Journal of Computer and System Sciences 82 (2016) 1180–1200

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

New techniques and tighter bounds for local computation algorithms

Omer Reingold [a], Shai Vardi [b,*,1]

[a] Samsung Research America, United States
[b] Weizmann Institute of Science, Rehovot, Israel

ARTICLE INFO

Article history:
Received 18 December 2015
Received in revised form 7 May 2016
Accepted 16 May 2016
Available online 3 June 2016

Keywords:
Local computation algorithms
Sublinear algorithms
Pseudorandomness

ABSTRACT

Given an input x and a search problem F, local computation algorithms (LCAs) implement access to specified locations of y in a legal output $y \in F(x)$, using polylogarithmic time and space. Parnas and Ron [27] and Mansour et al. [19] showed how to convert certain distributed and online algorithms to LCAs, respectively. In this work, we expand on those lines of work and develop new techniques for designing LCAs and bounding their space and time complexity.

1. Introduction

The need for *local computation algorithms* (LCAs) arises in situations when we require fast and space-efficient access to part of a solution to a computational problem, but we never need the entire solution at once. Consider, for instance, a network with millions of nodes, on which we would like to compute a maximal independent set (MIS). We are never required to compute the entire solution; instead, we are queried about specific nodes, and we have to reply whether or not they are part of the independent set. To accomplish this, we are allocated space polylogarithmic in the size of the graph, and are required to reply to each query in polylogarithmic time. Although at any single point in time, we can be queried as to whether a single node (or a small number of nodes) is in the solution, over a longer time period, we may be queried about all of the nodes. We therefore want all of our replies to be consistent with the same MIS.

Local computation algorithms were introduced by Rubinfeld et al. [31]. In that work, they gave LCAs for several problems such as hypergraph 2-coloring and MIS; their focus was on the time bounds of these algorithms. To give better space bounds, Alon et al. [1] showed how to use k-wise independent hash functions (instead of random functions), when an LCA makes at most k probes to the (hyper-)graph per query, and used it to obtain polylogarithmic time and space bounds for hypergraph 2-coloring and MIS. Mansour et al. [19] showed how to bound the number of probes of the algorithms of [1] by $O(\log n)$ (where n is the number of vertices), and showed that this query bound also applies to a variety of LCAs that are obtained via reductions from online algorithms (we expand more on this technique below). They were the first to design LCAs on graphs whose degree is not bounded, but sampled from some distribution. Their results give, for example, an LCA for maximal matching that requires $O(\log^3 n)$ space and runs in $O(\log^4 n)$ time (on these graphs). The analyses of [1] and [19] (and indeed of many other works, e.g., [11,18]) use an abstract structure called a *query tree*, introduced by Nguyen and

* Corresponding author.
E-mail addresses: omer.reingold@gmail.com (O. Reingold), shai.vardi@weizmann.ac.il (S. Vardi).

[1] Supported in part by the Google Europe Fellowship in Game Theory. This research was conducted while Shai was at Tel Aviv University.

http://dx.doi.org/10.1016/j.jcss.2016.05.007
0022-0000/

Onak [26], to bound the number of probes the LCA makes. Even et al. [7] showed how to obtain an acyclic orientation that defines an order on the vertices, so that an online algorithm can be simulated on this order (similarly to [19]), giving a deterministic LCA for MIS on constant bounded degree graphs that requires $O(\log^* n)$ probes. Several other papers, e.g., [1, 15,31] reduced LCAs to distributed algorithms; for example, Luby's MIS algorithm [17] is simulated for a constant number of rounds followed by a "brute-force cleanup" of the remaining components. Levi et al. [15] studied the complexity of LCAs as a function of the degree in bounded – (not necessarily constant) – degree graphs. They showed that MIS can be computed in polylogarithmic time for $\Delta = 2^{O(\sqrt[3]{\log \log n})}$, where Δ is the maximal degree. Ghaffari [9] improved this to $\Delta = 2^{O(\sqrt{\log \log n})}$.

In this work, we depart from the previous lines of proof, and develop new ideas for bounding the space and time bounds of LCAs. Our techniques are more general and can be applied to a wider family of graphs, and our proofs are simpler. This allows us to improve on the known results on LCAs in several important ways: we reduce the space and time bounds of many previous results; we expand and better characterize the family of LCAs for which these new bounds hold; and we expand the family of graphs on which these LCAs can be implemented.

1.1. Our contributions

We formalize a family of algorithms called *neighborhood-dependent online algorithms*. These algorithms compute some function f on the vertices (or edges) of a graph, and the value of $f(x)$ for every vertex x depends only on the value of f for the immediate neighbors of x that arrived before x. This family encompasses a large number of online algorithms: online algorithms for maximal matching, maximal independent set, and vertex/edge coloring, to name a few. It also includes many online load balancing algorithms, such as the algorithm of Azar et al. [2] and many of the variations thereof (e.g., [5,35,36]).

d-Light graphs Most of the results on LCAs thus-far have been on a very narrow family of graphs (for example, the LCAs of [1,7,15,31] only apply to graphs of bounded degree; [19,12] also study a narrow family of randomly-generated bipartite graphs). We take an important step forward in characterizing the graphs which admit LCAs: we introduce a family of graphs, which we call *d-light* graphs. The high-level idea behind d-light graphs is the following: given that some subgraph has already been exposed, the degree[2] of the next exposed vertex should be bounded by a distribution that has expected value d and whose tail is light. A formal definition is given in Section 4. We will see that this definition encompasses a broad range of graphs, for example:

- Graphs with degree bounded by d, where $d = O(\log \log n)$,
- Erdős–Rényi random graphs $G(n, p)$, where $p = d/n$ for any $d = O(\log \log n)$,
- Bipartite graphs on n consumers and m producers, where each consumer is connected to $d = O(\log \log n)$ producers uniformly at random.

LCAs from online algorithms Our main result is the following: Assume we are given a d-light graph (for constant d), $G = (V, E)$, where $|V| = n$. We are also given a combinatorial search problem F for G, for which there exists a (sufficiently efficient) neighborhood-dependent online algorithm. We show how to construct the following:

1. An LCA for F that requires time $O(\log n \log \log n)$ and space $O(\log^2 n)$ per query, with probability $1 - 1/\text{poly}(n)$.
2. An LCA for F that requires time $O(\log^2 n)$ and space $O(\log n \log \log n)$ per query, with probability $1 - 1/\text{poly}(n)$.

We further show, that in both of these cases, the LCAs require $O(\log \log n)$ time and $O(\log n)$ space in expectation.

For both of our LCAs we use the following general construction. When we are queried about the value of some vertex v in the solution, we simulate the execution of the online algorithm on the graph, for some randomized order of arrival of the vertices. Because the algorithm is neighborhood-dependent, we only need to probe the neighbors of v that arrived before it. Call this set U. For each of the vertices in $u \in U$, we need to probe the neighbors of u that arrived before u, and so on. We show how to generate a pseudorandom ordering on the vertices using a seed of length $O(\log n)$ such that any such query will require at most $O(\log n)$ probes to the graph w.h.p. We furthermore show that using this seed, the expected number of probes the LCA will need to make to the graph is constant. The difference between the two constructions is the following: as we discover the vertices we need to probe in order to simulate the online algorithm, we relabel the vertices with unique labels to reflect the ordering. Trivially, we can use labels of length $O(\log n)$, and hence we get the first construction. However, as we only need to order $O(\log n)$ vertices per query, we only require labels of length $O(\log \log n)$; our second construction shows that we can create shorter labels, albeit at the expense of a somewhat worse running time. We show that if we allow our LCAs to use polylogarithmic space and require polylogarithmic time, we can handle d-light graphs, where d is as large as $\Theta(\log \log n)$. We also show that this bound is tight, at least for LCAs constructed by using the current general techniques of simulating online algorithms.

In Table 1, we compare our results to previous results for two illustrative problems—MIS and load balancing.

[2] The vertex's degree in the graph, not only in the subgraph.

Table 1
Summary of complexities of LCAs.

Problem	Citation	Graph type	Time	Space
MIS	[1]	constant degree$^{\diamond}$	$O(\log^3 n)$	$O(\log^2 n)$
	[7]	constant degree$^{\diamond}$	$O(\log^* n)^{\dagger}$	–‡
	[9,15]	maximal degree Δ	$2^{O(\log^2 \Delta)} \log^3 n$	$2^{O(\log^2 \Delta)} \log^2 n$
	here	d-light$^{\natural}$	$O(\log n \log\log n)$	$O(\log^2 n)$
			$O(\log^2 n)$	$O(\log n \log\log n)$
Load balancing*	[19]	random bins*	$O(\log^4 n)$	$O(\log^3 n)$
	here	random bins*	$O(\log n \log\log n)$	$O(\log^2 n)$
			$O(\log^2 n)$	$O(\log n \log\log n)$

$^{\diamond}$ I.e., the maximal degree is at most some constant, independent of n.
† Indicates probe complexity; the time complexity is not explicitly given in the paper.
‡ Indicates enduring memory (or seed length—see Section 3 for formal model definitions); the total memory is not given in the paper.
$^{\natural}$ d-Light in this table refers to constant d.
* By "Load Balancing" we are referring to the local implementation of the online "power of d choices" algorithm of Azar et al. [2]. By "random bins", we mean the graph created when there are n balls and n bins and each ball chooses a constant d number of bins uniformly at random.

LCAs from distributed algorithms Parnas and Ron [27] (implicitly) gave a simple reduction from constant-time distributed labeling[3] algorithms to LCAs. We show that the Parnas–Ron reduction can be used on d-light graphs to obtain deterministic LCAs whose number of probes and running time is $O(\log n)$ w.h.p.

1.2. Related work

Recently, there has been much work on expanding our understanding of LCAs and their connection with other fields. Even et al. [8] showed how to use LCAs to develop faster distributed algorithms for approximations to maximum matching in bounded degree graphs. Hassidim et al. [12] introduced LCAs to the field of mechanism design. Mansour et al. [20] showed that there are some problems (such as maximum weight forest approximation) for which we can construct LCAs whose time complexity is independent of the size of the graph.

LCAs share a common motivation with several other fields: due to the sheer size of today's ever-growing networks and databases, polynomial-time (or even linear-time) algorithms are often too slow. Many of the approaches to problems on these giant entities therefore concern themselves with what can be done in sublinear time. The approaches have been very diverse: The field of property testing asks whether some mathematical object, such as a graph, has a certain property, or is "far" from having that property (for an introduction and a survey, see [10] and [29] respectively). Streaming algorithms are required to use limited memory and quickly process data streams which are presented as a sequence of items (see [23] for a comprehensive introduction, and [37] for a survey). Sublinear approximation algorithms give approximate solutions to optimization problems, (e.g., [27,26,30]). A major difference between all of these algorithms and LCAs is that LCAs require that the output will be correct on any query, while optimization problems usually require a correct output only on most queries. More importantly, LCAs require a consistent output for each query, rather than only approximating a given global property.

Another related area of research is distributed algorithms that run in time independent (or almost independent) of the size of the network, often dependent on the maximal degree, which is assumed to be a constant (e.g., [16,25] for some classic results and [4,28] for newer results). Recently, much focus has been given to constant time distributed algorithms (for a recent survey, see [33]). These algorithms have a nice relationship to LCAs, in that any distributed algorithm that runs in a constant number of rounds (and requires a small amount of time and space per computation) immediately gives an LCA for the problem, using the Parnas–Ron reduction, and LCAs can often be used to design distributed algorithms (as in [8]). We expand upon this is relationship between distributed algorithms and LCAs in Section 9.

1.3. Paper organization

In Section 3 we define our model and the class of online algorithms to which our reductions apply. In Section 4 we define d-light graphs, and bound the size of a neighborhood of subsets of vertices in these graphs. Sections 5–8 are devoted to the reduction of LCAs to online algorithms: in Section 5 we describe the pseudorandom generators we require; in Section 6 we prove our main result, that the number of probes made to the graph is $O(\log n)$ w.h.p;. In Section 7 we analyze the expected running time[4]; in Section 8 we describe the reduction in detail and prove the time and space complexities of our LCAs. In Section 9 we show that applying the Parnas–Ron reduction to d-light graphs gives LCAs whose running time is $O(\log n)$ w.h.p.

[3] A labeling algorithm is simply an algorithm that assigns vertices (or edges) labels that meet some set of constraints: a maximal independent set algorithm assigns each vertex a boolean label indicating whether or not it is in the independent set; a vertex coloring algorithm assigns each vertex a color.
[4] And transient space requirement, see Definition 3.1.

