CrossMark

Local Computation Algorithms for Graphs of Non-constant Degrees

Reut Levi[1] · **Ronitt Rubinfeld[2,3]** ·
Anak Yodpinyanee[2]

Abstract In the model of *local computation algorithms* (LCAs), we aim to compute the queried part of the output by examining only a small (sublinear) portion of the input. Many recently developed LCAs on graph problems achieve time and space complexities with very low dependence on n, the number of vertices. Nonetheless, these complexities are generally at least exponential in d, the upper bound on the degree of the input graph. Instead, we consider the case where parameter d can be moderately dependent on n, and aim for complexities with subexponential dependence on d, while maintaining polylogarithmic dependence on n. We present:

– a randomized LCA for computing maximal independent sets whose time and space complexities are quasi-polynomial in d and polylogarithmic in n;
– for constant $\varepsilon > 0$, a randomized LCA that provides a $(1 - \varepsilon)$-approximation to maximum matching with high probability, whose time and space complexities are polynomial in d and polylogarithmic in n.

Keywords Local computation algorithms · Randomized algorithms · Maximal independent set · Maximum matching

✉ Reut Levi
  reuti.levi@gmail.com

  Ronitt Rubinfeld
  ronitt@csail.mit.edu

  Anak Yodpinyanee
  anak@csail.mit.edu

[1] Ècole Normale Supèrieure and Universitè Paris Diderot, Paris, France

[2] CSAIL, MIT, Cambridge, MA 02139, USA

[3] The Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, Israel

1 Introduction

In the face of massive data sets, classical algorithmic models, where the algorithm reads the entire input, performs a full computation, then reports the entire output, are rendered infeasible. To handle these data sets, the model of *local computation algorithms* (LCAs) has been proposed. As defined in [37], these algorithms compute the queried part of the output by examining only a small (sublinear) portion of the input. Let us consider the problem of finding a maximal independent set (MIS) as an example. The algorithm $\mathscr{A}$ is given access to the input graph G, then it is asked a question: "is vertex v in the MIS?" The algorithm then explores only a small portion of G, and answers "yes" or "no." The set of vertices $\{v : \mathscr{A} \text{ answers "yes" on } v\}$ must indeed form a valid MIS of G. LCAs have been constructed for many problems, including MIS, maximal matching, approximate maximum matching, vertex coloring, and hypergraph coloring [4,14,27,28,36,37]. In our paper, we study MIS and approximate maximum matching; these are fundamental graph problems, well-studied in many frameworks, and moreover, tools and results for these problems have proven to be useful as building blocks for more sophisticated and specialized problems in the field.

The LCA framework is motivated by the circumstances where we focus on computing a small, specified portion of the output. This key characteristic of LCAs generalizes many other models from various contexts. For instance, LCAs may take the form of local filters and reconstructors [1,9–11,20,21,23,38]. Important applications include locally decodable codes (e.g., [40]), local decompression [13,30], and locally computable decisions for online algorithms and mechanism design [17,27]. There are a number of works on related models of local computation (e.g., [5,8,33,39]) as well as lower bounds for the LCA framework and other models (e.g., [16]).

Many recently developed LCAs on graph problems achieve time and space complexities with very low dependence on n, the number of vertices. Nonetheless, these complexities are at least exponential in d, the upper bound on the degree of the input graph. While these papers often consider d to be a constant, the large dependence on d may forbid practical uses of these algorithms. In this work we consider the case where the parameter d can be moderately dependent on n, and provide LCAs for complexities that have quasi-polynomial and even polynomial dependence on d, while maintaining polylogarithmic dependence on n. As noted in [27], whether there exist LCAs with polynomial dependence on d for these problems is an interesting open question. Our paper answers this question for the approximate maximum matching problem in the affirmative, and aims at providing techniques useful towards resolving other problems.

1.1 Our Contribution

This paper addresses the maximal independent set problem and the approximate maximum matching problem. The comparison between our results and other approaches is given in Table 1. All prior work on these problems only achieve time and query complexities with exponential dependence on d. Our paper provides the first LCAs whose complexities are quasi-polynomial and polynomial in d for these respective prob-

Table 1 The summary of complexities of various LCAs

Problem	Citation	Type	Time	Space
MIS	[37]	Randomized	$2^{O(d\log^2 d)}\log n$	$O(n)$
	[4]	Randomized	$2^{O(d\log^2 d)}\log^3 n$	$2^{O(d\log^2 d)}\log^2 n$
	[14]	Deterministic	$2^{O(d^2\log^2 d)}\log^* n^\dagger$	None
	[36]	Randomized	$2^{O(d)}\log^2 n$	$2^{O(d)}\log n\log\log n$
			$2^{O(d)}\log n\log\log n$	$2^{O(d)}\log^2 n$
	This paper	Randomized	$2^{O(\log^3 d)}\log^3 n$	$2^{O(\log^3 d)}\log^2 n$
Approximate maximum matching	[28]	Randomized	$O(\log^4 n)^\ddagger$	$O(\log^3 n)^\ddagger$
	[14]	Deterministic	$2^{\text{poly}(d)}\text{poly}(\log^* n)^\dagger$	None
	This paper	Randomized	$\text{poly}\{d,\log n\}$	$\text{poly}\{d,\log n\}$

For the approximate maximum matching problem, ε is assumed to be constant

$\dagger$ Query complexity, when time complexity is not explicitly given in the paper

$\ddagger$ Hidden dependence on d, which is at least $2^{O(d)}$ but not explicitly known

lems, while maintaining polylogarithmic dependence on n. More concretely, when d is non-constant, previously known LCAs have complexities with polylogarithmic dependence on n only when $d = O(\log\log n)$. Our LCAs maintain this dependence even when $d = \exp(\Theta((\log\log n)^{1/3}))$ for the MIS problem and $d = \text{poly}(\log n)$ for the approximate maximum matching problem. It is worth mentioning that our LCA for the MIS problem may be extended to handle other problems with reductions to MIS, such as maximal matching or $(d + 1)$-coloring, while maintaining similar asymptotic complexities.

1.2 Our Approaches

1.2.1 Maximal Independent Set

We provide an LCA for computing a MIS whose time and query complexities are quasi-polynomial in d. We construct a two-phase LCA similar to that of [37], which is based on Beck's algorithmic approach to Lovász local lemma [7]. In the first phase, we find a large partial solution that breaks the original graph into small connected components. The LCA for the first phase is obtained by applying the Parnas–Ron reduction which locally simulates distributed algorithms [34]. We improve upon the distributed algorithm used by [37] by occupying new insights from [6]. Via the Parnas–Ron reduction, the time and query complexities of the first phase of our algorithm remain subexponential in d. Then, in the second phase, we explore each component and solve our problems deterministically; the complexities of this phase are bounded by the component sizes. By employing a technique from [4], we reduce the amount of space required by our LCA so that it has roughly the same asymptotic bound as its time and query complexities.

1.2.2 Approximate Maximum Matching

We provide an LCA for computing a $(1 - \varepsilon)$-approximate maximum matching whose time and query complexities are polynomial in d. Our algorithm locally simulates the global algorithm based on Hopcroft and Karp's lemma [18]. This global algorithm begins with an empty matching, then for $\Theta(1/\varepsilon)$ iterations, augments the maintained matching with a maximal set of vertex-disjoint augmenting paths of increasing lengths. Yoshida et al. [41] show that this global algorithm has an efficient local simulation in expectation on the queries and the random tapes. We derive from their analysis that, on most random tapes, this simulation induces small query trees on most vertices. From this observation, we construct an efficient two-phase LCA as follows. In the first phase, we repeatedly check random tapes until we find a good tape such that the query trees for most vertices are small. This test can be performed by approximating the number of vertices whose query trees are significantly larger than the expected size through random sampling. In the second phase, we simulate the aforementioned algorithm using the acquired random tape. On queries for which the query trees are significantly large, we stop the computation and report that those edges do not belong to the matching. Our good tape from the first phase limits the number of such edges, allowing us to acquire the desired approximation with high probability.

1.3 Related Work

In our work we build on the Parnas–Ron reduction, proposed in their paper on approximating the size of a minimum vertex cover (VC) [34]. This reduction turns a k-round distributed algorithm into an LCA by examining all vertices at distance up to k from the queried vertex, then simulating the computation done by the distributed algorithm, invoking $d^{O(k)}$ queries to the input graph in total. Applying this technique on distributed subroutines with $O((\log d)/\varepsilon^3)$ and $O(\log d)$ rounds, they obtain a 2-approximation for the size of a minimum VC (with εn additive error) with query complexity $d^{O((\log d)/\varepsilon^3)}$, and a c-approximation for $c > 2$ using $d^{O(\log d)}/\varepsilon^2$ queries, respectively. Marko and Ron later improve this result to a $(2 + \varepsilon)$-approximation with query complexity $d^{O(\log(d/\varepsilon))}$ [29]. Rubinfeld et al. apply the same method on a subroutine for the MIS problem that requires $O(d \log d)$ rounds, which yields an LCA with query and time complexities $d^{O(d \log d)} \log n$ in [37].

