# On Derandomizing Local Distributed Algorithms

Mohsen Ghaffari
ETH Zurich
ghaffari@inf.ethz.ch

David G. Harris
University of Maryland, College Park
davidgharris29@gmail.com

Fabian Kuhn*
University of Freiburg
kuhn@cs.uni-freiburg.de

**Abstract**

The gap between the known randomized and deterministic local distributed algorithms underlies arguably the most fundamental and central open question in *distributed graph algorithms*. In this paper, we develop a generic and clean recipe for derandomizing randomized LOCAL algorithms and transforming them into efficient deterministic LOCAL algorithm. We also exhibit how this simple recipe leads to significant improvements on a number of problems, in cases resolving known open problems. Two sample end-results are as follows:

- An improved distributed hypergraph maximal matching algorithm, which improves on that of Fischer, Ghaffari, and Kuhn [FOCS'17], and leads to improved algorithms for edge-coloring, maximum matching approximation, and low out-degree edge orientation. The first gives an improved algorithm for Open Problem 11.4 of the book of Barenboim and Elkin, and the last gives the first positive resolution of their Open Problem 11.10.
- An improved distributed algorithm for the Lovász Local Lemma, which gets closer to a conjecture of Chang and Pettie [FOCS'17], and moreover leads to improved distributed algorithms for problems such as defective coloring and $k$-SAT.

*Supported by ERC Grant No. 336495 (ACDC).

# 1 Introduction and Related Work

The gap between the best known deterministic and randomized local distributed algorithms constitutes a central open question in the area of *distributed graph algorithms*. For many of the classic problems (e.g., maximal independent set (MIS) and $(\Delta+1)$-vertex coloring), $O(\log n)$-time randomized algorithms have been known since the pioneering work of Luby [Lub86] and Alon, Babai, and Itai [ABI86]. However, obtaining a $\operatorname{polylog} n$-time deterministic algorithm remains an intriguing open problem for 30 years now, which was first asked by Linial [Lin87]. The best known deterministic round complexity is $2^{O(\sqrt{\log n})}$, due to Panconesi and Srinivasan [PS95].

The issue is not limited to a few problems; these are just symptomatic of our lack of general tools and techniques for derandomization. Indeed, in their 2013 book on Distributed Graph Coloring [BE13], Barenboim and Elkin stated that "*perhaps the most fundamental open problem in this field is to understand the power and limitations of randomization*", and left the following as their first open problem[1]:

> OPEN PROBLEM 11.1 (Barenboim & Elkin [BE13]) Develop a general derandomization technique for the distributed message-passing model.

There is also a more modern and curious motive for developing derandomization techniques for distributed algorithms, even if one does not mind the use of randomness: efficient deterministic algorithms can help us obtain even more efficient *randomized* algorithms. More concretely, most of the recent developments in randomized algorithms use the *shattering technique* [Gha16, BEPS16, EPS15, GS17, FG17] which randomly breaks down the graph into small components, typically $\operatorname{polylog} n$-size, and then solves them separately via a deterministic algorithm. Once we develop faster deterministic algorithms, say via derandomization, we can speed up the corresponding randomized algorithms.

We next overview our contributions regarding this question. Before that, we review Linial's LOCAL model [Lin87, Pel00], which is the standard synchronous message passing model of distributed computing, and a sequential variant of it, called SLOCAL, introduced recently by Ghaffari, Kuhn, and Maus [GKM17], which will be instrumental in driving and explaining our results.

**The LOCAL Model:** The communication network is abstracted as an undirected $n$-node graph $G = (V, E)$ with maximum degree $\Delta$, and where each node has a $\Theta(\log n)$-bit unique identifier. Initially, nodes only know their neighbors in $G$. At the end, each node should know its own part of the solution, e.g., its color in coloring. Communication happens in synchronous rounds, where in each round each node can perform some arbitrary internal computations and it can exchange a possibly arbitrarily large message with each of its neighbors. The *time complexity* of an algorithm is defined as the number of rounds that are required until all nodes terminate. In the case of randomized algorithms, each node can in addition produce an arbitrarily long private random bit string before the computation starts.

A LOCAL model algorithm with time complexity $r$ can alternatively be defined as follows. When the computation starts, each node in parallel reads the initial states of the nodes in the $r$-hop neighborhood (including the private random bit strings for randomized algorithms). Based on that information, each node in parallel computes its ouput.

**The SLOCAL Model:** The SLOCAL is similar to the LOCAL model, in that each node can read its $r$-hop neighborhood in the graph $G$, for some parameter $r$. However, in the SLOCAL model, the neighborhoods are read sequentially. Formally, the nodes are processed in an arbitrary (adversarially chosen) order. When node $v$ is processed, $v$ can read its $r$-hop neighborhood and it computes and locally stores its output $y_v$ and

[1] Indeed, their Open Problems 11.2 to 11.5 also deal with the same question, directly asking for finding efficient deterministic algorithms for certain problems; in all cases, the known randomized algorithms satisfy the goal.

potentially additional information. When reading the $r$-hop neighborhood, $v$ reads all the information that has been locally stored by the previously processed nodes of $v$'s $r$-hop neighborhood. We call the parameter $r$ the *locality* or *complexity* or an SLOCAL algorithm.

Algorithms in the SLOCAL model can be seen as natural extensions of sequential greedy algorithms. In fact, the classic distributed graph problems such as MIS or $(\Delta+1)$-coloring have simple SLOCAL algorithms with locality 1: In order to determine whether a node $v$ is in the MIS or which color $v$ gets in a $(\Delta+1)$-coloring, it suffices to know the decisions of all the neighbors that have been processed before.

## 1.1 Our Contributions, Part I: Derandomization

In the first part of this paper, we present a simple and clean recipe for derandomizing local distributed algorithms. A simplified, and imprecise, version of our derandomization result is as follows:

**Theorem 1.1** (Derandomization—Informal and Simplified)**.** *Any $r$-round randomized* LOCAL *algorithm for a locally checkable problem can be transformed to a deterministic* SLOCAL *algorithm with locality $O(r)$. This* SLOCAL *algorithm can then be transformed to a deterministic* LOCAL *algorithm with complexity $\Delta^{O(r)}+O(r\log^* n)$, or $r\cdot 2^{O(\sqrt{\log n})}$, by using network decompositions.*

We note that it is necessary to assume that the problem solution is locally checkable. In Proposition 7.4, we provide a simple problem that is not locally checkable, for which a constant-time randomized distributed algorithm exists, but where the best deterministic SLOCAL algorithm (or consequently LOCAL algorithm) has at least polynomial locality. Moreover, the $2^{O(\sqrt{\log n})}$ term in the above round complexity is due to the currently best-known round complexity of computing $(O(\log n), O(\log n))$-network decompositions[2] [PS95]. Because of this overhead, unfortunately, the deterministic algorithms that we obtain out of this derandomization method may be much less efficient than their randomized counterparts. However, as we overview next, still in many cases we get algorithms that are far more efficient than existing algorithms, and in a few cases, they resolve known open problems; this is especially thanks to the first bound, combined with some other ideas that effectively reduce the maximum degree $\Delta$.

### 1.1.1 Hypergraph Maximal Matching and Ramifications

Hypergraph maximal matching was recently pointed out by Fischer, Ghaffari, and Kuhn [FGK17] as a clean and powerful problem, which admits reductions from several classic problems of the area. As the first concrete implication of our derandomization technique, we obtain an improved deterministic algorithm for maximal matching in hypergraphs:

**Theorem 1.2.** *There is an $O(r^2\log(n\Delta)\log n\log^4\Delta)$-round deterministic* LOCAL *algorithm that computes a maximal matching for any $n$-node hypergraph with maximum degree $\Delta$ and rank at most $r$ (the rank is the maximum number of vertices in a hyperedge).*

The solution of [FGK17] has complexity $\log^{O(\log r)}\Delta\cdot\log n$, and it thus is efficient essentially only for hypergraphs of rank $O(1)$. Improving this $r$-dependency was left as a main open question in [FGK17]. Theorem 1.2 improves the dependency to $\mathrm{poly}(r)$ and leads to efficient algorithms for rank up to $\mathrm{polylog}\, n$.

Consequently, thanks to known reductions [FGK17], this improved hypergraph maximal matching algorithm leads to efficient deterministic algorithms for a number of problems, including $(2\Delta-1)$-edge coloring, $(1-\varepsilon)$-approximation of maximum matching, and low out-degree orientation:

**Corollary 1.3.** *There is a deterministic distributed algorithm that computes a $(2\Delta-1)$-list-edge-coloring in $O(\log^4\Delta\log^2 n)$ rounds.*

[2]The formal definition of network decompositions will be reviewed later in Section 2.

**Corollary 1.4.** *There is a deterministic distributed algorithm that computes a $(1-\varepsilon)$-approximation of maximum matching in $O(\log^2 n \log^5 \Delta/\varepsilon^9)$ rounds.*

**Corollary 1.5.** *There is a deterministic distributed algorithm that computes an orientation with maximum out-degree at most $(1+\varepsilon)\lambda$ in any graph with arboricity $\lambda$, in $O(\log^{10} n \log^5 \Delta/\varepsilon^9)$ rounds.*

Corollary 1.3 gives an alternative solution for the problem of $(2\Delta-1)$-edge coloring, which was a well-known open problem since 1990s (listed as Open Problem 11.4 of [BE13]) and was resolved recently by Fischer et al. [FGK17]; indeed, the solution of Corollary 1.3 is more efficient than the $O(\log^8 n)$ algorithm of [FGK17]. Moreover, Corollary 1.5 gives the first positive resolution of Open Problem 11.10 of [BE13].

### 1.1.2 The Lovász Local Lemma and Extensions

The Lovász Local Lemma (LLL) is a powerful probabilistic tool; at a high level, it states that if one has a probability space $\Omega$ and a collection $\mathcal{B}$ of "bad" events in that space, then as long as the bad-events have low probability and are not too interdependent, then there is a positive probability that no event in $\mathcal{B}$ occurs; and in particular a configuration avoiding $\mathcal{B}$ exists. In its simplest, "symmetric" form, it states that if each bad-event has probability at most $p$, and each bad-event affects $d$ other bad-events, and $epd \leq 1$, then there is a positive probability that no bad-event occurs.

A distributed algorithm for the LLL is a key building-block for a number of distributed graph coloring algorithms, such as frugal or defective vertex-colorings. In addition, as Chang & Pettie noted in [CP17], the LLL plays a important role in the overall landscape of LOCAL algorithms; the reason is that in any randomized LOCAL$(r)$ algorithm, one may define a bad-event that a given node $v$ fails; this bad-event has low probability, and because of the locality of the procedure it only affects other nodes within radius $2r$.

As a result, [CP17] showed that if we have a distributed LLL algorithm running in time $t(n)$, then any LOCAL algorithm on a bounded-degree graph for a locally checkable problem running in $o(\log_\Delta n)$ rounds, can be sped up to time $t(n)$. Thus, in a sense, the LLL is a universal sublogarithmic LOCAL algorithm. They further conjectured that the LLL could run in time $O(\log\log n)$ (matching a lower bound of [BFH+16]), which in turn would allow a vast range of other LOCAL algorithms to run in $O(\log\log n)$ time.

There has been a long history of developing algorithmic versions of the LLL, including distributed algorithms. A breakthrough result of Moser & Tardos [MT10] gave one of the first general sequential algorithms for the LLL; they also discussed a parallel variant, which can easily be converted into a randomized LOCAL$(O(\log^2 n))$ algorith,. There have been a number of other algorithms developed specifically in the context of the LOCAL model [CPS17, FG17]. These typically require satisfying a stronger condition than the LLL, of the form $p(2d)^c \leq 1$, for some constant $c \geq 1$; we refer to as a *polynomially-weakened LLL criterion* (pLLL). Most LLL constructions can be adapted to a pLLL criterion, with only some small loss in constant terms.

More recently, Fischer & Ghaffari [FG17] described an algorithm running in $2^{O(\sqrt{\log\log n})}$ rounds, under the pLLL criterion $p(ed)^{32} < 1$, as long as $d < (\log\log n)^{1/5}$. Despite this significant limitation on degree, this algorithm nevertheless can be used to construct a number of combinatorial objects including defective coloring, frugal coloring, and vertex coloring in $2^{O(\sqrt{\log\log n})}$ time for arbitrary degree graphs.

In this paper, we give new LLL algorithms, that can be faster than those of [FG17] and that can be used for higher-degree graphs. The first algorithm gives the following guarantee:

**Theorem 1.6.** *Let $i \geq 1$ be a positive integer. If $(20000d^8)^i p \leq 1$ and $d \leq 2^{\sqrt{\log^{(i+1)} n}}$, then, w.h.p., a configuration avoiding $\mathcal{B}$ can be found in time*

$$(2\uparrow)^i \Big( O(\sqrt{\log^{(i+1)} n}) \Big),$$

*where $(2\uparrow)^{(i)}$ is the Knuth iterated-exponentation notation and $\log^{(j)}$ is the $j$-fold iterated logarithm.*

For example, with $i = 1$, the algorithm $\mathcal{A}_i$ requires $d \leq 2^{\sqrt{\log \log n}}$ and runs in $2^{O(\sqrt{\log \log n})}$ rounds—we thus have improved the dependence of $d$ by an exponential factor compared to [FG17]. For $i = 2$, the algorithm $\mathcal{A}_i$ requires $d \leq 2^{\sqrt{\log \log \log n}}$ and runs in $2^{2^{O(\sqrt{\log \log \log n})}}$ rounds. These faster algorithms make partial progress toward showing the conjecture of [CP17] for the running time of LLL on bounded-degree graphs.

The second LLL algorithm we develop does not make any requirement on the size of $d$, but is not as general in terms of the types of bad-events it can work on. It requires that the bad-events satisfy a different property which we refer to as *bounded fragility*. We show the following result:

**Theorem 1.7.** *Suppose that every bad-event $B \in \mathcal{B}$ has fragility at most $F$ and $Fd^{13} \leq 1$. Then there is an algorithm to find a configuration avoiding the bad-events in $2^{O(\sqrt{\log \log n})}$ rounds, w.h.p.*

This property is satisfied by a number of combinatorial problems such as $k$-SAT and defective vertex coloring. For example, we obtain the following algorithmic applications:

**Proposition 1.8.** *If a $k$-SAT formula $\Phi$ has $n$ variables and $m$ clauses and $d \leq (4/3)^{k/3} \approx 1.02238^k$, there is a distributed algorithm to find a satisfying solution to $\Phi$ is $2^{O(\sqrt{\log \log m})}$ rounds.*

**Proposition 1.9.** *Suppose $G$ has maximum degree $\Delta$ and $h \leq \Delta$. There is a distributed algorithm in $2^{O(\sqrt{\log \log n})}$ rounds to find an $h$-defective $k$-coloring with $k = O(\Delta/h)$.*

We note that Proposition 1.9 was shown by [FG17] using an ad-hoc algorithm based on recoloring vertices; our main contribution is to derive it as a black-box application of the more general Theorem 1.7. Proposition 1.8 appears to be the first algorithm for solving $k$-SAT in the distributed setting which makes no restriction on the maximum degree as a function of $n$.

We briefly summarize our algorithmic improvements. Many of the previous LLL algorithms, including our new ones, are based on a general method for constructing distributed graph algorithms by *graph shattering* [Gha16, BEPS16, EPS15, GS17, FG17]. These algorithms have two phases. The first phase satisfies most of the vertices in the graph; the remaining unsatisfied vertices have small connected components. The second phase applies a deterministic algorithm to solve the residual components.

Our derandomization method allows us to convert previous randomized LLL algorithms into deterministic ones; these deterministic algorithms can then be applied for the second phase. Indeed, we can do this recursively: we convert randomized LLL algorithms into deterministic ones; these new randomized algorithms are derandomized and become the deterministic algorithm to use in the second phase of shattering. This method of combining algorithms gives a speedup. This also significantly simplifyies the analysis, as it removes the need for the cumbersome "bootstrapping" step in the LLL algorithm of [FG17].

### 1.1.3 The Role of Randomization in the Sequential LOCAL Model

Besides the above concrete improvements, our derandomization method has implications that are helpful in understanding the bigger-picture aspects of the role and power of randomization in the LOCAL model. From Theorem 1.1, we can infer that the SLOCAL model does not need randomness, at least so long as we are not concerned with logarithmic factors. Particularly, we get that

**Theorem 1.10.** *Any randomized* SLOCAL *algorithm with locality $r(n)$ for any locally checkable problem can be transformed to a deterministic* SLOCAL *algorithm with locality $O(r(n) \log^2 n)$.*

This in some sense justifies the philosophy underlying the work of Ghaffari, Kuhn, and Maus [GKM17], who tried to decouple the challenges in this model into two parts: (A) a *locality* challenge, capturing the fact that each output is a function of a small neighborhood around it, and (B) *local coordination* and *symmetry breaking* challenges, capturing the fact that nearby nodes have to output different-looking outputs, e.g.

different colors, but all have to decide essentially at the same time. The SLOCAL model was an attempt to focus on the former, while removing the latter. This was with the intuitive hope that, while the latter seems to naturally benefit from randomization, locality on its own should not need randomization, at least so long as we ignore logarithmic factors. Theorem 1.10 now validates this mindset, by showing that the SLOCAL model indeed does not need randomness. That is, in some sense, in the LOCAL model, randomization helps mainly with local coordination and symmetry breaking challenges, but not with the locality challenge. We remark that, if we do care about logarithmic factors, then a gap appears also in SLOCAL:

**Theorem 1.11.** *The sinkless orientation problem in bounded degree graphs has randomized* SLOCAL *locality* $\Theta(\log\log\log n)$ *and deterministic* SLOCAL *locality* $\Theta(\log\log n)$.

This exhibits an exponential separation between the randomized and deterministic complexities in the SLOCAL model, akin to those observed in the LOCAL model—sinkless orientation in LOCAL requires $\Omega(\log n)$ rounds deterministically [CKP16] and $\Omega(\log\log n)$ rounds randomly [BFH+16], and both lower bounds are tight [GS17]. We also find it surprising that the placement of this tight gap is an exponential function lower in the SLOCAL model.

Interestingly, this also has the following take-home message: if there is significant separation in the LOCAL model in the regime of polylogarithmic complexities and higher, it must be for a reason very different than those of [CKP16, BFH+16, GS17] (for sinkless orientation) which show up in regime of sublogarithmic complexities; as the latter extend to sublogarithmic complexities of the SLOCAL model but do not appear in the higher complexity regime of SLOCAL.

