\left[\sum_{v \in V}(\mathrm{Proc}_t(v) - 1)\right] = O((n \log n)/x)$.

Proof. Recall that $p = c \cdot \frac{\log n}{r}$. Thus, for $r \geq 100c\alpha \log n$ it is bounded from above by $\frac{1}{100 \cdot \alpha}$. In each sample $S_{v,i}^{(t)}$, every neighbor of v is included independently with probability p.

Call a sample $S_{v,i}^{(t)}$ *good* if all sampled neighbors belong to strictly lower ideal peeling layers than v, i.e., $\ell(u) < \ell(v)$ for all $u \in S_{v,i}^{(t)}$ (Definitions 4 and 6). Since each vertex has at most 3α neighbors whose layer index is not strictly smaller than its own, we have

$$\Pr\left[S_{v,i}^{(t)} \text{ is good}\right] = (1-p)^{L_{\geq t}} \geq (1-p)^{3\alpha} \geq e^{-3/100} \geq 0.97.$$

Hence each sample is good with probability at least 0.97 and bad with probability at most 0.03. Let $i^\star$ be the index of the first good sample. Then

$$\Pr[i^\star = i] \leq (0.03)^{i-1} = q^{\,i-1}, \qquad q := 0.03.$$

Condition on the event that $i^\star = i$. By the definition of $\mathrm{Proc}_t(v)$, we have

$$\mathbb{E}[\mathrm{Proc}_t(v) - 1] \leq \sum_{i=1}^{\ell} \Pr[i^\star = i] \cdot 2^i \cdot \mathbb{E}\left[|S_{v,i}^{(t)}| + \sum_{u \in S_{v,i}^{(t)}} \mathrm{Proc}_{t-1}(u) \;\middle|\; i^\star = i \right].$$

Using the bound $\Pr[i^\star = i] \leq q^{\,i-1}$, this implies

$$\mathbb{E}[\mathrm{Proc}_t(v) - 1] \leq \sum_{i=1}^{\ell} q^{\,i-1} \cdot 2^i \, \mathbb{E}\left[\sum_{u \in S_{v,i}^{(t)}} |S_{v,i}^{(t)}| + \mathrm{Proc}_{t-1}(u) \;\middle|\; i^\star = i \right]. \tag{1}$$

Conditioned on the event that $S_{v,i}^{(t)}$ is a good sample, only neighbors u in strictly lower ideal layers than v can appear. Each such neighbor is included independently with probability p, so by linearity of expectation,

$$\mathbb{E}\left[|S_{v,i}^{(t)}| + \sum_{u \in S_{v,i}^{(t)}} \mathrm{Proc}_{t-1}(u) \;\middle|\; i^\star = i \right] = \sum_{\substack{u \in N(v) \\ \ell(u) < \ell(v)}} p \,(1 + \mathbb{E}[\mathrm{Proc}_{t-1}(u)]).$$

Plugging into (1) yields

$$\mathbb{E}[\mathrm{Proc}_t(v) - 1] \leq \sum_{i=1}^{\ell} q^{\,i-1} 2^i \sum_{\substack{u \in N(v) \\ \ell(u) < \ell(v)}} p\,(1 + \mathbb{E}[\mathrm{Proc}_{t-1}(u)]). \tag{2}$$

The geometric factors satisfy

$$\sum_{i=1}^{\ell} q^{\,i-1} 2^i \leq 2 \sum_{i=1}^{\infty} (2q)^{i-1} = \frac{2}{1-2q} = \frac{2}{1-0.06} < 3.$$

Therefore,

$$\mathbb{E}[\mathrm{Proc}_t(v) - 1] \leq 3 \sum_{\substack{u \in N(v) \\ \ell(u) < \ell(v)}} p\,(1 + \mathbb{E}[\mathrm{Proc}_{t-1}(u)]). \tag{3}$$

Summing (3) over all vertices v and rearranging gives

$$\sum_{v\in V}\mathbb{E}[\mathrm{Proc}_t(v)-1]\le 3p\sum_{u\in V}(1+\mathbb{E}[\mathrm{Proc}_{t-1}(u)])\cdot\left|v\in N(u):\ell(v)>\ell(u)\right|. \tag{4}$$