Many useful techniques for designing LCAs originate from algorithms for approximating the solution size in sublinear time via random sampling. For example, if we wish to approximate the size of the minimum VC, we may sample a number of vertices and check whether each of them belongs to the minimum VC or not. The main difference, however, is that an LCA must be able to compute the answer to every query, while an approximation algorithm is not required to produce a consistent answer for every sample, and may use other properties of the problem to infer its answer. We make use of a number of common techniques from these approximation algorithms.

A powerful technique for bounding the query and time complexities for these approximation algorithms is the query tree method from the Nguyen–Onak algorithm [31]. This method aims to convert *global* algorithms that operate on the entire input

into *local* algorithms that adaptively make queries when a new piece of information is needed. To illustrate this approach, let us consider the MIS problem as an example. Recall the sequential greedy algorithm where we maintain an initially empty set I, then iterate through the vertex set in a certain order, adding each vertex to I if it does not already have a neighbor in I. Each vertex v will be in the resulting MIS if and only if none of v's neighbors that precede v in our order is already in the MIS. From this observation, we may create a local simulation of this algorithm as follows. To determine whether v is in the MIS, we make recursive queries on each of v's neighbors that precede v in our order, and check whether each of them is in the MIS. Then, return YES if none of the queries on these neighbors returns YES; otherwise, return NO. The structure of our recursive queries forms a *query tree*, whose size governs the complexities of our local simulation.

Nguyen and Onak apply this approach on a random order of vertices so that the size of the query tree, which determines the time and query complexities, can be probabilistically bounded. This method is used in [4,17,27,28,36], giving query complexities with polylogarithmic dependence on n for various problems. Unfortunately, the expected query tree size is exponential in d, which is considered constant in these papers. For certain problems, a slight modification of the Nguyen-Onak algorithm reduces the expected query tree size to $O(\bar{d})$, where $\bar{d}$ is the average degree of the input graph [32,41]. This gives algorithms with poly($\bar{d}$) query complexity for approximating the sizes of maximum matching and minimum VC with multiplicative and additive errors. These sublinear-time algorithms for approximating solution sizes do not actually compute explicit approximate solutions, and thus do not lead directly to LCAs. Nonetheless, we build on the query tree method in order to obtain an LCA for the approximate maximum matching problem whose query and time complexities are polynomial in d.

Recently, a new method for bounding the query tree sizes using graph orientation is given in [14] based on graph coloring, which improves upon LCAs for several graph problems. They reduce the query complexity of their algorithm for the MIS problem to $d^{O(d^2 \log d)} \log^* n$, giving the lowest dependence on n currently known, as well as a new direction for developing deterministic LCAs. This approach can also be extended back to improve distributed algorithms for certain cases [15].

While all of these LCAs have complexities with exponential dependence on d for the problems studied in this paper, there has been no significant lower bound. To the best of our knowledge, the only lower bound is of $\Omega(\bar{d})$, which can be derived from the lower bound for approximation algorithms for the minimum VC problem, given by Parnas and Ron [34].

2 Preliminaries

2.1 Graphs

The input graph $G = (V, E)$ is a simple undirected graph with $|V| = n$ vertices and a bound on the degree d, which is allowed to be dependent on n. Both parameters n and d are known to the algorithm. Let $\bar{d}$ denote the average degree of the graph.

Each vertex $v \in V$ is represented as a unique positive ID from $[n] = \{1, \ldots, n\}$. For $v \in V$, let $\deg_G(v)$ denote the degree of v, $\Gamma_G(v)$ denote the set of neighbors of v, and $\Gamma_G^+(v) = \Gamma_G(V) \cup \{v\}$. For $U \subseteq V$, define $\Gamma_G^+(U) = \cup_{u \in U} \Gamma_G^+(u)$. The subscript G may be omitted when it is clear from the context.

We assume that the input graph G is given through an adjacency list oracle $\mathcal{O}^G$ which answers neighbor queries: given a vertex $v \in V$ and an index $i \in [d]$, the ith neighbor of v is returned if $i \leq \deg(v)$; otherwise, $\bot$ is returned. For simplicity, we will also allow a degree query which returns $\deg(v)$ when v is given; this can be simulated via a binary-search on $O(\log d)$ neighbor queries.

An *independent set* I is a set of vertices such that no two vertices in I are adjacent. An independent set I is a *maximal independent set* if no other vertex can be added to I without violating this condition.

A *matching* M is a set of edges such that no two distinct edges in M share a common endpoint. A matching is a *maximal matching* if no other edge can be added to M out violating this condition. Let $V(M)$ denote the set of matched vertices, and $|M|$ denote the size of the matching, defined to be the number of edges in M. A *maximum matching* is a matching of maximum size.

2.2 Local Computation Algorithms

We adopt the definition of local computation algorithms from [37], in the context of graph computation problems given an access to the adjacency list oracle $\mathcal{O}^G$.

Definition 1 A *local computation algorithm* $\mathscr{A}$ for a computation problem is a (randomized) algorithm with the following properties. $\mathscr{A}$ is given access to the adjacency list oracle $\mathcal{O}^G$ for the input graph G, a tape of random bits, and local read-write computation memory. When given an input (query) x, $\mathscr{A}$ must compute an answer for x. This answer must only depend on x, G, and the random bits. The answers given by $\mathscr{A}$ to all possible queries must be consistent; namely, all answers must constitute some valid solution to the computation problem.

The complexities of an LCA $\mathscr{A}$ can be measured in various different aspects, as follows.

– The *query complexity* is the maximum number of queries that $\mathscr{A}$ makes to $\mathcal{O}^G$ in order to compute an answer (to the computation problem) for any single query.[1]
– The *time complexity* is the maximum amount of time that $\mathscr{A}$ requires to compute an answer to any single query. We assume that each query to $\mathcal{O}^G$ takes a constant amount of time.
– The *space complexity* is the maximum total size of the random tape and the local computation memory used by $\mathscr{A}$ to answer any single query.
– The *success probability* is the probability that $\mathscr{A}$ consistently answers all queries.

---

[1] Some recent work on LCAs uses the alternative term "probe" to refer to queries that $\mathscr{A}$ makes to $\mathcal{O}^G$, in order to distinguish them from queries that we ask $\mathscr{A}$ to answer; the corresponding complexity is called the *probe complexity*. We choose not to adopt this notation since both types of queries will often be indistinguishable when we construct oracles for intermediate graphs during the execution of our algorithms. It will also contradict with the term "query tree," which is unanimously used in other work on this method.

In this paper, we refer to the time and query complexities rather exchangeably: while the time complexity may be much larger than the query complexity in certain cases, for all LCAs considered here, the time complexities are only roughly a factor of $O(\log n)$ larger than the query complexities. The space complexity of our LCAs are dominated by the size of the random tape, so we often refer to the space complexity as *seed length* instead. As for the success probability, we consider randomized LCAs that succeed *with high probability*; that is, the success probability can be amplified to reach $1 - n^{-c}$ for any positive constant c without asymptotically increasing other complexities.

2.3 Parnas–Ron Reduction

Some of our algorithms apply the reduction from distributed algorithms to LCAs proposed by Parnas and Ron [34]. This reduction was originally created as a subroutine for approximation algorithms. The model of distributed computation they consider is the $\mathscr{LOCAL}$ model, informally described as follows. The input graph G, which is the input of our computation problem, also describes the network graph on which the distributed algorithm operates: computation can be performed locally at each vertex, and each edge connects a pair of vertices that may communicate directly with each other. The algorithm operates in synchronous communication rounds. In a single round, each vertex may send an arbitrarily large message to each of its neighbors; once a vertex receive all messages from its neighbors, it may perform an unlimited amount of computation based on the information it currently has. The goal is to design distributed algorithms that minimize the number of rounds required for every vertex to determine its answer (e.g., whether it is in the MIS).

Suppose that when a distributed algorithm $\mathscr{A}$ is executed on a graph G with degree bounded by d, each vertex v is able to compute some function f within k communication rounds. Then this function f must only depend on the subgraph of G induced by vertices at distance at most k from v. We can then create an LCA $\mathscr{A}'$ that computes f by simulating $\mathscr{A}$. Namely, $\mathscr{A}'$ first queries the oracle to learn the structure of this subgraph, then makes the same decision on this subgraph as $\mathscr{A}$ would have done. In total, this reduction requires $d^{O(k)}$ queries to $\mathscr{O}^G$.

2.4 Construction of Random Bits and Orderings

While our LCAs rely on random bits and orderings, we do not require all bits or orderings to be truly random: LCAs tolerate some dependence or bias, as they only access a small portion of such random instance in each query. We now provide some definitions and theorems we will use to construct our LCAs.

2.4.1 Random Bits

We will use the following construction to generate k-wise independent random bits from the seed (truly random bits) given on the random tape.

Theorem 1 ([3]) *For* $1 \leq k \leq m$, *there is a construction of k-wise independent random bits* $x_1, \ldots, x_m$ *with seed length* $O(k \log m)$. *Furthermore, for* $1 \leq i \leq m$, *each* x_i *can be computed in space* $O(k \log m)$.

Note here that each random bit generated from this construction is either 0 or 1 with equal probability. Nonetheless, for any positive integer q, we may generate a random bit that is 1 with probability exactly $1/q$ using $O(\log q)$ such truly random bits.