2. Preliminaries

We denote the set of integers $\{1, 2, \ldots, n\}$ by $[n]$. We write $\mathbb{N}$ for the set of nonnegative integers. Let X be a random variable, distributed according to the Binomial distribution with parameters n and p; we denote this by $X \sim B(n, p)$.

Let $G = (V, E)$ be a simple undirected graph. The neighborhood of a vertex v, denoted $N(v)$, is the set of vertices that share an edge with v: $N(v) = \{u : (u, v) \in E\}$. The *degree* of a vertex v, is $|N(v)|$. The neighborhood of a set of vertices $S \subseteq V$, denoted $N(S)$, is the set of all vertices $\{u : v \in S, u \in N(v) \setminus S\}$. The *distance* between two vertices u and v, denoted $\text{dist}(u, v)$, is the minimal number of edges required to reach v from u (or vice-versa). For any vertex v, its k-neighborhood (for $k \geq 0$), denoted $N^k(v)$, is the set of all vertices at distance at most k from v. We only discuss *undirected* graphs, but both the definitions and algorithms easily extend to the directed case.

We assume that each vertex $v \in V$ has a unique label between 1 and some $n = \text{poly}(|V|)$. For simplicity, assume $n = |V|$ (or in other words $V = [n]$).

We use "with high probability" (w.h.p.) to mean with probability at least $1 - \frac{1}{\text{poly}(n)}$, where poly$n$ is some polynomial in n. Although we state our results with a specific (asymptotic) failure probability, such as $\frac{1}{n^2}$, they can all easily be extended to have a failure probability of $\frac{1}{n^c}$, for any constant c.

2.1. Stochastic dominance

We use the notion of *stochastic dominance*[5] for comparing distributions.

Definition 2.1 *(Stochastic dominance).* For any two distributions over the reals, X and Y, we say that X is *stochastically dominated by* Y if for every real number x it holds that $\Pr[X > x] \leq \Pr[Y > x]$. We denote this by $X \leq_{st} Y$.

We recall the following facts about stochastic dominance (see, e.g., [32]).

1. If $X \leq_{st} Y$ and $Y \leq_{st} Z$ then $X \leq_{st} Z$.
2. For any integer n, let $\{X_1, X_2, \ldots X_n\}$ and $\{Y_1, Y_2, \ldots Y_n\}$ be two sequences of independent random variables. If $\forall i, X_i \leq_{st} Y_i$, then $\sum_{j=1}^{n} X_j \leq_{st} \sum_{j=1}^{n} Y_j$.

We need the following observation and lemmas.

Observation 2.2. *For any $x \leq \alpha$, $B(x, \frac{d}{\alpha}) \leq_{st} B(\alpha, \frac{d}{\alpha})$.*

Proposition 2.3. *Let Y_1, Y_2 be independent discrete random variables. Let X_1, X_2 be (possibly) dependent discrete random variables, such that $X_1 \leq_{st} Y_1$, and conditioned on any realization of X_1, it holds that $X_2 \leq_{st} Y_2$, then*

$$X_1 + X_2 \leq_{st} Y_1 + Y_2.$$

Proof. For every realization x of X_1, define a different random variable for X_2. That is $X_2(x) = X_2 | X_1 = x$. Note $\forall x, X_2(x) \leq_{st} Y_2$. Further note that X_1 and Y_2 are independent. From the law of total probability,

$$\begin{aligned}
Pr[X_1 + X_2 > k] &= \sum_x Pr[x + X_2(x) > k] \cdot Pr[X_1 = x] \\
&= \sum_x Pr[X_2(x) > k - x] \cdot Pr[X_1 = x] \\
&\leq \sum_x Pr[Y_2 > k - x] \cdot Pr[X_1 = x] \\
&= \sum_x Pr[x + Y_2 > k] \cdot Pr[X_1 = x] \\
&= Pr[X_1 + Y_2 > k] \\
&\leq Pr[Y_1 + Y_2 > k]. \quad \square
\end{aligned}$$

This implies

[5] This is usually called *first-order stochastic dominance*. As this is the only measure of stochastic dominance we use, we omit the term "first-order" for brevity.

Lemma 2.4. *Let $\{Y_1, Y_2, \ldots, Y_N\}$ be a sequence of independent random variables. Let $\{X_1, X_2, \ldots, X_N\}$ be a series of (possibly) dependent random variables. If it holds that $X_1 \leq_{st} Y_1$, and for any $1 \leq i \leq N$, conditioned on any realization of $X_1, \ldots X_{i-1}$, it holds that $X_i \leq_{st} Y_i$, then*

$$\sum_{j=1}^{N} X_j \leq_{st} \sum_{j=1}^{N} Y_j.$$

Proof. We prove by induction on i that $\sum_{j=1}^{i} X_j \leq_{st} \sum_{j=1}^{i} Y_j$. For $i = 1$ we have that $X_1 \leq_{st} Y_1$. Assume that $\sum_{j=1}^{i} X_j \leq_{st} \sum_{j=1}^{i} Y_j$, we prove that $\sum_{j=1}^{i+1} X_j \leq_{st} \sum_{j=1}^{i+1} Y_j$. Let $Z_1 = \sum_{j=1}^{i} X_j$, $Z_2 = X_{i+1}$, $W_1 = \sum_{j=1}^{i} Y_j$, and $W_2 = Y_{i+1}$. Applying Proposition 2.3 with Z_1, Z_2, W_1, W_2 we get that $Z_1 + Z_2 \leq_{st} W_1 + W_2$ as required. □

For the second lemma we need (Lemma 2.9), we require the following well-known inequalities:

Fact 2.5. *For every $0 < x < 1$ and every $y > 0$,*

$$\left(1 - \frac{x}{y}\right)^{y} < e^{-x} < 1 - \frac{x}{2}.$$

Claim 2.6. *Let $2d < \alpha \leq n$. Let X and Y be random variables such that $X \sim B(1, \frac{d}{\alpha})$ and $Y \sim B(\lceil \frac{n^2}{\alpha} \rceil, \frac{2d}{n^2})$. Then $X \leq_{st} Y$.*

Proof. X is in fact a random variable with the Bernoulli distribution. As X can only take the values 0 or 1, to show stochastic dominance, it suffices to show that $\Pr[Y = 0] \leq \Pr[X = 0]$.

$$\Pr[Y = 0] = \left(1 - \frac{2d}{n^2}\right)^{\lceil \frac{n^2}{\alpha} \rceil} < e^{-2d/\alpha} < 1 - \frac{d}{\alpha} = \Pr[X = 0]. \quad \square$$

Claim 2.7. *Let X be a random variable such that $X \sim B(\alpha, \frac{d}{\alpha})$, and let $X_1, X_2, \ldots, X_{\alpha-1}$ be random variables duch that $\forall i, X_i \sim B(1, \frac{d}{\alpha})$. Then $X \leq_{st} \sum_{i=1}^{\alpha-1} X_i + 1$.*

The proof is immediate from the definition of the binomial distribution.

Claim 2.8. *Let Y be a random variable such that $Y \sim B(n^2, \frac{2d}{n^2})$ and let $Y_1, Y_2, \ldots, Y_{\alpha-1}$ be random variables such that $\forall i, Y_i \sim B(\lceil \frac{n^2}{\alpha} \rceil, \frac{2d}{n^2})$. Then $\sum_{i=1}^{\alpha-1} Y_i \leq_{st} Y$.*

Proof. It suffices to show that $(\alpha - 1)\lceil \frac{n^2}{\alpha} \rceil \leq n^2$.

$$(\alpha - 1)\left\lceil \frac{n^2}{\alpha} \right\rceil \leq (\alpha - 1)\left(\frac{n^2}{\alpha} + 1\right) \leq n^2 - n + \alpha - 1 < n^2,$$

because $\alpha \leq n$. □

Combining Claims 2.6, 2.7 and 2.8, we get

Lemma 2.9. *Let Z and X be random variables such that $Z \sim 2d + B(n^2, \frac{2d}{n^2})$ and $X \sim B(\alpha, \frac{d}{\alpha})$, where $d \leq \alpha \leq n$. Then $X \leq_{st} Z$.*

Proof. If $\alpha \leq 2d$, the lemma holds immediately. Assume $\alpha \geq 2d$. Then, using the notation of Claims 2.7 and 2.8

$$X \leq_{st} \sum_{i=1}^{\alpha-1} X_i + 1 \leq_{st} \sum_{i=1}^{\alpha-1} Y_i + 1 \leq_{st} B(n^2, \frac{2d}{n^2}) + 1 \leq_{st} Y + 1 \leq_{st} Z. \quad \square$$

3. The model

We assume the standard uniform-cost RAM model, in which the word size is $O(\log n)$ bits, where n is the input size, and it takes $O(1)$ to read and perform simple words operations.

We use the LCA model of Mansour et al. [20], which unifies the models of Rubinfeld et al. [31], (which focuses on time and space requirements) and Even et al. [7], (which focuses on the number of probes and the seed length). The following measures are used to quantify the efficiency of LCAs.

- *Number of probes*. An LCA can access a vertex in the input graph and ask for a neighbor's ID or a list of all of its neighbors' IDs. This is called a "probe".[6]
- *Running time*. The running time of an LCA, measured per query, is the number of word operations that the LCA is required to perform in order to output a reply. In the running time calculation, a probe to the graph is assumed to take $O(1)$ (as the reply to a probe can be a pointer to the list of neighbors). Note, however, that if we wish to perform an operation on a the entire list of neighbors and the list is of size ℓ, (such as committing the list to memory or finding the maximal ID), this will take $O(\ell)$ operations.
- *Enduring memory*. As a preprocessing step, before it is given its first query, the LCA can be allocated some memory to which it is allowed to write. Once the first query is given to the algorithm, it can only read from this memory and can never modify it. This can be viewed as the LCA being allowed to augment the input with some small number of bits. In all of the applications of this work, the enduring memory is used to store a random seed.
- *Transient memory*. This is simply the amount of memory (measured in the number of words) that an LCA requires in order to reply to a query. It does not include the enduring memory.
- *Failure probability*. The LCA is allowed to deviate from the number of probes, running time or transient memory that it requires per query. In this case, we say that the algorithm "fails". We require that, even if the LCA is queried on all vertices, the probability that it will fail on any of them is still very low. Note that the algorithm is never allowed to reply incorrectly—the failure is only a function of its complexity. Note further that the LCA is never allowed to use more enduring memory than it is allocated.

We define LCAs as follows.

Definition 3.1 *(Local computation algorithm).* A $(p(n), t(n), em(n), tm(n), \delta(n))$-*local computation algorithm* $\mathcal{A}$ for a computational problem is a (possibly randomized) algorithm that receives an input of size n, and a query x. Before the first query, $\mathcal{A}$ is allowed to write $em(n)$ bits to the enduring memory, and may only read from it thereafter. Algorithm $\mathcal{A}$ makes at most $p(n)$ probes to the input in order to reply to any query x, and does so in time $t(n)$ using $tm(n)$ transient memory (in addition to the enduring memory). The probability that $\mathcal{A}$ deviates from the probe, time or transient memory bounds is at most $\delta(n)$, which is called $\mathcal{A}$'s failure probability. Algorithm $\mathcal{A}$ must be *consistent*, that is, the algorithm's replies to all of the possible queries combine to a single feasible solution to the problem.

Algorithm characterization We study LCAs that can be derived from distributed and online algorithms. For simplicity, we only consider algorithms on vertices; an analogous definition holds for algorithms on edges, and our results hold for them as well. For online algorithms, we require that the algorithm will be neighborhood dependent in the sense that the value $f(v)$ is only a function of the values $\{f(u)\}$ for the neighbors u of v that the algorithm has already seen. Formally,

Definition 3.2 *(Neighborhood-dependent online graph algorithm).* Let $G = (V, E)$ be a graph, and let O be some arbitrary finite set. A *neighborhood-dependent online graph algorithm* $\mathcal{A}$ takes as input a vertex $v \in V$ and a sequence of pairs $\{(u_1, o_1), \ldots (u_\ell, o_\ell)\}$ where $\forall i, u_i \in V, o_i \in O$, and outputs a value $o \in O$. For every permutation Π of the vertices in V, define the output of $\mathcal{A}$ on G with respect to Π as follows. Denote by v_i^Π the vertex at location i under Π. Define $f(v_i^\Pi)$ recursively by invoking $\mathcal{A}$ on v_i^Π and the sequence of values $(v_j^\Pi, f(v_j^\Pi))$, such that $j < i$ and $(v_j^\Pi, v_i^\Pi) \in E$.