By the ideal peeling layers property, for every vertex u there are at most 3α neighbors v with $\ell(v)\ge\ell(u)$. Hence,

$$\sum_{v\in V}\mathbb{E}[\mathrm{Proc}_t(v)-1]\le 9\alpha p\sum_{u\in V}(1+\mathbb{E}[\mathrm{Proc}_{t-1}(u)])=9\alpha p\left(2|V|+\sum_{u\in V}\mathbb{E}[\mathrm{Proc}_{t-1}(u)-1]\right) \tag{5}$$

Let $R_t\stackrel{\text{def}}{=}\sum_{v\in V}\mathbb{E}[\mathrm{Proc}_t(v)-1]$. Then (5) implies

$$R_t\le 9\alpha p\left(2|V|+R_{t-1}\right).$$

Since $R_0=\sum_{v\in V}\mathbb{E}[\mathrm{Proc}_0(v)-1]=0$, and from the fact that $9\alpha p<0.5$ and that $9\alpha p=O((\log n)/x)$ it follows by induction that $R_t=O((n\log n)/x)$, as claimed ◁

▷ Claim 10. With high constant probability, the number of overflow vertices is at most $r/2$.

Proof. Observe that R_t is the expected total number of queries required to execute iteration t on *all* vertices, excluding only the first degree query performed on the starting vertex. By Claim 9,

$$\mathbb{E}\left[\sum_{v\in V}(\mathrm{Proc}_t(v)-1)\right]\le c_1(n\log n)/x,$$

where c_1 is a constant. Let

$$Z_t=\left|\left\{v\in V:\mathrm{Proc}_t(v)-1\ge c_2\cdot\frac{n\alpha\log n}{r^2}\right\}\right|,$$

where c_2 is a sufficiently large constant (the conditions it must satisfy will become apparent in the remainder of the proof). Then

$$Z_t\cdot c_2\frac{n\alpha}{r^2}\le\sum_{v\in V}(\mathrm{Proc}_t(v)-1).$$

Taking expectations,

$$\mathbb{E}[Z_t]\le\frac{c_1(n\log n)/x}{c_2\alpha n\log n/r^2}=O\left(\frac{c_1 r}{c_2}\right),$$

where we used $x=r/\alpha$. Choosing c_2 sufficiently large gives

$$\mathbb{E}[Z_t]\le\frac{r}{20}.$$

Therefore, by Markov's inequality,

$$\Pr\left[Z_t>\frac{r}{2}\right]\le\frac{\mathbb{E}[Z_t]}{r/2}\le\frac{1}{10}.$$

Thus, with high constant probability, at most $r/2$ vertices require at least $c_2\cdot\alpha n\log n/r^2$ additional queries, as claimed. ◁

3.2 Bounding the out-degree

▷ Claim 11. With high constant probability (over the randomness of the algorithm) the out-degree of every vertex is r.

Proof. Fix an iteration t, and consider only the randomness required by the algorithm for executing iterations $1, \ldots, t$, namely the sets $\{S_{v,i}^{(j)}\}_{j\in[t],\, i\in[\ell],\, v\in V}$. Let $U_{\leq t}$ denote the set of vertices that were unsuccessful in all iterations in $[t]$.

Let $v \in V$ be a vertex such that $|N(v) \cap U_{\leq t}| \geq r/2$. Fix an index $i \in [\ell]$. The probability that none of the vertices in $N(v) \cap U_{\leq t}$ belongs to $S_{v,i}^{(t)}$ is at most

$$(1-p)^{r/2} \leq e^{-pr/2} \leq 1/n^{c/2} \leq 1/n^2,$$

where the last inequality holds for an appropriate setting of c in the definition of p. By a union bound over all vertices v and all indices i, it follows that, with high probability, for every v and i, if $|N(v) \cap U_{\leq t}| \geq r/2$, then at least one vertex in $N(v) \cap U_{\leq t}$ belongs to $S_{v,i}^{(t+1)}$.