2.4.2 Random Orderings

For $n \geq 1$, let S_n denote the set of all permutations on $[n]$. Some of our LCAs make use of random permutations of the vertex set. Generating such uniformly random permutations requires $\Omega(n \log n)$ truly random bits. Nonetheless, we apply the method from [4] to construct good random permutations for our LCAs.

We generate our permutations by assigning a random value $r(v)$ to each element v, and rank our elements according to these values. More formally, an *ordering* of $[n]$ is an injective function $r : [n] \to \mathscr{R}$ where $\mathscr{R}$ is some totally ordered set. Let $v_1, \ldots, v_n$ be the elements of $[n]$ arranged according to their values mapped by r; that is, $r(v_1) < \cdots < r(v_n)$. We call the permutation $\pi = (v_1, \ldots, v_n)$ of $[n]$ corresponding to this ordering the *projection* of r onto S_n. We may refer to $r(v)$ as the *rank* of v. In our construction, the size of the range of $\mathscr{R}$ is polynomial in n.

A *random ordering* $\mathscr{D}$ of $[n]$ is a distribution over a family of orderings on $[n]$. For any integer $2 \leq k \leq n$, we say that a random ordering $\mathscr{D}$ is *k-wise independent* if for any subset $S \subseteq [n]$ of size k, the restriction of the projection onto S_n of $\mathscr{D}$ over S is uniform over all the $k!$ possible orderings among the k elements in S. A random ordering $\mathscr{D}'$ is *ε-almost k-wise independent* if there exists some k-wise independent random ordering $\mathscr{D}$ such that the statistical distance between $\mathscr{D}$ and $\mathscr{D}'$ is at most ε.

We shall use the following construction from Alon et al. [4].

Theorem 2 ([4]) *Let* $n \geq 2$ *be an integer and let* $2 \leq k \leq n$. *Then there is a construction of* $(1/n^2)$*-almost k-wise independent random ordering over* $[n]$ *whose seed length is* $O(k \log^2 n)$.

3 Maximal Independent Set

3.1 Overview

Our algorithm consists of two phases. The first phase of our algorithm computes a large MIS, using a variation of Luby's randomized distributed algorithm [26]. Luby's algorithm can be briefly described as follows. We begin with an initially empty independent set I, then repeatedly add more vertices to I. In each round, each vertex v tries to put itself into I with some probability. It succeeds if none of its neighbors also tries to do the same in that round; in this case, v is added to I, and $\Gamma^+(v)$ is removed from the graph.

By repeating this process with carefully chosen probabilities, we show that once the first phase terminates, the remaining graph contains no connected component of size

larger than $d^4 \log n$ with high probability. This phase is converted into an LCA via the Parnas–Ron reduction. Lastly, in the second phase, we locally compute an MIS of the remaining graph by simply exploring the component containing the queried vertex.

3.2 Distributed Algorithm: Phase 1

The goal of the first phase is to find an independent set I such that removing $\Gamma^+(I)$ breaks G into components of small sizes. We design our variation of Luby's algorithm based on Beck's algorithmic approach to Lovász local lemma [7]; this approach has been widely applied in many contexts (e.g., [2,6,22,37]). We design our algorithm based on the degree reduction idea from [6].[2] Our algorithm turns out to be very similar to the WEAK- MIS algorithm from a recent paper by Chung et al. [12]. We state their version, given in Algorithm 1, so that we may cite some of their results.

Let us say that a vertex v is *active* if $v \notin \Gamma^+(I)$; otherwise v is *inactive*. As similarly observed in [35], applying a round of Luby's algorithm with selection probability $1/(d+1)$ on a graph with maximum degree at most d makes each vertex of degree at least $d/2$ inactive with constant probability. To apply this observation, in each iteration, we first construct a graph G' of active vertices. Next, we apply Luby's algorithm so that each vertex of degree at least $d/2$ becomes inactive with constant probability. We then remove the remaining high-degree vertices from G' (even if they may still be active). As the maximum degree of G is halved, we repeat this similar process for $\lceil \log d \rceil$ stages until G' becomes edgeless, where every vertex can be added to I. Since each vertex becomes high-degree with respect to the maximum degree of G' at some stage, each iteration gives every vertex a constant probability to become inactive.

Algorithm 1 Chung et al.'s WEAK- MIS algorithm

1: **procedure** WEAK- MIS(G, d)
2:     $I \leftarrow \emptyset$
3:     **for** iteration $i = 1, \ldots, c_1 \log d$ **do**   ▷ c_1 is a sufficiently large constant
4:         $G' \leftarrow G[V \setminus \Gamma^+(I)]$
5:         **for** stage $j = 1, \ldots, \lceil \log d \rceil$ **do**
6:             $V_j \leftarrow \{v \in V(G') : \deg_{G'}(v) \geq d/2^j\}$
7:             each $v \in V(G')$ selects itself with probability $p_j = 1/(\frac{d}{2^{j-1}} + 1)$
8:             **if** v is the only vertex in $\Gamma^+_{G'}(v)$ that selects itself **then**
9:                 add v to I and remove $\Gamma^+_{G'}(v)$ from G'
10:             remove V_j from G'
11:         add $V(G')$ to I
12:     **return** I

Chung et al. use WEAK- MIS to construct an independent set such that the probability that each vertex remains active is only $1/\mathrm{poly}(d)$ [12]. We cite the following

---

[2] Applying a similar reduction on the unmodified version gives an LCA with complexities $2^{O(\log^3 d + \log d \log \log n)}$.

useful lemma that captures the key idea explained earlier, and include their proof for completeness.

Lemma 1 ([12]) *In Algorithm 1, if $v \in V_j$, then v remains active after stage j with probability at most $p = 1 - 1/4e^2$.*

Proof For each $u \in \Gamma_{G'}(v)$, let E_u denote the event where u is the only vertex in $\Gamma_{G'}^+(\{u, v\})$ that selects itself. Since the maximum degree in G' is at most $d/2^{j-1}$, then

$$\Pr[E_u] = p_j(1 - p_j)^{|\Gamma_{G'}^+(\{u,v\})|} \geq p_j(1 - p_j)^{2d/2^{j-1}} \geq \frac{p_j}{e^2}.$$

Notice that E_u is disjoint for each $u \in \Gamma_{G'}(v)$. Since $v \in V_j$, then $\deg_{G'}(v) \geq d/2^j$. Thus v becomes inactive with probability at least

$$\sum_{u \in \Gamma_{G'}(v)} \Pr[E_u] \geq \deg_{G'}(v) \cdot \left(\frac{p_j}{e^2}\right) \geq \frac{1}{4e^2},$$

as desired.   □

Observe that for v to remain active until the end of an iteration, it must be removed in step 10 due to its high degree. (If v were removed in line 9 or line 11, then v would have become inactive since either v or one of its neighbors is added to I.) Therefore, v must belong to one of the sets V_j. So, each vertex may remain active throughout the iteration with probability at most p. After $\Omega(\log d)$ iterations, the probability that each vertex remains active is only $1/\text{poly}(d)$, as desired.

Now we follow the analysis inspired by that of [6] to prove the guarantee on the maximum size of the remaining active components. Consider a set $S \subseteq V$ such that $\text{dist}_G(u, v) \geq 5$ for every distinct $u, v \in S$. We say that S is active if every $v \in S$ is active. As a generalization of the claim above, we show the following lemma.

Lemma 2 *Let $S \subseteq V$ be such that $\text{dist}_G(u, v) \geq 5$ for every distinct $u, v \in S$, then S remains active until WEAK- MIS terminates with probability at most $d^{-c_1'|S|}$ where $c_1' = -c_1 \log p$.*

Proof First let us consider an individual stage. Suppose that S is active at the beginning of this stage. By Lemma 1, each vertex $v \in S \cap V_j$ remains active after this stage with probability at most p. Notice that for each round of Luby's algorithm, whether v remains active or not only depends on the random choices of vertices within distance 2 from v: we need to know whether each $u \in \Gamma^+(v)$ is added to I, which can be determined based on the random choices of all vertices of $\Gamma^+(u)$. Since the vertices in S are at distance at least 5 away from one another, the events for all vertices in S are independent. Thus, the probability that S remains active after this stage is at most $p^{|S \cap V_j|}$.

Now we consider an individual iteration. Suppose that S is active at the beginning of this iteration. By applying an inductive argument on each stage, the probability that

S remains active at the end of this iteration is at most $p^{\sum_{j=1}^{\lceil \log d \rceil} |S \cap V_j|}$. Recall that for S to remain active after this iteration, every $v \in S$ must belong to some set V_j. So, $\sum_{j=1}^{\lceil \log d \rceil} |S \cap V_j| = |S|$. Thus, S remains active with probability at most $p^{|S|}$.

Lastly, we apply the inductive argument on each iteration to obtain the desired bound of $(p^{|S|})^{c_1 \log d} = d^{|S|(c_1 \log p)} = d^{-c_1'|S|}$. $\qquad\square$

Now we are ready to apply Beck's approach to prove the upper bound on the maximum size of the remaining active components ([7]; see also [37]). For completeness, we also provide the proofs that we adopt from other work.

