2018 IEEE 59th Annual Symposium on Foundations of Computer Science

On Derandomizing Local Distributed Algorithms

Mohsen Ghaffari
ETH Zurich
Email: ghaffari@inf.ethz.ch

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

Fabian Kuhn[†]
University of Freiburg
Email: 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 combine the method of conditional expectation with network decompositions to obtain a generic and clean recipe for derandomizing LOCAL algorithms. This leads to significant improvements on a number of problems, in cases resolving known open problems. Two main results are:**

- **An improved deterministic distributed algorithm for hypergraph maximal matching, improving on Fischer, Ghaffari, and Kuhn [FOCS'17]. This yields improved algorithms for edge-coloring, maximum matching approximation, and low out-degree edge orientation. The last result gives the first positive resolution in the Open Problem 11.10 in the book of Barenboim and Elkin.**
- **Improved randomized and deterministic distributed algorithms for the Lovász Local Lemma, which get closer to a conjecture of Chang and Pettie [FOCS'17].**

***Keywords*-LOCAL; distributed algorithms; hypergraph matching; Lovász Local Lemma; derandomization**

I. Introduction and Related Work

The gap between deterministic and randomized local distributed algorithms constitutes a central open question in the area of *distributed graph algorithms*. For many 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 [1] and Alon, Babai, and Itai [2]. However, obtaining a $\operatorname{polylog} n$-time deterministic algorithm remains an intriguing open problem for 30 years now, since it was posed by Linial [3]. The best known deterministic round complexity is $2^{O(\sqrt{\log n})}$, due to Panconesi and Srinivasan [4].

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 [5], 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 ([5]) 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. Most of the recent developments in randomized algorithms use the *shattering technique* [6], [7], [8], [9], [10], [11] 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.

Before we explain our results, we review Linial's LOCAL model [3], [12], which is the standard synchronous message passing model of distributed computing, and a sequential variant called SLOCAL, introduced recently by Ghaffari, Kuhn, and Maus[13].

The LOCAL Model: The communication network is abstracted as an undirected n-node graph $G = (V, E)$ with maximum degree Δ. 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. 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* is the number of rounds that required until all nodes terminate. For randomized algorithms, each node can in addition produce an arbitrarily long private random bit string before the computation starts.

The SLOCAL Model: There are two main obstacles to developing LOCAL algorithms: *locality* and *symmetry breaking*. In order to understand their roles separately, Ghaffari, Kuhn, and Maus [13] introduced the SLOCAL model, in which symmetry breaking is free and only locality becomes a bottleneck. In the SLOCAL model, the nodes are processed sequentially, in an arbitrary (adversarially chosen) order. When node v is processed, it reads its r-hop neighborhood (including all the information that has been locally stored by the previously processed nodes) and it computes and locally stores its output y_v and potentially

[†]Supported by ERC Grant No. 336495 (ACDC).

[1]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.

2575-8454/18/$31.00 ©2018 IEEE
DOI 10.1109/FOCS.2018.00069

additional information. We call the parameter r the *locality* of an SLOCAL algorithm.

The SLOCAL model can be seen as a natural extension 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.

The SLOCAL model is inherently sequential. It is useful because there are transformations from SLOCAL algorithms to LOCAL algorithms, which handles symmetry breaking in a "black-box" or generic way. By analyzing SLOCAL algorithms, we are therefore able to treat a number of diverse LOCAL problems in a unified and abstract fashion.

A. Our Contributions, Part I: Derandomization

In the first part of this paper, we present a simple and clean recipe for derandomizing local distributed algorithms.

Theorem I.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 round complexity $\Delta^{O(r)} + O(r \log^* n)$, or $r \cdot 2^{O(\sqrt{\log n})}$.*

We show Theorem I.1 by using the method of conditional expectation. In the SLOCAL model, the nodes are processed sequentially, and the method of conditional expectation allows the randomness of the nodes (in the randomized distributed algorithm) to be fixed in this order. In order to apply the method of conditional expectation, it is critical that the problem solution is locally checkable. We note that this assumption is generally necessary for efficiently derandomizing distributed algorithms. In Section VII-B, we provide a simple problem that is not locally checkable, which has a constant-time randomized distributed algorithm, but where the best deterministic LOCAL algorithm has at least polynomial locality.