Let R be a search problem on graphs. We say that $\mathcal{A}$ is a neighborhood-dependent online graph algorithm for R if for every graph G and every permutation Π, the output of $\mathcal{A}$ on G with respect to Π satisfies the relation defining R.

When reducing LCAs to online (or distributed) algorithms, the running time and space requirements of the LCAs will obviously depend on these of the online (distributed) algorithm. In order to avoid a cumbersome statement of the results, we assume that the algorithms are "well behaved", defined as follows. (For simplicity, we define the crispness on graph labeling online algorithms. It is easy to extend the definition of crisp online algorithms to other online algorithms.)

Definition 3.3 *(Crisp).* We define crispness of distributed and online algorithms:

[6] Any question that the algorithm can ask a vertex can be considered a probe; in this paper, we only require the ones detailed above.

- A distributed algorithm $\mathcal{A}$ is *crisp* if $\mathcal{A}$ requires computational time and space linear in the number of probes it makes, and the output of $\mathcal{A}$ has length $O(1)$.
- An online algorithm that assigns labels to nodes (resp. edges) of a graph, $\mathcal{A}$, is *crisp* if, whenever a node (edge) arrives, $\mathcal{A}$ requires time and space linear in the size of the subgraph that has already arrived to compute the output for that node (edge), and the output has length $O(1)$ per node (edge).

Most of the algorithms that we wish to convert to LCAs using the techniques of this paper are indeed crisp (it is easy to verify, for instance, that the greedy algorithms for maximal independent set and maximal matching are crisp). However, not all the algorithms we wish to handle are necessarily crisp; in the case of vertex coloring, for example, the output of the greedy algorithm can be $\log\Delta$, where Δ is the maximal degree of the graph. It is straightforward to extend our results to non-crisp algorithms. We note that the analysis of distributed algorithms often assumes that the "expensive" part of the computation is the message passing, while the processors themselves have unrestricted computational power. Nevertheless, many distributed algorithms are actually very efficient, and hence fall under the definition of "crisp".

4. *d*-Light graphs

A d-light graph is a probabilistic graph whose degrees behave "nicely". Informally, we would like the degree of every vertex to be distributed binomially, with mean d. Slightly more formally, when the graph is discovered one vertex at a time, the degree of the next discovered vertex, w.r.t. the undiscovered vertices, is upper bounded by a binomial distribution with mean d. For the formal definition, we use a *vertex exposure procedure*: a procedure $\mathcal{P}$ that is allowed to probe vertices in the graph (usually according to some predetermined set of rules), to obtain their list of neighbors.

Definition 4.1 *(Adaptive vertex exposure).* An *adaptive vertex exposure process* $\mathcal{P}$ is a process that receives a limited oracle access to a graph $G=(V,E)$ in the following sense: $\mathcal{P}$ maintains a set of vertices $S\subseteq V$, (initially $S=\emptyset$), and updates S iteratively. In the first iteration, $\mathcal{P}$ exposes an arbitrary vertex $v\in V$, learns the IDs of all of the neighbors of v, and adds v to S. In each subsequent iteration, $\mathcal{P}$ can choose to expose any $u\in N(S)$: u is added to S, and $\mathcal{P}$ learns the IDs of u's neighbors. If a subset $S\subseteq V$ was exposed by such a process, we say that S was *adaptively exposed*.

Definition 4.2 *(d-Light distribution).* Let $d>0$, $G=(V,E)$ be a graph sampled from some distribution, and $\mathcal{P}$ be an adaptive vertex exposure process. If, for every $S\subset V$ that is adaptively exposed by $\mathcal{P}$, conditioned on any instantiation of $S\cup N(S)$ and set of edges $E_S=\{(u,v)\in E \mid u\in S\}$, we have that for every $v\in N(S)\setminus S$ (or any v in the case that S is empty), there is a value $d\leq\alpha_S\leq|V|$, such that $|N(v)\setminus S|$ is stochastically dominated by $B(\alpha_S, d/\alpha_S)$, we say that the degree distribution of G is *d-light* (or, for simplicity, that G is d-light).

The family of d-light graphs includes many well-studied graphs in the literature; for example,

- d-regular graphs, or more generally, graphs with degree bounded by d (taking $\alpha=d$),
- The random graphs $G(n,p)$, where $p=\frac{d}{n-1}$. Each one of the $\binom{n}{2}$ edges is selected independently with probability p; therefore the degree of each vertex is distributed according to the binomial distribution $B(n-1,\frac{d}{n-1})$. Note that in general, the degrees are not independent (for example, if one vertex has degree $n-1$ then all other vertices are connected to this vertex). However, once a subset $W\subset V$ has been exposed (as well as the edges in the cut $(W, G\setminus W)$), for any $v\notin W$, $|N(v)\setminus W|\sim B(n-|W|-1,\frac{d}{n-1})$, which is stochastically dominated by $B(n-1,\frac{d}{n-1})$.
- Bipartite graphs $G=(M\cup W,E)$ where $|M|=|W|=n$. Each vertex $m\in M$ man chooses d vertices $w\in W$ uniformly at random. This scenario is quite common—it can be found in the context of, e.g., stable matchings [13,14] (M and W represent the sets of men and women respectively), and load balancing [2,5] (M is the set of balls/jobs and W the set of bins/machines). It is easy to see that these are d-light bipartite graphs: the degrees of the vertices $m\in M$ obey $deg(m)\sim B(d,1)$, and the degrees of the vertices $w\in W$ obey $deg(w)\sim B(n,d/n)$, as each vertex $m\in M$ will be a neighbor of w with probability d/n.

For simplicity, we henceforth assume that d is a constant. We can allow d to be $O(\log\log n)$ while retaining polylogarithmic complexity measures. We do not explicitly compute the bounds for super-constant values of d; we expand on this point briefly in Remark 6.9, after the proof of our main result.

Bounding the neighborhood of exposed sets The results of this paper depend on a crucial property: any exposed subgraph of a d-light graph does not have "too many" neighbors. For motivation, consider the property: "Every connected subgraph with s vertices has at most $O(d\cdot s)$ neighbors". While this desirable holds trivially for graphs of degree bounded by d, it does not necessarily hold for d-light graphs. We could ask for a weaker property: "Every connected subgraph with s vertices has at most $O(d\cdot s)$ neighbors w.h.p.". Unfortunately, this does not hold for d-light graphs either. We therefore ask for two weaker properties, given by Properties 4.4 and 4.5. The first is that when we adaptively expose a large enough connected subgraph, the number of neighbors it has is unlikely to be much larger than the subgraph itself. The second is that if we adaptively

expose a small enough connected subgraph, the number of neighbors it has is unlikely to be more than $c \log n$, for some constant c.

Note that we count the number of edges $e = (u, v)$ for which at least one endpoint (u) is in S. The other (v) may or may not be in S.

The proofs of Properties 4.4 and 4.5 rely on the following proposition.

Proposition 4.3. *Let $\{X_{i,j}\}_{i\in[m], j\in[n^2]}$ be $n^2 \cdot m$ independent Bernoulli random variables such that $Pr[X_{i,j} = 1] = 2d/n^2$ for every $i \in [m]$ and $j \in [n^2]$. For every d-light graph $G = (V, E)$ with $|V| = n$ and every adaptively exposed subset of the vertices $S \subseteq V$ of size m,*

$$|\{(u, v) \in E \mid u \in S\}| \leq_{st} 2dm + \sum_{i=1}^{m}\sum_{j=1}^{n^2} X_{i,j}.$$

Proof. Denote $|S| = m$. Label the vertices of S: $1, 2, \ldots m$, according to the order of exposure. That is, vertex 1 is the first vertex that was exposed, and so on. Denote by S_i the set of vertices $\{1, 2, \ldots, i\}$, and by Y_i the random variable representing the number of neighbors of vertex i that are not in S_{i-1}. That is, $Y_i = |N(S_i) \setminus S_{i-1}|$. The quantity we would like to bound, $|\{(u, v) \in E \mid u \in S\}|$, is at most $\sum_{i=1}^{m} Y_i$.

From the definition of a d-light distribution, conditioned on every possible realization of S_{i-1}, $N(S_{i-1})$ and the edges adjacent to S_{i-1}, there exists some $d \leq \alpha_i \leq n$ such that $Y_i \leq_{st} B(\alpha_i, d/\alpha_i)$. By Lemma 2.9, under the same conditioning, $Y_i \leq_{st} Z \sim 2d + B(n^2, \frac{2d}{n^2})$. This implies that conditioned on any realization of $Y_1, \ldots, Y_{i-1}$ we have that $Y_i \leq_{st} Z$. By Lemma 2.4 we now have that $\sum_{i=1}^{m} Y_i$ is stochastically dominated by the sum of m independent copies of Z.

Note that for every i we have that $\sum_{j=1}^{n^2} X_{i,j} \sim B(n^2, 2d/n^2)$. By the definition of Z we now have that

$$\sum_{i=1}^{m} Y_i \leq_{st} 2dm + \sum_{i=1}^{m}\sum_{j=1}^{n^2} X_{i,j}. \quad \square$$

Property 4.4. *There exists some constant $c > 0$, such that for every d-light graph $G = (V, E)$ with $|V| = n$ and every adaptively exposed subset of the vertices $S \subseteq V$ of size at least $c \log n$, we have that $\Pr[|\{(u, v) \in E \mid u \in S\}| > 6d|S|] \leq 1/n^5$. In particular, $\Pr[|N(S)| > (6d - 1)|S|] \leq 1/n^5$.*

Proof. Letting $|S| = m$, from Proposition 4.3,

$$|\{(u, v) \in E \mid u \in S\}| \leq_{st} 2dm + \sum_{i=1}^{m}\sum_{j=1}^{n^2} X_{i,j}.$$

By the linearity of expectation, $\mathbb{E}[\sum_{i=1}^{m}\sum_{j=1}^{n^2} X_{i,j}] = 2dm$. By the Chernoff bound, there exists a constant c such that when $m \geq c \log n$ it holds that

$$\Pr[\sum_{i=1}^{m}\sum_{j=1}^{n^2} X_{i,j} > 4dm] \leq 1/n^5. \quad \square$$

Property 4.5. *There exist constants $c'' > c' > 0$, such that for every d-light graph $G = (V, E)$ with $|V| = n$ and every adaptively exposed subset of the vertices $S \subseteq V$ of size at most $c' \log n$, we have that $\Pr[|\{(u, v) \in E \mid u \in S\}| > c'' \log n] \leq 1/n^5$. In particular, $\Pr[|N(S)| > c'' \log n] \leq 1/n^5$.*

Proof. Similarly to the proof of Property 4.4, let $|S| = m$. From Proposition 4.3, $|\{(u, v) \in E \mid u \in S\}| \leq_{st} 2dm + \sum_{i=1}^{m}\sum_{j=1}^{n^2} X_{i,j}$. By the linearity of expectation, $\mathbb{E}[\sum_{i=1}^{m}\sum_{j=1}^{n^2} X_{i,j}] \leq 2dc' \log n$. By the Chernoff bound (of Theorem Appendix C.1), taking $c'' > 4edc'$, it holds that

$$\Pr[\sum_{i=1}^{m}\sum_{j=1}^{n^2} X_{i,j} > c'' \log n] \leq 1/n^5. \quad \square$$

5. Static and adaptive almost k-wise independent random orderings

5.1. Static k-wise independent hash functions

The probe and time bounds of the LCAs that we derive from reductions to online algorithms depend on the fact that we use some randomness to determine the order; as we want the replies of the LCA to be consistent with the same solution, we must use the same randomness every time. Therefore, we need to store this randomness. Storing a permutation of the vertices requires $\log n$ bits for each vertex, for a total of $n \log n$ bits. Unfortunately, this is the best we can do asymptotically (see, e.g., [3]). We therefore use *pseudorandom generators*. A pseudorandom generator is an algorithm that takes as input a short, perfectly random seed and then returns a (much longer) sequence of bits that "looks" random. We clearly sacrifice some randomness when we do this: the permutation we get is defined by the random seed, and if the seed is of length k, there are only 2^k possible distinct values of the seed. The notions of k-wise independent hash functions and *almost* k-wise independent hash functions were introduced by [6].

Definition 5.1 *(k-Wise independent hash functions).* For $n, L, k \in \mathbb{N}$ such that $k \leq n$, a family of functions $\mathcal{H} = \{h : [n] \to [L]\}$ is *k-wise independent* if for all distinct $x_1, x_2, \ldots, x_k \in [n]$, when H is sampled uniformly from $\mathcal{H}$ we have that the random variables $H(x_1), H(x_2), \ldots, H(x_k)$ are independent and uniformly distributed in $[L]$.

To quantify what we mean by "almost" k-wise independence, we use the notion of *statistical distance*.