## 1.2 Our Contributions, Part II: Limitations of Derandomization

In the second part of the paper, we exhibit some form of limitations for any derandomization technique by presenting conditional hardness results on derandomization for some classic and well-studied distributed problems, including set cover approximation, minimum dominating set approximation, and computing neighborhood covers. Formally, these conditional-hardness results are by showing that these problems are P-SLOCAL-complete, in the framework set forth by Ghaffari, Kuhn, and Maus [GKM17]. A problem $\mathcal{P}$ is called P-SLOCAL-complete if $\mathcal{P}$ can be solved deterministically with polylogarithmic locality in the SLOCAL model and if a polylog-time deterministic distributed algorithm for $\mathcal{P}$ would imply such algorithms for all problems that can be solved deterministically with polylogarithmic locality in the SLOCAL model. We provide an informal explanation here; please see Section 7.3 for a more detailed explanation.

For the above three problems, rather satisfactory polylog-time randomized LOCAL algorithms have been known for many years, e.g., [BBR97, BRS94, RV98, DMP+05, JRS02, KW05, KMW06, LS93]. However, there are no known efficient deterministic algorithms. We show that devising efficient deterministic algorithms for them may be hard; at least as hard as some well-known open problems of the area. More concretely, if one finds polylogarithmic-time deterministic algorithms for any of these problems, that solution can be transformed into a polylogarithmic-time deterministic LOCAL algorithm for computing a $(O(\log n), O(\log n))$ network decomposition, and would consecutively imply polylogarithmic-time deterministic LOCAL algorithms for any problem that admits an SLOCAL algorithm with polylogarithmic locality—most notably, including computing an MIS. Hence, devising polylogarithmic-time deterministic algorithms for these problems is at least as hard as doing that for MIS, which remains a well-known open problem in the area since Linial explicitly asked for it in 1987 [Lin87].

Regarding the P-SLOCAL-completeness of neighborhood cover, we would like to hightlight the implication. The neighborhood cover problem asks for a collection of connected clusters of nodes such that each cluster has $\operatorname{polylog} n$ diameter, each node is contained in at most $\operatorname{polylog} n$ clusters, and each 1-hop neighborhood of the graph is completely contained in at least one cluster. The problem has been one of the central problems in the study of *local distributed graph algorithms* since the work of Awerbuch and

Peleg [AP90], closely related to another central problem, *network decompositions*, introduced by Awerbuch et al. [AGLP89]. By classic results of [ABCP96], it has been known that an efficient deterministic network decomposition algorithm can be transformed into an efficient deterministic neighborhood cover algorithm. Our result shows for the first time that the converse is also true: an efficient deterministic neighborhood cover algorithm can be transformed into one for network decomposition.

Finally, we also show P-SLOCAL-completeness for a particular problem of computing a maximal independent set of certain star structures in the graph.Again, we defer the formal description to Section 7.3. The significance of this last result is that it shows a clean problem which can be solved easily using a simple and natural greedy method—with locality 2 in the SLOCAL model—and yet it is P-SLOCAL-complete.

### 1.3 Organization of the Paper

The remainder of the paper is structured as follows. In Section 2, we formally define the various types of randomized and deterministic local algorithms and in particular the respective complexity classes that we use throughout the paper. Section 3 then proves the basic derandomization routine, which will be used as the main technical tool for most of our new results. The section also discusses various direct consequences of the result. In Section 4, we show how to use the derandomization procedure to obtain a better deterministic algorithm for computing a maximal matching in a hypergraph, and we show some consequences of this improved hypergraph matching algorithm. Section 5 formally proves all the results regarding the LLL that are discussed in Section 1.1.2. Finally, in Section 7, we discuss the limitations to derandomization and the completeness results that are outlined in Section 1.2. In addition, in Appendix A, we discuss some results on graph shattering and how our derandomization results can be used for shattering algorithms.

## 2 Model and Definitions

**Notation:** For a graph $G = (V, E)$ and a subset $X \subseteq V$, we define $G[X]$ to be the vertex-induced subgraph. For any integer $r \geq 1$, $G^r$ is the graph on vertex set $V$ and with an edge between any two nodes $u$, $v$ that are at distance at most $r$. Further, $\Delta(G)$ denotes the maximum degree of a graph $G$. Likewise, for a hypergraph $H = (V, E)$, we define $\Delta(H)$ to be the maximum degree, i.e., the maximum, over all vertices $v \in V$, of the number of edges $e \in E$ such that $v \in e$. We define the *rank* of hypergraph $H$ to be the maximum cardinality of any edge. We often write $\Delta$ instead of $\Delta(G)$ or $\Delta(H)$, if it is clear from context.

**Distributed Graph Problems:** We deal with *distributed graph problems* of the following form. We are given a simple, undirected graph $G = (V, E)$ that models the network. Initially, each node $v \in V$ gets some private input $x_v$. Each node has a unique ID, which is also part of the initial input. At the end of an algorithm, each node $v \in V$ needs to output a value $y_v$. Let $\boldsymbol{x}$ and $\boldsymbol{y}$ the vectors of all the inputs and outputs, respectively. An algorithm solving a distributed graph problem needs to guarantee that the triple $(G, \boldsymbol{x}, \boldsymbol{y})$ satisfies the specification of the graph problem. We assume that whether a given input-output pair satisfies the specification of a graph problem can only depend on the topology of $G$ and it has to be independent of the assignment of IDs to the nodes. For simplicity, we assume that all nodes also know a common polynomial upper bound on the number of nodes $n$. For a more formal definition, we refer to [GKM17]. We study distributed graph problems in the LOCAL and the SLOCAL models that were introduced in Section 1.

**Complexity Classes:** We next define the complexity classes that we use in the paper. We start by defining the classes for deterministic algorithms. Throughout, we only consider complexities as a function of the number of nodes $n$. For a more general and formal definition, we refer to [GKM17].

**LOCAL($t(n)$):** Distributed graph problems that can be solved by a *deterministic* LOCAL *algorithm* with time complexity at most $t(n)$.

**SLOCAL($t(n)$):** Distributed graph problems that can be solved by a *deterministic* SLOCAL *algorithm* with locality at most $t(n)$.

We further use the notion of local decision problems introduced by Fraigniaud, Korman, and Peleg in [FKP13]. A local decision problem tests whether a given labelled graph satisfies a certain property. A distributed algorithm is said to solve a given decision problem if the following holds. If the input satisfies the property, all nodes output "yes", otherwise, at least one node needs to output "no". When phrased as a distributed graph problem, each node receives its label as input. In [FKP13], the following complexity class is defined for local decision problems.

**LD($t(n)$):** Contains the set of local decision problems that can be *decided* by a *deterministic* LOCAL *algorithm* with time complexity at most $t(n)$.

We distinguish two kinds randomized algorithms: Monte Carlo and Las Vegas algorithms. A *distributed Monte Carlo* algorithm has a fixed time complexity and it guarantees that the solution solves the graph problem $\mathcal{P}$ with probability strictly larger than $1 - 1/n$. For a *distributed Las Vegas* algorithm, we also assume that the time complexity is fixed. However, in addition to the output of the graph problem, each node also outputs a bit $F_v$ indicating whether the algorithm failed locally at $v$. If $F_v = 0$ for all nodes $v \in V$, it is guaranteed that the computed output solves $\mathcal{P}$. Furthermore, it is guaranteed that $\sum_{v \in V} \mathbb{E}[F_v] < 1$. The complexity classes for the two kinds of randomized distributed algorithms are as defined as follows.[3]

**RLOCAL($t(n)$):** Distributed graph problems that can be solved by a *randomized Monte Carlo algorithm* in the LOCAL model with time complexity at most $t(n)$.

**ZLOCAL($t(n)$):** Distributed graph problems that can be solved by a *randomized Las Vegas algorithm* in the LOCAL model with time complexity at most $t(n)$.

We clearly have $\mathsf{RLOCAL}(t(n)) \subseteq \mathsf{ZLOCAL}(t(n))$. In addition, if a the solution to a graph problem can be checked by a local decision algorithm in $\mathsf{LD}(d)$, we can transform a $\mathsf{RLOCAL}(t)$ Monte Carlo algorithm into a $\mathsf{ZLOCAL}(t + d)$ Las Vegas algorithm.

In the context of the LOCAL model, we often think of an algorithm as *efficient* if its time complexity is at most polylogarithmic in $n$. We therefore define shortcut notations for above complexity classes when $t(n) = \operatorname{polylog} n$; specifically, we use **P-LOCAL**, **P-SLOCAL**, **P-RLOCAL**, **P-ZLOCAL**, and **P-LD** to denote $\mathsf{LOCAL}(\operatorname{polylog} n)$, $\mathsf{SLOCAL}(\operatorname{polylog} n)$, $\mathsf{RLOCAL}(\operatorname{polylog} n)$, $\mathsf{ZLOCAL}(\operatorname{polylog} n)$, and $\mathsf{LD}(\operatorname{polylog} n)$.

**Network Decomposition:** A graph problem that plays a central role in the area and also in the context of this paper is the problem of computing a network decomposition.

**Definition 2.1** (Network Decomposition)**.** [AGLP89] *A $\big(d(n), c(n)\big)$-decomposition of an $n$-node graph $G = (V, E)$ is a partition of $V$ into clusters such that every cluster has (weak or strong) diameter at most $d(n)$ and the cluster graph is properly colored with colors $\{1, \ldots, c(n)\}$.*

It was shown in [AP90, LS93] that every graph has an $\big(O(\log n), O(\log n)\big)$-decomposition. Further, as shown in [GKM17], the algorithm of [AP90, LS93] directly leads to an $\mathsf{SLOCAL}(O(\log^2 n))$-algorithm for

[3] This definition only guarantees a *positive* (not high) success probability. This initially seems to be much weaker than the corresponding definitions in complexity theory. We later show in Proposition 3.6 that any algorithm satisfying this weak property can be amplified to have much higher success probability.

computing an $\big(O(\log n), O(\log n)\big)$-decomposition. In [LS93], Linial and Saks describe a $O(\log^2 n)$-time distributed Las Vegas algorithm to compute a $\big(O(\log n), O(\log n)\big)$-decomposition. Further, in [GKM17], it was shown that the problem of computing a $(d(n), c(n))$-decomposition is P-SLOCAL-complete for any $d(n), c(n) = O(\log^k n)$ for any constant $k \geq 1$. As a consequence, there in particular is a polylog-time distributed Las Vegas algorithm for every problem in P-SLOCAL. The best deterministic distributed algorithm to compute a $\big(O(\log n), O(\log n)\big)$-decomposition is by Panconesi and Srinivasan [PS95] and it has time complexity $2^{O(\sqrt{\log n})}$.

# 3 Basic Derandomization of Local Algorithms

In the previous section, we have defined two classes of randomized distributed algorithms. Our derandomization technique only applies to Las Vegas algorithms; however, this is only a slight restriction, as most randomized graph problems (including, for instance, all locally-checkable ones), can be converted into Las Vegas algorithms. The formal statement of our basic derandomization result is given as follows:

**Theorem 3.1.** *Let $\mathcal{P}$ be a distributed graph problem $\mathcal{P}$ which has a $\mathsf{ZLOCAL}(r)$ algorithm $\mathcal{A}$. When running $\mathcal{A}$ on a graph $G = (V, E)$, for each $v \in V$ let $R_v$ be the private random bit string used by node $v$. Then there is a deterministic $\mathsf{SLOCAL}(2r)$-algorithm that assigns values to all $R_v$ such that when (deterministically) running $\mathcal{A}$ with those values, it solves $\mathcal{P}$.*

*Proof.* We construct the algorithm $\mathcal{A}'$ via the method of conditional expectations. Consider a randomized run of $\mathcal{A}$. Recall that in a Las Vegas algorithm $\mathcal{A}$, at the end, every node $v$ sets a flag $F_v$ such that if $F_v = 0$ for all $v \in V$, the solution is guaranteed to be correct. Let us further define the random variable $F := \sum_{v \in V} F_v$ such that $F > 0$ only if $\mathcal{A}$ fails to compute a solution for $\mathcal{P}$. By definition, we have $\mathbb{E}[F] = \sum_{v \in V} \mathbb{E}[F_v] < 1$.

We now show how to design a deterministic $\mathsf{SLOCAL}$-algorithm $\mathcal{A}'$ based on the randomized distributed algorithm $\mathcal{A}$. We number the $n$ nodes in an arbitrary way from $v_1, \ldots, v_n$ and assume that $\mathcal{A}'$ processes the nodes in this order. When processing node $v_i$, $\mathcal{A}'$ needs to fix the value of $R_{v_i}$. The goal of the algorithm is to pick values $\rho_i$ for all $i \in \{1, \ldots, n\}$ such that for all $i \in \{0, \ldots, n\}$, it holds that

$$\mathbb{E}\left[F \,\middle|\, \{R_{v_1} = \rho_1, \ldots, R_{v_i} = \rho_i\}\right] \leq \mathbb{E}[F] < 1. \tag{1}$$

After processing all $n$ nodes, all values $R_{v_i}$ are set to a fixed value and for $i = n$, the conditional expectation in Equation (1) is equal to the number of failed local conditions when running $\mathcal{A}$ with these values for $R_{v_i}$. Because $\mathbb{E}[F] < 1$, Equation (1) thus implies that $\mathbb{E}[F \,|\, R_{v_1} = \rho_1, \ldots, R_{v_n} = \rho_n] = 0$ and thus the algorithm succeeds in solving $\mathcal{P}$. It remains to show how the values $\rho_i$ can be found such that Equation (1) is satisfied for all $i \in \{0, \ldots, n\}$ and such that when processing node $v_i$, the $\mathsf{SLOCAL}$-algorithm $\mathcal{A}'$ only needs to query the $2r$-neighborhood of $v_i$.

If the values of $\rho_1, \ldots, \rho_{i-1}$ are already given such that $\mathbb{E}[F \,|\, R_{v_1} = \rho_1, \ldots, R_{v_{i-1}} = \rho_{i-1}] \leq \mathbb{E}[F]$, Equation (1) can clearly be satisfied by choosing $\rho_i$ to minimize $\mathbb{E}[F \,|\, R_{v_1} = \rho_1, \ldots, R_{v_i} = \rho_i]$. Further note that for each node $v$, the output of $v$ in the distributed algorithm only depends on the initial state of the $r$-hop neighborhood of $v$ and thus also the value of $F_v$ only depends on $R_u$ for nodes $u$ within distance $r$ from $v$. Because the values of $R_u$ are mutually independent, for each node $v$ and all $S \subset V$, we therefore have

$$\mathbb{E}\left[F_v \,\middle|\, \bigwedge_{u \in S} (R_u = \rho_u)\right] = \mathbb{E}\left[F_v \,\middle|\, \bigwedge_{u \in S \,:\, d_G(u,v) \leq r} (R_u = \rho_u)\right]. \tag{2}$$

We can therefore fix the value of $\rho_i$ as follows.

$$\rho_i \;=\; \arg\min_{\rho} \mathbb{E}\left[F \,\middle|\, R_{v_1} = \rho_1, \ldots, R_{v_{i-1}} = \rho_{i-1}, R_{v_i} = \rho\right]$$

$$= \arg\min_{\rho} \sum_{v \in V} \mathbb{E}\left[F_v \,\middle|\, R_{v_1} = \rho_1, \ldots, R_{v_{i-1}} = \rho_{i-1}, R_{v_i} = \rho\right]$$

$$= \arg\min_{r} \sum_{v\,:\,d_G(v_i,v)\leq r} \mathbb{E}\left[F_v \,\middle|\, R_i = \rho \wedge \bigwedge_{j<i\,:\,d_G(v,v_j)\leq r} (R_{v_j} = \rho_j)\right]$$

The last equation follows because for $v$ at distance more than $r$ from $v_i$, by Equation (2), $\mathbb{E}\left[F_v \,\middle|\, R_{v_1} = \rho_1, \ldots, R_{v_{i-1}} = \rho_{i-1}, R_{v_i} = \rho\right]$ does not depend on the value of $\rho$. Thus, when determining the value of $\rho_i$, $\mathcal{A}'$ needs to evaluate conditional expectations of $F_v$ for all $v$ within distance at most $r$ from $v_i$. In order to do this, it is sufficient to read the current state of the $(2r)$-neighborhood of $v_i$. □

The key tool to turn an SLOCAL algorithm back into a distributed algorithm is the computation of network decompositions (cf. Definition 2.1). The formal statement of how to use network decompositions is given by the following proposition, which was implicitly proven in [GKM17].

**Proposition 3.2.** *Suppose we have an* $\mathsf{SLOCAL}(r(n))$ *algorithm* $\mathcal{A}$ *for a graph problem* $\mathcal{P}$ *and a* $(d(n), c(n))$*-network decomposition of* $G^{r(n)}$*. Then* $\mathcal{P}$ *can be solved in* $O\big((d(n)+1)c(n)r(n)\big)$ *rounds in the* LOCAL *model. If the* SLOCAL *algorithm* $\mathcal{A}$ *is deterministic, the resulting* LOCAL *algorithm is also deterministic.*

*Proof.* We use the network decomposition of $G^{r(n)}$ to run the SLOCAL algorithm $\mathcal{A}$ in the distributed setting. We run $\mathcal{A}$ by processing the nodes according to the increasing lexicographic order given by the cluster color of a node and the ID of the node. A network decomposition of $G^{r(n)}$ guarantees that nodes of different clusters of the same color are at least $r(n)+1$ hops away from each other. Hence, when processing the nodes of a given color, different cluster cannot interfere with each other and we can locally simulate the execution of $\mathcal{A}$ in each cluster. In the LOCAL model, this local simulation can be done in $r(n) \cdot (d(n)+1)$ rounds for each cluster. The claim of the proposition then follows because we have to iterate over the $c(n)$ cluster colors. □

We next show how we can convert back and forth between ZLOCAL, SLOCAL, and LOCAL algorithms.

**Proposition 3.3.** *We have*

*(1) Any* $\mathsf{SLOCAL}(r)$ *algorithm* $\mathcal{A}$ *to solve a graph problem* $\mathcal{P}$ *on* $G$ *can be executed deterministically in* $\min\left\{r \cdot 2^{O(\sqrt{\log n})}, O\big(r \cdot (\Delta(G^r) + \log^* n)\big)\right\}$ *rounds in the* LOCAL *model.*

*(2) Any* $\mathsf{ZLOCAL}(r)$ *algorithm* $\mathcal{A}$ *to solve a graph problem* $\mathcal{P}$ *on* $G$ *can be executed determinstically in* $\min\left\{r \cdot 2^{O(\sqrt{\log n})}, O\big(r \cdot (\Delta(G^{2r}) + \log^* n)\big)\right\}$ *rounds in the* LOCAL *model.*

*Proof.* We first show part (1). To get the first bound, we can compute an $\big(O(\log n), O(\log n)\big)$-decomposition of $G^r$ in time $r \cdot 2^{O(\sqrt{\log n})}$ by using the algorithm of [PS95]. By Proposition 3.2 we can use this to simulate $\mathcal{A}$ in $O(r \log^2 n) \leq r2^{O(\sqrt{\log n})}$ rounds. For the second bound, we compute a $\Delta(G^r)$-coloring of $G^r$ in time $O\big(r(\Delta(G^r) + \log^* n)\big)$ by using the algorithm of [BEK15]. This can be viewed as as $(0, \Delta(G^r))$-decomposition of $G^r$. By Proposition 3.2 we can use this to simulate $\mathcal{A}$ in $O(r\Delta(G^r))$ rounds.