The transformation from the SLOCAL model to the LOCAL model by using network decompositions follows from [13]. It is well-known that network decompositions are a powerful generic tool to obtain efficient distributed algorithms. Theorem I.1 proves that network decompositions are even sufficiently powerful to turn any randomized distributed algorithm for a locally checkable problem into an efficient deterministic algorithm.[2] The $2^{O(\sqrt{\log n})}$ term in the round complexity of Theorem I.1 is due to the currently best-known round complexity of computing $(O(\log n), O(\log n))$-network decompositions [4]. Because of this overhead, unfortunately, the deterministic algorithms obtained from this derandomization method may be much less efficient than their randomized counterparts. However, we can still get many derandomized algorithms that are far more efficient than existing algorithms.

1) Hypergraph Maximal Matching and Ramifications: Hypergraph maximal matching was recently pointed out by Fischer, Ghaffari, and Kuhn [14] as a clean subroutine admitting reductions from diverse classic problems. As the first concrete implication of our derandomization technique, we obtain an improved deterministic algorithm:

Theorem I.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 Δ and rank at most r (the rank is the maximum number of vertices in a hyperedge).*

The algorithm of [14] has complexity $(\log \Delta)^{O(\log r)} \cdot \log n$, which is efficient only for hypergraphs of essentially constant rank. Theorem I.2 gives efficient algorithms for $r = \operatorname{polylog} n$.

Using known reductions [14], this algorithm leads to efficient deterministic algorithms for a number of problems, including edge coloring, approximate maximum matching, and low out-degree orientation:

Corollary I.3. *There is a deterministic distributed algorithm that computes a $(2\Delta - 1)$-list-edge-coloring in $O(\log^2 n \log^4 \Delta)$ rounds.*

Corollary I.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 I.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 λ, in $O(\log^{10} n \log^5 \Delta/\varepsilon^9)$ rounds.*

Theorem I.3 gives an alternative, more efficient, solution for $(2\Delta - 1)$-edge coloring, which was a well-known open problem since 1990s (listed as Open Problem 11.4 of [5]) and was resolved recently by Fischer et al.[14]. Moreover, Theorem I.5 gives the first positive resolution of Open Problem 11.10 of [5]. In a recent paper, Ghaffari, Kuhn, Maus, and Uitto [15] gave efficient deterministic algorithms for edge coloring with $(1+\varepsilon)\Delta$ and $\frac{3}{2}\Delta$ colors; these results critically depend on the stronger maximum matching approximation guaranteed by Theorem I.4.

2) The Lovász Local Lemma: The Lovász Local Lemma (LLL) is a powerful probabilistic tool; at a high level, it states that if one has a probability space Ω 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 $\mathcal{B}$

[2] Actually, together with the fact that $(O(\log n), O(\log n))$-decompositions can be computed in randomized polylogarithmic time in the LOCAL model, Theorem I.1 exactly captures the set of problems for which network decompositions can be used.

occurs; 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 at most 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 graph algorithms, such as frugal or defective vertex-colorings. In addition, as Chang & Pettie noted [16], the LLL plays a important role in the overall landscape of LOCAL algorithms: 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 problem. They further conjectured that the LLL could be solved in time $O(\log \log n)$ (matching a lower bound of [17]), 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 [18] 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 algorithm running in $O(\log^2 n)$ rounds. There have been a number of other algorithms developed specifically in the context of the LOCAL model [19], [10]. 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).

More recently, Fischer and Ghaffari [10] described an algorithm running in $2^{\mathrm{poly}(d)+O(\sqrt{\log \log n})}$ rounds, under the pLLL criterion $p(ed)^{32} < 1$. Despite the exponential dependence 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.