Definition 5.2 *(Statistical distance).* For random variables X and Y taking values in $\mathcal{U}$, their *statistical distance* is

$$\Delta(X, Y) = max_{D \subseteq \mathcal{U}} |\Pr[X \in D] - \Pr[Y \in D]|.$$

For $\epsilon \geq 0$, we say that X and Y are ϵ-close if $\Delta(X, Y) \leq \epsilon$.

Definition 5.3 *(ϵ-Almost k-wise independent hash functions).* For $n, L, k \in \mathbb{N}$ such that $k \leq n$, let Y be a random variable sampled uniformly at random from $[L]^k$. For $\epsilon \geq 0$, a family of functions $\mathcal{H} = \{h : [n] \to [L]\}$ is *ϵ-almost k-wise independent* if for all distinct $x_1, x_2, \ldots, x_k \in [n]$, we have that $\langle H(x_1), H(x_2), \ldots, H(x_k)\rangle$ and Y are ϵ-close, when H is sampled uniformly from $\mathcal{H}$.

Naor and Naor [24] showed that relaxing from k-wise to almost k-wise independence can imply significant savings in the family size.

5.2. Simple almost k-wise random orderings

A family of *orderings* $\Pi = \{\Pi_r\}_{r \in R}$, indexed by R is *k-wise independent* if for every subset $S \subseteq [n]$ of size k, the projection of Π onto S (denoted by $\Pi(S)$) is uniformly distributed over all $k!$ possible permutations of S. Denote this uniform distribution by $U(S)$. We have that Π is *ϵ-almost k-wise independent* if for every k-element subset S we have that $\Delta(\Pi(S), U(S)) \leq \epsilon$ (where Δ is the statistical distance, see Definition 5.2). One can give adaptive versions of these definitions (in the spirit of Definition 5.7).

Past works on LCAs (e.g., [1,12,15]) have used k-wise and almost k-wise independent orderings to handle the derandomized ordering of vertices. The general idea is the following: let $m = O(n^4)$. Generate a random number in $[m]$ for each vertex. Assuming there are no collisions, this defines an ordering on the vertices. The probability that there is any collision is $O(1/n^2)$.[7] Alon et al. [1] showed that we can use a seed of length $O(\log^3 n)$ to obtain such an ordering.

The following "warm up" theorem shows that if $L = \text{poly}(n)$ and $\mathcal{H} = \{h : [n] \to [L]\}$ is a family of k-wise independent hash functions, then $\Pi = \{\Pi_h\}_{h \in \mathcal{H}}$ is a family of $1/\text{poly}(n)$-almost k-wise independent orderings, and that this requires a seed of length $O(k \log n)$. This implies that if we require (almost) $O(\log n)$-wise independence, we only need a seed of length $O(\log^2 n)$.

Theorem 5.4. *For every $n, k \in \mathbb{N}, \epsilon > 0$, such that $k \leq n$, there exists a construction of (adaptive) ϵ-almost k-wise independent random ordering of $[n]$ whose seed length is $O(k \log n)$.*

The proof of Theorem 5.4 follows immediately from Lemma 5.5 and Theorem 5.6.

[7] For k-wise independent hash functions this is immediate; for ϵ-almost k-wise independent hash functions, it is easy to show, for an appropriate choice of ϵ.

Lemma 5.5. *For every $n, L, k \in \mathbb{N}, \epsilon > 0$, such that $k \leq n$ and $L \geq k^2/\epsilon$, if $\mathcal{H} = \{h : [n] \to [L]\}$ is k-wise independent then $\Pi = \{\Pi_h\}_{h \in \mathcal{H}}$ is a family of ϵ-almost k-wise independent ordering.*

Proof. Fix the set S. There is probability smaller than ϵ on the choice of $h \in \mathcal{H}$ that there exist two distinct values i and j in S such that $h(i) = h(j)$. Conditioned on such collision not occurring the order is uniform by the definition of k-wise independent hashing. □

Theorem 5.6 *(cf. [34] Proposition 3.33, see also Theorem 5.9). For $m, k \in \mathbb{N}$ such that $k \leq m$ and m is a power of 2, there exists a family of functions $\mathcal{H} = \{h : [m] \to [m]\}$ that is k-wise independent, whose seed length is $k \log m$.*

Theorem 5.4 is an improvement over the seed length bound of [1], and could potentially be further improved, but one can observe that a lower bound on the seed length of almost $\log n$-wise independent ordering is $\Omega(\log n \log\log n)$: Consider any set S of $\log n$ elements. For (truly) $\log n$-wise independent ordering, the relative order of the elements in S is selected uniformly among the $(\log n)!$ possible orderings. Therefore the Shannon entropy of this relative order is $\Omega(\log n \log\log n)$, and since this relative order is a function of the seed, the seed length must be $\Omega(\log n \log\log n)$. A similar argument applies to the entropy of an ϵ-almost log n-wise independent ordering.

5.3. Adaptive k-wise independent hash functions

Another interpretation of ϵ-almost k-wise independent hash functions is as functions that are indistinguishable from uniform for a static distinguisher that is allowed to query the function in at most k places. In other words, we can imagine the following game: the (computationally unbounded) distinguisher $\mathcal{D}$ selects k inputs $x_1, x_2, \ldots, x_k \in [n]$, and gets in return values $F(x_1), F(x_2), \ldots, F(x_k)$, where $F : [n] \to [L]$ is either chosen from $\mathcal{H}$ (which we denote by $\mathcal{D}^{F \leftarrow \mathcal{H}}$), or is selected uniformly at random. $\mathcal{H}$ is ϵ-almost k-wise independent if no such $\mathcal{D}$ can differentiate the two cases with advantage larger than ϵ. In this paper we will need to consider *adaptive* distinguishers that can select each x_i based on the values $F(x_1), F(x_2), \ldots, F(x_{i-1})$.

Definition 5.7 *(Adaptive ϵ-almost k-wise independent hash functions).* For $n, L, k \in \mathbb{N}$ such that $k \leq n$ and for $\epsilon \geq 0$, a family of functions $\mathcal{H} = \{h : [n] \to [L]\}$ is *adaptive ϵ-almost k-wise independent* if for every (computationally unbounded) distinguisher $\mathcal{D}$ that makes at most k queries to an oracle F it holds that

$$|\Pr[\mathcal{D}^{F \leftarrow \mathcal{H}} = 1] - \Pr[\mathcal{D}^{F \leftarrow \mathcal{G}} = 1]| \leq \epsilon,$$

where $\mathcal{G}$ is the set of all functions $F : [n] \to [L]$.

Maurer and Pietrzak [21] showed a very efficient way to transform a family of (static) almost k-wise independent functions into a family of adaptive almost k-wise independent functions with similar parameters. For our purposes, it is enough to note that every family of (static) almost k-wise independent function is in itself also adaptive almost k-wise independent. While the parameters deteriorate under this reduction, they are still good enough for our purposes. We provide the proof in Appendix B for completeness.

Lemma 5.8. *[21] For $n, L, k \in \mathbb{N}$ such that $k \leq n$ and for $\epsilon \geq 0$, every family of functions $\mathcal{H} = \{h : [n] \to [L]\}$ that is ϵ-almost k-wise independent is also adaptive ϵL^k-almost k-wise independent.*

In particular, Lemma 5.8 implies that k-wise independent functions are also adaptive k-wise independent, and we get the following theorem:

Theorem 5.9 *(cf. [34] Proposition 3.33 and Lemma 5.8). For $n, k \in \mathbb{N}$ such that $k \leq n$ and n is a power of 2, there exists a family of functions $\mathcal{H} = \{h : [n] \to [n]\}$ that is adaptive k-wise independent, whose seed length is $k \log n$. The time required to evaluate each h is $O(k)$ word operations and the memory required per evaluation is $O(\log n)$ bits (in addition to the memory for the seed).*

5.4. Reducing the seed size to $O(\log n)$

Let us now consider what happens if we let L be much smaller, namely a constant. This will reduce the seed length to $O(k + \log n)$ (see below). However, we will lose the almost k-wise independence (for sub-constant error) of the ordering: Consider two variables $i < j$. Conditioned on $h(i) \neq h(j)$, the order of i and j under the permutation defined by h (denoted Π_h) is uniform. But with constant probability $h(i) = h(j)$. Nevertheless, we will show that a constant L suffices for our purposes.

We employ a recent result of Meka et al. [22] (which, using Lemma 5.8, also applies to *adaptive* almost k-wise independence). The following is specialized from their work to the parameters we mostly care about in this work.

Theorem 5.10 *([22] and Lemma 5.8). For every $n, L, k \in \mathbb{N}$ and $\epsilon > 0$ such that n and L are powers of 2, and such that $k \cdot L = O(\log n)$ and $1/\epsilon = poly(n)$, there is a family of adaptive ϵ-almost k-wise independent functions $\mathcal{H} = \{h : [n] \to [L]\}$ such that choosing a random function from $\mathcal{H}$ takes $O(\log n)$ random bits. The time required to evaluate each h is $O(\log k)$ word operations and the (enduring) memory is $O(\log n)$ bits.*

The property of almost k-wise independent hash functions H that we will need is that an algorithm querying H will not query "too many" preimages of any particular output of H. Specifically, if the algorithm queries H more than $cL \log n$ times, none of the values will appear more than twice their expected number. More formally:

Proposition 5.11. *There exists a constant c such that following holds. Let $n, L, k \in \mathbb{N}$ be such that $k \leq n$ and let $\epsilon \geq 0$. Let $\mathcal{H} = \{h : [n] \to [L]\}$ be a family of functions that is adaptive ϵ-almost k-wise independent. Let $\mathcal{A}$ be any procedure with oracle access to H sampled uniformly from $\mathcal{H}$ and let ℓ be any value in $[L]$. Define the random variable m to be the number of queries $\mathcal{A}$ makes and the random variables $x_1, x_2, \ldots, x_m$ to be those queries. Then conditioned on $cL \log n \leq m \leq k$, it holds that:*

$$\Pr[|\{x_i | H(x_i) = \ell\}| > 2m/L] \leq \frac{1}{2n^5} + \epsilon.$$

Proof. Consider any $\mathcal{A}$ as in the theorem and assume for the sake of contradiction that

$$\Pr[|\{x_i | H(x_i) = \ell\}| > 2m/L \mid cL \log n \leq m \leq k] > \frac{1}{2n^5} + \epsilon.$$

Consider $\mathcal{A}$ with access to a function F selected uniformly at random from $\mathcal{G}$, the set of all functions $f : [n] \to [L]$. Define m' to be the number of queries $\mathcal{A}$ makes in such a case and let $x'_1, \ldots, x'_{m'}$ be the set of queries. By the Chernoff bound, conditioned on any fixing of m', we have that $\Pr[|\{x'_i | F(x'_i) = \ell\}| > 2m'/L]$ is exponentially small in m'/L. Therefore, by a union bound, for a sufficiently large c we have that $\Pr[|\{x_i | F(x_i) = \ell\}| > 2m'/L \mid cL \log n \leq m' \leq k] \leq \frac{1}{2n^5}$ (note that $L < n$, otherwise the theorem is trivially true). We thus have that $\mathcal{A}$ distinguishes the distribution $\mathcal{H}$ from the distribution $\mathcal{G}$ with k queries, with advantage ϵ, in contradiction to $\mathcal{H}$ being ϵ-almost adaptive k-wise independent. □

6. Upper bounding the number of probes

In order to show that a family of hash functions with a constant range (L) is sufficient in order to bound our probe complexity, we first require some definitions.

Definition 6.1 *(Level, Levelhood).* Let $G = (V, E)$ be a graph. Let $h : V \to \mathbb{N}$ be a function that assigns each vertex an integer. For each $v \in V$, we call $h(v)$ the *level* of v. Denote the restriction of a set of vertices $S \subseteq V$ to only vertices of a certain level ℓ, $\{x \in S : h(x) = \ell\}$, by $S\|_\ell$.

Let $S \subseteq V$, and let $N_\ell(S)$ be the neighbors of S that are in level ℓ. That is $N_\ell(S) = \{u \in V : u \in N(S), h(u) = \ell\}$. The ℓ^{th} *levelhood of* S (denoted $\Psi_\ell(S)$), is defined recursively as follows. $\Psi_\ell(\emptyset) = \emptyset$. $\Psi_\ell(S) = \{S \cup \Psi_\ell(N_\ell(S))\}$. In other words, we initialize $\Psi_\ell(S) = S$, and add to $\Psi_\ell(S)$ neighbors of level ℓ, until we cannot add any more vertices.