For part (2), Theorem 3.1 gives a $\mathsf{SLOCAL}(2r)$ algorithm to determine a set of random bits that make $\mathcal{A}$ succeed; we can turn this into a deterministic LOCAL algorithm by using part (1). We can then simulate $\mathcal{A}$ (which is at this point a deterministic algorithm) in $O(r)$ additional rounds. □

The basic derandomization technique in Theorem 3.1 only shows how to get an SLOCAL algorithm to determine the private randomness $R_v$ of the distributed ZLOCAL algorithm for every node $v$. We can extend this to get a deterministic SLOCAL algorithm for the original graph problem.

**Proposition 3.4.** *We have the inclusion* $\mathsf{ZLOCAL}(r) \subseteq \mathsf{SLOCAL}(4r)$.

*Proof.* Let $\mathcal{P}$ be a graph problem in class $\mathsf{ZLOCAL}(r)$, with a corresopnding algorithm $\mathcal{A}$. Theorem 3.1 shows that we can determine a setting for the random bits which causes $\mathcal{A}$ to succeed. Once these random bits are determined, we can clearly execute $\mathcal{A}$ as a deterministic (and hence $\mathsf{SLOCAL}$) algorithm with locality $r$. Thus, we can solve $\mathcal{P}$ by composing two $\mathsf{SLOCAL}$ algorithms, with localities $r_1 = 2r$ and $r_2 = r$, respectively. As shown in Lemma 2.3 of [GKM17], this composition can be realized as a *single* $\mathsf{SLOCAL}$ algorithm with locality $r' = r_1 + 2r_2 = 4r$.[4] □

Using the same basic techiques, we can also prove Theorem 1.10.

**Proof of Theorem 1.10.** Assume that we are given an $n$-node graph $G$ and a randomized $\mathsf{SLOCAL}$ algorithm $\mathcal{A}$ with locality $r(n)$. Using the randomized network decomposition algorithm of [LS93], we can compute an $\big(O(\log n), O(\log n)\big)$-decomposition of $G^{r(n)}$ in $O(r(n)\log^2 n)$ rounds. Using Proposition 3.2, the randomized $\mathsf{SLOCAL}$ algorithm $\mathcal{A}$ can therefore be transformed into a randomized distributed algorithm with time complexity $O(r(n)\log^2 n)$. Because the graph problem that is solved by $\mathcal{A}$ is assumed to be locally checkable, we can then use Proposition 3.4 to convert this randomized distributed algorithm into a deterministic $\mathsf{SLOCAL}$ algorithm with locality $O(r\log^2 n)$. □

### 3.1 Alternative Definition of Las Vegas Algorithms

The definition of Las Vegas algorithms is somewhat non-standard, and does not match up in every way with the definitions of Las Vegas algorithms one encounters in, e.g., complexity theory. We next relate our definition of Las Vegas algorithms to an alternate, more standard, definition.

**Definition 3.5** (Zero-Error Las-Vegas Algorithm)**.** *We define an $r$-round* zero-error distributed Las Vegas algorithm *$\mathcal{A}$ to be a randomized $\mathsf{LOCAL}$ algorithm for a given graph problem $\mathcal{P}$ such that a) when all nodes terminate, $\mathcal{A}$ always computes a correct solution for $\mathcal{P}$, b) with probability 1, all nodes terminate eventually, and c) with probability strictly larger than $1 - 1/n$, all nodes terminate after at most $r$ rounds.*

**Proposition 3.6.** *Any $r$-round zero-error distributed Las Vegas algorithm can be turned into a $\mathsf{ZLOCAL}(r)$ algorithm. Further, any $\mathsf{SLOCAL}(r)$ algorithm can be turned into a $O(r\log^2 n)$-rounds zero-error distributed Las Vegas algorithm.*

*Proof.* For the first part, any $r$-round zero-error Las Vegas algorithm $\mathcal{A}$ can directly be turned into a $\mathsf{ZLOCAL}(r)$-algorithm as follows: We stop the computation of $\mathcal{A}$ after $r$ rounds. Every node $v$ that has terminated by that time sets $F_v := 0$, all other nodes set $F_v := 1$.

For the second part, we use the randomized, distributed network decomposition algorithm of [LS93]. The algorithm can be used as an $O(\log^2 n)$-round zero-error distributed Las Vegas algorithm to compute an $\big(O(\log n), O(\log n)\big)$-decomposition of a graph $G$.

We construct our zero-error algorithm in phases of $O(\log n)$ rounds. In phase $i$, we use the network algorithm to construct the cluster of color $i$, and, in the same manner as Proposition 3.2, we execute the $\mathsf{SLOCAL}$ algorithm on the color-$i$ vertices. □

Proposition 3.6 shows that the two definitions of Las Vegas algorithms are equivalent up to polylogarithmic factors. It is somewhat remarkable that we are able to convert an algorithm which succeeds with merely positive probability into one that terminates with high probability and succeeds with probability one.

[4] Lemma 2.3 of the conference paper [GKM17] only claims a locality of $r' = 2(r_1 + r_2)$. The tight bound and a full proof appear as Lemma 2.2 in the full version of [GKM17], which is available at https://arxiv.org/abs/1611.02663.

# 4 Improved Deterministic Hypergraph Maximal Matching

We consider a hypergraph $H$ on $n$ nodes, maximum degree at most $\Delta$, and rank at most $r$. Although the LOCAL model is defined for graphs, there is a very similar model for hypergraphs: in a single communication round, each node $u$ can send a message to each node $v$ for which $u$ and $v$ are contained in a common hyperedge. The objective of the section is to compute a maximal matching of $H$, that is, a maximal set of pairwise disjoint hyperedges.

Our construction is based on a method of partitioning the hyperedges of a hypergraph $H$ into two classes, so hyperedges of each node are roughly split into two equal parts, which we refer to as *hypergraph degree splitting*. This degree splitting procedure uses the derandomization lemma Theorem 3.1 as its core tool.

**Definition 4.1** (Hypergraph Degree Splitting)**.** *Let $H = (V, E)$ be a hypergraph and let $\delta \geq 1$ and $\varepsilon > 0$ be two parameters. A $(\varepsilon, \delta)$-degree splitting of $H$ is a coloring of the hyperedges with two colors red and blue such that for each node $v \in V$ of degree $\deg_H(v) \geq \delta$, at least $\frac{1-\varepsilon}{2} \cdot \deg_H(u)$ of the hyperedges of $u$ are colored red and at least $\frac{1-\varepsilon}{2} \cdot \deg_H(u)$ of the hyperedges of $u$ are colored blue.*

**Lemma 4.2.** *Let $H$ be an $n$-node hypergraph with maximum degree at most $\Delta$ and rank at most $r$ and let $\varepsilon > 0$ be a parameter. Then, for every $\varepsilon > 0$, there is a deterministic $O(r\varepsilon^{-2}\log(n\Delta))$-round algorithm in the* LOCAL *model that computes a $\left(\varepsilon, \frac{8\ln(n\Delta)}{\varepsilon^2}\right)$-degree splitting of $H$.*

*Proof.* For convenience, we define $\delta := \frac{8}{\varepsilon^2} \cdot \ln(n\Delta)$ and we assume that $n \geq n_0$ for a sufficiently large constant $n_0 \geq 1$ (for constant $n$, the statement of the lemma is trivial). As a first step, we reduce the problem on $H$ to the hypergraph splitting problem on a low-degree hypergraph $H'$. To construct the graph $H'$, we divide each node of $H$ or degree $\geq 2\delta$ into virtual nodes, each of degree $\Theta(\delta)$. More specifically, for each node $u \in V$, we replace $u$ by $\ell_u := \max\{1, \lfloor \deg_H(u)/\delta \rfloor\}$ virtual nodes $u_1, \ldots, u_{\ell_u}$ and we assign each of the hyperedges of $u$ to exactly one of the virtual nodes $u_1, \ldots, u_{\ell_u}$. If $\ell_u > 1$, we divide the hyperedges that each virtual node $u_i$ has degree at least $\delta$ and less than $2\delta$. The graph $H'$ then is a hypergraph with maximum degree less than $2\delta$ and computing $(\varepsilon, \delta)$-degree splitting of $H'$ immediately implies a $(\varepsilon, \delta)$-degree splitting of $H$.

We thus need to show how to efficiently compute an $(\varepsilon, \delta)$-degree splitting of the low-degree hypergraph $H'$. Instead of directly working on $H'$, it is more convenient to define the algorithm on the following bipartite graph $B = (U_B \cup V_B, E_B)$. The graph $B$ has one node in $U_B$ for every node $u$ of $H'$ and it has one node in $V_B$ for every hyperedge of $H'$. A node $u \in U_B$ and a node $v \in V_B$ are connected by an edge if and only if the node of $H'$ corresponding to $u$ is contained in the hyperedge of $H'$ corresponding to $v$. Clearly, any $r$-round computation on $H'$ can be simulated in $B$ in at most $2r$ rounds. An $(\varepsilon, \delta)$-splitting of $H'$ now corresponds to a red/blue-coloring of the nodes in $V_B$ such that every node of degree $d \geq \delta$ in $U_B$ has at least $(1-\varepsilon)\frac{d}{2}$ red and at least $(1-\varepsilon)\frac{d}{2}$ blue neighbors in $V_B$.

We first claim that such a red/blue coloring of $B$, and thus an $(\varepsilon, \delta)$-degree splitting of $H'$, can be computed by a trivial Las Vegas algorithm. Each node in $V_B$ colors itself red or blue independently with probability $1/2$. For a node $u \in U_B$, let $X_u$ and $Y_u$ be the number red and blue neighbors in $V_B$ after this random coloring step. If the degree of $u$ is less than $\delta$, the coloring does not need to satisfy any condition. Otherwise, we know that the degree of $u$ is in $[\delta, 2\delta)$. The probability that a node $u$ gets too few red hyperedges can therefore be upper bounded by a standard Chernoff bound as follows:

$$\mathbb{P}\left(X_u < (1-\varepsilon) \cdot \frac{\deg_{H'}(u)}{2}\right) \leq e^{-\varepsilon^2 \deg_B(u)/4} \leq e^{-\varepsilon^2\delta/4} = \frac{1}{(n\Delta)^2}$$

The corresponding bound for $Y_u$ is obtained in the same way. In order to show that the algorithm computes a red/blue coloring with desired properties w.h.p., we first need to bound the number of nodes $N$ of the bipartite graph $B$. If $\Delta < 2\delta$, the hypergraphs $H$ and $H'$ are identical and the number of nodes in $U_B$ is

equal to the $n$, the number of nodes of the hypergraph $H$. Otherwise, we have $|U_B| \leq n\Delta/\delta \leq n\Delta/2$. The number of nodes in $V_B$ is equal to the number of hyperedges of $H$ and it can therefore also be upper bounded by $|V_B| \leq n\Delta/2$. Overall, the number of nodes of $B$ is thus at most $N \leq n\Delta$. By a union bound, the probability that some node $U_B$ has too few red or blue neighbors in $V_B$ is at most

$$\mathbb{P}\left(\bigvee_{u\in U_B} \max\{X_u, Y_u\} < (1-\varepsilon)\cdot\frac{\deg_{H'}(u)}{2}\right) < 2\cdot|U_B|\cdot\frac{1}{(n\Delta)^2} \leq 2\cdot\frac{n\Delta}{2}\cdot\frac{1}{(n\Delta)^2} \leq \frac{1}{N}.$$

This procedure can be computed in 0 rounds (without communicating), and the correctness can be verified in 1 round. So it is a $\mathsf{ZLOCAL}(1)$ algorithm. Since $B^2$ has maximum degree $O(r\log(n\Delta)/\varepsilon^2)$, Proposition 3.3 shows that it can be executed as a deterministic algorithm in $O(r\Delta(B^2) + r\log^* n) \leq O(r\log(n\Delta)/\epsilon^2)$ rounds. This concludes the proof. □

**Lemma 4.3.** *Let $H = (V, E)$ be a hypergraph of maximum degree $\Delta$, and let $U \subseteq V$ be a set of vertices such that $d(u) \geq \delta$ for every $u \in U$. Then there is a deterministic distributed algorithm, running in $O(r\Delta\log r + \log r\log^* n)$ rounds, to find a matching $M \subseteq E$ such that*

$$\sum_{e\in M}|e\cap U| \geq \Omega\left(\frac{|U|\delta}{r\Delta}\right).$$

*Proof.* Let $k = \lceil\log_2(r+1)\rceil$ and for $i = 0,\dots,k$ let $E_i \subseteq E$ denote the set of edges $e$ such that $2^i \leq |e\cap U| < 2^{i+1}$. We will sequentially construct matchings $M_k,\dots,M_0$, as follows. At stage $i$, we define $E_i'$ to be the set of edges $e \in E_i$ which do not intersect with any edge $f \in M_{i+1}\cup\cdots\cup M_k$; we find a maximal matching $M_i$ of the hypergraph $H_i = (V_i, E_i')$. We then finish by outputting $M = M_0 \cup M_1 \cup \cdots \cup M_k$.

This method of construction ensures that, for any $i = 0,\dots,k$, the matching $M_i \cup M_{i+1} \cup \cdots \cup M_k$ is a maximal matching of the hypergraph $(V, E_i \cup E_{i+1} \cup \cdots \cup E_k)$. Since the line graph of $H$ has maximum degree $s = r\Delta$, this shows that

$$|M_i| + \cdots + |M_k| \geq \frac{|E_i| + \cdots + |E_k|}{s}.$$

For any $u \in U$, let $d_i(u)$ denote the number of edges $e \in E_i$ with $u \in e$. By double-counting, we have

$$|E_i| \geq \frac{\sum_{u\in U} d_i(u)}{2^{i+1}}$$

and $\sum_i d_i(u) \geq \delta$ for every $u \in U$.

We now compute:

$$\begin{aligned}
\sum_{e\in M}|e\cap U| &= \sum_{i=0}^{k}\sum_{e\in M_i}|e\cap U| \geq \sum_{i=0}^{k}2^i|M_i| = \sum_{i=0}^{k}2^{i-1}\sum_{j=i}^{k}|M_i| \\
&\geq \sum_{i=0}^{k}2^{i-1}/s\sum_{j=i}^{k}|E_j| \\
&= \sum_{j=0}^{k}2^j|E_j|/s = \sum_{j=0}^{k}2^j\frac{\sum_{u\in U}d_j(u)}{2^{j+1}s} \\
&= \sum_{j=0}^{k}\frac{\sum_{u\in U}d_j(u)}{2s} \geq \frac{|U|\delta}{2s}
\end{aligned}$$

This procedure goes through $O(\log r)$ stages. In each stage $i$, we compute a maximal matching of the hypergraph $H_i$, whose line graph has maximum degree $s$; so this can be achieved in $O(s + \log^*(n\Delta))$ time per stage. (We note that $\Delta \leq 2^n$, so $\log^* \Delta \leq O(\log^* n)$). □

**Lemma 4.4.** *Given a rank-$r$ hypergraph $H = (V, F)$ of degree $\Delta = \Omega(\log n)$, there is a deterministic $O(r \log r \log n + r \log(n\Delta) \log^3 \Delta)$-round* LOCAL *algorithm to find a matching $M$ of $H$ with the following property: if $U_+$ denotes the set of vertices of $H$ of degree at least $\Delta/2$, then the edges of $M$ contain at least an $\Omega(1/r)$ fraction of the vertices of $U_+$.*

*Proof.* The problem is trivial when $n \leq O(1)$ so we assume without loss of generality that $n \geq n_0$ for a sufficiently large constant.

Let $H_+$ be the sub-hypergraph of $H$ induced by all the hyperedges that have at least one node in $U_+$. We reduce the degree of $H_+$ by repeatedly applying the hypergraph degree splitting of Lemma 4.2. Let $F_0 \subseteq F$ be the set of edges of $H$ which contain at least one node in $U_+$.

Let $F_0 \subseteq F$ be the set of edges of $H$ which contain at least one node in $U_+$. For an integer $t \geq 1$, we will define parameters $\varepsilon_1, \ldots, \varepsilon_t > 0$ and $\delta_1, \ldots, \delta_t \geq 1$, and construct edge sets $F_1, \ldots, F_t$ such that $F_i \subseteq F_{i-1}$ as follows. For each $i \in \{1, \ldots, t\}$, we use Lemma 4.2 to compute a $(\varepsilon_i, \delta_i)$-splitting of the hypergraph $(V, F_{i-1})$; we define $F_i$ to be the resulting hyperedges that are colored red.

We choose the parameters as follows:

$$\begin{aligned} t &:= \max\{1, \lfloor \log\Delta - \log\log n - 14 \rfloor\} \\ \delta_i &:= \frac{\Delta}{2^{i+1}} \\ \varepsilon_i &:= \max\left\{ \frac{1}{4\log\Delta}, \sqrt{\frac{16\ln(n\Delta/2^{i-1})}{\Delta/2^i}} \right\} \end{aligned}$$

Let us first observe that $\sum_{j=1}^{t} \varepsilon_j \leq 1/2$. Assuming $n \geq n_0$ for a sufficiently large constant $n_0$, we have

$$\begin{aligned} \sum_{i=1}^{t} \varepsilon_i &\leq \sum_{i=1}^{t} \frac{1}{4\log\Delta} + \sum_{i=1}^{t} \sqrt{\frac{16\ln(n\Delta/2^{i-1})}{\Delta/2^i}} \\ &\overset{(t\leq\log\Delta)}{\leq} \frac{1}{4} + \sum_{j=\log\Delta - t}^{\infty} \sqrt{\frac{16\big(\ln n + (j+1)\ln 2\big)}{2^j}} \\ &\overset{(n\geq n_0)}{\leq} \frac{1}{4} + \sum_{s=14}^{\infty} \sqrt{\frac{32\ln n + s}{2^s \cdot \ln n}} \\ &\overset{(n\geq n_0)}{\leq} \frac{1}{4} + \sum_{s=14}^{\infty} \sqrt{\frac{32}{1.8^s}} < \frac{1}{2}. \end{aligned}$$

Let $H_i = (V, F_i)$ and $\Delta_i = \Delta(H_i)$. We show by induction that $\Delta_i \leq (1 + 2\sum_{j=1}^{i} \varepsilon_j)\Delta/2^i \leq \Delta/2^{i-1}$, and any node in $U_+$ has degree at least $(1 - \sum_{j=1}^{i} \varepsilon_j)\Delta/2^{i+1} \geq \Delta/2^{i+2}$ in $H_i$. For $i = 0$, these bounds clearly hold because by definition, each node of $H_+$ has degree at most $\Delta$ and each node in $U_+$ has degree at least $\Delta/2$ in $H_+ = H_0$.