We give new LLL algorithms that can be faster than those of [10] and that can be used for higher-degree graphs. The main algorithm is summarized as follows:

Theorem I.6. *Let i be an integer in the range $1 \leq i \leq \log^* n - 2\log^* \log^* n$. If $20000d^8p \leq 1$, then a configuration avoiding $\mathcal{B}$ can be found by a randomized* LOCAL *algorithm in time $\exp^{(i)}\big(O(\log d + \sqrt{\log^{(i+1)} n})\big)$ and by a deterministic* LOCAL *in time $\exp^{(i)}\big(O(\log d + \sqrt{\log^{(i)} n})\big)$.*

For example, with $i = 1$, the randomized algorithm runs in $\mathrm{poly}(d) \times 2^{O(\sqrt{\log \log n})}$ rounds—this improves the dependence of d by an exponential factor compared to [10]. For $i = 2$, the algorithm can run in time $2^{2^{O(\sqrt{\log \log \log n})}}$, under the condition that $d \leq 2^{\sqrt{\log \log \log n}}$. This makes partial progress toward showing the conjecture of [16] for the running time of LLL on bounded-degree graphs.

The final 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 requires that the bad-events satisfy a different property which we refer to as *bounded fragility*. We show the following result:

Theorem I.7. *Suppose that every bad-event $B \in \mathcal{B}$ has fragility at most $F \leq e^{-10}d^{-12}$. Then there is a randomized algorithm to find a configuration avoiding $\mathcal{B}$ in $2^{O(\sqrt{\log \log n})}$ rounds, w.h.p.*

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*[6], [7], [8], [9], [10], [11]. 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. This gives us two new, randomized LLL algorithms. The first runs in time $2^{O(\sqrt{\log \log n})}$ for $d \leq 2^{\sqrt{\log \log n}}$. The second runs in time $O(d^2 + \log^* n)$ for LLL instances satisfying a stronger slack condition $p \leq 1/\operatorname{polylog}(n)$. These can be combined via the bootstrapping methodology of [10], giving us the stated runtime bounds.

3) The Role of Randomization in the SLOCAL Model: Besides the above concrete improvements, our derandomization method has bigger-picture implications on the role of randomization in the LOCAL model. From Theorem I.1, we show that randomized and deterministic complexities are very close in the SLOCAL model:

Theorem I.8. *Any randomized* SLOCAL *algorithm with locality $r(n)$ for a locally checkable problem can be transformed to a deterministic* SLOCAL *algorithm with locality $O(r(n)\log^2 n)$.*

As we have discussed, the SLOCAL model aims to decouple the challenges of locality from those of symmetry breaking. This was with the intuitive hope that, while the latter seems to naturally benefit from randomization, locality on its own should not need randomization. This is partially validated by Theorem I.8, as the SLOCAL model can only gain polylogarithmic factors from randomness. We remark that, if we do care about logarithmic factors, then a gap appears also in SLOCAL:

Theorem I.9. *Sinkless orientation 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 randomized and deterministic complexities in the SLOCAL

model, akin to those observed in the LOCAL model—sinkless orientation in LOCAL requires $\Theta(\log n)$ rounds deterministically[20] and $\Theta(\log \log n)$ rounds randomly [17], [20], [9]. We find it surprising that this gap appears an exponential lower in the SLOCAL model.

B. Our Contributions, Part II: Limits of Derandomization

In the second part of the paper, we exhibit limitations on derandomization technique. We present 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, we show that these problems are P-SLOCAL-complete, as defined by Ghaffari, Kuhn, and Maus [13].

For the above three problems, rather satisfactory polylog-time randomized LOCAL algorithms have been known for many years, e.g., [21], [22], [23], [24], [25], [26], [27], [28]. However, there are no known efficient deterministic algorithms. We show that devising efficient deterministic algorithms for them may be hard. Concretely, a polylogarithmic-time deterministic algorithm for any of these problems can be transformed into a polylogarithmic-time deterministic LOCAL algorithm for computing a $(O(\log n), O(\log n))$ network decomposition and thereby imply polylogarithmic-time deterministic LOCAL algorithms for all P-SLOCAL problems. The latter class includes, most notably, computing an MIS. Hence, devising polylogarithmic-time deterministic algorithms for these problems is at least as hard as doing so for MIS, which remains a well-known open problem in the area since Linial explicitly asked for it in 1987 [3].