We define the *rank* of a vertex to be the concatenation of its level and ID. Formally,

Definition 6.2 *(Ranking).* Let L be some positive integer. A function $r : [n] \to [L]$ is a *ranking* of $[n]$, where $r(i)$ is called the *level* of i. The *ordering* Π_r which corresponds to r is a permutation on $[n]$ obtained by defining $\Pi_r(i)$ for every $i \in [n]$ to be its position according to the monotone increasing order of the relabeling $i \mapsto (r(i), i)$. In other words, for every $i, j \in [n]$ we have that $\Pi_r(i) < \Pi_r(j)$ if and only if $r(i) < r(j)$ or $r(i) = r(j)$ and $i < j$. The pair $(r(i), i)$ is called the *rank* of i.

Informally, vertices of higher rank "arrive earlier".

Definition 6.3 *(Relevant vicinity).* The *relevant vicinity* of a vertex v, denoted $\Im_h(v)$ (relative to a hash function $h : V \to [L]$), is defined constructively as follows. Let Π_h be the permutation defined by h, as in Definition 6.2. Initialize $\Im_h(v) = \{v\}$. For each $u \in \Im_h(v)$, add to $\Im_h(v)$ all vertices $w \in N(u) : \Pi_h(w) < \Pi_h(u)$. Continue adding vertices until no more can be added. The relevant vicinity of a set of vertices U is the union of the relevant vicinities of the vertices in U.

The relevant vicinity of a vertex v is exactly the vertices that our LCA will simulate the online algorithms on when queried about v.[8] Because we cannot make any assumptions about the original labeling of the vertices, we upper bound

[8] In specific cases, better implementations exist that do not need to explore the entire relevant vicinity. For example, once an LCA for maximal independent set sees that a neighbor of the inquired vertex is in the independent set, it knows that the inquired vertex is not (and can halt the exploration). Nevertheless, in order not to make any assumptions on the online algorithm, we assume that the algorithm explores the entire relevant vicinity.

the size of the relevant vicinity by defining the *containing vicinity*, where we assume that the worst case always holds. That is, if two neighbors have the same level, the one that is queried first appears before the other in the permutation. The size of the containing vicinity is clearly an upper bound on the size of a relevant vicinity, for the same hash function.

Definition 6.4 *(Containing vicinity).* The *containing vicinity* of a vertex v (relative to a hash function $h : V \to [L]$), given that $h(v) = \ell$, is $\Psi_L(\Psi_{L-1}(\ldots\Psi_{\ell+1}(\Psi_\ell(v))\ldots))$. In other words, let S_ℓ be the ℓ^{th} levelhood of v, and for $i \in \{\ell+1,\ldots,L\}$, $S_i = \Psi_i(S_{i-1})$. The containing vicinity is then S_L. The containing vicinity of a set of vertices U is the union of the containing vicinities of the vertices in U.

The following lemma is our main result.

Lemma 6.5. *Let $G = (V, E)$ be a d-light graph, $|V| = n$, and let L be an integer such that $L > 24d$. Let c be such that Proposition 5.11 and Property 4.4 hold, and let $\kappa = 2^L c \log n$. Let $k = 6d\kappa$, and let h be an adaptive ϵ-almost k-wise independent hash function, $h : V \to [L]$. For any adaptively exposed set of vertices $U \subseteq V$, whose size is at most $c \log n$, the relevant vicinity of U has size at most κ with probability at least $1 - \frac{1}{n^4} + \frac{1}{n^5}$.*

Corollary 6.6. *Let the conditions of Lemma 6.5 hold. For any vertex $v \in G$, the relevant vicinity of v has size at most κ with probability at least $1 - \frac{1}{n^4} + \frac{1}{n^5}$.*

Given the following claim, the proof of Lemma 6.5 is straightforward, and is presented after the claim.

Claim 6.7. *Let the conditions of Lemma 6.5 hold; assume without loss of generality that for all $v \in U$, $h(v) = 1$, and let $S_0 = U$, $S_1 = \Psi_1(S_0), \ldots, S_{\ell+1} = \Psi_{\ell+1}(S_\ell), \ldots, S_L = \Psi_L(S_{L-1})$. Then for all $0 \le i \le L$,*

$$\Pr[|S_i| \le 2^i c \log n \wedge |S_{i+1}| \ge 2^{i+1} c \log n] \le \frac{2}{n^5}.$$

Proof. Fix i and denote the bad event for i as B_i. That is, $B_i \equiv |S_i| \le 2^i c \log n \wedge |S_{i+1}| \ge 2^{i+1} c \log n$. We make the following observation:

$$B_i \Rightarrow |S_{i+1}| > 2^{i+1} c \log n \wedge |S_{i+1}\|_{i+1}| > \frac{|S_{i+1}|}{2}, \tag{1}$$

because $S_{i+1}\|_{i+1} = S_{i+1} \setminus S_i$. In other words, this means that if B_i occurs then the majority of elements in S_{i+1} must have come from the $(i+1)^{th}$-levelhood of S_i.

For all $0 < i \le L$ let $T_i = S_i \cup N(S_i)$. Define two bad events:

$$B_i^1 \equiv |S_{i+1}| > 2^{i+1} c \log n \wedge |T_{i+1}| > 6d|S_{i+1}|$$

$$B_i^2 \equiv |S_{i+1}| > 2^{i+1} c \log n \wedge |T_{i+1}\|_{i+1}| > \frac{12d}{L}|S_i + 1|$$

From Equation (1), the definition of L, and the fact that $T_{i+1}\|_{i+1} = S_{i+1}\|_{i+1}$, we get $B_i \Rightarrow B_i^2$.

By Property 4.4,

$$\Pr[B_i^1] \le \frac{1}{n^5}, \tag{2}$$

because S_{i+1} is an adaptively exposed subset, $T_{i+1} = S_{i+1} \cup N(S_{i+1})$ and the size of S_{i+1} satisfies the conditions of the proposition. Given $|S_{i+1}| > 2^{i+1} c \log n$, it holds that $|T_{i+1}| > 2^{i+1} c \log n$ because $S_{i+1} \subseteq T_{i+1}$. Also note that T_{i+1} was defined based on the value of h on elements in T_{i+1} only. Therefore, the conditions of Proposition 5.11 hold for $|T_{i+1}|$.

$$\begin{aligned}\Pr[B_i^2 : \neg B_i^1] &\le \Pr[|T_{i+1}\|_{i+1}| > \frac{12d}{L}|S_{i+1}| : |T_{i+1}| < 6d|S_{i+1}|] \\ &\le \Pr[|T_{i+1}\|_{i+1}| > \frac{2|T_{i+1}|}{L}] \\ &\le \frac{1}{n^5},\end{aligned} \tag{3}$$

where the last inequality is due to Proposition 5.11.

Because

$$\Pr[B_i^2] = \Pr[B_i^2 : B_i^1]\Pr[B_i^1] + \Pr[B_i^2 : \neg B_i^1]\Pr[\neg B_i^1],$$

from Equations (2) and (3), the claim follows. □

Proof of Lemma 6.5. We need to show that

$$\Pr[|S_L| > 2^L c \log n] < \frac{1}{n^4} - \frac{1}{n^5}.$$

We show that for $0 \le i \le L$, $\Pr[|S_i| > 2^i c \log n] < \frac{2i}{n^5}$, by induction. For the base of the induction, $|S_0| = 1$, and the claim holds. For the inductive step, assume that $\Pr[|S_i| > 2^i c \log n] < \frac{2i}{n^5}$. Then

$$\begin{aligned}\Pr[|S_{i+1}| > 2^{i+1} c \log n] &= \Pr[|S_{i+1}| > 2^{i+1} c \log n : |S_i| > 2^i c \log n] \Pr[|S_i| > 2^i c \log n] \\ &\quad + \Pr[|S_{i+1}| > 2^{i+1} c \log n : |S_i| \le 2^i c \log n] \Pr[|S_i| \le 2^i c \log n].\end{aligned}$$

From the inductive step and Claim 6.7, using the union bound, the lemma follows. □

From Lemma 6.5 and Property 4.4, we immediately get

Corollary 6.8. *Let $G = (V, E)$ be a d-light graph, where $|V| = n$, and let L be an integer such that $L > 24d$. Let c be such that Proposition 5.11 and Property 4.4 hold, and let $\kappa = 2^L c \log n$. Let $k = 6d\kappa$, and let h be an adaptive k-wise ϵ-almost independent hash function. For any vertex $v \in G$, let S_L be the relevant vicinity of v. Then*

$$\Pr[|\{(u, w) \in E \mid u \in S_L\}| > k] < 1/n^4.$$

Corollary 6.8 essentially shows the following: Assume that $G = (V, E)$ is a d-light graph, and there is some function $F : V \to \mathcal{R}$ that is computable by a neighborhood-dependent online algorithm $\mathcal{A}$. Then, in order to compute $F(v)$ for any vertex $v \in V$, we only need to look at a logarithmic number of vertices and edges with high probability. Note that in order to calculate the relevant vicinity, we need to look at all the vertices in the relevant vicinity and all of their neighbors (to make sure that we have not overlooked any vertex). This is upper bound by the number of edges which have an endpoint in the relevant vicinity, as the relevant vicinity is connected. Furthermore, as we will see in Section 8, we would like to store the subgraph induced by the relevant vicinity, and for this, we need to store all of the edges.

Applying a union bound over all the vertices gives that the number of probes we need to make per query (i.e., each time the LCA is queried about a vertex) is $O(\log n)$ with probability at least $1 - 1/n^3$ even if we are queried about all the vertices.

Remark 6.9. It is easy to verify that if $d = \Theta(\log\log n)$ the number of probes per query is polylogarithmic, by letting $\kappa = \text{polylog} n$. (One also needs to make similar adjustments to Proposition 5.11 and Property 4.4.) As $\kappa = 2^{O(d)} \text{polylog} n$, if $d = \omega(\log\log n)$, we cannot bound the number of probes by $\text{polylog} n$ using Lemma 6.5. However, we show in Appendix A that this is a limitation of the algorithm, not of the proof.

7. Expected size of the relevant vicinity

In Section 8, we show constructions of the subgraph induced by the relevant vicinity whose running times and transient memory requirements are dependent on t_v and t_e, the size of the relevant vicinity and the number of edges adjacent to the relevant vicinity, respectively. All the dependencies can be upper bound by $O(t_e^2)$. In this section, we prove that the expected value of t_e^2 in a d-light graph is a constant (when d is a constant).

Recall that we use ϵ-almost k-wise independent hash functions. For this section, we require that $\epsilon < \frac{1}{d^{2k}}$.

Proposition 7.1. *For any d-light graph $G = (V, E)$ and any vertex $v \in V$, the expected number of simple paths of length t originating from v is at most d^t.*

We prove a slightly more general claim, from which Proposition 7.1 immediately follows (taking S in the proposition to be the empty set).

Claim 7.2. *For any d-light graph $G = (V, E)$, any adaptively exposed subset $S \subseteq V$, and any vertex $v \in N(S) \setminus S$ (or any vertex v if S is empty), the expected number of simple paths of length t originating from v and not intersecting with S is at most d^t.*

Proof. The proof is by induction on t. For the base of the induction, $t = 0$, and there is a single simple path (the empty path). For the inductive step, let $t > 0$ and assume that the claim holds for $t - 1$. We show that it holds for t. Given S, let $S' = S \cup \{v\}$. Let a be a random variable representing the number of neighbors of v that are not in S; that is, $a = |N(v) \setminus S|$. Because G is d-light, $\mathbb{E}(a) \le d$. Fixing a, label the neighbors of v that are not in S by $w_1, w_2, \ldots, w_a$. By the inductive hypothesis (as S' is also adaptively exposed), for all $i = 1, \ldots, a$, the expected number of simple paths of length

$t-1$ originating from w_i and not intersecting with S' is at most d^{t-1}. The expected number of simple paths of length t originating from v and not intersecting with S is therefore upper bounded by

$$\sum_j \Pr[a=j]\, j \cdot d^{t-1} = \mathbb{E}[a] d^{t-1} \le d^t. \quad \square$$

For any simple path p originating at some vertex v, we would like to determine whether all the vertices on p are in the relevant vicinity of v. As we cannot make any assumptions about the original labels of the vertices, we upper bound this by the probability that the levels of the vertices on the path are non-decreasing.

Definition 7.3 *(Legal path).* We say that a path $p = v \rightsquigarrow u$ is *legal* if it is simple and the labels of the vertices on p are in non decreasing order.

Definition 7.4 *(Prefix-legal path).* We say that a path $p = v \rightsquigarrow u$ of length t is *prefix-legal* if it is simple, and the prefix of p of length $t-1$ is legal.