For the induction step, we first show that for each $i$, we can apply the degree splitting algorithm of Lemma 4.2 to compite a $(\varepsilon_i, \delta_i)$-degree splitting of $H_{i-1}$; specifically, we need to show that $\delta_i \geq 8\ln(n\Delta_{i-1})/\varepsilon_i^2$. By induction hypothesis, we have

$$\frac{8\ln(n\Delta_{i-1})}{\varepsilon_i^2} \leq \frac{8\ln(n\Delta/2^{i-1})}{\frac{16\ln(n\Delta/2^{i-1})}{\Delta/2^i}} = \frac{\Delta}{2^{i+1}} = \delta_i.$$

By induction hypothesis, each node in $U_+$ has degree at least $\delta_i$ in $H_{i-1}$. The minimum degree of any node of $U_+$ in $H_i$ is therefore at $(1-\varepsilon_i)/2$ times the the minimum degree of any node of $U_+$ in $H_{i-1}$. Thus, the minimum degree of any node of $U_+$ in $H_i$ is at least

$$\left(1-\sum_{j=1}^{i-1}\varepsilon_j\right)\frac{\Delta}{2^i}\cdot\frac{1-\varepsilon_i}{2}=\left(1-\sum_{j=1}^{i-1}\varepsilon_j\right)(1-\varepsilon_i)\cdot\frac{\Delta}{2^{i+1}}\geq\left(1-\sum_{j=1}^{i}\varepsilon_j\right)\cdot\frac{\Delta}{2^{i+1}}\geq\frac{\Delta}{2^{i+2}}.$$

Finally, note that maximum degree of $H_i$ is at most $(1+\varepsilon_i)/2$ times the maximum degree of $H_{i-1}$. So

$$\Delta_i\leq\left(1+2\sum_{j=1}^{i-1}\varepsilon_j\right)(1+\varepsilon_i)\cdot\frac{\Delta}{2^i}\leq\left(1+2\sum_{j=1}^{i}\varepsilon_j\right)\cdot\frac{\Delta}{2^i}\leq\frac{\Delta}{2^{i-1}}.$$

where here we make use of the inequality $\sum_{j=1}^{i}\varepsilon_j\leq\sum_{j=1}^{t}\varepsilon_j\leq 1/2$.

We finish by applying Lemma 4.3 on the hypergraph $H_t$ and vertex set $U_+$. Note that $H_t$ has maximum degree $O(\log n)$, and every node of $U_+$ has degree $\Theta(\log n)$. So, Lemma 4.3 runs in time $O(r\log n\log r)$. The resulting matching $M$ contains at least $\Omega(|U_+|/r)$ vertices of $U_+$. Since $t=O(\log\Delta)$ and $\epsilon_i\geq\frac{1}{4}\log\Delta$, each application of Lemma 4.2 takes time $O(r\log(n\Delta)\log^2\Delta)$. □

By applying the hypergraph degree splitting of Lemma 4.4 repeatedly, we can now prove Theorem 1.2.

**Theorem 1.2.** *Let $H=(U,F)$ be an $n$-node hypergraph with maximum degree at most $\Delta$ and rank at most $r$. Then, a maximal matching of $H$ can be computed in $O(r^2\log(n\Delta)\log n\log^4\Delta)$ rounds in the* LOCAL *model.*

*Proof.* If $\Delta\leq r\log^2 n$, then simply note that the line graph of $H$ has degree at most $r\Delta$, and so the deterministic MIS algorithm of [BEK15] runs in time $O(r\Delta+\log^*(n\Delta))\leq O(r^2\log^2 n)$, and we are done. So let us assume that $\Delta\geq r\log^2 n$.

Each time we apply Lemma 4.4, we reduce the number of vertices in $U_+$ by a factor of $\Omega(1/r)$. Thus, after $O(r\log n)$ applications, we reduce the degree by a factor of $1/2$. So after $\log\Delta$ applications, the degree is reduced to $O(\log n)$.

At this stage, we note that the residual line graph of $H$ has degree at most $O(r\log n)$. We can thus use a deterministic MIS algorithm on it, in time $O(r\log n+\log^*(n\Delta))$.

Each application of Lemma 4.4 uses $O(r\log r\log n+r\log(n\Delta)\log^3\Delta)$ time. Our assumption that $\Delta\geq r\log^2 n$ ensures that the first term is at most $O(r\log\Delta\log n)$, which is dominated by the second term. □

### 4.1 Implications on Edge-Coloring, Maximum Matching, and Low-Degree Orientation

*Proof of Corollary 1.3.* The proof follows immediately from Theorem 1.2, combined with a reduction of Fischer, Ghaffari, and Kuhn [FGK17] that reduces $(2\Delta-1)$-list-edge-coloring to hypergraph maximal matching on hypergraphs of rank 3, with $O(m+n)$ vertices—where $m$ denotes the number of the edges of the graph—and maximum degree $O(\Delta^2)$, The complexity is thus $O(\log^2 n\log^4\Delta)$. □

*Proof of Corollary 1.4.* The algorithms follows the approach of Hopcroft and Karp [HK73]. Augmenting the matching $M$ with $P$ means replacing the matching edges in $P\cap M$ with the edges $P\setminus M$. For each $\ell=1$ to $2(1/\varepsilon)-1$, we find a maximal set of vertex-disjoint augmenting paths of length $\ell$, and we augment them all. Given a matching $M$, an augmenting path $P$ with respect to $M$ is a path that starts with an unmatched vertex, alternates between non-matching and matching edges, and finally ends in an unmatched vertex. Hopcroft and Karp [HK73] show that this produces a $(1+\varepsilon)$-approximation of maximum matching.

See also [LPSP15], where a similar method is used obtain a $O(\log n/\varepsilon^3)$-round randomized distributed algorithm for $(1+\varepsilon)$-approximation of maximum matching, by applying Luby's Algorithm [Lub86].

We now discuss how to compute a maximal set of vertex-disjoint augmenting paths of a given length $\ell \leq 2(1/\varepsilon) - 1$, by formulating it as a hypergraph maximal matching for a hypergraph of rank at most $1/\varepsilon + 1$: we create a hypergraph $H$ by including one vertex for each unmatched node and also one vertex for each matching edge. Then, each augmenting path is simply a hyperedge made of its elements, i.e., its unmatched vertices and its matching edges. This hypergraph has rank at most $1/\varepsilon + 1$, maximum degree at most $\Delta^{2(1/\varepsilon)}$, and the number of its vertices is no more than $n$. Moreover, a single round of communication on this hypergraph can be simulated in $O(1/\varepsilon)$ rounds of the base graph, simply because each hyperedge spans a path of length at most $O(1/\varepsilon)$. Hence, placing these parameters in the bound of Theorem 1.2, we get complexity $O(1/\varepsilon \cdot (1/\varepsilon)^2 \log(n\Delta^{2/\varepsilon}) \log n \log^4(\Delta^{2/\varepsilon})) = O(\log^2 n \log^5 \Delta/\varepsilon^8)$ rounds for each one value of $\ell \in [1, 2/\varepsilon - 1]$. Thus, the overall complexity is $O(\log^2 n \log^5 \Delta/\varepsilon^9)$. □

*Proof of Corollary 1.5.* We follow the approach of Ghaffari and Su [GS17], which iteratively improves the orientation by reducing its maximum out-degree via another special notion of augmenting paths. Let $D = \lceil \lambda(1+\varepsilon) \rceil$. Given an arbitrary orientation, Ghaffari and Su call a path $P$ an augmenting path for this orientation if $P$ is a directed path that starts in a node with out-degree at least $D+1$ and ends in a node with out-degree at most $D-1$. Augmenting this path means reversing the direction of all of its edges. This would improve the orientation by decreasing the out-degree of one of the nodes whose out-degree is above the budget $D$, without creating a new node with out-degree above the threshold.

Let $G_0$ be the graph with our initial arbitrary orientation. Define $G'_0$ to be a directed graph obtained by adding a source node $s$ and a sink node $t$ to $G_0$. Then, we add $outdeg_{G_0}(u) - D$ edges from $s$ to every node $u$ with outdegree at least $D+1$, and $D - outdeg_{G_0}(u)$ edges from every node $u$ with outdegree at most $D-1$ to $t$. We improve the orientation gradually in $\ell = O(\log n/\varepsilon)$ iterations. In the $i^{th}$ iteration, we find a maximal set of edge-disjoint augmenting paths of length $3+i$ from $s$ to $t$ in $G'_i$, and then we reverse all these augmenting paths. The resulting graph is called $G'_{i+1}$. Ghaffari and Su [GS17, Lemma D.6] showed that each time the length of the augmenting path increases by at least one, and at the end, no augmenting paths of length at most $\ell = O(\log n/\varepsilon)$ remains. They used this to prove that there must be no node of out-degree $D+1$ left, at the end of the process, as any such node would imply the existence of an augmenting path of length at most $\ell = O(\log n/\varepsilon)$ [GS17, Lemma D.9].

The only algorithmic piece that remains to be explained is computing a maximal set of edge-disjoint augmenting paths of length at most $3+i < \ell$, in a given orientation. We do using Theorem 1.2, by viewing each edge as one vertex of our hypergraph, and each augmenting path of length at most $3+i < \ell$ as one hyperedge. The round complexity is at most $O((\log n/\varepsilon) \cdot (\log n/\varepsilon)^2 \log(n\Delta^{O(\log n/\varepsilon)}) \log n \log^4(\Delta^{O(\log n/\varepsilon)})) = O(\log^9 n \log^5 \Delta/\varepsilon^8)$ , where the first term $(\log n/\varepsilon)$ factor is because simulating each hyperedge needs $\ell = O(\log n/\varepsilon)$ rounds. This is the complexity for each iteration. For $\ell = O(\log n/\varepsilon)$ iterations, the total complexity is $O(\log^{10} n \log^5 \Delta/\varepsilon^9)$. □

# 5 The Lovász Local Lemma

We will consider a somewhat restricted case of the LLL, which can be described as follows. The probability space $\Omega$ is defined by variables $X(1), \ldots, X(v)$; each $X(i)$ takes on values from some countable domain $D$. The variables $X(1), \ldots, X(v)$ are all mutually independent. Every bad event $B\mathcal{B}$ is a boolean function of a set of variables $S_B \subseteq [v]$. We say that a configuration $X$ *avoids* $\mathcal{B}$ if every $B \in \mathcal{B}$ is false on $X$.

We define a *dependency graph* $H$ for $\mathcal{B}$, to be an undirected graph on vertex set $\mathcal{B}$, with an edge $(A, B)$ if $S_A \cap S_B \neq \emptyset$; we write this as $A \sim B$. We will also define $B \sim B$ for every $B \in \mathcal{B}$. With this notation, we can state the LLL in its simplest "symmetric" form: if the dependency graph $H$ has maximum degree

$d-1$, and every bad-event $B \in \mathcal{B}$ has $P_\Omega(B) \leq p$, and $epd \leq 1$, then there is a positive probability that no $B \in \mathcal{B}$ occurs.

There are a number of combinatorial objects that can be constructed via the LLL. Thus, a distributed algorithm for the LLL is a key building-block for a number of distributed graph coloring algorithms, such as frugal or defective vertex-colorings. In such settings, we have a communication graph $G$ and a variable for every vertex $x \in G$, and have a bad-event $B_x$ indicating that the color fails for $x$ in some way; for example, in a frugal coloring, a bad-event for $x$ may be that $x$ has too many neighbors with a given color. Each vertex $x$ typically only uses information about its neighbor within a radius of $r$, where $r$ is very small (and typically is $O(1)$). In this case, the dependency graph $H$ is essentially the same as $G^r$. To avoid further confusion, we will assume that the communication graph is the same as the dependency graph; since communication rounds on $H$ may be simulated in $O(r)$ rounds of $G$, this typically only changes the overall runtime of our algorithms by a small (typically constant) factor. Thus, we will also assume that $|\mathcal{B}| = n$ and $d = \Delta + 1$.

## 5.1 Previous Distributed LLL Algorithms

The LLL, in its original form, only shows that there is an exponentially small probability of avoiding the events of $\mathcal{B}$, so it does not directly give efficient algorithms. There has been a long history of developing algorithmic versions of the LLL, including distributed algorithms. A breakthrough result of Moser & Tardos [MT10] gave one of the first general serial algorithms for the LLL; they also discussed a parallel variant, which can easily be converted into an distributed algorithm running in $O(\log^2 n)$ rounds. This algorithm converges under essentially the same conditions as the probabilistic LLL, namely it requires $epd(1+\epsilon) \leq 1$ for some constant $\epsilon > 0$.

In [CPS17], Chung, Pettie & Su began to investigate the algorithmic LLL specifically in the context of distributed computations. They give an algorithm running in $O((\log^2 d)(\log n))$ rounds (subsequently improved to $O((\log d)(\log n))$ by [Gha16]). They also discuss an alternate algorithm running in $O(\frac{\log n}{\log(epd^2)})$ rounds, under the stronger LLL criterion $epd^2 < 1$. They further showed that the criterion $epd^2 < 1$, although significantly weaker than the LLL itself, is still usable to construct a number of combinatorial objects. They used this procedure to construct objects such as frugal colorings and defective colorings in $O(\log n)$ time. We refer to this type of weakened LLL condition as a *polynomially-weakened LLL criterion* (pLLL). We will use this algorithm as a key subroutine; note in particular that if we satisfy the pLLL criterion $pd^3 < 1$, then this algorithm runs ins $O(\frac{\log n}{\log d})$ rounds.

More recently, Fischer & Ghaffari [FG17] have algorithm running in $2^{O(\sqrt{\log\log n})}$ rounds, under the pLLL criterion $p(ed)^{32} < 1$, as long as $d < (\log\log n)^{1/5}$. Although the general LLL algorithm of [FG17] has a significant limitation on degree, they nevertheless be used to construct a number of graph colorings including defective coloring, frugal coloring, and vertex coloring in $2^{O(\sqrt{\log\log n})}$ time (for arbitrary degree graphs).

On the other hand, [BFH+16] has shown a $\Omega(\log\log n)$ lower bound on the round complexity of distributed LLL, even under a pLLL criterion.

In addition to these general algorithms, there have been a number of algorithmic approaches to more specialized LLL instances. In [CPS17], Chang et al. have investigated the LLL when the graph $G$ is a tree; they develop an $O(\log\log n)$ round algorithm in that case. In [Har17], Harris developed an $O(\log^3 n)$-round algorithm for a form of the LLL known as the Lopsided Lovász Local Lemma (which applies to more general types of probability spaces). Finally, there have been a number of parallel PRAM algorithms developed for the LLL, including [MT10, HH17]; these are often similar to distributed LLL algorithms but are not directly comparable.

## 5.2 Graph Shattering

The LLL algorithms use a general technique for building distributed graph algorithms known as *graph shattering*. In the first phase, there is some random process. Most of the vertices $v \in G$ are satisfied in this random phase, and one fixes the choices these "good" vertices make. In the second phase, we let $R \subseteq V$ denote the unsatisfied vertices. These vertices are very sparse, and the connected components of $G[R]$ are relatively small. One can then use a deterministic algorithm to solve the residual problem on each component of $G[R]$.

We have seen a general method for converting randomized algorithms into deterministic ones. This has a particularly nice structure in the context of shattering. In Appendix A, we describe some general results on converting randomized algorithms into shattering-based algorithms. We summarize these results here:

**Theorem 5.1.** *Suppose each vertex survives to a residual graph $R$ with probability at most $(e\Delta)^{-4c}$, and this bound holds even for adversarial choices of the random bits outside the $c$-hop neighborhood of $v$ for some constant $c \geq 1$.*

*Suppose that the residual problem can be solved via a $\mathsf{SLOCAL}(r)$ procedure; then the residual problem can be solved w.h.p. in the $\mathsf{LOCAL}$ model in $O\big(r(\Delta(G^r) + \log^* n)\big)$ rounds. Furthermore, if $r \leq 2^{O(\sqrt{\log\log n})}$, then the residual problem can be solved w.h.p. in the $\mathsf{LOCAL}$ model in $2^{O(\sqrt{\log\log n})}$ rounds.*

**Proposition 5.2.** *Suppose each vertex survives to a residual graph $R$ with probability at most $(e\Delta)^{-4c}$, and this bound holds even for adversarial choices of the random bits outside the $c$-hop neighborhood of $v$ for some constant $c \geq 1$.*

*Suppose that there is a $\mathsf{ZLOCAL}(r)$ algorithm $\mathcal{A}$ on $G[X]$, for any vertex subset $X \subseteq V$ of size $|X| \leq N = O(\Delta^{2c} \log n)$.*

*Then the residual problem can be solved in $O(r\Delta(G^{2r}) + r \log^* n)$ rounds. Furthermore, if $r \leq 2^{O(\sqrt{\log\log n})}$, then the residual problem can be solved in $2^{O(\sqrt{\log\log n})}$ rounds.*

## 5.3 The LLL for Low-Degree Graphs

As we have discussed, the algorithm of [FG17] is very fast, running in just $2^{O(\sqrt{\log\log n})}$, but only works for graphs whose degree is very low. In this section, we will use one of the core subroutines of [FG17] to obtain an algorithm running in $2^{O(\sqrt{\log\log n})}$ rounds for much larger degree; specifically, we will allow the degree to become as large as $d = 2^{\sqrt{\log\log n}}$, an exponential improvement over [FG17].

In analyzing this and other LLL algorithms, it is convenient to extend the domain $D$ by adding an additional symbol denoted ?; we say $X(i) = ?$ to indicate that variable $X(i)$ is not determined, but will be later drawn from $D$ with its original sampling probability. We let $\overline{D} = D \cup \{?\}$. We always assume that $D$ is finite.

Given any vector $x \in \overline{D}^v$, and an event $E$ on the space $\Omega$, we define the *marginal probability* of $E$ with respect to $x$ to the probability that $E$ holds, if all variables $i$ with $X(i) = ?$ are resampled from the original distribution. Note that if $x \in D^v$ then any event has marginal probability with respect to $x$ of either zero or one. Also, if $x = (?, \ldots, ?)$, then the marginal probability of $E$ with respect to $x$ is simply $P_\Omega(E)$.