Theorem 3 WEAK- MIS(G, d) *computes an independent set I of an input graph G within $O(\log^2 d)$ communication rounds, such that the subgraph of G induced by active vertices contains no connected component of size larger than $d^4 \log n$ with probability at least $1 - 1/\text{poly}(n)$.*

Proof Let T be a tree embedded on the *distance*-5 graph, defined as $(V, \{(u, v) : \text{dist}_G(u, v) = 5\})$, such that $\text{dist}_G(u, v) \geq 5$ for every distinct $u, v \in V(T)$. Let $S = V(T)$ be its vertex set, and $s = |S|$ denote its size. By Lemma 2, the probability that S remains active is $d^{-c_1's}$. First we give the following claim that bounds the number of distinct possible trees. $\qquad\square$

Claim 3 ([37]) *There are at most $n(4d^5)^s$ distinct trees T of size s embedded on the distance-5 graph.*

Proof of Claim 3 First, recall that the Catalan numbers count non-isomorphic ordered trees (rooted trees such that the children of each vertex are ordered; also known as (rooted) plane trees). Specifically, the number of such trees with s vertices is

$$C_{s-1} = \frac{1}{s-1}\binom{2s-2}{s-1} < 4^s,$$

where C_{s-1} denote the $(s-1)^{\text{th}}$ Catalan number.

Consider a specific rooted tree T^* on s vertices. Let us bound the number of distinct trees T on the distance-5 graph that are isomorphic to T^*. Consider the vertices of T^* in pre-order. The root of T^* can be mapped to any of the n vertices. Each of its $s - 1$ subsequent vertices must be adjacent to the previously chosen vertex of T that corresponds to its parent in T^*, so the number of possible choices for each vertex is bounded by the maximum degree. Recall that the distance-5 graph has maximum degree at most $d(d - 1)^4 < d^5$. Thus there are at most $n(d^5)^s$ possible trees T isomorphic to T^*.

Since there are less than 4^s non-isomorphic ordered trees on s vertices, and we may construct up to $n(d^5)^s$ trees on the distance-5 graph according to each structure, then there are at most $n(4d^5)^s$ distinct possible trees T, as desired. $\qquad\square$

Since there are at most $n(4d^5)^s$ distinct trees of size s embedded on the distance-5 graph, the expected number of such trees whose vertex sets remain active is at most $n(4d^5)^s \cdot d^{-c_1's}$. Note that we may obtain any arbitrary large constant c_1' by increasing

the constant c_1 in Algorithm 1. Thus, for $s = \log n$, this quantity is bounded above by $1/\text{poly}(n)$. By Markov's inequality, all such sets S are inactive with probability at least $1 - 1/\text{poly}(n)$.

Next, we consider the following observation from [7].

Claim 4 ([7]) *Any connected subgraph $H \subseteq G$ of size at least sd^4 contains some set S of size s such that $dist_G(u, v) \geq 5$ for every distinct $u, v \in S$.*

Proof of Claim 4 We prove this claim by constructing $S = \{v_1, \ldots, v_s\}$ for the connected subgraph H as follows. We choose the first vertex v_1 from H arbitrarily. Next, we choose each subsequent vertex v_j so that the maximum distance from v_j to the previously chosen vertices $v_1, \ldots, v_{j-1}$ is exactly 5. To show that such vertex v_j exists, notice that the number of vertices that are at distance at most 4 from every previously chosen vertex is at most $(j-1)(1 + d + d(d-1) + d(d-1)^2 + d(d-1)^3) < sd^4$ (for $d \geq 2$). Since there are at least sd^4 vertices in H, then there exists some vertex $u \in H$ at distance at least 5 from all previously chosen vertices. As H is connected, there exists a path in H from v_1 to u. Then one of the vertices on this path must be at distance exactly 5 from $\{v_1, \ldots, v_{j-1}\}$, satisfying the required condition for v_j.

Notice that from our construction, all vertices in S must be at distance at least 5 from one another. Since each vertex v_j is at distance exactly 5 from some previously chosen vertex v_i, then v_i and v_j are adjacent in the distance-5 graph. Thus, S is a vertex set for some tree T embedded on the distance-5 graph. □

Recall that all such sets S are inactive with probability at least $1 - 1/\text{poly}(n)$. Since a connected subgraph of G of size at least $d^4 \log n$ must contain some such set S, the probability that any large connected component remains active after WEAK- MIS terminates is also bounded above by $1/\text{poly}(n)$. □

3.3 Constructing the LCA: Phase 1

We now provide the Parnas–Ron reduction of WEAK- MIS in Algorithm 1 into an LCA. Let us start with a single stage, given as procedure LC- MIS- STAGE in Algorithm 2. The given parameters are the graph G' (via oracle access), the degree bound d, the queried vertex v, the iteration number i and the stage number j. Given a vertex v, this procedure returns one of the three states of v at the end of iteration i, stage j:

– YES if $v \in I$ (so it is removed from G')
– NO if $v \notin I$ and it is removed from G'
– $\perp$ otherwise, indicating that v is still in G'

For simplicity, we assume that the local algorithm has access to random bits in the form of a publicly accessible function $B : V \times [c_1 \log d] \times [\lceil \log d \rceil] \rightarrow \{0, 1\}$ such that $B(v, i, j)$ returns 1 with the selection probability p_j (from Algorithm 1, line 7) and returns 0 otherwise. This function can be replaced by a constant number of memory accesses to the random tape, and we will explore how to reduce the required amount of random bits in Sect. 3.5. Note also that we are explicitly giving the oracle $\mathscr{O}^{G'}$ as a parameter rather than the actual graph G'. This is because we will eventually

Algorithm 2 LCA of Phase 1 for a single stage of WEAK- MIS

```
1: procedure LC- MIS- STAGE(𝒪^{G'}, d, v, i, j)
2:     if j = 0 then return ⊥
3:     if LC- MIS- STAGE(𝒪^{G'}, d, v, i, j − 1) ≠ ⊥ then
4:         return LC- MIS- STAGE(𝒪^{G'}, d, v, i, j − 1)
5:     for each u within distance 2 from v on G' do
6:         if LC- MIS- STAGE(𝒪^{G'}, d, u, i, j − 1) ≠ ⊥ then
7:             status(u) ← removed
8:         else
9:             if B(u, i, j) = 1 then status(u) ← selected
10:            else status(u) ← not selected
11:    if status(v) = selected AND ∀u ∈ Γ_{G'}(v): status(u) ≠ selected then
12:        return YES
13:    if status(v) = not selected then
14:        if ∃u ∈ Γ_{G'}(v): status(u) = selected AND ∀w ∈ Γ_{G'}(u): status(w) ≠ selected then
15:            return NO
16:    if |{u ∈ Γ_{G'}(v) : status(u) ≠ removed}| ≥ d/2^j then
17:        return NO
18:    return ⊥
```

simulate oracles for other graphs, but we never concretely create such graphs during the entire computation. Observe that each call to LC- MIS- STAGE on stage j invokes up to $O(d^2)$ calls on stage $j − 1$. Simulating all stages by invoking this function with $j = \lceil \log d \rceil$ translates to $d^{O(\log d)}$ queries to the the base level $j = 0$.

Next we give the LCA LC- MIS- ITERATION for computing a single iteration in Algorithm 3. The parameters are similar to that of LC- MIS- STAGE, but the return values are slightly different:

– YES if $v \in I$ (so it is inactive)
– NO if $v \in \Gamma_G^+(I) \setminus I$ (so it is inactive)
– ⊥ otherwise, indicating that v is still active

In case v is still active by the end of iteration $i − 1$, we must simulate iteration i using LC- MIS- STAGE. We must return YES if $v \in I$ (line 7 of Algorithm 1); which may occur in two cases. It can be added to I in some stage, making LC- MIS- STAGE returns YES (corresponding to line 9 of Algorithm 1). It may also remain in G' through all iterations, making LC- MIS- STAGE return ⊥; v is then added to I because it is isolated in G' (line 11 of Algorithm 1). We note that the conditions in lines 6 and 8 of Algorithm 3 handle both cases, and are written abbreviately as "≠ NO."

To implement this LCA, we must simulate an adjacency list oracle for G' using the given oracle for G, which can be done as follows. For a query on vertex v, we call LC- MIS- ITERATION($\mathcal{O}^G, d, u, i − 1$) on all vertices u at distance at most 2 away from v. This allows us to determine whether each neighbor of v is still active at the beginning of iteration i, as well as providing degree queries. We then modify the ordering of the remaining neighbors (by preserving the original ordering, for example) to consistently answer neighbor queries. That is, $\mathcal{O}^{G'}$ can be simulated through at most $O(d^2)$ function calls of the form LC- MIS- ITERATION($\mathcal{O}^G, d, u, i − 1$).