Proposition 7.5. *Let $G = (V, E)$ be a d-light graph and let the conditions of Lemma 6.5 hold. That is, $k, \kappa = O(\log n)$ and $L > 24d$. For any $c > 0$ there exists a value $L = \mathrm{poly}(d)$ for which the following holds: Let h be an adaptive k-wise ϵ-almost independent hash function, $h : V \to [L]$. For any path p of length $2ed^c \le t' < k-1$ originating at some vertex v, the probability that p is legal is at most $d^{-ct'} + \epsilon$.*

Proof. For any simple path p of length t' from v, there are $L^{t'}$ possible values for the levels of the vertices of p. Let the values be $r_0, r_1, \ldots, r_{t'-1}$. We define $t'+1$ new variables $a_0 = r_0$, $a_1 = r_1 - r_0$, …, $a_{t'-1} = r_{t'-1} - r_{t'-2}$, $a_{t'} = L - r_{t'-1}$. Clearly, the a_i's uniquely define the r_i's and p is legal if and only if $a_0, \ldots, a_{t'}$ are all non-negative.

Note that the $\sum_{i=0}^{t'} a_i = L$; hence computing the number of possible legal values of $a_0, \ldots, a_{t'}$ is the same as computing the number of ways of placing L identical balls in $t'+1$ distinct bins,[9] where each bin represents a vertex and if there are k balls in bin y, then $r_y = r_{y-1} + k$. This is known to be $\binom{t'+L}{t'}$. Therefore, assuming the choices of the levels are all uniform,

$$\Pr[p \text{ is legal}] = \frac{1}{L^{t'}} \binom{t'+L}{t'} \le \left(\frac{e(t'+L)}{Lt'}\right)^{t'}.$$

From the definition of ϵ-almost adaptive k-wise independent hash functions, we immediately get

$$\Pr[p \text{ is legal}] \le \left(\frac{e(t'+L)}{Lt'}\right)^{t'} + \epsilon.$$

The result follows, by setting $L = 2ed^c$. $\square$

The following corollary is immediate, setting $t' = t-1$ in Proposition 7.5.

Corollary 7.6. *Let $G = (V, E)$ be a d-light graph and let the conditions of Lemma 6.5 hold. That is, $k, \kappa = O(\log n)$ and $L > 24d$. For any $c > 0$ there exists a value $L = \mathrm{poly}(d)$ for which the following holds: Let h be an adaptive k-wise almost ϵ-independent hash function, $h : V \to [L]$. For any path p of length $2ed^c < t < k$ originating at some vertex v, the probability that p is prefix-legal is at most $d^{c(1-t)} + \epsilon$.*

As a warm-up, we first show that the expected number of edges adjacent to a relevant vicinity is a constant.

Lemma 7.7. *Let $G = (V, E)$ be a d-light graph and let the conditions of Proposition 7.5 hold. That is, $k, \kappa = O(\log n)$ and $L = \mathrm{poly}(d)$. Let h be an adaptive k-wise ϵ-almost independent hash function, $h : V \to [L]$. Then the expected number of edges adjacent to a relevant vicinity in G is $O(1)$.*

Proof. Let v be any vertex, let S_L be the relevant vicinity of v, and let E_L be the set of edges with at least one endpoint in S_L. Let $c > 2$ be a constant. The total number of edges at a constant distance from v is a constant; we ignore these edges and only count edges at distance at least $2ed^c$ from v. Let $(u, w) \in E$ be any edge at distance at least $2ed^c$ from v and denote by $p^{\le k}_{(v,u,w)}$ the indicator random variable whose value is 1 if there exists a prefix-legal path of length at most k

[9] Alternatively, one can view this as placing t' separators among L balls.

from v to w whose last edge is (u, w), and 0 otherwise. Similarly, denote by $p^{>k}_{(v,u,w)}$ the random variable whose value is 1 if there exists a prefix-legal path of length greater than k from v to w whose last edge is (u, w), and 0 otherwise. For any $(u, w) \in E$,

$$\Pr[(u, w) \in E_L] \leq \Pr[p^{\leq k}_{(v,u,w)}] + \Pr[p^{>k}_{(v,u,w)}].$$

Therefore,

$$\begin{aligned}
\mathbb{E}[|E_L|] &\leq \sum_{(u,w)\in E} \Pr[(u, w) \in E_L] \\
&\leq \sum_{(u,w)\in E} \Pr[p^{\leq k}_{(v,u,w)}] + \sum_{(u,w)\in E} \Pr[p^{>k}_{(v,u,w)}] \\
&\leq \sum_{(u,w)\in E} \Pr[p^{\leq k}_{(v,u,w)}] + n^2 \Pr[|S_L| > k-1] \\
&\leq \sum_{(u,w)\in E} \Pr[p^{\leq k}_{(v,u,w)}] + 1 \qquad (4) \\
&\leq \sum_{\substack{\alpha\text{: paths of} \\ \text{length} \leq k \text{ from } v}} \Pr[\text{path } \alpha \text{ is prefix-legal }] + 1 \\
&\leq \sum_{t\leq k} \left(\sum_{\substack{\alpha\text{: paths of} \\ \text{length } t \text{ from } v}} \Pr[\text{path } \alpha \text{ is prefix-legal }] \right) + 1 \\
&\leq \sum_{t\leq k} (d^{t+c-ct} + \epsilon) + 1 = O(1) \qquad (5)
\end{aligned}$$

where Inequality (4) is due to Lemma 6.5, and Inequality (5) is due to Proposition 7.1 and Corollary 7.6. □

We now turn to bounding the expectation of the square of the number of edges in a vertex's relevant vicinity.

Lemma 7.8. *Let $G = (V, E)$ be a d-light graph and let the conditions of Proposition 7.5 hold. That is, $k, \kappa = O(\log n)$ and $L = \text{poly}(d)$. Furthermore let $c > 3$ be a constant. Let h be an adaptive k-wise ϵ-almost independent hash function, $h : V \to [L]$. Denote by E_L the number of edges that have at least one endpoint in the relevant vicinity of some vertex v. Then, $\mathbb{E}[|E_L|^2] = O(1)$.*

Proof. For simplicity, we ignore edges that are very close to v, similarly to Lemma 7.7: they cannot asymptotically affect the value we want to bound. For any edge $e = (u, w) \in E$, let $\mathbb{I}_e$ be an indicator variable whose value is 1 if $e \in E_L$ and 0 otherwise. Let $\mathbb{I}_{|S_L|>k}$ be an indicator variable whose value is 1 if $|S_L| > k$ and 0 otherwise. Then

$$|E_L|^2 = \left(\sum_{e\in E} \mathbb{I}_e \right)^2 = \sum_{e\in E} \mathbb{I}_e \sum_{f\in E} \mathbb{I}_f.$$

Let $p^{\leq k}_{(v,u,w)}$ and $p^{>k}_{(v,u,w)}$ be as in the proof of Lemma 7.7.

$$\begin{aligned}
|E_L|^2 &\leq \sum_{(u,w)\in E} (p^{\leq k}_{(v,u,w)} + p^{>k}_{(v,u,w)}) \sum_{(x,y)\in E} (p^{\leq k}_{(v,x,y)} + p^{>k}_{(v,x,y)}) \\
&= \sum_{(u,w)\in E} \sum_{(x,y)\in E} (p^{\leq k}_{(v,u,w)} + p^{>k}_{(v,u,w)})(p^{\leq k}_{(v,x,y)} + p^{>k}_{(v,x,y)}) \\
&\leq \sum_{(u,w)\in E} \sum_{(x,y)\in E} (p^{\leq k}_{(v,u,w)} p^{\leq k}_{(v,x,y)} + 3\mathbb{I}_{|S_L|>k-1}) \\
&\leq 3n^4 \mathbb{I}_{|S_L|>k-1} + \sum_{(u,w)\in E} \sum_{(x,y)\in E} (p^{\leq k}_{(v,u,w)} p^{\leq k}_{(v,x,y)}). \qquad (6)
\end{aligned}$$

For every vertex $u \in V$, let σ^t_u denote the number of simple paths from v to u of length t, and label these paths arbitrarily by $q^t_u(i), i = 1, 2, \ldots \sigma^t_u$. For each path $q^t_u(i)$, let $\hat{q}^t_u(i)$ be the random variable whose value is 1 if $q^t_u(i)$ is prefix-legal, and 0 otherwise. Let Λ^t_v denote the total number of simple paths of length t originating in v.

$$\sum_{(u,w)\in E}\sum_{(x,y)\in E}(p^{\le k}_{(v,u,w)}p^{\le k}_{(v,x,y)}) \le \sum_{w\in V}\sum_{t\le k}\sum_{i=1}^{\sigma^t_w}\sum_{y\in V}\sum_{s\le k}\sum_{j=1}^{\sigma^s_y}\hat{q}^t_w(i)\hat{q}^s_y(j)$$

$$\le 2\sum_{w\in V}\sum_{t\le k}\sum_{i=1}^{\sigma^t_w}\sum_{y\in V}\sum_{s\le t}\sum_{j=1}^{\sigma^s_y}\hat{q}^t_w(i)\hat{q}^s_y(j) \tag{7}$$

$$= 2\sum_{w\in V}\sum_{t\le k}\sum_{i=1}^{\sigma^t_w}\hat{q}^t_w(i)\sum_{y\in V}\sum_{s\le t}\sum_{j=1}^{\sigma^s_y}\hat{q}^s_y(j)$$

$$\le 2\sum_{w\in V}\sum_{t\le k}\sum_{i=1}^{\sigma^t_w}\hat{q}^t_w(i)\sum_{s\le t}\Lambda^s_y,$$

where Inequality (7) is because we order the paths by length and either path can be longer.

$$\mathbb{E}\left[\sum_{(u,w)\in E}\sum_{(x,y)\in E}(p^{\le k}_{(v,u,w)}p^{\le k}_{(v,x,y)})\right] \le 2\sum_{t\le k}\sum_{\substack{\alpha:\text{ paths of}\\ \text{length } t \text{ from } v}}\Pr[\alpha \text{ is prefix-legal}]\cdot\mathbb{E}\left[\sum_{s\le t}\Lambda^s_y\right]$$

$$\le 2\sum_{t\le k}d^{t+c-ct}\cdot\mathbb{E}\left[\sum_{s\le t}\Lambda^s_y\right]$$

$$\le 2\sum_{t\le k}(d^{t+c-ct}+\epsilon)d^{t+1}$$

$$\le 2\sum_{t\le k}(d^{3t+c-ct}+\epsilon') = O(1). \tag{8}$$

From Corollary 7.6 we know that we can choose an $L = O(d)$ such that Inequality (8) holds. From Inequalities (6) and (8), Lemma 6.5 and the linearity of expectation, the lemma follows. □

8. Bounding running time and transient memory

Let $G = (V, E)$ be a d-light graph, $d > 0$. Let F be a search problem on V, and assume that there exists a neighborhood-dependent online algorithm $\mathcal{A}$ for F. We show how we can use the results of the previous sections to construct an LCA for F. Given a query $v \in V$, we would like to generate a permutation Π on V, build the relevant vicinity of v relative to Π, and simulate $\mathcal{A}$ on these vertices in the order of Π. Because $\mathcal{A}$ is neighborhood-dependent, we do not need to probe any vertices outside the relevant vicinity in order to correctly compute the output of $\mathcal{A}$ on v, and so our LCA will output a reply consistent with the execution of $\mathcal{A}$ on the vertices, if they arrive according to Π. In order to simulate $\mathcal{A}$ on the correct order, we need to store the relevant vicinity and label the vertices in a way that defines the ordering. We show two ways of doing this. The first gives a better time bound, at the expense of a worse space bound. The second gives a better space bound, at the expense of a worse time bound. It remains an open problem whether we can achieve "the best of both worlds"—an LCA requiring $O(\log n \log\log n)$ time and transient space (or even $O(\log n)$). We note that in expectation, both our LCAs require $O(\log\log n)$ time and $O(\log n)$ transient memory.

Theorem 8.1. *Let $G = (V, E)$ be a d-light graph, where $d > 0$ is a constant. Let F be a neighborhood-dependent search problem on V. Assume $\mathcal{A}$ is a crisp neighborhood-dependent online algorithm that correctly computes F on any order of arrival of the vertices. Then*

1. *There is an $(O(\log n), O(\log n \log\log n), O(\log n), O(\log^2 n), 1/n)$-LCA for F.*
2. *There is an $(O(\log n), O(\log^2 n), O(\log n), O(\log n \log\log n), 1/n)$-LCA for F.*

Furthermore, both LCAs require, in expectation, $O(\log\log n)$ time and $O(\log n)$ space.