We summarize the main subroutine of [FG17] as Algorithm 1:

**Algorithm 1** Distributed LLL algorithm

Initialize $K \leftarrow \emptyset$; this will be the set of *frozen variables.*
Initialize $X = (?, \ldots, ?)$.
Compute a $d^2 + 1$-coloring $\chi$ of $G^2$.
**for** $i = 1, \ldots, d^2 + 1$ **do**:
    **for** each bad-event $B$ with $\chi(B) = i$ **do**
        **for** each $j \in S_x$ **do**
            **if** $j \notin K$ and $X(j) = ?$ **then**
                Draw $X(j)$ from its distribution under $\Omega$
                **if** any $A \sim B$ has marginal probability at least $(ed)^8 p$ under $X$ **then**
                    Update $K \leftarrow K \cup S_A$

Let $X' \in \overline{D}$ be the final value of vector $X$, and let $\mathcal{A} \subseteq \mathcal{B}$ be the set of all bad-evens whose marginal probability under $X'$ is non-zero. We use the following key facts about Algorithm 1:

**Theorem 5.3** ( [FG17]). *Suppose that the domain $D = \{0, 1\}$ and for every $i \in [v]$ we have $P_\Omega(X_i = 0) = P_\Omega(X_i = 1) = 1/2$. Then Algorithm 1 runs in $O(d^2 + \log^* n)$ rounds. At its termination:*

1. *Every $B \in \mathcal{B}$ has marginal probability under $X'$ of at most $2(ed)^8 p$.*
2. *For any $B \in \mathcal{B}$, we have $\mathbb{P}(B \in \mathcal{A}) \leq (ed)^{-8}$; furthermore, this probability bound holds even if the random bits made outside a two-hop radius of $B$ are chosen adversarially.*

*Proof.* Note that Steps (5) — (10) can be simulated in a single round of $G$, since all the bad-events $B$ with $\chi(B) = i$ do not interact during this step. Step (3) can be implemented via a deterministic algorithm in $\tilde{O}(d) + O(\log^* n)$ rounds [FHK16]. Thus, the overall run-time for this process is $O(d^2 + \log^* n)$. The probabilistic bounds are shown in [FG17]. □

Our shattering framework allows give a more straightforward overall LLL algorithm, without the need for the bootstrapping step.

**Theorem 5.4.** *Suppose $20000pd^8 \leq 1$ for some absolute constant $C$ and $d \leq 2^{O(\sqrt{\log \log n})}$. Then one can find a configuration avoiding $\mathcal{B}$ after $2^{O(\sqrt{\log \log n})}$ rounds w.h.p.*

*Proof.* First, note that it is possible to convert the LLL instance into a probability space $\Omega'$ over a domain $D' = \{0, 1\}$ and a larger variable set $v' > v$, with the probability distribution $\mathbb{P}_{\Omega'}(X_i = 1) = \mathbb{P}_{\Omega'}(X_i = 0) = 1/2$. By selecting $v'$ to be sufficiently large, we can ensure that $\mathbb{P}_{\Omega'}(B) \leq 2\mathbb{P}_\Omega(B)$ for every $B \in \mathcal{B}$.

We now run Algorithm 1 on the space $\Omega', D'$ with $p' = 2p$ to obtain a partial solution $X' \in \overline{D'}$. Now, consider the residual problem of converting the partial solution $X'$ into a full solution $X$; this can be viewed as an LLL instance on the bad-events $\mathcal{A}$.

By Theorem 5.3, each $B \in \mathcal{A}$ has marginal probability at most $2(ed)^8 p'$. If $ed \times (2e^8 d^8 p') \leq 1/2$, then $\mathcal{A}$ can be viewed as an LLL instance satisfying the symmetric LLL criterion with slack $\epsilon = 1/2$. The algorithm of [MT10] can thus be used as a Las Vegas procedure to this residual problem; on an $N$-node subgraph of $G$ it would require $O(\log^2 N)$ rounds. This condition $ed(4e^8 d^8 p) \leq 1/2$ is ensured by our condition $20000d^8 p \leq 1$.

By Theorem 5.3, each $B \in \mathcal{B}$ survives to the residual with probability $(ed)^{-8}$, and so apply Corollary 5.2 with $c = 2$. Also, in this case, we have $N = \Delta^8 \log n$, and so the algorithm of [MT10] requires $O(\log \log n + \log \Delta) \leq O(\log \log n)$ rounds to succeed with probability $1 - N^{-2}$. So the overall algorithm terminates w.h.p. after $2^{O(\sqrt{\log \log n})}$ rounds. □

We can obtain a faster LLL algorithm by using our LLL algorithms recursively, when $d$ is very small. The key idea is that, after applying Algorithm 1, we have a new LLL instance on graphs of size $N = \Delta^8 \log n$; when $\Delta$ is small this is approximately $\log n$. We can run the randomized LLL algorithm on each of these components.[5]

We will state our round complexity using Knuth's iterated-exponentiation notation, which is defined recursively

$$(2\uparrow)^0(x) = x$$
$$(2\uparrow)^{i+1}(x) = 2^{(2\uparrow)^i(x)}$$

Thus, for instance, $(2\uparrow)^1(x) = 2^x$, and $(2\uparrow)^2(x) = 2^{2^x}$, etc.

**Theorem 1.6.** *For any integer $i \geq 1$, there is an algorithm $\mathcal{A}_i$ with the following guarantee. If $(20000d^8)^i p \leq 1$ and $d \leq 2^{\sqrt{\log^{(i+1)} n}}$, then a configuration avoiding $\mathcal{B}$ in*

$$(2\uparrow)^i\left(c_i\sqrt{\log^{(i+1)} n}\right)$$

*rounds whp, where $c_i$ is a constants and $\log^{(j)}$ is the $j$-fold iteration logarithm.*

*Proof.* We show this by induction on $i$. When $i = 1$, we are requiring that $d \leq 2^{\sqrt{\log\log n}}$ and $p(20000d^8) \leq 1$, and we are trying to show a round complexity of $2^{O(\sqrt{\log\log n})}$; this has already been shown in Theorem 5.4.

We move on to the induction step. The algorithm $\mathcal{A}_i$ is defined by running Algorithm 1 and then using $\mathcal{A}_{i-1}$ on the residual. Note that the residual problem is an LLL instance, and so it is locally checkable with radius $O(1)$; in particular, $\mathcal{A}_{i-1}$ can be treated as a Las Vegas algorithm. So we apply Corollary 5.2 with $c = 2$; each $B \in \mathcal{B}$ survives to the residual with probability $(erd)^{-8}$. The resulting residual problem is an LLL instance, in which every remaining bad-event has marginal probability $q = 2(ed)^8 p' \leq 4e^8 d^8 p$ and which has a size of $N = \Delta^8 \log n$.

In order to apply algorithm $\mathcal{A}_{i-1}$, we require that $d \leq 2^{\sqrt{\log^{(i)} N}}$ and $(20000d^8)^{i-1} q \leq 1$. These conditions are satisfied when $d \leq 2^{\sqrt{\log^{(i+1)} n}}$ and $p(20000d^8)^i \leq 1$. Thus, we can apply $\mathcal{A}_{i-1}$ to the residual instances. On any component of size $N$, the residual algorithm $\mathcal{A}_{i-1}$ runs in time

$$r = (2\uparrow)^{i-1}\left(c_i\sqrt{\log^{(i)}(\Delta^8 \log n)}\right).$$

We note that $\Delta^8 \ll \log n$, and so $r \leq (2\uparrow)^{i-1}(c'\sqrt{\log^{(i+1)} n})$ for some constant $c'$. Including the running time of Algorithm 1, the overall run-time is

$$O(r\Delta^{2r} + r\log^* n) \leq O(\Delta^{3\times(2\uparrow)^{i-1}(c'\sqrt{\log^{(i+1)} n})} + \Delta^2 + (\log^* n)^2)$$

The terms $\Delta^2$ and $(\log^* n)^2$ are negligible. We may further simplify this expression as

$$\Delta^{3(2\uparrow)^{i-1}(c'\sqrt{\log^{(i+1)} n})} = 2^{3\log\Delta\times(2\uparrow)^{i-1}(c'\sqrt{\log^{(i+1)} n})} \leq 2^{\log^{(i+1)} n\times(2\uparrow)^{i-1}(c'\sqrt{\log^{(i+1)} n})}$$
$$= (2\uparrow)^i(O(\sqrt{\log^{(i+1)} n}))$$

□

[5] Note that the reader should not confuse this method of combinining LLL algorithms, by using one LLL algorithm to solve the residual problem, with the method of combining LLL equations by bootstrapping as used in [FG17]; in the latter, *the LLL algorithm itself* becomes the subject of the LLL.

# 6 The LLL for High-Degree Graphs

There are two main shortcomings with the distributed LLL algorithm of Section 5. First, it is only a pLLL criterion; second, it requires $\Delta \leq 2^{O(\sqrt{\log \log n})}$. Despite these shortcomings, one can use the algorithm of Section 5 to get distributed algorithms running in time $2^{O(\sqrt{\log \log n})}$, without any condition on $\Delta$, for numerous graph problems, such as defective vertex coloring. However, removing this degree restriction seems to require problem-specific and ad-hoc techniques, as in [FG17]; for example, in defective vertex coloring, certain vertices are re-colored using a secondary set of colors, instead of resampling a bad-event directly. This negates to a large extent one of the main advantages to developing general LLL algorithms.

In this section, we describe an alternative LLL algorithm, whose run-time does not depend on $\Delta$. We use this algorithm to provide distributed algorithms for problems such as defective and frugal vertex coloring, running in time $2^{O(\sqrt{\log \log n})}$. This matches the run-time of [FG17] for these problems; however, the algorithms can be described in a much more generic, high-level way. In addition, we provide algorithms for other LLL problems, such as $k$-SAT with bounded clause overlap, that do not appear directly possible in the framework of Theorem 5.4 or the algorithm of [FG17].

Our algorithm is quite similar to an LLL algorithm of [CPS17]; the key definition underlying the algorithms is that of a *dangerous* event.

**Definition 6.1** (Dangerous event)**.** *Let $B \in \mathcal{B}$ and $x \in D^v$. For any $U \subseteq N(B)$, define $y_U \in \overline{D}^v$ by*

$$y_U(i) = \begin{cases} ? & \text{if } i \in \bigcup_{A \in U} S_A \\ x(i) & \text{otherwise} \end{cases}$$

*We refer to $y_U$ as the* reversion of $x$ with respect to $U$*. We say that $B$ is $q$-*dangerous with respect to $x$*, if there is any $U \subseteq N(B)$ such that the marginal probability of $B$ with respect to $y_U$ is at least $q$.*

The key to analyzing our algorithm, as in [CPS17], will be to bound the probability than an event is $q$-dangerous. In [CPS17], the following bound was provided:

**Proposition 6.2** ( [CPS17])**.** *Any $B \in \mathcal{B}$ is $q$-dangerous with probability at most $2^d p/q$.*

Unfortunately, this bound is exponential in $d$, and so this typically leads to criteria which are more weaker than the LLL (i.e., bounds which are exponential in $d$ as opposed to polynomial in $d$.) We are not aware of any stronger bound, *as a function solely of $p$ and $d$*.

However, in many problem instances, we can use the specific form of the bad-events $\mathcal{B}$ to give much stronger bounds on the probability of being $q$-dangerous. Our LLL criterion will thus use more information than just $p$ and $d$. Nevertheless, the LLL criterion will be *local*, in the sense that it uses only information directly affecting any given bad-event (and not global information such as $n$). Furthermore, this bound can be often be computed fairly readily, and once it is computed no further information about $\mathcal{B}$ will be used.

We will define a statistic we refer to as *fragility* and use this to compute the probability of a bad-event becoming dangerous.

**Definition 6.3.** *Let $B$ be an event on variables $X(i), \ldots, X(v)$ and and $X_0, X_1 \in D^v$. For any vector $a \in \{0,1\}^v$, define a configuration $Z_a \in D^v$ by*

$$Z_a(i) = X_{a(i)}(i)$$

*Define the event $E$ by*

$$E = \bigvee_{a \in \{0,1\}^v} [B \text{ holds on configuration } Z_a]$$

*where $[\mathcal{P}]$ is the Iverson notation.*

*The* fragility of $B$ (denoted $f(B)$ *is defined to be the probability of $E$, when $X_0, X_1$ are drawn independently from $\Omega$.*

As an example of how to compute $f(E)$, let us consider the case of any event which depends on only a small number of coordinates.

**Definition 6.4** ($s$-witnessable event)**.** *Given a Boolean event $E$ on $X(1), \ldots, X(v)$, we say that $E$ is* $s$-witnessable*, if for any configuration of the variables $X \in D^v$ for which $E$ is true, there exists indices $i_1, \ldots, i_s$ with the following property: for any $X' \in D^v$ such that $X(i_j) = X'(i_j)$ for $j = 1, \ldots, s$, the event $E$ is true on $X'$. We say that indices $i_1, \ldots, i_s$* witness *that $E$ is true on $X$.*

As an example, consider a $\beta$-frugal coloring. Suppose that we assign colors to each vertex in a graph, and let $X(i)$ denote the color assigned to vertex $i$. For any vertex $i$, let $B$ be the event that the coloring assigned to $i$ fails to be frugal, i.e. some color appears at least $\beta + 1$ times in the neighborhood of $i$. This event is $\beta + 1$-witnessable, although it depends on $\Delta$ variables.

**Theorem 6.5.** *If $B$ is an $s$-witnessable boolean function, then $f(B) \leq 2^s \mathbb{P}_\Omega(B)$.*

*Proof.* Define the event $E$ that there exists some $a \in \{0,1\}^s$ such that $B$ is true on $Z_a$ as in Definition 6.3. We claim that if $E$ holds, then there must be at least $2^{v-s}$ vectors $a' \in \{0,1\}^v$ such that $B$ holds on $Z_{a'}$. For, suppose that we fix $X_0, X_1$ that $B$ holds on $Z_a$. By hypothesis, there must exist indices $i_1, \ldots, i_s$ such that any configuration $X'$ which agrees with $Z_a$ on coordinates $i_1, \ldots, i_s$ also satisfies $B$. Thus, changing the entries of $a$ outside the coordinates $i_1, \ldots, i_s$ does not falsify $B$.

Thus, for any fixed choice of $X_0, X_1$, we have

$$[E] \leq \frac{\sum_{a \in \{0,1\}^v} [B \text{ holds on configuration } Z_a]}{2^{v-s}}$$

But note that for a fixed choice of $a$, the configuration $Z_a$ is drawn according to the distribution $\Omega$, so that overall we have $\mathbb{P}_{X_0, X_1 \sim \Omega}(E) \leq \frac{2^v \mathbb{P}_\Omega(B)}{2^{v-s}} = 2^s \mathbb{P}_\Omega(B)$. □

Another example of fragility is for a large-deviation event.

**Theorem 6.6.** *Consider an event $B$ of the form*

$$B \equiv \sum_{i,j} c_{ij} [X(i) = j] \geq t$$

*where $c_{ij} \in [0,1]$ and $[X(i) = j]$ is the indicator random variable. Let $\mu = \sum_{i,j} \mathbb{P}_\Omega(X(i) = j)$. Then for $t \geq 2\mu(1+\delta)$ we have*

$$f(B) \leq \Big( \frac{e^\delta}{(1+\delta)^{1+\delta}} \Big)^{2\mu}$$

*Proof.* Suppose we draw $X_0, X_1$ independently from $\Omega$. Then observe that $B$ is true of some configuration $Z_a$ if it is true on the configuration $Z_{a^*}$, where we define

$$a^* = \begin{cases} 0 & \text{if } c_{i,X_0(i)} \geq c_{i,X_1(i)} \\ 1 & \text{if } c_{i,X_1(0)} < c_{i,X_1(i)} \end{cases}$$

Thus, if we define $Y_i = \max(c_{i,X_0(i)}, c_{i,X_1(i)})$, we see that $B$ holds on some $Z_{a^*}$ iff $\sum_i Y_i \geq t$. Each $Y_i$ is bounded in the range $[0,1]$ and $\mathbb{E}[Y_i] \leq \mathbb{E}[c_{i,X_0(i)} + c_{i,X_1(i)}] = 2 \sum_j \mathbb{P}(X(i) = j)$, and the variables $Y_i$ are independent. The stated bound therefore follows from Chernoff's bound. □

We now connect the fragility of an event to the probability it becomes dangerous.

**Proposition 6.7.** *For any bad-event $B$ and any $q \in [0,1]$ we have $\mathbb{P}(B \text{ is } q\text{-dangerous}) \leq f(B)/q$.*

*Proof.* Consider drawing $X_0$ according to the distribution $\Omega$. Suppose there is some subset $Y \subseteq N(B)$ such that the reversion of $X_0$ with respect to $Y$ causes the marginal probability of $B$ to exceed $q$. Define the vector $a$ by

$$a(i) = \begin{cases} 1 & \text{if } i \in \bigcup_{A \in Y} S_A \\ 0 & \text{otherwise} \end{cases}$$

By definition of $q$-dangerous, when we draw $X_1$ according to the distribution $\Omega$, the probability that $B$ holds on $Z_a$ is greater than $q$.

Thus, when $X_0, X_1$ are independently drawn from $\Omega$, the probability that $B$ is true on some configuration $Z_a$ is at least $\mathbb{P}(B \text{ is } q\text{-dangerous}) \times q$. □

## 6.1 The Algorithm

With these definitions given, we can now provide an outline for the LLL algorithm:

**Algorithm 2** Distributed LLL algorithm

1: Draw $X \in D^v$ from the distribution $\Omega$
2: Construct the set $M \subseteq \mathcal{B}$ consisting of all bad-events which are $(ed)^{-3}$-dangerous with respect to $X$.
3: Form the vector $Y \in \overline{D}^v$ by

$$Y(i) = \begin{cases} ? & \text{if } i \in \bigcup_{B \in M} S_B \\ X(i) & \text{otherwise} \end{cases}$$

4: Form the set $\mathcal{A} \subseteq \mathcal{B}$ by

$$\mathcal{A} = \{B \in \mathcal{B} \mid N(B) \cap M \neq \emptyset\}$$

5: Fix all the non-? entries of $Y$, and use the deterministic LLL algorithm on (each connected component) of $\mathcal{A}$ to fill in all the ? entries of $Y$.

**Proposition 6.8.** *Let $F = \max_{B \in \mathcal{B}} f(B)$. Any bad-event $A$ goes into $\mathcal{A}$ with probability at most $eFd^4$, and this depends only on the random bits within the two-hop neighborhood of $A$.*

*Proof.* Each $B$ goes into $M$ if it is $q$-dangerous with respect to $X$, where $q = (ed)^{-3}$; by Proposition 6.7 this has probability at most $f(B)/q \leq F/q$. Also, the event that $B$ goes into $M$ can be determined solely by the values of $X(i)$ for $i \in S_B$. Thus, the event that $B$ goes into $M$ is not affected by random choices made outside the neighborhood of $B$.

The event $A$ goes into $\mathcal{A}$ only if there is some $B \in N(A)$ with $B \in M$. Taking a union bound over all such $B \in N(A)$, we see that $\mathbb{P}(A \in \mathcal{A}) \leq \sum_{B \in N(A)} \mathbb{P}(B \in M) \leq dF/q$. Also, this only depends on random variables within the two-hop neighborhood of $A$. □

**Theorem 1.7.** *Suppose that $f(B) \leq F$ for every $B \in \mathcal{B}$ and*

$$Fd^{13} \leq 1$$

*Then Algorithm 2 terminates with a satisfying assignment in $2^{O(\sqrt{\log \log n})}$ rounds whp.*

*Proof.* Algorithm 2 takes only $O(1)$ rounds to determine membership in $M$, etc. The main running time will be dominated by Step (5).