We would like to highlight an implication of the hardness of neighborhood cover. This has been a central problem in the study of local distributed graph algorithms since the work of Awerbuch and Peleg[29], closely related to another central problem, *network decompositions*, introduced by Awerbuch et al.[30]. By classic results of [31], 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 show P-SLOCAL-completeness for the problem of computing an MIS of a type of star. Significantly, this problem 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.

II. 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 Δ 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. For simplicity, we assume that all nodes also know a common polynomial upper bound on the number of nodes n.

Complexity Classes: 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 [13].

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

We distinguish two kinds of 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 flag $F_v \in \{0, 1\}$, which serves as an indicator of whether the algorithm failed locally at v. If $F_v = 0$ for every node v, it is guaranteed that the computed output solves $\mathcal{P}$. Furthermore, it is guaranteed that $\sum_{v \in V} \mathbb{E}[F_v] < 1$.

This definition of Las Vegas algorithms initially appear very different from standard notions of success for randomized LOCAL algorithms. In Section III-A, we show how this definition is equivalent (up to polylogarithmic factors) with more standard definitions. However, we adopt this definition in terms of indicator variables F_v for technical reasons.

- **RLOCAL**$(t(n))$: Distributed graph problems that can be solved by a *randomized Monte Carlo algorithm* in the **LOCAL** model with time complexity $t(n)$.
- **ZLOCAL**$(t(n))$: Distributed graph problems that can be solved by a *randomized Las Vegas algorithm* in the **LOCAL** model with time complexity $t(n)$.
- **RSLOCAL**$(t(n))$: Distributed graph problems that can be solved by a *randomized Monte Carlo* **SLOCAL** *algorithm* with locality $t(n)$.

We clearly have $\mathsf{RLOCAL}(t(n)) \subseteq \mathsf{ZLOCAL}(t(n))$. In addition, if the validity of a solution to a graph problem can be locally checked by exploring the d-neighborhood of each node (cf. [32] for a formal definition of local decision problems), then a $\mathsf{RLOCAL}(t)$ algorithm yields a $\mathsf{ZLOCAL}(t+d)$ algorithm. We will show in Section VII-B an example of a problem in $\mathsf{RLOCAL}(0)$ but not in $\mathsf{ZLOCAL}(o(\sqrt{n}))$.

We often think of an LOCAL algorithm as *efficient* if it has time complexity at most $\operatorname{polylog} n$. We therefore define **P-LOCAL**, **P-SLOCAL**, **P-RLOCAL**, and **P-ZLOCAL** as respectively $\mathsf{LOCAL}(\operatorname{polylog} n)$, $\mathsf{SLOCAL}(\operatorname{polylog} n)$, $\mathsf{RLOCAL}(\operatorname{polylog} n)$, and $\mathsf{ZLOCAL}(\operatorname{polylog} n)$.

Network decomposition: This graph structure plays a central role in this paper and LOCAL algorithms.

Definition II.1 (Network Decomposition)**.** [30] *A $\big(d(n), c(n)\big)$-decomposition of an n-node graph $G = (V, E)$ is a partition of $V = V_1 \sqcup \cdots \sqcup V_{c(n)}$, with the property that each induced subgraph $G[V_i]$ has diameter at most $d(n)$.*

It was shown in [29], [28] that every graph has an $\big(O(\log n), O(\log n)\big)$-decomposition. Further, as shown in [13], this directly leads to an $\mathsf{SLOCAL}(O(\log^2 n))$-algorithm for computing an $\big(O(\log n), O(\log n)\big)$-decomposition. In [28], 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 [13], it was shown that the problem of computing a $(d(n), c(n))$-decomposition is P-SLOCAL-complete for $d(n), c(n) = O(\log^k n)$ and any constant $k \geq 1$. As a consequence, P-SLOCAL $\subseteq$ P-ZLOCAL. The best deterministic distributed algorithm to compute a $\big(O(\log n), O(\log n)\big)$-decomposition, due to Panconesi and Srinivasan [4], has time complexity $2^{O(\sqrt{\log n})}$.