Using this subroutine, the LCA LC- MIS- PHASE1 for WEAK- MIS can be written compactly as given in Algorithm 2. Via a similar inductive argument, we can show that

---

Algorithm 3 LCA of Phase 1 for a single iteration of WEAK- MIS

1: **procedure** LC- MIS- ITERATION($\mathcal{O}^G, d, v, i$)
2:     **if** $i = 0$ **then return** $\perp$
3:     **if** LC- MIS- ITERATION($\mathcal{O}^G, d, v, i - 1$) $\neq \perp$ **then**
4:         **return** LC- MIS- ITERATION($\mathcal{O}^G, d, v, i - 1$)
5:     $\mathcal{O}^{G'} \leftarrow$ oracle for the subgraph induced by $\{u : $ LC- MIS- ITERATION($\mathcal{O}^G, d, u, i - 1$) $= \perp\}$
6:     **if** LC- MIS- STAGE($\mathcal{O}^{G'}, d, v, i, \lceil \log d \rceil$) $\neq$ NO **then**
7:         **return** YES
8:     **else if** $\exists u \in \Gamma_{G'}(v)$: LC- MIS- STAGE($\mathcal{O}^{G'}, d, u, i, \lceil \log d \rceil$) $\neq$ NO **then**
9:         **return** NO
10:     **return** $\perp$

---

each call to LC- MIS- PHASE1 translates to $d^{O(\log^2 d)} = 2^{O(\log^3 d)}$ calls to the original oracle $\mathcal{O}^G$. The running time for the LCA is clearly given by the same bound. The memory usage is given by the amount of random bits of B, which is $O(n \log^2 d)$. We summarize the behavior of this LCA through the following lemma.

Lemma 5 LC- MIS- PHASE1 *is a local computation algorithm that computes the distributed* WEAK- MIS *algorithm with time complexity* $2^{O(\log^3 d)}$ *and space complexity* $O(n \log^2 d)$.

Algorithm 4 LCA of Phase 1 (WEAK- MIS)

1: **procedure** LC- MIS- PHASE1($\mathcal{O}^G, d, v$)
2:     **return** LC- MIS- ITERATION($\mathcal{O}^G, d, v, c_1 \log d$)

3.4 Constructing the LCA: Phase 2 and the Full LCA

Let G'' be graph G induced by active vertices after WEAK- MIS terminates. By Theorem 3, with high probability, G'' contains no component of size exceeding $d^4 \log n$. Therefore, to determine whether an active vertex v is in the MIS, we first apply a breadth-first search until all vertices in C_v, the component containing v, are reached. Then we compute an MIS deterministically and consistently (by choosing the lexicographically first MIS, for instance). This procedure is summarized as LC- PHASE2 in Algorithm 5.

Algorithm 5 LCA of Phase 2

1: **procedure** LC- PHASE2($\mathcal{O}^{G''}, d, v$)
2:     breadth-first search for $d^4 \log n$ steps on G'' to find C_v, the component of G'' containing v
3:     **if** $|C_v| > d^4 \log n$ **then report** ERROR
4:     deterministically compute an MIS I_{C_v} of C_v
5:     **if** $v \in I_{C_v}$ **then return** YES
6:     **else return** NO

The algorithm only reports ERROR when the component size exceeds the bound from Theorem 3. Clearly, a call to LC- PHASE2 makes at most poly(d) $\log n$ queries

to $\mathscr{O}^{G''}$ in total. The lexicographically first maximal independent set of C_v can be computed via a simple greedy algorithm with $O(d \cdot |C_v|)$ time complexity. Overall, both time and query complexities are poly$\{d, \log n\}$.

Combining both phases, we obtain the LCA LC- MIS for computing an MIS as given in Algorithm 6. We now prove the following theorem.

Algorithm 6 LCA for computing a maximal independent set

1: **procedure** LC- MIS($\mathscr{O}^G, d, v$)
2:    **if** LC- MIS- PHASE1($\mathscr{O}^G, d, v$) $\neq \perp$ **then**
3:       **return** LC- MIS- PHASE1($\mathscr{O}^G, d, v$)
4:    $\mathscr{O}^{G''} \leftarrow$ oracle for the subgraph induced by $\{u :$ LC- MIS- PHASE1($\mathscr{O}^G, d, u$) $= \perp\}$
5:    **return** LC- MIS- PHASE2($\mathscr{O}^{G''}, d, v$)

Theorem 4 *There exists a randomized local computation algorithm that computes a maximal independent set of G with time complexity $2^{O(\log^3 d)} \log n$ and space complexity $O(n \log^2 d)$.*

Proof To obtain the time complexity, recall from Lemma 5 that each call to LC-PHASE1 can be answered within $2^{O(\log^3 d)}$ time using $2^{O(\log^3 d)}$ queries to $\mathscr{O}^G$. Thus the adjacency list oracle $\mathscr{O}^{G''}$ can be simulated with the same complexities. The LCA for Phase 2 makes poly$(d) \log n$ queries to $\mathscr{O}^{G''}$, resulting in $2^{O(\log^3 d)} \log n$ total computation time and queries. The required amount of space is dominated by the random bits used in Phase 1. The algorithm only fails when it finds a component of size larger than $d^4 \log n$, which may only occur with probability $1/\text{poly}(n)$ as guaranteed by Theorem 3. $\qquad\square$

3.5 Reducing Space Usage

In this section, we directly apply the approach from [4] to reduce the amount of random bits used by our LCA, thus proving the following theorem.

Theorem 5 *thmresmem There exists a randomized local computation algorithm that computes a maximal independent set of G with seed of length $2^{O(\log^3 d)} \log^2 n$ and time complexity $2^{O(\log^3 d)} \log^3 n$.*

Proof Observe that the MIS algorithm in Theorem 4 constructed throughout this section does not require fully independent random bits in function B. Our algorithm can answer a query for any vertex v by exploring up to $q = 2^{O(\log^3 d)} \log n$ vertices in total. So, random bits used by vertices not explored by a query do not affect our answer. One bit is used by each vertex in each round of Phase 1, and thus we only create $b = O(\log^2 d)$ random bits for each vertex. Therefore, out of $O(n \log^2 d)$ bits from function B, only $q \cdot b$ bits are relevant for each query.

To generate random bits for function B, we will apply Theorem 2.4.2. As we may require bits of function B to be 1 with probability as low as $1/(d + 1)$, we

will need up to $\lceil \log d \rceil$ bits from this construction to obtain one bit for function B. So in our case, we have $k = \lceil \log d \rceil \cdot q \cdot b = 2^{O(\log^3 d)} \log n$ and $m = \lceil \log d \rceil \cdot O(n \log^2 d) = O(n \log^3 d)$. Thus the amount of space can be reduced to $O(k \log m) = 2^{O(\log^3 d)} \log n \cdot \log(n \log^3 d) = 2^{O(\log^3 d)} \log^2 n$. Similarly, the required amount of time for computing each random bit becomes $2^{O(\log^3 d)} \log^2 n$.   □

3.6 Other Remarks

The proposed LCA for the maximal independent set problem can be used as a building block for other problems. For example, a $(d+1)$-coloring can be computed by finding a maximal independent set of $G \times K_{d+1}$ where K_{d+1} is a clique of size $d+1$, resulting in an LCA within the same asymptotic complexities (see also [24]).

We may apply our MIS algorithm to the line graph $L(G)$ in order to compute a maximal matching on G within the same asymptotic complexities as Theorem 5. Nonetheless, in [19], Israeli and Itai proposed a randomized distributed algorithm which takes $O(\log n)$ rounds to compute a maximal matching. Similarly to the MIS problem, Barenboim et al. also create a variant of this algorithm that, within $O(\log d)$ rounds, finds a large matching that breaks the remaining graph into small connected components [6]. Specifically, by running Algorithm 7, the remaining graph satisfies the following lemma.

Algorithm 7 Barenboim et al.'s variant of Israeli and Itai's algorithm for computing a partial matching (simplified)

1: **procedure** DISTRIBUTED- MM- PHASE1(G, d)
2:     initialize matching $M = \emptyset$
3:     **for** $i = 1, \ldots, c_2 \log d$ **do**   ▷ c_2 is a sufficiently large constant
4:         initialize directed graphs $F_1 = (V, \emptyset)$ and $F_2 = (V, \emptyset)$
5:         each vertex s chooses a neighbor t (if any) uniformly at random,
                    then add (s, t) to $E(F_1)$
6:         each vertex t with positive in-degree in F_1 chooses a vertex $s \in \{s' : (s', t) \in E(F_1)\}$
                    with highest ID, then add (s, t) to $E(F_2)$
7:         each node v with positive degree in F_2 chooses a bit $b(v)$ as follows:
                    if v has an outgoing edge but no incoming edge, $b(v) = 0$
                    if v has an ingoing edge but no outcoming edge, $b(v) = 1$
                    otherwise, chooses $b(v) \in \{0, 1\}$ uniformly at random
8:         add every edge $(s, t) \in E(F_2)$ such that $b(s) = 0$ and $b(t) = 1$ to M,
                    and remove matched vertices from G

Lemma 6 ([6]) DISTRIBUTED- MM- PHASE1(G, d) *computes a partial matching M of an input graph G within $O(\log d)$ communication rounds, such that the remaining graph contains no connected component of size larger than $O(d^4 \log n)$ with probability at least $1 - 1/\text{poly}(n)$.*

The proof of this lemma also makes use of Beck's analysis, but contains a more complicated argument which shows that with constant probability, each vertex loses some constant fraction of its neighbors in every round. Thus, applying such matching

subroutine for $O(\log d)$ rounds suffices to remove or isolate each vertex with probability $1 - 1/\text{poly}(d)$. Converting this lemma into a two-phase LCA in a similar fashion, we obtain an LCA with a better dependence on d:

Theorem 6 *There exists a randomized local computation algorithm that computes a maximal matching of G with seed of length $2^{O(\log^2 d)} \log^2 n$ and time complexity $2^{O(\log^2 d)} \log^3 n$.*

4 Approximate Maximum Matching

In this section, we aim to construct an LCA that provides a $(1 - \varepsilon)$-approximation to maximum matching. To do so, we first address the simpler problem of computing a $(1-\varepsilon)$-approximation to MIS. We say that a set of vertices $\tilde{I}$ is a $(1-\varepsilon)$-approximation to MIS if it is a subset of some maximal independent set I and satisfies $|\tilde{I}| \geq (1-\varepsilon)|I|$. We present this LCA because it allows us to explain our LCA for the approximate maximum matching problem in a much clearer manner, as they share both their main principles and the two-phase structure. Moreover, this LCA is useful as a subroutine for other problems. For example, an approximation to MIS of the line graph $L(G)$ readily yields a $(1/2 - \varepsilon)$-approximation to maximum matching of G.