Proof. We show two methods of constructing an LCA from the online algorithm $\mathcal{A}$. In both, given a query $v \in V$, we use adjacency lists to store the containing vicinity, $\Psi(v)$. We use a single bit to indicate for each vertex, whether it is in the relevant vicinity, $\Im(v)$. This means that for each vertex in $\Im(v)$, we keep a list of all of its neighbors, but for vertices that are in $\Psi(v) \setminus \Im(v)$, we don't need to keep such a list. We denote the number of vertices in the relevant vicinity, $|\Im(v)|$, by t_v, and the total number of edges stored by t_e. Note that $t_e \ge |\Psi(v)|$. This adjacency list representation of $\Psi(v)$ is generated slightly differently in the two constructions. In both cases we label this data structure by $D(v)$. For clarity, we abuse the notation, and use the same name, u, for $u \in V$, and for the vertex which represents u in $D(v)$.

Construction 1. The first time the LCA is invoked, it chooses a random function h from a family of adaptive ϵ-almost k-wise independent hash functions as in Theorem 5.10 and Lemma 6.5. The LCA receives as a query a vertex v, and computes $h(v)$. It then discovers the relevant vicinity using DFS. For each vertex u that it encounters, it relabels the vertex $(h(u), u)$. The LCA simulates $\mathcal{A}$ on the vertices arriving in the order induced by the new labels.

The size of the new label for any vertex u is $|(h(u), u)| = O(\log n)$. In addition, we need to store $\mathcal{A}(u)$ for every vertex $u \in \Im(v)$ (which is $O(1)$ because we assume $\mathcal{A}$ is crisp). Overall, because $|(h(u), u)| = O(\log n)$, the space required for the LCA is upper bounded by $O((t_e + t_v)\log n + t_v)$. From Corollary 6.8, $t_v, t_e = O(\log n)$. In expectation, by Lemma 7.8, $\mathbb{E}[t_e + t_v] = O(1)$. This gives us the required space bounds. To analyze the running time, we make the following observation.

Observation 8.2. *In Construction 1, given $u \in D(v)$, we can access $u \in V$ in $O(1)$.*

We look at each stage of the construction separately:

1. Constructing $D(v)$ is done by DFS, which takes time $O(t_v + t_e)$, as well as the time it takes to generate $|\Im(v)|$ labels, which requires invoking h at most t_e times, and, by Theorem 5.10, this requires $O(\text{loglog}\, n)$ time per label.
2. Sorting the labels takes $O(t_v \log t_v)$.
3. Simulating $\mathcal{A}$ on $\Psi(v)$ now takes $O(t_e)$ (as $\mathcal{A}$ is crisp).

Given the high probability upper bounds and expected values of t_v and t_e, the first part of the theorem follows. (Note that if $\mathcal{A}$ is not crisp in the sense that computing $F(v)$ is more than linear in the number of neighbors of v, this must be taken into account in the running time.)

In the first construction method, we give each vertex a label of length $O(\log n)$. This seems wasteful, considering we know that the expected size of the relevant vicinity is $O(1)$, and that its size is $O(\log n)$ w.h.p. We therefore give a more space-efficient method of constructing the induced subgraph.

Construction 2. As in the first method, the first time the LCA is invoked, it chooses a random function h from a family of adaptive ϵ-almost k-wise independent hash functions as in Theorem 5.10 and Lemma 6.5. Again, we would like to construct the induced subgraph of the relevant vicinity, but to save memory we will not hold the original labels of the vertices (which require $\log n$ bits to represent), but rather new labels that require at most $\text{loglog}\, n$ bits to describe (logarithmic in the size of the relevant vicinity). As before, the LCA receives as a query a vertex v, and computes $h(v)$. We still use $(h(v), v)$ to determine the ordering, however we do not commit this ranking to memory. We initialize $S = \{v\}$, and give v the label 1. In each round i, we look at $N(S)$, and choose the vertex u with the highest rank $(h(u), u)$, out of all the vertices in $N(S)$ which have a lower rank than their neighbor in S. We then add u to S, and give it the label $i+1$. When we have discovered the entire relevant vicinity, we simulate $\mathcal{A}$ on the vertices in the reverse order of the new labels.

Note on the required data structure To efficiently build S, we need to use a slightly different data structure used for storing the adjacency lists than in Construction 1; in fact, we have two adjacency list data structures. The first, $D_1(v)$, contains only the vertices in the relevant vicinity (not vertices in $\Psi(v) \setminus \Im(v)$). The second, $D_2(v)$, holds the neighbors of the vertices of S which have not yet been added to S. In $D_1(v)$, each vertex is represented by its new label. In $D_2(v)$, each vertex is represented only by its level. We need D_2 to avoid recalculating $h(u)$ more than once for each vertex u. In both $D_1(v)$ and $D_2(v)$, for each edge (i, j) which represents the edge (v_i, v_j), we also store the position of this edge among the edges that leave v_i, and its direction of discovery.

Correctness Note that v is the vertex of highest rank in its relevant vicinity, and indeed it holds by induction that at step i, the subgraph we expose will contain vertices $v_1 = v, v_2, \ldots, v_i$ that have the highest ranking in the relevant vicinity of v (and such that v_i has the ith highest rank). This guarantees the correctness of $\mathcal{A}$—the reverse order of the labels is exactly the correct ranking of the vertices of the relevant vicinity.

Complexity The size of the new label for any vertex u is $O(\log t_e)$. In addition, we need to store, for each edge, its position relative to the edges, the edge's direction of discovery, and $\mathcal{A}(u)$ for every vertex $u \in \Im(v)$. Because the graph is d-light, we know that the degree of each edge is $O(\log n)$ w.h.p., and so keeping the relative position of each edge will require $O(\text{loglog}\, n)$ bits w.h.p. Overall the space required for $D_1(v)$ is upper bound by $O((t_e + t_v)(\log t_e) + t_v + t_e \text{loglog}\, n)$. The space required for $D_2(v)$ is $O(t_e \cdot |L|) = O(t_e)$. From Corollary 6.8, and Lemma 7.8, we have the required space bounds. The expected space bound is due to the length of the seed, $O(\log n)$.

Observation 8.3. *In Construction 2, given $u \in D_1(v) \cup D_2(v)$, we can access $u \in V$ in $O(t_v)$.*

Proof. Given $u \in D_1(v)$ (or $D_2(v)$), we find v by DFS from u. As the edges are directed, and $D_1(v)$ and $D_2(v)$ are acyclic, this takes $O(t_v)$. Note that the space required for this DFS may be as much as $t_v \text{loglog}\, n$, but we use that amount of space regardless. We can store the path $v \rightsquigarrow u$ using the relative locations of the edges, and follow this path on G to find u. □

Similarly to Construction 1, in order to bound the complexity, we look at each stage of the construction separately:

1. Before we add a vertex to S, we need find the vertex u with the lowest $(h(u), u)$ among all vertices in $\Psi(v) \setminus \Im(v)$. This is done by going over all of these vertices to find the minimum, using a DFS on G and the subgraph concurrently, which takes $O(t_e)$.
2. Once we have chosen which vertex u to add to S, we update $D_1(v)$ and $D_2(v)$ to include it. When we look at u's neighbors, though, we don't know whether they are already in $D_1(v)$ or $D_2(v)$, as we don't have pointers to the original vertices in G. From Observation 8.3, though, finding this out takes $O(t_v)$ per neighbor, and updating $D_1(v)$ and $D_2(v)$ takes a further $O(1)$ per neighbor.
3. Because of $D_2(v)$, we only need to generate h once for each vertex in $\Psi(v)$. This takes $O(t_e \ \mathrm{loglog} n)$.
4. Reversing the order of the labels takes $O(t_v)$.
5. Simulating $\mathcal{A}$ on $\Psi(v)$ takes $O(t_e)$.

Stages 3, 4 and 5 require $O(t_e \log\log n)$ time in total. Stage 1 accounts for $O({t_e}^2)$, and 2 for $O(t_e t_v)$ overall. Lemma 7.8 gives the required expected time bound. Note that we have not discounted the possibility that $\mathcal{A}$ requires the original labels (or "names") of the vertices, in order to compute $F(v)$. When we encounter a vertex, we can always give $\mathcal{A}$ the name of the vertex by exploring the original representation of the graph (the additional time needed is bounded by $O({t_e}^2)$, the same bound as for Stage 1).

The worst case running time and space of the LCA are $O(\log^2 n)$ and $O(\log n \log\log n)$ respectively (w.h.p.). (Note that if $\mathcal{A}$ is not crisp in the sense that $|F(v)|$ is not a constant, this must be taken into account in the space bounds.) □

Our results immediately extend to the case that $d = O(\log\log n)$ (see Remark 6.9):

Theorem 8.4. *Let $G = (V, E)$ be a d-light graph, where $d = O(\log\log n)$. Let F be a neighborhood-dependent search problem on V. Assume A is a crisp neighborhood-dependent online algorithm that correctly computes F on any order of arrival of the vertices. Then there is an $(O(\mathrm{polylog} n), O(\mathrm{polylog} n), O(\mathrm{polylog} n), O(\mathrm{polylog} n), 1/n)$-LCA for F.*

In Appendix A we show that the techniques of the paper do not hold for graphs where the expected degree is $\omega(\log\log n)$, and so the results of this section are, in this sense, tight.

9. Reducing LCAs to distributed algorithms

Let $G = (V, E)$ be a graph, and let O be some arbitrary finite set. Parnas and Ron [27] noticed the following: Assume that there is a (crisp) distributed algorithm $\mathcal{A}$ that computes some function $F : V \to O$. If $\mathcal{A}$ terminates after a constant number of rounds, ℓ, then it is suffices to simulate $\mathcal{A}$ on the ℓ-neighborhood of any vertex v to obtain the value of $F(v)$ (Theorem 9.1). It is easy to see why this is true: if $\mathcal{A}$ terminates after ℓ rounds, no information from a vertex whose distance from v is greater than ℓ can reach v. In fact, this reasoning shows that it suffices to simulate the first round of $\mathcal{A}$ on the $N_\ell(v)$, the second round on $N_{\ell-1}(v)$, and so on. For simplicity, we assume that the distributed algorithm is simulated on the entire neighborhood for ℓ rounds, as this does not asymptotically affect the running time.[10] This reduction immediately allows us to convert any ℓ-time distributed algorithm to an $(O(d^\ell), O(\ell d^\ell), 0, O(\ell d^\ell), 0)$-LCA for graphs of degree bounded by d.

Theorem 9.1 *([27]). Let $G = (V, E)$ be a distributed network with degree at most d. Let $\mathcal{D}$ be a deterministic distributed algorithm that computes a labeling $D(G)$ in k rounds. Then it is possible, for any node $v \in V$ to compute $D(v)$ in time and space complexity $O(d^k)$ using a single processor, where the algorithm uses only neighbor and degree queries.*

Applying the Parnas–Ron reduction to d-light graphs guarantees that the number of probes, running time and transient memory are $O(\log^\ell n)$ w.h.p. (if the distributed algorithm requires ℓ rounds). We show that this analysis is far from tight, in fact these complexity measures are all $O(\log n)$.[11] This result is formally stated below as Theorem 9.3; we prove it by bounding the size of the ℓ-neighborhood of v, for any $v \in V$.

Bounding the neighborhood size Recall that $N_i(v)$ is the set of vertices at distance at most i from v.

Claim 9.2. *For any constant integer $i > 0$, there exists a constant c_i such that for any d-light graph $G = (V, E)$ and vertex $v \in V$, it holds that $\Pr[|N_i(v)| \le c_i \log n] \ge 1 - \frac{1}{n^4}$.*

[10] There is a small nuance: to simulate the second round of $\mathcal{A}$ on u, a vertex at distance ℓ from v, we might need to simulate the first round of $\mathcal{A}$ on u's neighbors, which may be at distance $\ell+1$ from v. But for the purpose of the simulation, we can assume that u has no neighbors that are at distance $\ell+1$ from v, as anything that happens to u after the first round cannot affect v. Therefore, once we have uncovered the ℓ-neighborhood of v, we can ignore the rest of the graph.

[11] ℓ appears exponentially in the constant.

Proof. Let $\mathcal{N}^i$ be the random variable representing the number of vertices in the i-neighborhood of vertex v. We prove by induction that $\Pr[\mathcal{N}^i \leq c_i \log n] \geq 1 - \frac{i}{n^5}$, where $c_i = (4ed)^{i-1}c_1$.