III. Basic Derandomization of Local Algorithms

In the previous section, we 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 Monte Carlo graph algorithms (including, for instance, all locally-checkable problems), can be converted into Las Vegas algorithms. The formal statement of our basic derandomization result is the following:

Theorem III.1. *Let $\mathcal{P}$ be a graph problem which has a $\mathsf{ZLOCAL}(r)$ algorithm A. When running 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 A with those values, it solves $\mathcal{P}$.*

Proof: Consider a randomized run of algorithm A. In it, every node v sets a flag F_v. Let us further define the random variable $F := \sum_{v \in V} F_v$ such that $F \geq 1$ iff 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 SLOCAL-algorithm A' via the method of conditional expectation. Suppose that A' processes the nodes in some arbitrary order $v_1, \ldots, v_n$. When processing node v_i, A' needs to fix the value of R_{v_i}. The algorithm A' will choose values ρ_i for each $i \in \{0, \ldots, n\}$ to ensure 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 Equation (1) is equal to the final value of $\sum_v F_v$ when running A with these values for R_{v_i}. Because $\mathbb{E}[F] < 1$, Equation (1) thus implies that $\mathbb{E}[F \,|\, R_{v_1} = \rho_1, \ldots, R_{v_n} = \rho_n] < 1$ and thus the algorithm succeeds.

It remains to show how to set ρ_i to satisfy Equation (1). 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 ρ_i to minimize $\mathbb{E}[F \,|\, R_{v_1} = \rho_1, \ldots, R_{v_i} = \rho_i]$. Further note that the output of any node 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. Therefore,

$$\mathbb{E}\Big[F_v \,\Big|\, \bigwedge_{u \in S} (R_u = \rho_u)\Big] = \mathbb{E}\Big[F_v \,\Big|\, \bigwedge_{\substack{u \in S \\ d_G(u,v) \leq r}} (R_u = \rho_u)\Big] \tag{2}$$

So we can fix the value of ρ_i by setting:

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

Thus 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. ∎

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

Proposition III.2. *Any algorithm $A \in \mathsf{SLOCAL}(r) \cup \mathsf{ZLOCAL}(r)$ to solve a graph problem $\mathcal{P}$ on G yields a deterministic LOCAL algorithm in $\min\Big\{r \cdot 2^{O(\sqrt{\log n})}, O\big(r \cdot (\Delta(G^{2r}) + \log^* n)\big)\Big\}$ rounds.*

Proof: We show this only for $A \in \mathsf{SLOCAL}(r)$; the case of $A \in \mathsf{ZLOCAL}(r)$ is similar. 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 [4]. In [13], it shown how a $(d(n), c(n))$-network decomposition allow us to simulate a $\mathsf{SLOCAL}(r(n))$ algorithm in $\mathsf{LOCAL}(r(n)(d(n)+1)c(n))$ rounds. So we can simulate A in $O(r \log^2 n) \leq r 2^{O(\sqrt{\log n})}$ rounds. For the second bound, we compute a $\Delta(G^{2r})$-coloring of G^{2r} in time $O\big(r(\Delta(G^{2r}) + \log^* n)\big)$ by using the algorithm of [33].

This can be viewed as as $(0, \Delta(G^{2r}))$-decomposition of G^{2r}, which allows us to simulate A in $O(r\Delta(G^{2r}))$ rounds. ■

Proposition III.3. $\mathsf{ZLOCAL}(r) \subseteq \mathsf{SLOCAL}(4r)$.

Proof of Theorem I.8*:* Consider an n-node graph G and an $\mathsf{RSLOCAL}(r(n))$ algorithm A for a graph problem $\mathcal{P}$. We compute an $\big(O(\log n), O(\log n)\big)$-network-decomposition of $G^{r(n)}$ in $O(r(n)\log^2 n)$ round using the randomized algorithm of [28]. This transforms A into a randomized distributed algorithm with time complexity $O(r(n)\log^2 n)$. Because $\mathcal{P}$ is locally checkable, Theorem III.3 converts this randomized distributed algorithm into a deterministic $\mathsf{SLOCAL}[O(r(n)\log^2 n)]$ algorithm. ■

A. Alternative Definition of Las Vegas Algorithms

The definition of Las Vegas algorithms is non-standard and does not seem to definitions in e.g. complexity theory. We next show how our definition relates to other notions of zero-error randomization.