Each $B \in \mathcal{B}$ survives to the residual problem $\mathcal{A}$ with probability $P \leq eFd^4$, and this depends only on the $c$-hop neighborhood for $c = 2$. Note that $P(e\Delta)^{4c} \leq e^9 eFd^4 d^8 \leq 1$, so we may apply Theorem 5.1. The residual problem can be viewed as an LLL instance with probability $q$ and dependency $d$. Since $edq^3 \leq 1$, the randomized algorithm of [CPS17] can solve such LLL instances on graphs of size $N$ in $O(\log_d N)$ rounds. Here, $N = d^8 \log n$, so this takes time $r = O(\frac{\log d + \log n \log n}{\log d}) \leq O(\log \log n)$.

We have $r \leq 2^{O(\sqrt{\log \log n})}$ and so by Theorem 5.1, the overall time to solve the residual problem is $2^{O(\sqrt{\log \log n})}$. □

### 6.2 Application: $k$-SAT

Consider a boolean formula $\Phi$, in which each clause contains $k$ literals, and clause overlaps with at most $d$ other clauses. A classic application of the LLL is to show that as long as $d \leq 2^k/e$, then the formula is satisfiable; further, as shown by [GST16] this bound is asymptotically tight. Using our Theorem 1.7, we are able to show a qualitatively similar bound.

**Proposition 1.8.** *If $\Phi$ has $n$ variables and $m$ clauses and $d \leq (4/3)^{k/3} \approx 1.02238^k$, there is a distributed algorithm to find a satisfying solution to $\Phi$ is $2^{O(\sqrt{\log \log m})}$ rounds.*

*Proof.* The probability space $\Omega$ is defined by selecting each variable $X(i)$ to be true or false with probability $1/2$. We have a bad-event for each clause, that it becomes violated; each bad-event $B$ has probability $p = 2^{-k}$. Thus, in total, there are $m$ bad-events.

Consider some clause $C$, wlg. $C = x_1 \vee x_2 \vee \cdots \vee x_k$. Consider drawing $X_0, X_1$ independently from $\Omega$; if $X_0(i) = X_1(i) = T$ for some $i \in [k]$, then necessarily $C$ holds on every configuration $Z_a$. Thus, a necessary event for $B$ to fail on some $Z_a$ is for $X_0(i) = F$ or $X_1(i) = F$. This has probabiltiy $3/4$ for each $i$ and by independence we have

$$f(B) \leq (3/4)^k$$

So in our problem, $F = (3/4)^k$ (by contrast, $p = 2^{-k}$). By Theorem 1.7, we need to satisfy

$$(3/4)^k d^{13} \leq 1$$

i.e. $d \leq (4/3)^{k/13}$. □

### 6.3 Application: Defective Coloring

A $h$-defective $k$-coloring of a graph $G = (V, E)$, is a mapping $\phi : V \to \{1, \ldots, c\}$, with the property that every vertex $v$ has at most $f$ neighbors $w$ with $\phi(v) = \phi(w)$. A proper vertex coloring is a 0-defective coloring. A classical application of the iterated LLL is to show that a graph with maximum degree $\Delta$ has an $h$-defective $k$-coloring with $k = O(\Delta/f)$, for any integer $f \geq 0$. In [FG17], a distributed algorithm was given to find such a coloring in $2^{O(\sqrt{\log \log n})}$ rounds. When $\Delta$ is small, this is a straightforward application of the distributed pLLL algorithm. For large values of $\Delta$, this required a somewhat specialized recoloring argument. We will show how to replace this recoloring argument with an application of Theorem 1.7.

**Proposition 6.9.** *Suppose that there is a map on the vertex set $\phi_1 : V \to \{0, \ldots, k_1 - 1\}$. Define $V_i = \phi_1^{-1}(i)$ for $i = 1, \ldots, k_1$. Suppose that each induced subgraph $G[V_i]$ has an $h$-defective $k_2$-coloring. Then $G$ has an $h$-defective $k$-coloring with $k = k_1 k_2$*

*Proof.* Let $\phi_{2,i}$ be the coloring of $G[V_i]$. Define the coloring map $\phi : V \to \{0, \ldots, k - 1\}$ by $\phi(v) = k_2 i + \phi_{2,i}(v)$ for $v \in V_i$ □

**Lemma 6.10.** *Suppose $G$ has maximum degree $\Delta$. There is a distributed algorithm in $2^{O(\sqrt{\log\log n})}$ rounds to find an $h$-defective $k$-coloring with $h = O(\log \Delta)$ and $k = \frac{\Delta}{\log \Delta}$.*

*Proof.* The probability space $\Omega$ assigns each vertex to a class $V_i$ with probability $1/k$. A bad-event $B_v$ is that a $v$ such that $v$ has more than $K \log \Delta$ neighbors in some class $V_i$.

A bad-event $B_v$ only depends on the allocation of the neighbors of $v$, and so only affects another $B_w$ if $d(v, w) \leq 2$. So the LLL dependency $d$ satisfies $d \leq \Delta^2$.

We next compute $f(B_v)$. We can write $B_v$ as $B_v = B_{v,1} \cup B_{v,2} \cup \cdots \cup B_{v,k}$, where $B_{v,i}$ is the event that $v$ has too many neighbors in $V_i$. So $f(B_v) = f(B_{v,1} \cup \cdots \cup B_{v,k}) \leq f(B_{v,1}) + \cdots + f(B_{v,k}) \leq \Delta f(B_{v,1})$.

Now, note that $B_{v,1}$ can be interpreted as a large-deviation event; here $\mu$ is the expected number of neighbors of $v$ entering $V_i$, which is at most $\Delta/k = \log \Delta$. By Theorem 6.6, we therefore have

$$f(B_{v,1}) \leq \Big(\frac{e^\delta}{(1+\delta)^{1+\delta}}\Big)^{2\mu}$$

where $\mu = \log \Delta$ and $\delta = K/2 - 2$. When $K = 100$, simple calculations show that $f(B_{v,1}) \leq \Delta^{-285}$.

We can therefore apply Theorem 1.7 to find a configuration avoiding all bad-events, as

$$Fd^{13} \leq (\Delta \times \Delta^{-285}) \times (\Delta^2)^{13} = \Delta^{-258} \leq 1$$

□

**Proposition 1.9.** *Suppose $G$ has maximum degree $\Delta$ and $h \leq \Delta$. There is a distributed algorithm in $2^{O(\sqrt{\log\log n})}$ rounds to find an $h$-defective $k$-coloring with $k = O(\Delta/h)$.*

*Proof.* We apply Lemma 6.10 to the original graph, thereby obtaining a coloring $\phi_1$ with $k_1 = \Delta/\log \Delta$ colors, such that each color class of $\phi_1$ has maximum degree $\Delta_1 = O(\log \Delta)$.

We apply Lemma 6.10 again to each color class of $\phi_1$; each color-class has a $\Delta_2$-defective $k_2'$-coloring with $k_2' = \frac{\Delta_1}{\log \Delta_1}$. By Proposition 6.9, this yields a $\Delta_2$-defective coloring $\phi_2$ with $k_2 = k_1 k_2 = O(\frac{\Delta}{\log\log \Delta})$ colors. Note that this step can be carried out in parallel for each color class of $\phi_1$, so the overall running time is $2^{O(\sqrt{\log\log n})}$.

Each color class of $\phi_2$ has maximum degree $\Delta_2 = O(\log\log \Delta)$. This degree is small enough that we can use the standard pLLL construction to find a $h$-defective coloring $\phi_3$ of each color class with $k_3' = O(\Delta_2/h)$ colors. This step can be carried out in parallel for each class of $\phi_2$, so overall it also takes $2^{O(\sqrt{\log\log n})}$ rounds.

Applying Proposition 6.9, this yields a final $h$-defective coloring with $k_3' k_2 = O(\Delta/h)$ colors. □

# 7 Obstacles to Derandomizing Local Algorithms

In this section, we discuss possible limitations to derandomizing local algorithms.

## 7.1 An Exponential Separation in the SLOCAL Model

A key consequence of Theorem 3.1 is that in the SLOCAL model, for locally checkable problems, randomized algorithms are no more powerful than deterministic algorithms, if we do not care about logarithmic factors. Phrased differently, any randomized SLOCAL$(t(n))$ algorithm for an LCL problem with constant locality —or even $\mathrm{poly}(\log n)$ locality—can be derandomized to a deterministic SLOCAL$(t(n) \cdot \log^2 n)$ algorithm. In this section, we show that once we start caring about logarithmic factors, an exponential separation shows up. More concretely, we show that the same problem of sinkless orientation, which was shown to exhibit an exponential separation between randomized and deterministic complexities in the LOCAL model,

exhibits an exponential separation also in the SLOCAL model. However, the placement of the bounds are different, and in fact surprising to us.

In the LOCAL model, Brandt et al. [BFH+16] showed that randomized sinkless orientation requires $\Omega(\log\log n)$ round complexity and Chang et al. [CKP16] showed that deterministic sinkless orientation requires $\Omega(\log n)$ round complexity. These were complemented by matching randomized $O(\log\log n)$ and deterministic $O(\log n)$ upper bounds by Ghaffari and Su [GS17]. In contrast, in the SLOCAL model, the tight complexities are an exponential lower: we show in Theorems 7.1 to 7.3 that in the SLOCAL model, the sinkless orientation has deterministic complexity $\Theta(\log\log n)$ and randomized complexity $\Theta(\log\log\log n)$.

A take-home message from this exponential separation for the SLOCAL model is that, one might suspect that similar to this exponential gap of the SLOCAL model that disappears once we move up to the land of higher complexities and ignore logarithmic factors, perhaps a similar phenomena happens also in the LOCAL model as well and the gap between randomized and deterministic LOCAL algorithms for LCL problems can be at most factors of $\log n$. More concretely, we can at least say that if a significant gap persists in the LOCAL model in the land of $\Omega(\log n)$ complexity, it must be for a very different reason than the gap in the lower land of $o(\log n)$ complexities—e.g., that of sinkless orientation—because the latter disappears in the SLOCAL model.

**Theorem 7.1.** *Any deterministic* SLOCAL *algorithm for sinkless orientation, even on $\Delta$-regular graphs, has locality $\Omega(\log_\Delta \log n)$. Moreover, any randomized* SLOCAL *algorithm for sinkless orientation, even on $\Delta$-regular graphs, has locality $\Omega(\log_\Delta \log\log n)$.*

*Proof.* Suppose that there is a deterministic $\mathsf{SLOCAL}(t)$ sinkless orientation algorithm. By Proposition 3.3, this yields a deterministic $\mathsf{LOCAL}(t\Delta^t + t\log^* n)$ algorithm. As shown by Chang et al. [CKP16], deterministic LOCAL sinkless orientation algorithms must have round complexity $\Omega(\log n)$. So we know that $t\Delta^t + t\log^* n \geq \Omega(\log n)$, which can easily be seen to imply that $t \geq \Omega(\log_\Delta \log n)$.

The proof for the $\Omega(\log_\Delta \log\log n)$ lower bound on the locality of randomized SLOCAL algorithms is similar, modulo a small change that in this case the contradiction will be with the $\Omega(\log\log n)$ lower bound of Brandt et al. [BFH+16] on the round complexity of randomized LOCAL sinkless orientation algorithms. □

**Theorem 7.2.** *There is a deterministic* $\mathsf{SLOCAL}(O(\log\log n))$ *algorithm to compute a sinkless orientation of any graph with minimum degree at least* 3*.*

*Proof.* Follows immediately from the randomized LOCAL sinkless orientation of Ghaffari and Su [GS17], which has round complexity $O(\log\log n)$, combined with Theorem 3.1, and noticing that sinkless orientation is in $\mathsf{LD}(1)$. □

**Theorem 7.3.** *There is a randomized* $\mathsf{SLOCAL}(O(\log\log\log n))$ *to compute a sinkless orientation of any graph with minimum degree at least* 3*.*

*Proof.* We here describe a cleaner version of the algorithm that works for the special case of regular graphs with degree $d \geq 500$. This special case already suffices for our goal of exhibiting an exponential separation in the SLOCAL model, because the lower bounds work on this special case. This algorithm can be extended to the general case of arbitrary graphs with minimum degree at least 3, by adding a few small steps, similar to those of of Ghaffari and Su [GS17, Appendix A.2], which have SLOCAL complexity $O(1)$.

Below, we describe the algorithm as a two-pass randomized $\mathsf{SLOCAL}(O(\log\log\log n))$ algorithm. This algorithm is borrowed from Ghaffari and Su [GS17] almost verbatim, modulo the small change that the second pass/phase uses a deterministic SLOCAL algorithm. As shown in Ghaffari, Kuhn, and Maus [GKM17], this can then be transformed into a single-pass randomized $\mathsf{SLOCAL}(O(\log\log\log n))$ algorithm.

**Algorithm 3** A Randomized SLOCAL Algorithm for Sinkless Orientation

**Pass 1:**
Mark each edge with probability $\frac{1}{4}$.
For each marked edge, orient it randomly with probability $1/2$ for each direction.
For each node $v$, mark $v$ as *a bad node* of the following types according to these rules:

- Type I. If $v$ has more than $d/2$ marked edges incident to it.
- Type II. If $v$ is not Type I but it has at least one neighbor of Type I.
- Type III. If $v$ is not Type I or Type II but it has no outgoing marked edges.

Unmark all the edges incident to Type I nodes.
Orient unmarked edges that both of their endpoints are good nodes arbitrarily.
Consider unmarked edges with exactly one good endpoint as a half-edge only attached to the bad-node.

**Pass 2:**
If the node under process is bad, then run the deterministic SLOCAL algorithm of Theorem 7.2 on this node and the components induced by the bad nodes around it.

The first pass clearly has locality $O(1)$. By the analysis of Ghaffari and Su [GS17], each connected component on bad nodes has size at most $N = O(\log n)$ and therefore, by Theorem 7.2, the second pass has locality $O(\log \log N) = O(\log \log \log n)$. □

## 7.2 Impossibility of Derandomization for Non-Locally-Checkable Problems

In this section, we show a simple toy-problem which shows that the local checkability in our derandomization is an important property, and it is not there just for technicalities. More concretely, we show a problem that can be solved by a zero-round Monte Carlo algorithm, but any deterministic algorithm for it needs at least $\sqrt{n}$ rounds. Thus, any derandomization for this problem must have a polynomial (not polylogarithmic) overhead. will have a significant overhead, way beyond poly-logarithmic factors.

The problem is simple: Given a cycle of size $n$, where $n$ is known to all nodes, we should mark $(1 \pm o(1))\sqrt{n}$ nodes. The trivial zero-round randomized algorithm, which marks each node with probability $1/\sqrt{n}$, achieves this goal with high probability. We next argue that any deterministic algorithm needs $\Omega(\sqrt{n})$ rounds.

**Proposition 7.4.** *Any deterministic* LOCAL *algorithm for the cycle-marking problem needs* $\Omega(\sqrt{n})$ *rounds.*

*Proof.* Suppose there is a deterministic algorithm $\mathcal{A}$ that solves the problem in $T = \sqrt{n}/100$ rounds. Consider some $50\sqrt{n}$ disjoint cycles each of length $n$, where the $i^{th}$ cycle is made of nodes with IDs $(i-1)n+1$ to $in$. We run $\mathcal{A}$ on each of these cycles. Algorithm $\mathcal{A}$ must mark at least one node in each of these cycles (in fact at least $(1-o(1))\sqrt{n}$ many). Then, we select one of those marked nodes per cycle and we carve out a $2T$ neighborhood of that one marked node from the cycle. We then append $50\sqrt{n}$ of these together to form a new cycle of length $n$.

In the new cycle, each previous mark will remain marked, because it is more than $T$ hops far from the cut points and it does not notice that it is carved out and put in a new graph. More nodes might be marked on the boundaries, but that does not hurt. This way, overall, we have at least $50\sqrt{n}$ marked node and so $\mathcal{A}$ fails on this new $n$-node cycle. □

## 7.3 Complete Problems

We next prove completeness results for several natural and widely-studied distributed graph problems. To relate different problems to each other, we use the notion of *locality-preserving reductions* as introduced as

formally defined in [GKM17]. A distributed graph problem $\mathcal{A}$ is called *polylog-reducible* to a distributed graph problem $\mathcal{B}$ iff a $\operatorname{polylog} n$-time deterministic LOCAL algorithm for $\mathcal{A}$ (for all possible $n$-node graphs) implies a $\operatorname{polylog} n$-time deterministic LOCAL algorithm for $\mathcal{B}$. In addition, a distributed graph problem $\mathcal{A}$ is called P-SLOCAL-complete if a) the problem $\mathcal{A}$ is in the class P-SLOCALand b) every other distributed graph problem in P-SLOCAL is polylog-reducible to $\mathcal{A}$. As a consequence, if any P-SLOCAL-complete problem could be solved in $\operatorname{polylog} n$ deterministic time in the LOCAL model, then we would have P-LOCAL = P-SLOCAL and thus every problem in P-SLOCAL(and thus by Proposition 3.4 also all problems in P-ZLOCAL) could also be solved in $\operatorname{polylog} n$ time deterministically in the LOCAL model. The following completeness proofs can therefore be understood as conditional lower bounds. Efficiently derandomizing any of the following problems would require to efficiently derandomize the whole class P-ZLOCAL of polylog-time randomized distributed Las Vegas algorithms.

In the *minimum set cover* problem, we are given a set of elements $X$ and a set $\mathcal{S} \subseteq 2^X$ of subsetst of $X$ such that for every $x \in X$, there is at least one set $A \in \mathcal{S}$ such that $x \in A$. The objective is to find a smallest possible collection of sets $\mathcal{C} \subseteq \mathcal{S}$ such that $\bigcup_{A \in C} = X$. In the *distributed set cover problem* and often also more generally in distributed linear programming algorithms (cf. [BBR97, KMW06, PY93]), the set system $(X, \mathcal{S})$ is modelled as a bipartite graph with a node for every element $x \in X$ and every set $A \in \mathcal{S}$. There is an edge between $x \in X$ and $A \in \mathcal{S}$ if and only if $x \in A$.

**Theorem 7.5.** *Approximating the distributed set cover problem by any factor $\alpha = \operatorname{polylog}(n)$ is* P-SLOCAL-*complete.*

*Proof.* We first note that [GKM17] has shown that the problem of computing a $(1+\varepsilon)$-factor approximation for the distributed set cover problem is in the class P-SLOCALfor any $\varepsilon \geq 1/\operatorname{polylog} n$.

To show that the problem is also P-SLOCAL-hard, we reduce from the problem of computing a $\operatorname{polylog}(n)$-color conflict-free multicoloring (abbreviated CFM) of an $n$-node hypergraph $H = (V, E)$ with $|E| \leq \operatorname{poly}(n)$. This problem was shown to be P-SLOCAL-complete in Theorem 1.10 of [GKM17] (even for almost uniform hypergraphs). A $q$ color multicoloring of a hypergraph $H = (U, F)$ assigns a non-empty set of colors from $\{1, \ldots, q\}$ to each node in $U$ soch that for each hyperedge $f \in F$, there is one color that is assigned to exactly one node in $f$ [Smo13].