4.1 LCA for Computing an Approximate Maximal Independent Set

We now describe an LCA that provides a $(1 - \varepsilon)$-approximation to MIS. More specifically, with high probability, the set of vertices $\tilde{I}$ that our LCA answers YES must be a $(1 - \varepsilon)$-approximation to MIS as defined above. Our algorithm is based on a local simulation of the classical greedy algorithm for computing an MIS. This greedy algorithm iterates over the vertex set according to some arbitrary order and adds a vertex to the constructed independent set if and only if none of its neighbors has been added previously.

Algorithm 8 summarizes the local simulation of the greedy algorithm as suggested by Nguyen and Onak and further analyzed by Yoshida et al. [31,41]. In addition to the adjacency list oracle $\mathscr{O}^G$ and the queried vertex v, LS- MIS also receives as an input a permutation π on the vertices. The set of vertices v such that LS- MIS($\mathscr{O}^G, \pi, v$) returns YES is the lexicographically first MIS according to π, which is also the MIS outputted by the greedy algorithm when the vertices are iterated in this order.

Algorithm 8 Local simulation of the greedy MIS algorithm

1: **procedure** LS- MIS($\mathscr{O}^G, \pi, v$)
2:     query $\mathscr{O}^G$ for all neighbors of v
3:     let $v_1, \ldots, v_k$ be the neighbors of v sorted according to π
4:     **for** $i = 1, \ldots, k$ **do**
5:         **if** v_i precedes v in π **then**
6:             compute LS- MIS($\mathscr{O}^G, \pi, v_i$)
7:             **if** LS- MIS($\mathscr{O}^G, \pi, v_i$) = YES **then return** NO
8:     **return** YES

We now give an overview of the construction of our LCA. From the analysis of Yoshida et al. we obtain that in expectation over the queries and the random permutations, the query complexity of Algorithm 8 is polynomial in d, the degree bound [41]. More formally, let $R_\pi^G(v)$ denote the number of (recursive) calls to LS- MIS during the evaluation of LS- MIS$(\mathscr{O}^G, \pi, v)$. This result from Yoshida et al. can be succinctly stated as follows.

Theorem 7 ([41]) *For any graph $G = (V, E)$ with n vertices and m edges,*

$$\mathrm{E}_{\pi \in S_n, v \in V}[R_\pi^G(v)] \leq 1 + \frac{m}{n} .$$

However, the query complexity of an LCA is determined by the maximum number of queries invoked in order to answer any single query, so this expected bound does not readily imply an efficient LCA. We may alter Algorithm 8 so that it computes an independent set that is relatively large but not necessarily maximal as follows. If a query on a vertex v recursively invokes too many other queries, then the simulation is terminated and the algorithm immediately returns NO, i.e., that v does not belong to our independent set. We show that Theorem 7 implies that for at least a constant fraction of the orderings, this modification yields a good approximation ratio. Nonetheless, an LCA must succeed with high probability rather than only a constant probability. We resolve this issue by repeatedly sampling permutations until we obtain a sufficiently good one.

While generating a truly random permutation π requires $\Omega(n)$ random bits, we show that we may generate sufficiently good permutations by projecting random orderings from a distribution of small entropy onto S_n. Using the construction from Alon et al. as stated in Theorem 2, it follows that a small seed whose size is polylogarithmic in n suffices for our algorithm. From these outlined ideas, we are now ready to prove the following result.

Theorem 8 *There exists a randomized $(1-\varepsilon)$-approximation local computation algorithm for maximal independent set with random seed of length $O((d^2/\varepsilon^2) \log^2 n \log \log n + (d^3/\varepsilon) \log^3 n)$ and query complexity $O((d^4/\varepsilon^2) \log^2 n \log \log n)$.*

Proof Our algorithm builds on Algorithm 8 and consists of two phases as follows. The input of the algorithm is a random seed (tape) s and a queried vertex $v \in V$. In the first phase, using s and graph queries, we find a good permutation $\pi = \pi_G(s)$ over the vertex set. Note that this good permutation π is dependent only on s and G, and therefore must be identical for any query. Then in the second phase, we decide whether v belongs to our independent set using the altered local simulation that limits the number of invoked queries.

Formally, we say that $\pi \in S_n$ is *good* if $\Pr_{v \in V}\left[R_\pi^G(v) > \ell\right] \leq \gamma$ where $\gamma = \varepsilon/d$, $\ell = 6t/\gamma$ and $t = m/n+1$. Let δ denote the error probability of the algorithm. (For our purposes, we will later substitute $\delta = 1/\mathrm{poly}(n)$ to obtain an LCA that succeeds with high probability.) We claim that, given access to orderings which are drawn from an ℓ-wise independent family of orderings, D_ℓ, Algorithm 9 computes a good permutation π with the desired success probability. This algorithm checks for each constructed

permutation π_i whether it is good by approximating the fraction p_{π_i} of vertices whose induced query trees have size exceeding ℓ. The set S of sampled vertices is sufficiently large that we obtain a good approximation $\tilde{p}_{\pi_i}$ of p_{π_i}, and note that checking whether $R^G_{\pi_i}(v) > \ell$ can be accomplished by simply simulating Algorithm 8 until ℓ recursive calls are invoked.

Algorithm 9 Phase 1 of the LCA for the approximate MIS algorithm (finding a good ordering)

1: **procedure** LC- AMIS- PHASE1($\mathscr{O}^G, s$)
2:     let S be a multi-set of $\Theta(\log(1/\delta)\log\log(1/\delta)/\gamma^2)$ vertices chosen uniformly at random.
3:     **for** $i = 1, \ldots, h = \Theta(\log(1/\delta))$ **do**
4:         let π_i be the projection of an ordering independently drawn from D_ℓ
5:         let $\tilde{p}_{\pi_i}$ denote the fraction of vertices $v \in S$ for which $R^G_{\pi_i}(v) > \ell$
6:         **if** $\tilde{p}_{\pi_i} < 3\gamma/4$ **then**
7:             **return** $\pi = \pi_i$
8:     **report** ERROR

Now we show that with probability at least $1 - \delta$, Algorithm 9 finds a good π. Let $H_{\pi,v}$ be the indicator variable such that $H_{\pi,v} = 1$ when $R^G_\pi(v) > \ell$, and $H_{\pi,v} = 0$ otherwise. By Markov's inequality and Theorem 7 it holds that

$$\Pr_{\pi \in S_n, v \in V}[R^G_\pi(v) > \ell] \leq \gamma/6 \ .$$

That is,

$$\mathrm{E}_{v \in V}[\mathrm{E}_{\pi \in S_n}[H_{\pi,v}]] \leq \gamma/6 \ . \tag{1}$$

Next we aim to justify that we may obtain our random permutations by projecting orderings from $D_{\ell d}$ instead of drawing directly from S_n. More specifically, we claim that for a fixed v, $H_{\pi,v}$ is identically distributed regardless of whether π is drawn uniformly from S_n or obtained by projecting an ordering r drawn from $D_{\ell d}$. To establish this claim, consider the evaluation of $H_{\pi,v}$ by a decision tree $\mathscr{T}_{G,v}$, where each inner vertex of $\mathscr{T}_{G,v}$ represents a query to the rank $r(u)$ identified with the queried vertex $u \in V$. In order to evaluate $H_{\pi,v}$ we proceed down the tree $\mathscr{T}_{G,v}$ according to π until we reach a leaf. After at most ℓd queries to π, the value of the indicator $H_{\pi,v}$ is determined. Therefore, the depth of $\mathscr{T}_{G,v}$ is at most ℓd, and each leaf is associated with either 0 or 1. In other words, each leaf is identified with a sequence of at most ℓd vertices and their corresponding ranks.