Definition III.4 (Zero-error distributed algorithm)**.** *A zero-error distributed algorithm is a randomized* LOCAL *algorithm A for a graph problem $\mathcal{P}$ such that when all nodes terminate, A always computes a correct solution for $\mathcal{P}$.*

Proposition III.5. *For a graph problem $\mathcal{P}$, the following are equivalent:*

1) $\mathcal{P} \in \mathsf{P\text{-}ZLOCAL}$
2) $\mathcal{P} \in \mathsf{P\text{-}SLOCAL}$
3) *$\mathcal{P}$ has a zero-error distributed algorithm with the property that for any integer $\gamma \geq 1$, the algorithm terminates within $\gamma\,\mathrm{polylog}(n)$ rounds with probability at least $1 - n^{-\gamma}$.*
4) *$\mathcal{P}$ has a zero-error distributed algorithm with expected termination time $\mathrm{polylog}(n)$.*

IV. Hypergraph Maximal Matching

We consider a hypergraph H on n nodes, maximum degree at most Δ, 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 partitioning the hyperedges into two classes, so each node's neighborhood is roughly split into two equal parts. This degree splitting procedure uses the derandomization lemma Theorem III.1 as its core tool.

Lemma IV.1. *Let H be an n-node hypergraph with maximum degree at most Δ and rank at most r. Then, for every $\varepsilon > 0$, there is a deterministic $O(r\log(n\Delta)/\varepsilon^2)$-round* LOCAL *algorithm to compute a red/blue coloring of the hyperedges of H, with the property that for every node $v \in V$ satisfying $\deg_H(v) \geq \delta = \frac{8\ln(n\Delta)}{\varepsilon^2}$, 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.*

Proof: As a first step, we reduce the problem on H to the hypergraph splitting problem on a low-degree hypergraph H'. To construct H', we divide each node of H of 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. The hypergraph H' has at most $n\lceil \Delta/\delta \rceil$ vertices, maximum degree 2δ. It suffices to solve the problem on H', as this also solves it for H.

Instead of directly working on H', it is more convenient to define the algorithm on its bipartite incidence 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.

We first claim that such a (ε, δ)-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 δ, the coloring does not need to satisfy any condition. Otherwise, we know that the degree of u is in $[\delta, 2\delta)$. A Chernoff bound calculation shows $\mathbb{P}\left(X_u < (1-\varepsilon) \cdot \frac{\deg_{H'}(u)}{2}\right) \leq \frac{1}{(n\Delta)^2}$ and similarly for Y_u.

As $|U_B| \leq n\Delta$, the expected number of failing nodes is therefore at most $n\Delta \cdot \frac{1}{(n\Delta)^2} < 1$. This procedure can be computed and 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 III.2 gives a deterministic algorithm in $O(r\Delta(B^2) + r\log^* n) \leq O(r\log(n\Delta)/\epsilon^2)$ rounds. ■

Lemma IV.2. *Given a rank-r hypergraph $H = (V, E)$ of maximum 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 Sketch: We reduce the degree of H by repeatedly applying the hypergraph degree splitting of Theorem IV.1. After $O(\log\Delta)$ iterations, the degree is reduced to $O(\log n)$ and every vertex in U_+ still has degree $\Theta(\log n)$.

To finish, we find a matching M in the residual graph H'.

This step uses standard deterministic MIS algorithms. Note that the line graph of H' has maximum degree $O(r \log n)$; it is known that one can find an MIS in a graph G in $O(\Delta(G)+\log^* n)$ rounds. ∎

By applying the hypergraph degree splitting of Lemma IV.2 repeatedly, we can now prove Theorem I.2.