Assume that there is a deterministic polylogarithmic-time algorithm for computing a $\alpha$-approximate solution for a given distributed set cover problemm. We show how to use this algorithm to obtain a polylogarithmic-time deterministic CFM algorithm for $H$ with $q = \operatorname{poly}(\alpha \cdot \log n)$ colors. Specifically, we show how to use the set cover approximation algorithm to reduce the problem of computing a $q$-color CFM on $H$ to the problem of computing a $q-1$-color CFM on a hypergraph on at most $n/2$ vertices; repeating this $O(\log n)$ times then gives the required reduction.

Let $F_-$ be the set of hyperedges of $H$ of size at most $\delta := C\alpha(\ln n + 1)$, for some constant $C$, and let $F_+ := F \setminus F_-$. In Lemma 6.2 of [GKM17], it is shown that for hypergraphs of $\operatorname{polylog} n$ rank, a CFM with $\operatorname{polylog} n$ colors can be computed deterministically in $\operatorname{polylog} n$ time in the LOCAL model. As $|F| \leq \operatorname{poly}(n)$ and $\alpha \leq \operatorname{polylog}(n)$, we can therefore compute a $\operatorname{polylog} n$-color CFM of $H_- := (V, F_-)$ in polylogarithmic deterministic time.

It remains to also compute a CFM of the hypergraph $H_+ := (V, F_+)$. Note that as $H_+$ has minimum rank $\delta$, it has a fractional vertex cover of size $|V|/\delta$. (Assign each vertex a weight of $1/\delta$. By a standard probabilitic argument using independent rounding and alteration, this shows that $H_+$ has a vertex cover of size $O(\frac{|V| \log n}{\delta})$. Using our approximation algorithm for set cover, we therefore can obtain a vertex cover $U$ of size $|U| \leq O(\frac{n\alpha \log n}{\delta})$.

Let $F'_+ = \{f \cap U \mid f \in F_+\}$. Note that if we can obtain a $q'$-color CFM of the hypergraph $(U, F_+)$, then we can extend it to a $q'+1$-color CFM of $H$ by assigning one additional color to every node in $V - U$. Also note that if $C$ is sufficiently large constant, then we have $|U| \leq n/2$. □

Given a graph $G = (V, E)$, the minimum dominating set (MDS) problem asks for a smallest possible set $D \subseteq V$ of nodes such that for all $u \in V$, either $u \in D$ or some neighbor $v$ of $u$ is in $D$. The MDS problem is essentially equivalent to the minimum set cover problem, and the following theorem shows that also approximating MDS is P-SLOCAL-complete. Note that MDS is one of the most widely studied problems in the LOCAL model. Polylogarithmic-time randomized distributed approximation algorithms have been known for a long time, e.g., [DMP+05, JRS02, KW05, KMW06].

**Theorem 7.6.** *Approximating the distributed MDS by a polylogarithmic factor is* P-SLOCAL*-complete.*

*Proof.* Since the dominating set problem is a special case of the set cover problem, the problem is clearly also in the class P-SLOCAL. To show that the problem is P-SLOCAL-hard, we reduce from the distributed set cover problem. Assume that we are given a distributed set cover problem $(X, \mathcal{S})$ with elements $X$ and sets $\mathcal{S}$. Recall the communication graph of the problem is given by the bipartite graph $G$ with nodes $X \cup \mathcal{S}$ and an edge between $x \in X$ and $A \in \mathcal{S}$ if and only if $x \in A$. Note that for the set cover instance to be solvable, for each $x \in X$, there needs to be at least one set $A \in \mathcal{S}$ for which $x \in A$. We define a graph $G'$ based on $G$ as follows. $G'$ has the same set of nodes as $G$ and it has all the edges of $G$. In addition, in $G'$, there is an edge between any two nodes $A$ and $B$ in $\mathcal{S}$ for which the corresponding sets share an element (i.e., all $A$ and $B$ for which $A \cup B \neq \emptyset$). Note that one communication round on the graph $G'$ can be simulated in $O(1)$ round on graph $G$.

We claim that the MDS size of $G'$ is equal to the minimum set cover size of the set cover instance $(X, \mathcal{S})$ and that any dominating set $D$ of $G'$ can directly be converted to a set cover of size at most $|D|$ for $(X, \mathcal{S})$. Consider a set cover $C \subseteq \mathcal{S}$. By the construction of $G'$, the set $C$ is also a dominating set of $G'$. Further, let $D$ be a dominating set of $G'$. If $D$ contains nodes in $X$, we can convert $D$ into a dominating set $D'$ of size $|D'| \leq |D|$ such that $D'$ only contains nodes from $\mathcal{S}$. For each $x \in D \cap X$, we replace $x$ by some neighborhing node $A \in \mathcal{S}$ (recall that such a neighbor always exists). Since $A$ is connected to all $B \in \mathcal{S}$ for which $x \in B$, $A$ covers all nodes of $G'$ that were covered by $x$ and thus $D'$ is still a dominating set and because $D' \subseteq \mathcal{S}$, it is also directly corresponds to a solution of the set cover instance $(X, \mathcal{S})$. Given an $\alpha$-approximate dominating set of $G'$, we therefore directly get an $\alpha$-approximate set cover solution. □

We next consider a natural greedy packing problem that has a similar flavor as MIS, one of the four classic symmetry breaking problems. Note that we do now know whether the MIS problem is P-SLOCAL-complete. Assume that we are given an $n$-node bipartite graph $G = (L \cup R, E)$. We call $L$ the left side and $R$ the right side of $G$. For a positive integer $k$, we define a $k$-star of $G$ to be a left-side node $u \in L$ together with $k$ right-side neighbors $v_1 \ldots, v_k \in R$. A *maximal independent $k$-star set* is a maximal set of pairwise vertex-disjoint $k$-stars of $G$. Note that the problem of finding a maximal independent $k$-star set can actually be modelled as finding an MIS of a graph $H$: The graph $H$ has a node for every $k$-star of $G$, two nodes $H$ are connected if the respective $k$-stars are not vertex-disjoint. However, if $G$ has $n$ nodes, the graph $H$ can have up to $O(n^{k+1})$ nodes and thus a polylogarithmic-time MIS algorithm on $H$ does not directly lead to a polylogarithmic-time algorithm for finding a maximal independent $k$-star set on $G$.

**Theorem 7.7.** *Let $G = (L \cup R, E)$ be an $n$-node bipartite graph, where every node in $L$ has degree at most $\Delta$. For every $\lambda \geq 1/\operatorname{polylog} n$, the problem of finding a maximal independent $\lceil \lambda \Delta \rceil$-star set of $G$ is* P-SLOCAL*-complete.*

*Proof.* As a maximal independent $k$-star set can be computed by a simple sequential greedy algorithm, clearly the problem is in P-SLOCAL. In fact, when processing a node $u \in L$, we can decide whether one can add a $k$-star with $u$ as the center by inspecting the 2-hop neighborhood of $u$. The problem therefore even is in the class SLOCAL(2).

To show P-SLOCAL-hardness, we reduce from the minimum set cover problem. Consider a minimum set cover problem $(X, \mathcal{S})$ and the corresponding bipartite graph $G = (L \cup R, E)$, where $L$ corresponds to

the sets $\mathcal{S}$, the $R$ corresponds to the set of elements $X$. Assume that the maximum degree of the nodes in $L$ is at most $\Delta$. All of the sets in set cover instance have size at most $\Delta$, and so the minimum set cover has size at least $|X|/\Delta$.

Assume that we are given a maximal independent $\lceil \lambda\Delta/\rceil$-star set of $G$. Because each star contains $\lceil \lambda\Delta/\rceil$ nodes from $X$, such a set can consist of at most $|X|/(\lambda\Delta)$ starts. Further, by adding all the sets corresponding to the centers of the stars to the set cover, by the maximality of the star set, the maximum set size of the remaining set cover instance is at most $(1-\lambda)\Delta$. Repeating $O\big(\frac{\log\Delta}{\lambda}\big)$ therefore yields a set cover solution that is optimal up to a factor $O\big(\frac{\log\Delta}{\lambda^2}\big)$. □

The last problem we consider in this section is *sparse neighborhood cover* of a graph. [AP90].

**Definition 7.8** (Sparse neighborhood cover, adapted from [AP90])**.** *Let $G=(V,E)$ be a graph and let $r \geq 1$ be an integer parameter. A sparse $r$-neighborhood cover of $G$ is a collection of clusters $C_1,\dots,C_k$ such that each cluster has diameter at most $r \cdot \operatorname{polylog} n$, such that each $r$-hop neighborhood of $G$ is completely contained in at least one of the clusters, and such that every node of $G$ is contained in at most $\operatorname{polylog} n$ of the clusters.*

It is shown in [AP90] that such neighborhood covers exist (even when replacing all the $\operatorname{polylog} n$ terms in the definition by terms of order $\log n$). Sparse neighborhood covers are a fundamental structure with a large number of applications in distributed systems [Pel00]. We note that it is sufficient to study sparse neighborhood covers for $r=1$. When computing a sparse 1-neighborhood cover for $G^r$, one obtains a sparse $r$-neighborhood cover for $G$.

**Theorem 7.9.** *For any $1 \leq r \leq \operatorname{polylog} n$, the problem of computing a sparse neighborhood cover of a graph is* P-SLOCAL*-complete.*

*Proof.* The sequential construction of [AP90] to construct sparse neighborhood covers almost directly gives a multi-phase SLOCAL algorithm. The clusters are constructed iteratively in $O(\log n)$ passes over all the nodes. Each cluster is constructed by a ball growing argument starting from the center node of the cluster. To construct a cluster around a node $u$, only the $O(r\log n)$-hop neighborhood of $u$ has to be inspected. The construction can thus be turned into a concatenation of $O(\log n)$ SLOCAL $(O(r\log n))$-algorithms. Lemma 2.3 of [GKM17] then implies that the construction can be turned into a single SLOCAL $(O(r\log^2 n))$-algorithm. For $r \leq \operatorname{polylog} n$, the problem of computing a sparse neighborhood cover is therefore in P-SLOCAL.

To show that the problem is P-SLOCAL-hard, we reduce from the MDS problem. Assume that we want to compute a dominating set of a graph $G=(V,E)$. We first compute a sparse 1-neighborhood cover of $G$. Assume that each node of $G$ is contained in at most $\delta \leq \operatorname{polylog} n$ of the clusters of the neighborhood cover.

For each cluster $C$ of the 1-neighborhood cover of $G$, we define a set cover problem as follows. Let $C_{inside}$ be the set of node in $C$ that are not at the boundary (the nodes for which all neighbors are also in $C$). The set of elements of the set cover instance is $C_{inside}$ and we define a $A_v$ for each node $v \in C$, where $A_v$ contains all the nodes of $C_{inside}$ that are in the 1-neighborhood of $v$. Note that if compute a solution to the set cover instance for cluster $C$ and we add all the nodes of $C$ corresponding to the chosen sets to our dominating set, then the dominating set definitely covers all the nodes in $C_{inside}$. We compute a dominating set of $G$ by constructing an optimal set cover solution for each cluster and taking the union of the resulting sets of nodes.[6] First note that the resulting set is a valid dominating set because each 1-neighborhood of $G$ is completely contained in some cluster and thus for every node $v \in V$, there is some cluster that contains $v$ and where $v$ is not on the boundary.

[6] If we want the local computations at the nodes to be polynomial, it is sufficient to just take the greedy set cover solution for each cluster.

It remains to show that the constructed dominating set is a good approximation. For this purpose, we also compute an optimal solution for the dual LP of the fractional set cover LP for each cluster $C$. The dual LP assigns a value $y_v$ to each element, i.e., to each node $v \in C_{inside}$ such that for each node $u \in C$, the sum of the $y_v$ values of its neighbors is at most 1. Because the set cover LP has an integrality gap of at most $O(\log n)$, the total sum of the $y_v$ values in cluster $C$ are at least a $1/O(\log n)$-fraction of the size of the set cover for cluster $C$. Consider the combined dual solution that is obtained by adding up the dual solutions of all clusters. Clearly, the sum is also at least a $1/O(\log n)$-factor of the total size of the constructed dominating set. Further, consider the sum of the combined dual values in the 1-hop neighborhood of nodes node $v$. Because in each cluster $C$, only the nodes in $C_{inside}$ get positive dual values, only the clusters that contain $v$ contributed to the combined dual values in the 1-hop neighborhood of $v$. Therefore, because each node is contained in at most $\delta$ of the clusters, the total sum of the combined dual values in the 1-hop neighborhood of $v$ is at most $\delta$. By dividing each combined dual value by $\delta$, we therefore obtain a feasible solution for the dual LP of the dominating set problem on $G$. The constructed dominating set is at most an $O(\delta \log n)$-factor larger than the size of this LP and the dominating set is thus also at most an $O(\delta \log n)$-factor larger than an optimal dominating set. The claim of the theorem now follows because $\delta \leq \operatorname{polylog} n$ and because by Theorem 7.6, computing a $\operatorname{polylog}$-approximate dominating set is P-SLOCAL-complete. □

The above theorem can be also be interpreted as follows. It was clear before that given a polylog-time deterministic distributed algorithm for network decomposition, we can also get such an algorithm for computing sparse neighborhood covers. The above theorem shows that also the converse is true: Given a polylog-time deterministic distributed algorithm for sparse neighborhood covers, we can get a polylog-time deterministic distributed algorithm to compute a $(O(\log n), O(\log n))$-network decomposition. Hence, up to polylogarithmic factors, the complexity of the two key graph clustering variants are equivalent in the LOCAL model.

## Acknowledgements

We would like to thank Janosch Deurer, Manuela Fischer, Juho Hirvonen, Yannic Maus, Jara Uitto, and Simon Weidner for interesting discussions on different aspects of the paper.

## References

[ABCP96] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Fast network decompositions and covers. *J. of Parallel and Distributed Computing*, 39(2):105–114, 1996.

[ABI86] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. of Algorithms*, 7(4):567–583, 1986.

[AGLP89] B. Awerbuch, A. V. Goldberg, M. Luby, and S. A. Plotkin. Network decomposition and locality in distributed computation. In *Proc. 30th Symp. on Found. of Computer Science (FOCS)*, pages 364–369, 1989.

[AP90] B. Awerbuch and D. Peleg. Sparse partitions. In *Proc. 31st IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 503–513, 1990.

[BBR97] Y. Bartal, J. W. Byers, and D. Raz. Global optimization using local information with applications to flow control. In *Proc. 38th IEEE Symp. on the Foundations of Computer Science (FOCS)*, pages 303–312, 1997.

[BE13] L. Barenboim and M. Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Morgan & Claypool Publishers, 2013.

[BEK15] L. Barenboim, M. Elkin, and F. Kuhn. Distributed $(\Delta+1)$-coloring in linear (in $\Delta$) time. *SIAM J. Computing*, 43(1):72–95, 2015.

[BEPS16] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider. The locality of distributed symmetry breaking. *Journal of the ACM (JACM)*, 63(3):20, 2016.

[BFH+16] S. Brandt, O. Fischer, J. Hirvonen, B. Keller, T. Lempiäinen, J. Rybicki, J. Suomela, and J. Uitto. A lower bound for the distributed Lovász local lemma. In *Proc. 48th ACM Symp. on Theory of Computing (STOC)*, pages 479–488, 2016.

[BRS94] B. Berger, J. Rompel, and P. W. Shor. Efficient NC algorithms for set cover with applications to learning and geometry. *Journal of Computer and System Sciences*, 49(3):454–477, 1994.

[CKP16] Y.-J. Chang, T. Kopelowitz, and S. Pettie. An exponential separation between randomized and deterministic complexity in the LOCAL model. In *Proc. 57th IEEE Symp. on Foundations of Computer Science (FOCS)*, 2016.

[CP17] Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the local model. *arXiv preprint arXiv:1704.06297*, 2017.

[CPS17] K.-M. Chung, S. Pettie, and H.-H. Su. Distributed algorithms for the Lovász local lemma and graph coloring. *Distributed Computing*, 30(4):261–280, 2017.

[DMP+05] D. Dubhashi, A. Mei, A. Panconesia, J. Radhakrishnan, and A. Srinivasan. Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons. *J. of Computer and System Sciences (JCSS)*, 71(4):467–479, 2005.

[EPS15] M. Elkin, S. Pettie, and H.-H. Su. (2$\Delta$- 1)-edge-coloring is much easier than maximal matching in the distributed setting. In *Proc. 26th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 355–370, 2015.

[FG17] M. Fischer and M. Ghaffari. Sublogarithmic distributed algorithms for Lovász local lemma with implications on complexity hierarchies. In *Proc. 31st Symp. on Distributed Computing (DISC)*, 2017.

[FGK17] M. Fischer, M. Ghaffari, and F. Kuhn. Deterministic distributed edge-coloring via hypergraph maximal matching. In *Proc. 58th IEEE Symp. on Foundations of Computer Science (FOCS)*, 2017.

[FHK16] P. Fraigniaud, M. Heinrich, and A. Kosowski. Local conflict coloring. In *Proc. 57th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 625–634, 2016.

[FKP13] P. Fraigniaud, A. Korman, and D. Peleg. Towards a complexity theory for local distributed computing. *J. of the ACM*, 60(5):35, 2013.

[Gha16] M. Ghaffari. An improved distributed algorithm for maximal independent set. In *Proc. 27th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 270–277, 2016.

[GKM17] M. Ghaffari, F. Kuhn, and Y. Maus. On the complexity of local distributed graph problems. In *Proc. 49th ACM Symp. on Theory of Computing (STOC)*, pages 784–797, 2017.

[GS17] M. Ghaffari and H.-H. Su. Distributed degree splitting, edge coloring, and orientations. In *Proc. 28th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 2505–2523, 2017.

[GST16] H. Gebauer, T. Szabó, and G. Tardos. The Local Lemma is asymptotically tight for SAT. *J. of the ACM*, 63(5):43, 2016.

[Har17] D. G. Harris. Parallel algorithms for the Lopsided Lovász local lemma. *arXiv preprint arXiv:1702.02547*, 2017.

[HH17] B. Haeupler and D. G. Harris. Parallel algorithms and concentration bounds for the Lovász local lemma via witness DAGs. In *Proc. 28th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1170–1187, 2017.

[HK73] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

[JRS02] L. Jia, R. Rajaraman, and R. Suel. An efficient distributed algorithm for constructing small dominating sets. *Distributed Computing*, 15(4):193–205, 2002.

[KMW06] F. Kuhn, T. Moscibroda, and R. Wattenhofer. The price of being near-sighted. In *Proc. 17th Symp. on Discrete Algorithms (SODA)*, pages 980–989, 2006.

[KW05] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. *Distributed Computing*, 17(4):303–310, 2005.