Let s_1 and s_2 be a pair of leaves such that: (1) s_1 and s_2 are identified with the same sequence of vertices, $v_1, \ldots, v_k$; and (2) the ranks of $v_1, \ldots, v_k$ in s_1 and s_2 induce the same permutation on $v_1, \ldots, v_k$. Clearly the values corresponding to leaves s_1 and s_2 are identical. Therefore, our claim follows from the definition of k-wise independent ordering. Thus, from Eq. (1), we obtain

$$\mathrm{E}_{v \in V}[\mathrm{E}_{r \in D_\ell}[H_{\pi,v}]] \leq \gamma/6 \ . \tag{2}$$

Next, we say that $\pi \in S_n$ is *very good* if $\Pr_{v \in V}\left[R_\pi^G(v) > \ell\right] \leq \gamma/2$. By Markov's inequality and Equation (2), an ordering r drawn from D_ℓ induces a very good permutation with probability at least $2/3$. Therefore an ordering r drawn according to a distribution that is $(1/n^2)$-almost (ℓd)-wise independent random ordering over $[n]$ induces a very good permutation with probability at least $2/3 - 1/n^2$. So by generating $\Theta(\log(1/\delta))$ permutations π_i's from such random orderings, with probability greater than $1 - \delta/2$, at least one of them must be very good.

Let E_1 denote the event that none of the permutations in $\{\pi_i\}_{i=1}^h$ is very good, so E_1 occurs with probability smaller than $\delta/2$. Let E_2 denote the event that there exists $j \in [h]$ such that $|p_{\pi_j} - \tilde{p}_{\pi_j}| > \gamma/4$. By Chernoff's bound and the union bound, E_2 occurs with probability at most $\delta/2$. Henceforth, we assume that neither E_1 nor E_2 occur. Thus for every $j \in [h]$ such that $\tilde{p}_{\pi_j} \leq 3\gamma/4$, it holds that π_j is good; i.e., $p_{\pi_j} \leq \gamma$. Therefore, the algorithm does not report an error and the permutation returned by the algorithm is good. By the union bound, this happens with probability at least $1 - \delta$, as desired.

Theorem 2 implies that a random seed of length $O((d^3/\varepsilon)\log^2 n)$ suffices in order to obtain a $(1/n^2)$-almost (ℓd)-wise independent random ordering over $[n]$. Since Algorithm 9 draws $O(\log(1/\delta))$ such random orderings, overall a seed of length $O((d^3/\varepsilon)\log(1/\delta)\log^2 n)$ suffices. Additionally, we need $O((d^2/\varepsilon^2)\log(1/\delta)\log\log(1/\delta)\log n)$ more random bits to determine the set S of sampled vertices for evaluating $\tilde{p}_\pi$. For Algorithm 9 to succeed with high probability, it suffices to substitute $\delta = 1/\mathrm{poly}(n)$, which yields the seed length claimed in the theorem statement.

Finally, we now turn to describe the second phase of the algorithm. Let I_π denote the maximal independent set greedily created by $\pi \in S_n$, as defined by Algorithm 8. Consider the following LCA $\mathscr{L}_\pi$ for a good permutation π. On query v, $\mathscr{L}_\pi$ simulates the execution of LS- MIS$(\mathscr{O}^G, \pi, v)$ until it performs up to ℓ recursive calls to LS- MIS. If LS- MIS$(\mathscr{O}^G, \pi, v)$ terminates within ℓ steps, $\mathscr{L}_\pi$ returns the answer given by LS- MIS$(\mathscr{O}^G, \pi, v)$. Otherwise it simply returns NO.

Let $I'_\pi = I_\pi \cap V'_\pi$ where $V'_\pi = \{v : R_\pi^G(v) \leq \ell\}$, and notice that $\mathscr{L}_\pi$ answers YES precisely on I'. Since π is good, we have that $V'_\pi \geq (1-\gamma)n$ and so $|I'_\pi| \geq |I_\pi| - \gamma n$. Since for any maximal independent set I it holds that $|I| \geq n/d$, we obtain $|I'_\pi| \geq (1-\varepsilon)|I_\pi|$. In other words, $\mathscr{L}_\pi$ computes a $(1-\varepsilon)$-approximate MIS, as required. □

4.2 LCA for Computing an Approximate Maximum Matching

Now we turn to describe an LCA that provides a $(1 - \varepsilon)$-approximation to maximum matching with high probability. Our LCA locally simulates the global algorithm based on the following theorem by Hopcroft and Karp:

Lemma 7 [18] *Let M and M* be a matching and a maximum matching in G. If the shortest augmenting path with respect to M has length $2i - 1$, then $|M| \geq (1 - 1/i)|M^*|$.*

This global algorithm is used in the many contexts, including distributed computing and approximation (e.g., [25,31,41]). This algorithm can be summarized as follows. Let M_0 be an empty matching, and let $k = \lceil 1/\varepsilon \rceil$. We repeat the following process for each $i = 1, \ldots, k$. Let P_i denote the set of augmenting paths of length $2i - 1$ with respect to M_{i-1}. We compute a maximal set A_i of vertex-disjoint augmenting paths, then augment these paths to M_{i-1} to obtain M_i; that is, we set $M_i = M_{i-1} \oplus A_i$ where $\oplus$ denotes the symmetric difference between two sets. As a result, M_k will be a $(1 - \varepsilon)$-approximate maximum matching.

The local simulation of this algorithm uses the formula $M_i = M_{i-1} \oplus A_i$ to determine whether the queried edge e is in M_k in a recursive fashion. To determine A_i, consider the following observation. Let $H_i = (P_i, E_i)$ be the graph whose vertex set corresponds to the augmenting paths, and $(u, v) \in E_i$ if and only if the augmenting paths u and v share some vertex. A_i is a maximal set of vertex-disjoint augmenting paths; that is, A_i is an MIS of H_i. In order to compute locally whether a path is in A_i or not, Yoshida et al. apply Algorithm 8 on H_i. They show that in expectation over the queries and the random permutations of paths in $P_1, \ldots, P_k$, the query complexity is polynomial in d. More formally, let $\boldsymbol{\pi} = (\pi^1, \ldots, \pi^k)$ where π^i is a permutation of vertices (augmenting paths) in P_i, and let $Q_{\boldsymbol{\pi}, k}^G(e)$ denote the query complexity of this local simulation when queried on $e \in E$. Yoshida et al. prove the following theorem.

Theorem 9 ([41]) *For any graph $G = (V, E)$ with n vertices and maximum degree d,*

$$\mathrm{E}[Q_{\boldsymbol{\pi}, k}^G(e)] \le d^{6k^2} k^{O(k)},$$

where $\boldsymbol{\pi} = (\pi^1, \ldots, \pi^k)$ and the expectation is taken over the uniform distribution over $S_{|P_1|} \times \ldots \times S_{|P_k|} \times |E|$.

Based on this theorem, we convert the aforementioned local simulation into an LCA as similarly done in the previous section for the approximate MIS problem. We now prove the following theorem.

Theorem 10 *There is a randomized $(1 - \varepsilon)$-approximation local computation algorithm for maximum matching with random seed of length $O(d^{6k^2+1} k^{O(k)} \log^3 n \log \log n)$ and query complexity $O(d^{6k^2+2} k^{O(k)} \log^2 n \log \log n)$ where $k = \Theta(1/\varepsilon)$.*

Proof We build on the local algorithm of Yoshida et al. ([41]) where we take $k = \lceil 2/\varepsilon \rceil$ so that M_k is a $(1 - \varepsilon/2)$-approximate maximum matching. The proof for this theorem follows the structure of the proof of Theorem 8. Our LCA contains two phases: the first phase finds a sequence of good permutations, then the second phase locally simulates the approximate maximum matching algorithm using those permutations. We say that a sequence of permutations $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_k) \in S_{|V_1|} \times \ldots \times S_{|V_k|}$ is *good* if $\Pr_{e \in E}\left[Q_{\boldsymbol{\pi}}^G(e) > \ell\right] \le \gamma$ where $\gamma = \varepsilon/(2d)$, $\ell = 6t/\gamma$ and $t = d^{6k^2} k^{O(k)}$.

In the first phase, rather than individually generating a permutation π^k of $\boldsymbol{\pi}$ on each P_i, we create a single permutation π over $[m_{n,k}]$ where $m_{n,k} = \sum_{i=1}^{k} \binom{n}{k} \ge \sum_{i=1}^{k} |P_i|$. In terms of constructing random orderings, this simply implies extending

the domain of our orderings to cover all augmenting paths from all k iterations. Then for each $i \in [k]$, the permutation π^k of paths P_i can be obtained by restricting $\boldsymbol{\pi}$ to P_i (i.e., considering the relative order among paths in P_i). Since the algorithm queries $\boldsymbol{\pi}$ on at most ℓ locations, we conclude that Theorem 9 holds for any $\boldsymbol{\pi}$ that is ℓ-wise independent. Similarly to the proof of Theorem 10, with probability at least $1 - 1/\mathrm{poly}(n)$, we shall find a good $\boldsymbol{\pi}$ by testing $\Theta(\log n)$ random orderings obtained via the construction of Alon et al. in Theorem 2. Therefore, to this end we need at most $O(\ell \log^3 n + (d^2/\varepsilon^2) \log^2 n \log \log n)$ random bits, as required.