For the base, from Property 4.5, taking $S = \{v\}$, it holds that there exists a constant c_1 such that, $\Pr[\mathcal{N}^1 > c_1 \log n] \leq 1/n^5$.

Assuming that the claim holds for all integers smaller than i, we show that it holds for i. We use the law of total probability.

$$\begin{aligned}\Pr[\mathcal{N}^i > c_i \log n] &= \Pr[\mathcal{N}^i > c_i \log n | \mathcal{N}^{i-1} \leq c_{i-1} \log n] \Pr[\mathcal{N}^{i-1} \leq c_{i-1} \log n] \\ &\quad + \Pr[\mathcal{N}^i > c_i \log n | \mathcal{N}^{i-1} > c_{i-1} \log n] \Pr[\mathcal{N}^{i-1} > c_{i-1} \log n] \\ &\leq \Pr[\mathcal{N}^i > c_i \log n | \mathcal{N}^{i-1} \leq c_{i-1} \log n] + \Pr[\mathcal{N}^{i-1} > c_{i-1} \log n] \\ &\leq \frac{1}{n^5} + \frac{i-1}{n^5},\end{aligned}$$

where the last inequality uses Property 4.5 and the inductive hypothesis. □

We conclude the following.

Theorem 9.3. *Let $G = (V, E)$ be a d-light graph, where $d > 0$ is a constant, let O be some finite set and let $F : V \to O$ be some function on the vertices. Assume that $\mathcal{A}$ is a crisp constant-time distributed algorithm for F. Then there is an $(O(\log n), O(\log n), 0, O(\log n), \frac{1}{n^4})$-LCA for F.*

Proof. From Claim 9.2, the ℓ-neighborhood of any vertex is number of probes required to simulate the distributed algorithm is $O(\log n)$, for some constant $c > 0$, with probability at least $\frac{1}{n^5}$. Assume that the distributed algorithm $\mathcal{A}$ runs for ℓ rounds. As $\mathcal{A}$ is crisp, simulating it on this neighborhood requires time $\ell \cdot O(1) \cdot O(\log n)$, and this is clearly also an upper bound on the required transient memory. A union bound gives the required failure probability. □

Acknowledgments

We wish to thank Yishay Mansour and the anonymous referees for their useful comments.

Appendix A. Tightness with respect to *d*-light graphs

Our results hold for d-light graphs where $d = O(\text{loglog} n)$. We show that at least, using the technique of simulating an online algorithm on a random ordering of the vertices, we cannot do better. We do this by showing that the expected relevant vicinity of the root of a complete binary tree of a d-regular graph is $\Omega(2^{d/2})$, and hence for $d = \omega(\text{loglog} n)$, the expected size will be super-polylogarithmic.

Lemma Appendix A.1. *Let T be a complete d-regular binary tree rooted at v, and let Π be a uniformly random permutation on the vertices. The expected size of the relevant vicinity of v relative to Π is at least $2^{d/2}$.*

Proof. Let X_ℓ be a random variable for the number of vertices on level ℓ of the tree that are in the relevant vicinity. There are exactly d^ℓ vertices on level ℓ. The probability that each one is in the relevant vicinity is at least $\frac{1}{\ell!}$, as this is the probability when the permutation on the vertices is truly random.

$$\mathbb{E}[X_\ell] \geq \frac{d^\ell}{\ell!} \geq \left(\frac{d}{\ell}\right)^\ell$$

Taking $\ell = d/2$ gives that $\mathbb{E}[X_\ell] \geq 2^{d/2}$. □

Appendix B. Proof of Lemma 5.8

Lemma 5.8. *([21]) For $n, L, k \in \mathbb{N}$ such that $k \leq n$ and for $\epsilon \geq 0$, every family of functions $\mathcal{H} = \{h : [n] \to [L]\}$ that is ϵ-almost k-wise independent is also adaptive ϵL^k-almost k-wise independent.*

Proof. The proof is by a simulation argument. Consider an adaptive distinguisher $\mathcal{D}$ that makes at most k queries; assume without loss of generality that $\mathcal{D}$ always makes *exactly* k distinct queries. We can define the following static distinguisher $\mathcal{D}'$ with oracle access to some function F as follows: $\mathcal{D}'$ samples k distinct outputs $y_1, y_2, \ldots, y_k \in [L]$ uniformly at random. $\mathcal{D}'$ then simulates $\mathcal{D}$ by answering the ith query x_i of $\mathcal{D}$ with y_i. Let σ be the bit $\mathcal{D}$ would have output in this simulation. Now $\mathcal{D}'$ queries F for $x_1, x_2, \ldots x_k$ (note that $\mathcal{D}'$ makes all of its queries simultaneously and is therefore static). If the replies obtained are consistent with the simulation (i.e. $y_i = F(x_i)$ for every i) then $\mathcal{D}'$ outputs σ. Otherwise, it outputs 0. By definition,

$$\Pr[\mathcal{D}'^{F\leftarrow\mathcal{H}}=1]=L^{-k}\Pr[\mathcal{D}^{F\leftarrow\mathcal{H}}=1].$$

This implies that for every H and G

$$|\Pr[\mathcal{D}'^{F\leftarrow\mathcal{H}}=1]-\Pr[\mathcal{D}'^{F\leftarrow\mathcal{G}}=1]|=L^{-k}|\Pr[\mathcal{D}^{F\leftarrow\mathcal{H}}=1]-\Pr[\mathcal{D}^{F\leftarrow\mathcal{G}}=1]|.$$

The proof follows. □

Appendix C. A Chernoff bound

We require the following Chernoff bound:

Theorem Appendix C.1 *(Chernoff bound). Let X be a binomially distributed random variable, $X\sim B(n,p)$, such that $\mu=np$. Then, for $\lambda>2e-1$ it holds that*

$$\Pr[X>(1+\lambda)\mu]<2^{-\mu\lambda}.$$

Proof. Substituting $\lambda\geq 2e-1$ into the standard Chernoff bound gives

$$\begin{aligned}\Pr[X>(1+\lambda)\mu]&\leq\left(\frac{e^{\lambda}}{(1+\lambda)^{1+\lambda}}\right)^{\mu}\\&\leq\left(\frac{e^{\lambda}}{(2e)^{1+\lambda}}\right)^{\mu}\\&=\leq 2^{-\mu\lambda}\end{aligned}$$ □

References

[1] Noga Alon, Ronitt Rubinfeld, Shai Vardi, Ning Xie, Space-efficient local computation algorithms, in: Proc. 22nd ACM-SIAM Symposium on Discrete Algorithms, SODA, 2012, pp. 1132–1139.
[2] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, Eli Upfal, Balanced allocations, SIAM J. Comput. 29 (1) (1999) 180–200.
[3] Jérémy Barbay, Gonzalo Navarro, Compressed representations of permutations, and applications, in: 26th International Symposium on Theoretical Aspects of Computer Science, STACS, 2009, pp. 111–122.
[4] Leonid Barenboim, Michael Elkin, Seth Pettie, Johannes Schneider, The locality of distributed symmetry breaking, in: FOCS, 2012, pp. 321–330.
[5] Petra Berenbrink, André Brinkmann, Tom Friedetzky, Lars Nagel, Balls into non-uniform bins, J. Parallel Distrib. Comput. 74 (2) (2014) 2065–2076.
[6] J. Lawrence Carter, Mark N. Wegman, Universal classes of hash functions, J. Comput. Syst. Sci. 18 (2) (1979) 143–154.
[7] Guy Even, Moti Medina, Dana Ron, Deterministic stateless centralized local algorithms for bounded degree graphs, in: 22th Annual European Symposium on Algorithms, ESA, 2014, pp. 394–405.
[8] Guy Even, Moti Medina, Dana Ron, Distributed maximum matching in bounded degree graphs, in: Proceedings of the 2015 International Conference on Distributed Computing and Networking, ICDCN, 2015, pp. 18:1–18:10.
[9] Mohsen Ghaffari, An improved distributed algorithm for maximal independent set, in: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10–12, 2016, 2016, pp. 270–277.
[10] Oded Goldreich, A brief introduction to property testing, in: Studies in Complexity and Cryptography, 2011, pp. 465–469.
[11] Avinatan Hassidim, Jonathan A. Kelner, Huy N. Nguyen, Krzysztof Onak, Local graph partitions for approximation and testing, in: FOCS, 2009, pp. 22–31.
[12] Avinatan Hassidim, Yishay Mansour, Shai Vardi, Local computation mechanism design, in: ACM Conference on Economics and Computation, EC'14, 2014, pp. 601–616.
[13] Nicole Immorlica, Mohammad Mahdian, Marriage, honesty, and stability, in: SODA, 2005, pp. 53–62.
[14] Fuhito Kojima, Parag A. Pathak, Incentives and stability in large two-sided matching markets, Am. Econ. Rev. 99 (3) (2009) 608–627.
[15] Reut Levi, Ronitt Rubinfeld, Anak Yodpinyanee, Brief announcement: local computation algorithms for graphs of non-constant degrees, in: Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA, 2015, pp. 59–61.
[16] Nathan Linial, Locality in distributed graph algorithms, SIAM J. Comput. 21 (1) (1992).
[17] Michael Luby, A simple parallel algorithm for the maximal independent set problem, SIAM J. Comput. 15 (4) (1986) 1036–1053.
[18] Yishay Mansour, Shai Vardi, A local computation approximation scheme to maximum matching, in: APPROX-RANDOM, 2013, pp. 260–273.
[19] Yishay Mansour, Aviad Rubinstein, Shai Vardi, Ning Xie, Converting online algorithms to local computation algorithms, in: Proc. 39th International Colloquium on Automata, Languages and Programming, ICALP, 2012, pp. 653–664.
[20] Yishay Mansour, Boaz Patt-Shamir, Shai Vardi, Constant-time local computation algorithms, in: Approximation and Online Algorithms – 13th International Workshop, WAOA, 2015, pp. 110–121.
[21] Ueli Maurer, Krzysztof Pietrzak, Composition of random systems: when two weak make one strong, in: Moni Naor (Ed.), Theory of Cryptography, in: Lecture Notes in Computer Science, vol. 2951, Springer, Berlin Heidelberg, ISBN 978-3-540-21000-9, 2004, pp. 410–427.
[22] Raghu Meka, Omer Reingold, Guy N. Rothblum, Ron D. Rothblum, Fast pseudorandomness for independence and load balancing—(extended abstract), in: ICALP (1), 2014, pp. 859–870.
[23] S. Muthukrishnan, Data streams: algorithms and applications, Found. Trends Theor. Comput. Sci. 1 (2) (2005).
[24] Joseph Naor, Moni Naor, Small-bias probability spaces: efficient constructions and applications, in: STOC, 1990, pp. 213–223.
[25] Moni Naor, Larry J. Stockmeyer, What can be computed locally? SIAM J. Comput. 24 (6) (1995) 1259–1277.
[26] Huy N. Nguyen, Krzystof Onak, Constant-time approximation algorithms via local improvements, in: Proc. 49th Annual IEEE Symposium on Foundations of Computer Science, FOCS, 2008, pp. 327–336.
[27] M. Parnas, D. Ron, Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms, Theor. Comput. Sci. 381 (1–3) (2007).
[28] Seth Pettie, Hsin-Hao Su, Fast distributed coloring algorithms for triangle-free graphs, in: ICALP (2), 2013, pp. 681–693.

[29] Ron Dana, Algorithmic and analysis techniques in property testing, Found. Trends Theor. Comput. Sci. 5 (2) (2009) 73–205.
[30] Ronitt Rubinfeld, Asaf Shapira, Sublinear time algorithms, SIAM J. Discrete Math. 25 (4) (2011) 1562–1588.
[31] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, Ning Xie, Fast local computation algorithms, in: Proc. 2nd Symposium on Innovations in Computer Science, ICS, 2011, pp. 223–238.
[32] M. Shaked, J.G. Shanthikumar, Stochastic Orders and Their Applications, Academic Press, Inc., San Diego, CA, USA, 1994.
[33] Jukka Suomela, Survey of local algorithms, ACM Comput. Surv. 45 (2) (2013) 24.
[34] Salil P. Vadhan, Pseudorandomness, Found. Trends Theor. Comput. Sci. 7 (1–3) (2011) 1–336.
[35] Vöcking Berthold, How asymmetry helps load balancing, J. ACM 50 (4) (2003) 568–589.
[36] Udi Wieder, Balanced allocations with heterogenous balls, in: Proc. 19th ACM Symposium on Parallel Algorithms and Architectures, SPAA, 2007, pp. 188–193.
[37] Jian Zhang, A survey on streaming algorithms for massive graphs, in: Managing and Mining Graph Data, Springer, 2010, pp. 393–420.