[Lin87] N. Linial. Distributive graph algorithms – global solutions from local data. In *Proc. 28th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 331–335, 1987.

[LPSP15] Z. Lotker, B. Patt-Shamir, and S. Pettie. Improved distributed approximate matching. *Journal of the ACM (JACM)*, 62(5), 2015.

[LS93] N. Linial and M. Saks. Low diameter graph decompositions. *Combinatorica*, 13(4):441–454, 1993.

[Lub86] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036–1053, 1986.

[MT10] R. A. Moser and G. Tardos. A constructive proof of the general Lovász local lemma. *J. of the ACM*, 57(2):11, 2010.

[Pel00] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

[PS95] A. Panconesi and A. Srinivasan. On the complexity of distributed network decomposition. *J. of Algorithms*, 20(2):581–592, 1995.

[PY93] C. Papadimitriou and M. Yannakakis. Linear programming without the matrix. In *Proc. 25th ACM Symp. on Theory of Computing (STOC)*, pages 121–129, 1993.

[RV98] S. Rajagopalan and V. V. Vazirani. Primal-dual RNC approximation algorithms for set cover and covering integer programs. *SIAM Journal on Computing*, 28(2):525–540, 1998.

[Smo13] S. Smorodinsky. Conflict-free coloring and its applications. In *GeometryIntuitive, Discrete, and Convex*, pages 331–389. Springer, 2013.

# A Residual problems and shattering

In this section, we provide an overview of the shattering method for distributed algorithms, and in particular how our derandomization allows us to randomized algorithms for the second "clean-up" phase. The shattering method was developed first in [BEPS16], and variants have been applied to many graph algorithms since then such as [Gha16, BEPS16, EPS15]. In this section, we aim to provide a comprehensive and unified treatment of these types of algorithms. Also, while our arguments will closely parallel those of [BEPS16], the latter does not give us precisely the parameters we will need for our LLL algorithms.

The shattering method has two phases. In the first phase, there is some random process. Most of the vertices $v \in G$ are satisfied in this random phase, and we will fix the choices these "good" vertices make. In the second phase, we let $R \subseteq V$ denote the unsatisfied vertices. These vertices are very sparse, and the connected components of $G[R]$ are relatively small. We then a deterministic algorithm to solve the residual problem on on (each component of $G[R]$). We assume that each vertex $v$ survives to $R$ with probability $p$, and this bound holds even for adversarial choices for the random bits outside the $c$-hop neighborhood of $v$. We assume here that $c \geq 1$ is some constant, and all asymptotic notations in this section may hide dependence upon $c$. We will assume throughout this section that $p$ satisfies

$$p \leq (e\Delta)^{-4c}$$

**Definition A.1.** *Given a graph $G = (V, E)$ and a vertex set $W \subseteq V$, we say that $W$ is* connected in $G$ *if for every $w, w' \in W$ there is a path in $G$ from $w$ to $w'$. We say that $G$ is* connected *if $V$ is connected in $G$.*

**Definition A.2.** *Given a graph $G = (V, E)$ and a parameter $c \geq 1$, we say that a set $U \subseteq V$ is a* $c$-backbone *of $G$ if $U$ is an independent set of $G^c$, and $G^{3c}[U]$ is connected.*

*We say that a set $W \subseteq V$ is* $c, m$-backbone-free *in $G$ if there is no $c$-backbone $U \subseteq W$ with $|U| \geq m$.*

**Proposition A.3.** *If $G$ has maximum degree $\Delta$ and $x$ is a vertex of $G$, then there are at most $(e\Delta)^{3cm}$ distinct $c$-backbones $U$ such that $|U| = m, x \in U$.*

*Proof.* Define $r = \Delta^{3c}$. For any $v \in V$, define

$$\tau_i(v) = \sum_{\substack{c\text{-backbones } U \\ |U| \leq i \\ U \ni v}} (er)^{-|U|}$$

We also define $\tau_\infty(v) = \lim_{i \to \infty} \tau_i(v)$. Thus

$$\sum_{\substack{c\text{-backbones } U \\ U \ni x \\ |U| \geq m}} 1 = (er)^m \sum_{\substack{c\text{-backbones } U \\ U \ni v \\ |U| = m}} (er)^{-|U|} \leq (er)^m \sum_{\substack{c\text{-backbones } U \\ U \ni v}} (er)^{-|U|} = (er)^m \tau_\infty(x)$$

We show by induction on $i$ that $\tau_i(v) \leq 1/r$ for all integers $i \geq 0$ and $v \in V$. Consider some $c$-backbone $U$ for $x$ of size at most $i$. Let $v_1, \dots, v_k$ denote the elements of $U$ within distance $3c$ of $x$; we must have $k \leq r$. For each $j = 1, \dots, k$, let $U_j$ denote the vertices $u \in U$ which are reachable from $v_j$ but not $x, v_1, \dots, v_{j-1}$ via paths in $G^{2c}$. Clearly $U_1, \dots, U_k$ are $c$-backbones containing $v_1, \dots, v_k$ respectively, and have size strictly less than $i$. Also

$$(er)^{|U|} = (er)^{-1} (er)^{-|U_1|} \dots (er)^{-|U_k|}$$

Summing over all choices of possible $U_1, \ldots, U_k$ and using the induction hypothesis, gives us the upper bound

$$\sum_{\substack{c\text{-backbones } U \\ |U| \le i \\ U \ni v \\ |U \cap N_2(v)| = \{v_1, \ldots, v_k\}}} \le (er)^{-1} \tau_{i-1}(v_1) \ldots \tau_{i-1}(v_k) \le (er)^{-1} r^{-k}$$

Summing over all possible choices for $k$ and $v_1, \ldots, v_k$ gives:

$$\tau_i(v) \le \sum_{k=0}^{r} \binom{r}{k} \frac{1}{er^{k+1}} = \frac{(1+1/r)^r}{er} \le 1/r$$

This implies that $\tau_\infty(v) \le 1/r$ for all $v \in V$, and the claim follows. □

**Proposition A.4.** *Suppose that $G = (V, E)$ and $S \subseteq V$. Suppose that there is some $W \subseteq S$ such that $W$ is an independent set of $G^c$ and $W$ is connected in $G^c[S]$. Then $G$ has a $c$-backbone $U \subseteq S$ with $|U| \ge |W|$.*

*Proof.* Let $S' \subseteq S$ denote the set of vertices $s \in S$ such that $s$ is reachable from $W$ via paths in $G^c[S]$. Since $W$ is connected in $G^c[S]$, the graph $G^c[S']$ is connected.

Let $U$ be a maximal independent set of $G^c[S']$. We claim that $|U| \ge |W|$. Note that for every $u \in U$, there is at most one $w \in W$ with $d(u, w) \le c$ (since $W$ is independent in $G^c$). So if $|U| < |W|$, there must exist some $w \in W$ which has $d(u, w) > c$ for all $u \in U$. Since $W \subseteq S'$, this would contradicting maximality of $W$.

Next we claim that $G^{3c}[U]$ is connected. For, consider any pair $u, u' \in U$. Since $U \subseteq S'$ and $G^c[S']$ is connected, there is a path $w = x_1, x_2, \ldots, x_k = w'$ with $x_1, \ldots, x_k \in S'$ and $d_G(x_i, x_{i+1}) \le c$. By maximality of $U$, for each $i = 1, \ldots, k$ there is some $v_i \in U$ with $d_G(v_i, x_i) \le c$. Now note that $w, v_1, v_2, \ldots, v_k, w'$ is a path in $G^{3c}[U]$. □

**Proposition A.5.** *Whp, $R$ is $c, m$-backbone-free for $m = \Omega(\log n)$.*

*Proof.* First, observe that if there is a $c, m'$-backbone $U'$ in $R$ of size $m' > m$, then one can form remove nodes as needed to form a $c, m$-backbone $U$. Thus, it suffices to show that there are no $c, m$-backbones in $R$.

Consider $x \in V$. There are at most $(e\Delta)^{3cm}$ possible backbones of size $m$ including $x$. Since a backbone is an independent of $G^{3c}$, each such backbone survives to $R$ with probability $p^m$. Summing over all $x$, we see that the expected number of surviving $c, m$-backbones is at most $n(e\Delta)^{3cm} p^m \le n(e\Delta)^{-cm}$; this is smaller than $n^{-100}$ for $m \ge \Omega(\frac{\log n}{\log \Delta})$. □

**Theorem A.6.** *Whp every connected component of $G[R]$ has size at most $O(\Delta^{2c} \log n)$.*

*Proof.* Let $T$ be a connected component of $G[R]$ and let $W$ be chosen to be maximal such that $W \subseteq T$ and $W$ is an independent set of $G^c$. Since $G^c$ has maximum degree $\Delta$, we have $|W| \ge |T|/(\Delta + 1)^c$.

We also claim that $G^{3c}[W]$ is connected. For, consider any vertices $x, x' \in T$. There must exist a path $x = y_1, y_2, \ldots, y_k = x'$ in $G[R]$. By maximality of $U$, for each $i = 1, \ldots, k$ there is some $v_i \in U$ with $d_G(v_i, x_i) \le c$ (otherwise one could add $x_i$ to $U$). Thus $x, v_1, v_2, \ldots, v_k, x'$ is a path in $G^{3c}[W]$.

So, $W$ is a $c, m$-backbone with $m \ge |T|/(\Delta + 1)^c$. By Proposition A.5, whp we have $m \le O(\log n)$. This implies that $|T| \le m(\Delta + 1)^c \log n \le O(\Delta^{2c} \log n)$. □

**Theorem A.7.** *Whp, one can obtain an $(O(\log \log n), O((\log \log n)^2)$-network-decomposition of $G[R]^r$ in $r 2^{O(\sqrt{\log \log n})}$ rounds, for any integer $r \ge 1$.*

*Proof.* We begin by finding a $(2, O(\log\log n))$-ruling set $W$ for the graph $J = G^c[R]$. This step can be performed in $O(\log\log n)$ rounds using the algorithm of Schneider, Elkin, Wattenhoffer, and succeeds with probability at least $1 - n^{-100}$. This ensures that every $v \in R$ has distance $d_J(v, W) \leq t = \Theta(\log\log n)$.

For each $v \in R$, let $s(v)$ be the vertex $w \in W$ which minimizes $d_G(v, w)$ (breaking ties arbitrarily). Since $W$ is a ruling set we have $d_J(v, s(v)) \leq t$, and this implies $d_G(v, s(v)) \leq ct$.

Now consider the graph $H$, on vertex set $W$, and with an edge $(w, w')$ if there are $v, v' \in R$ with $s(v) = w, s(v') = w'$ and $d_{G[R]}(v, v') \leq r$.

We claim that every connected component of $H$ must have size at most $m = \log n$. For, suppose that $T \subseteq W$ is connected in $H$. We claim that $T$ is connected in $J$. Since $W$ is a connected component of $H$, it suffices to show that for any edge of $H$ between vertices $w, w' \in T$, there is a path from $w$ to $w'$ in $J$. By definition, there are $v, v' \in R$ with $s(v) = w, s(v') = w'$ and $v \sim v'$. So $d_J(v, w) \leq t < \infty$ and $d_J(v', w') \leq t < \infty$. Also, $d_{G[R]}(v, v') \leq r < \infty$ and thus $d_J(v, v') < \infty$ as well.

In addition, since $W$ is a ruling set of $J$, it must be that $T$ is an independent set of $G^c$. Applying Proposition A.4 with $S = R$, we see that $G$ has a $c$-backbone $U \subseteq R$ of size $|U| \geq |T|$; by Proposition A.5 this implies whp that $|T| \leq O(\log n)$.

Since every connected component of $H$ has size $O(logn)$, we can use the deterministic algorithm of [PS95] in $2^{O(\sqrt{\log\log n})}$ rounds to obtain an $(\lambda, D)$-network-decomposition of $H$ with $\lambda, D \leq O(\log\log n)$. These are rounds with respect to communication on the graph $H$, each of which can be simulated in $O(rt)$ rounds on the graph $G$. Thus, this step requires $r2^{O(\sqrt{\log\log n})}$ communications rounds in $G$ in total.

Let $X_1, \ldots, X_\lambda$ denote the color classes of this decomposition. Let $L = G[R]^r$; we now generate a network-decomposition $X'_1, \ldots, X'_\lambda$ of the graph $L$, specifically we set $X'_j = s^{-1}(X_j)$..

In order to show this decomposition works, consider some $X'_j$ and consider the induced subgraph $K = L[X'_j]$. Now observe that for any vertex $x \in K$, we have $d_K(x, s(x)) \leq ct$. For by definition, $s(x)$ is the closest vertex of $W$ to $x$ in $G[R]$. By definition of $s$, every vertex $y$ along the shortest path from $x$ to $s(x)$ will also have $s(y) = s(x)$. Since $d_G(x, s(x)) \leq ct$, this implies $d_K(x, s(x)) \leq ct$ as well.

Let $v, v' \in K$ with $d_K(v, v') < \infty$; we need to show that $d_K(v, v') \leq O((\log\log n)^2)$.

There is a path $v = a_1, a_2, \ldots, a_\ell = v'$ in $K$. For each $i = 1, \ldots, \ell$ let $w_i = s(a_i)$ where each $w_i \in X_j$. Each $(w_i, w_{i+1})$ is an edge of $H[X_j]$ and so $w_1, w_\ell$ are connected in $H[X_j]$.

Since $H[X_j]$ has diameter $O(\log\log n)$, there must exist a path $w_1 = u_1, \ldots, u_k = w_\ell$ in $H[X_j]$ with $k \leq O(\log\log n)$. Also, since each $(u_i, u_{i+1})$ is an edge in $H$, there must be $b_1, \ldots, b_k \in R$ such that $u_i = s(b_i)$ and $d_J(b_i, b_{i+1}) \leq r$. Since $s(b_i) \in X_j$, we see that $b_1, \ldots, b_k \in X'_j$.

Since $d_{G[R]}(b_i, b_{i+1}) \leq r$, we have that $b_i \sim b_{i+1}$ in the graph $L$, which implies that $d_K(b_i, b_{i+1}) \leq 1$.

We may thus compute $d_K(u_i, u_{i+1}) \leq d_K(u_i, b_i) + d_K(b_i, b_{i+1}) + d_K(b_{i+1}, u_{i+1}) \leq 2ct + 1$. This implies that $d_K(w_1, w_\ell) \leq k(2ct+1)$, and therefore $d_K(a_1, a_\ell) \leq d_K(a_1, w_1) + d_K(w_1, w_\ell) + d_K(w_\ell, a_\ell) \leq 2ct + k(2ct + 1) \leq O((\log\log n)^2)$. □

**Theorem 5.1.** *Suppose each vertex survives to a residual graph $R$ with probability at most $(e\Delta)^{-4c}$, and this bound holds even for adversarial choices for the random bits outside the $c$-hop neighborhood of $v$ for some constant $c \geq 1$.*

*Suppose that the residual problem can be solved via a $\mathsf{SLOCAL}(r)$ procedure. Then the residual problem can be solved whp in the LOCAL model with $O(r\Delta(G^r) + r\log^* n)$ rounds. Furthermore, if $r \leq 2^{O(\sqrt{\log\log n})}$, then the residual problem can be solved whp in the LOCAL model with $2^{O(\sqrt{\log\log n})}$ rounds.*

*Proof.* After we run the first phase, whp every connected component of $G[R]$ has size at most $N$.

For the first result, note that by Proposition 3.3, the randomized algorithm $\mathcal{A}$ leads to a deterministic algorithm in $O(r\Delta(G^{2r}) + r\log^* N)$ rounds. We can run this algorithm on each component of the residual graph.

For the second result, use Theorem A.7 to get a $(D, C)$-network decomposition of $G[R]^r$ in $2^{O(\sqrt{\log\log n})}$ rounds, with $C = O(\log\log n)$ and $D = O((\log\log n)^2)$. By Proposition 3.2, this allows us to run $\mathcal{A}$ in $O(CDr) = 2^{O(\sqrt{\log\log n})}$ rounds deterministically. □

We note that Theorem 5.1 can be used to obtain cleaner formulations of a number of previous shattering-based algorithms. We give one simple example, based on an algorithm of [Gha16] for maximal independent set.

**Proposition A.8** ( [Gha16]). *Let $G$ be a graph of maximum degree $\Delta$. There is a LOCAL algorithm to find an MIS of $G$ in $O(\log\Delta) + 2^{\sqrt{O(\log\log n)}})$ rounds whp.*

*Proof.* The algorithm of [Gha16] has two phases. The first, randomized phase has the following guarantee: if we run it for $\Omega(\log\Delta + \log(1/\epsilon))$ rounds, then any given vertex $v$ is removed from the residual (either due to $v$ or a neighbor going into the MIS) with probability $1 - \epsilon$; furthermore, this depends only on the random bits within a two-hop neighborhood of $v$.

We now apply Theorem 5.1. In particular, setting $\epsilon = \Delta^{-10}$ and $c = 2$, we see that a vertex survives to the residual with probability $p = \Delta^{-10} \leq (e\Delta)^{-4c}$. Also, the residual problem clearly can be solved as an $\mathsf{SLOCAL}(r)$ procedure for $r = 1$ — each vertex checks if a neighbor is already in the MIS, and if not, enters the MIS itself. So, Theorem 5.1 shows that the residual problem can be solved in $2^{O(\sqrt{\log\log n})}$ rounds whp. So the overall time is $O(\log\Delta)$ (for the first phase) plus $2^{O(\sqrt{\log\log n})}$ (for the residual phase.) □

We also note that Proposition 3.3 gives us a rich source of SLOCAL algorithms. In particular, by combining Proposition 3.3 with Theorem 5.1, we are able to use Las Vegas algorithms for the second phase of shattering algorithms.

**Proposition 5.2.** *Suppose each vertex survives to a residual graph $R$ with probability at most $(e\Delta)^{-4c}$, and this bound holds even for adversarial choices for the random bits outside the $c$-hop neighborhood of $v$ for some constant $c \geq 1$.*

*Suppose that there is a $\mathsf{ZLOCAL}(r)$ algorithm $\mathcal{A}$ on $G[X]$, for any vertex subset $X \subseteq V$ of size $|X| \leq N = O(\Delta^{2c}\log n)$. Then the residual problem can be solved in $O(r\Delta(G^{2r}) + r\log^* n)$ rounds. Furthermore, if $r \leq 2^{O(\sqrt{\log\log n})}$, then the residual problem can be solved in $2^{O(\sqrt{\log\log n})}$ rounds.*

*Proof.* For the first result, note that by Proposition 3.3, the randomized algorithm $\mathcal{A}$ leads to a deterministic algorithm in $O(r\Delta(G^{2r}) + r\log^* N)$ rounds. We can run this algorithm on each component of the residual graph. For the second result, note that by Proposition 3.3, the randomized algorithm $\mathcal{A}$ leads to a $\mathsf{SLOCAL}(4r)$ algorithm; now apply Theorem 5.1. □