Now that we obtain a good sequence of permutations $\boldsymbol{\pi}$ from the first phase, we again perform the altered version of local algorithm that returns NO as soon as the simulation invokes too many queries. Let $M_{\boldsymbol{\pi}}$ denote the matching obtained by the algorithm of Yoshida et al. when executed with the permutations from $\boldsymbol{\pi}$. Let $M'_{\boldsymbol{\pi}} = M_{\boldsymbol{\pi}} \cap E'_{\boldsymbol{\pi}}$ where $E'_{\boldsymbol{\pi}} = \{e : Q^G_{\boldsymbol{\pi}}(e) \leq \ell\}$. Similarly to the proof of Theorem 8, given $\boldsymbol{\pi}$, we obtain an LCA that answers according to $M'_{\boldsymbol{\pi}}$ with query complexity ℓ. If $\boldsymbol{\pi}$ is good, we have that $E'_{\boldsymbol{\pi}} \geq (1-\gamma)|E|$ and so $|M'_{\boldsymbol{\pi}}| \geq |M_{\boldsymbol{\pi}}| - \gamma|E| \geq (1-\varepsilon/2)|M^*| - \gamma|E|$ where M^* denotes a maximum matching. Since for any maximal matching M it holds that $|M| \geq |E|/(2d)$ we obtain $|M'_{\boldsymbol{\pi}}| \geq (1-\varepsilon)|M^*|$, as required. $\square$

Compliance with Ethical Standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Ailon, N., Chazelle, B., Comandur, S., Liu, D.: Property-preserving data reconstruction. Algorithmica **51**(2), 160–182 (2008)
2. Alon, N.: A parallel algorithmic version of the local lemma. Random Struct. Algorithms **2**(4), 367–378 (1991)
3. Alon, N., Babai, L., Itai, A.: A fast and simple randomized parallel algorithm for the maximal independent set problem. J. Algorithms **7**(4), 567–583 (1986)
4. Alon, N., Rubinfeld, R., Vardi, S., Xie, N.: Space-efficient local computation algorithms. In: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1132–1139. SIAM (2012)
5. Andersen, R., Chung, F., Lang, K.: Local graph partitioning using pagerank vectors. In: Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on, pp. 475–486. IEEE (2006)
6. Barenboim, L., Elkin, M., Pettie, S., Schneider, J.: The locality of distributed symmetry breaking. In: Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on, pp. 321–330. IEEE (2012)
7. Beck, J.: An algorithmic approach to the Lovász local lemma. I. Random Struct. Algorithms **2**(4), 343–365 (1991)
8. Borgs, C., Brautbar, M., Chayes, J.T., Khanna, S., Lucier, B.: The power of local information in social networks. In: Goldberg, P.W. (ed.) Internet and Network Economics, Proceedings of the 8th Interna-

tional Workshop, WINE 2012, Liverpool, UK, December 10–12, 2012. Lecture Notes in Computer Science, vol. 7695, pp. 406–419. Springer (2012)

9. Brakerski, Z.: Local property restoring. Unpublished manuscript (2008)
10. Campagna, A., Guo, A., Rubinfeld, R.: Local reconstructors and tolerant testers for connectivity and diameter. In: Raghavendra, P., Raskhodnikova, S., Jansen, K., Rolim, J.D.P. (eds.) Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, Proceedings of the 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21–23, 2013. Lecture Notes in Computer Science, vol. 8096, pp. 411–424. Springer, Berlin (2013)
11. Chazelle, B., Seshadhri, C.: Online geometric reconstruction. J. ACM (JACM) **58**(4), 14 (2011)
12. Chung, K.M., Pettie, S., Su, H.H.: Distributed algorithms for the lovász local lemma and graph coloring. In: Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing, pp. 134–143. ACM (2014)
13. Dutta, A., Levi, R., Ron, D., Rubinfeld, R.: A simple online competitive adaptation of lempel-ziv compression with efficient random access support. In: Data Compression Conference (DCC), 2013, pp. 113–122. IEEE (2013)
14. Even, G., Medina, M., Ron, D.: Deterministic stateless centralized local algorithms for bounded degree graphs. In: Schulz, A.S., Wagner, D. (eds.) Algorithms - ESA 2014, Proceedings of the 22th Annual European Symposium, Wroclaw, Poland, September 8–10, 2014. Lecture Notes in Computer Science, vol. 8737, pp. 394–405. Springer, Berlin (2014)
15. Even, G., Medina, M., Ron, D.: Distributed maximum matching in bounded degree graphs. arXiv preprint arXiv:1407.7882 (2014)
16. Gamarnik, D., Sudan, M.: Limits of local algorithms over sparse random graphs. In: Proceedings of the 5th Conference on Innovations in Theoretical Computer Science, pp. 369–376. ACM (2014)
17. Hassidim, A., Mansour, Y., Vardi, S.: Local computation mechanism design. In: Proceedings of the Fifteenth ACM Conference on Economics and Computation, pp. 601–616. ACM (2014)
18. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. **2**(4), 225–231 (1973)
19. Israeli, A., Itai, A.: A fast and simple randomized parallel algorithm for maximal matching. Inf. Process. Lett. **22**(2), 77–80 (1986)
20. Jha, M., Raskhodnikova, S.: Testing and reconstruction of lipschitz functions with applications to data privacy. SIAM J. Comput. **42**(2), 700–731 (2013)
21. Kale, S., Peres, Y., Seshadhri, C.: Noise tolerance of expanders and sublinear expander reconstruction. In: Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on, pp. 719–728. IEEE (2008)
22. Kelsen, P.: Fast parallel matching in expander graphs. In: Proceedings of the Fifth Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 293–299. ACM (1993)
23. Levi, R., Ron, D., Rubinfeld, R.: Local algorithms for sparse spanning graphs. In: Jansen, K., Rolim, J.D.P., Devanur, N.R., Moore, C. (eds.) Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4–6, 2014, Barcelona, Spain. LIPIcs, vol. 28, p. 826. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2014)
24. Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. **21**(1), 193–201 (1992)
25. Lotker, Z., Patt-Shamir, B., Rosén, A.: Distributed approximate matching. SIAM J. Comput. **39**(2), 445–460 (2009)
26. Luby, M.: A simple parallel algorithm for the maximal independent set problem. SIAM J. Comput. **15**(4), 1036–1053 (1986)
27. Mansour, Y., Rubinstein, A., Vardi, S., Xie, N.: Converting online algorithms to local computation algorithms. In: Czumaj, A., Mehlhorn, K., Pitts, A.M., Wattenhofer, R. (eds.) Automata, Languages, and Programming, Proceedings of the 39th International Colloquium, ICALP 2012, Warwick, UK, July 9–13, 2012. Lecture Notes in Computer Science, vol. 7391, pp. 653–664. Springer, Berlin (2012)
28. Mansour, Y., Vardi, S.: A local computation approximation scheme to maximum matching. In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, pp. 260–273. Springer, Berlin (2013)
29. Marko, S., Ron, D.: Approximating the distance to properties in bounded-degree and general sparse graphs. ACM Trans. Algorithms (TALG) **5**(2), 22 (2009)
30. Muthukrishnan, S., Strauss, M., Zheng, X.: Workload-optimal histograms on streams. In: Brodal, G.S., Leonardi, S. (eds.) Algorithms - ESA 2005, Proceedings of the 13th Annual European Symposium,

Palma de Mallorca, Spain, October 3–6, 2005. Lecture Notes in Computer Science, vol. 3669, pp. 734–745. Springer, Berlin (2005)

31. Nguyen, H.N., Onak, K.: Constant-time approximation algorithms via local improvements. In: Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on, pp. 327–336. IEEE (2008)

32. Onak, K., Ron, D., Rosen, M., Rubinfeld, R.: A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1123–1131. SIAM (2012)

33. Orecchia, L., Zhu, Z.A.: Flow-based algorithms for local graph clustering. In: Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1267–1286. SIAM (2014)

34. Parnas, M., Ron, D.: Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. Theor. Comput. Sci. **381**(1), 183–196 (2007)

35. Peleg, D.: Distributed computing: a locality-sensitive approach. Siam, Philadelphia. Technical report, ISBN 0-89871-464-8 (2000)

36. Reingold, O., Vardi, S.: New techniques and tighter bounds for local computation algorithms. arXiv preprint arXiv:1404.5398 (2014)

37. Rubinfeld, R., Tamir, G., Vardi, S., Xie, N.: Fast local computation algorithms. In: Innovations in Computer Science—ICS 2010, Tsinghua University, Beijing, China, January 7–9, 2011. Proceedings, pp. 223–238 (2011). http://conference.itcs.tsinghua.edu.cn/ICS2011/content/papers/36.html

38. Saks, M., Seshadhri, C.: Local monotonicity reconstruction. SIAM J. Comput. **39**(7), 2897–2926 (2010)

39. Spielman, D.A., Teng, S.H.: A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. SIAM J. Comput. **42**(1), 1–26 (2013)

40. Sudan, M., Trevisan, L., Vadhan, S.: Pseudorandom generators without the xor lemma. In: Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, pp. 537–546. ACM (1999)

41. Yoshida, Y., Yamamoto, M., Ito, H.: Improved constant-time approximation algorithms for maximum matchings and other optimization problems. SIAM J. Comput. **41**(4), 1074–1093 (2012)