Consequently, it holds that $level(v) \geq t+1$ (note that a vertex v can be successful in iteration $t+1$ only if there exists $i \in [\ell]$ such that all vertices in $S_{v,i}^{(t+1)}$ were successful in one of the iterations in $[t]$ (this is a necessary but not a sufficient condition since the query budget is divided between the different procedures). We conclude that, with high probability, every vertex has at most $r/2$ neighbors whose level is greater than or equal to its own, excluding overflow vertices. Since w.h.c.p. the number of overflow vertices is bounded by $r/2$, this concludes the proof. ◁

We now prove Theorem 1.

Proof of Theorem 1. The claim on the query complexity follows from the construction as the probe complexity per edge query is $O\left(\frac{\alpha n \log^2 n}{r^2}\right)$, where the additional $\log n$ factor comes from trying $T = \Theta(\log n)$ iterations, and each iteration is stopped after budget $B = \Theta(\alpha n \log n / r^2)$. The claim on the success probability of the algorithm follows from Claim 11. ◀

References

1. Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1132–1139. SIAM, 2012. doi:10.1137/1.9781611973099.89.
2. Leonid Barenboim and Michael Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using nash-williams decomposition. *Distributed Comput.*, 22(5-6):363–379, 2010. doi:10.1007/S00446-009-0088-2.
3. Amartya Shankha Biswas, Talya Eden, Quanquan C. Liu, Ronitt Rubinfeld, and Slobodan Mitrovic. Massively parallel algorithms for small subgraph counting. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPIcs*, pages 39:1–39:28. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.39.
4. Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985. doi:10.1137/0214017.
5. Jiangqi Dai, Mohsen Ghaffari, and Julian Portmann. Constant approximation of arboricity in near-optimal sublinear time. In *Proceedings of the 2025 IEEE 66th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2025. doi:10.1109/FOCS63196.2025.00030.

6 Talya Eden, Saleet Mossel, and Dana Ron. Approximating the arboricity in sublinear time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9–12, 2022*, pages 2404–2425. SIAM, 2022. doi:10.1137/1.9781611977073.96.

7 Manuela Fischer, Mohsen Ghaffari, and Fabian Kuhn. Deterministic distributed edge-coloring via hypergraph maximal matching. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 180–191. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.25.

8 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, Hyatt Regency, New Brunswick, New Jersey, USA, October 9-11, 2016*, pages 625–634. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.73.

9 Mohsen Ghaffari and Hsin-Hao Su. Distributed degree splitting, edge coloring, and orientations. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2505–2523. SIAM, 2017. doi:10.1137/1.9781611974782.166.

10 Meng He, Ganggui Tang, and Norbert Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation – 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*, volume 8889 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 2014. doi:10.1007/978-3-319-13075-0_11.

11 Akash Kumar, C. Seshadhri, and Andrew Stolman. Random walks and forbidden minors III: poly(d/ϵ)-time partition oracles for minor-free graph classes. In *62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2021)*, pages 441–452. IEEE, 2021. doi:10.1109/FOCS52345.2021.00050.

12 Yishay Mansour and Shai Vardi. A local computation approximation scheme to maximum matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2013)*, volume 8096 of *Lecture Notes in Computer Science*, pages 260–273. Springer, 2013. doi:10.1007/978-3-642-40328-6_19.

13 David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of Algorithms*, 3(3):187–200, 1983.

14 Slobodan Mitrović, Ronitt Rubinfeld, and Mihir Singhal. Locally Computing Edge Orientations. In *32nd Annual European Symposium on Algorithms (ESA 2024)*, volume 308 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 89:1–89:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ESA.2024.89.

15 Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theor. Comput. Sci.*, 381(1-3):183–196, 2007. doi:10.1016/J.TCS.2007.04.040.

16 Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In Bernard Chazelle, editor, *Innovations in Computer Science – ICS 2011, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 223–238. Tsinghua University Press, 2011. URL: http://conference.iiis.tsinghua.edu.cn/ICS2011/content/papers/36.html.

17 Shay Solomon and Nicole Wein. Improved dynamic graph coloring. *ACM Journal of Experimental Algorithmics*, 26(1), June 2020. doi:10.1145/3392724.