Theorem I.2. *Let H be an n-node hypergraph with maximum degree at most Δ 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: Each application Theorem IV.2 reduces 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 finish with a deterministic MIS algorithm on the line graph. ∎

A. Implications of Hypergraph Matching

Proof of Theorem I.3*:* This follows immediately from Theorem I.2, combined with a reduction of Fischer, Ghaffari, and Kuhn [14] that reduces $(2\Delta-1)$-list-edge-coloring of a graph $G=(V,E)$ to hypergraph maximal matching on hypergraphs of rank 3, with $O(|V|+|E|)$ vertices and maximum degree $O(\Delta(G)^2)$, The complexity is thus $O(\log^2 n \log^4 \Delta)$. ∎

Proof of Theorem I.4*:* The algorithm follows the approach of Hopcroft and Karp [34] based on repeated augmentation of the matching. For each $\ell = 1$ to $2(1/\varepsilon)-1$, we find a maximal set of vertex-disjoint augmenting paths of length ℓ, and we augment them all.

To compute a maximal set of vertex-disjoint augmenting paths of a given length ℓ, we form the hypergraph H on vertex set V, with a hyperedge for every augmenting path $v_1, \ldots, v_\ell$. This hypergraph has rank at most ℓ and maximum degree at most Δ^ℓ; every maximal matching of H corresponds to a length-ℓ vertex-disjoint augmenting path. Moreover, a single round of communication on H can be simulated in $O(\ell)$ rounds of the base graph G. Placing these parameters in the bound of Theorem I.2, we get complexity $O(\ell \cdot \ell^2 \log(n\Delta^\ell) \log n \log^4(\Delta^\ell)) = O(\log^2 n \log^5 \Delta/\varepsilon^8)$ rounds for each value of ℓ. Thus, the overall complexity is $O(\log^2 n \log^5 \Delta/\varepsilon^9)$. ∎

Proof of Theorem I.5*:* We follow the approach of Ghaffari and Su [9], which iteratively improves the orientation by reducing its maximum out-degree via another type of augmenting path. 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.

Let G_0 be the graph with our initial arbitrary orientation. 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 $3+i$, and then we reverse all these augmenting paths. The resulting graph is called G_{i+1}. Ghaffari and Su showed that at the end of this process, the graph G_ℓ has maximum out-degree D [9, Lemma D.9].

To compute a maximal set of edge-disjoint augmenting paths of length in G_i, we use use Theorem I.2 by viewing each edge as one vertex of our hypergraph, and each augmenting path as a hyperedge. By calculations similar to Theorem I.4, we get a a total complexity of $O(\log^{10} n \log^5 \Delta/\varepsilon^9)$. ∎

V. The Lovász Local Lemma

We will consider a somewhat restricted case of the LLL, which can be described as follows. The probability space Ω is defined by variables $X(1), \ldots, X(v)$ on some domain D. Every bad event $B \in \mathcal{B}$ 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 $A \sim B$. 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 $\mathbb{P}_\Omega(B) \leq p$, and $epd \leq 1$, then there is a positive probability that no $B \in \mathcal{B}$ occurs.

A distributed LLL algorithm is a key building-block for graph 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 coloring 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 r-hop neighborhood, where r is very small (and typically is $O(1)$). In this case, the dependency graph H is essentially the same as G^r. Bearing this structure in mind, we assume throughout that $|\mathcal{B}| = n$ and $d = \Delta + 1$.

A. 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 [18] 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. In [19], 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 [8]). 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 weaker than the LLL, is still usable to construct a number of combinatorial objects such as frugal colorings and defective colorings. We refer to this type of weakened LLL condition as a *polynomially-weakened LLL criterion* (pLLL).

More recently, Fischer & Ghaffari [10] have shown an algorithm running in $2^{O(\sqrt{\log \log n})}$ rounds, under the pLLL criterion $p(ed)^{32} < 1$, as long as $d < (\log \log n)^{1/5}$. Although the general LLL algorithm of [10] has a significant limitation on degree, they nevertheless used it 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, [17] 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 [35], Chang et al. have shown an $O(\log \log n)$ algorithm when the underlying graph G is a tree. In [36], Harris developed an $O(\log^3 n)$-round algorithm for a form of the LLL known as the Lopsided Lovász Local Lemma. Finally, there have been a number of parallel PRAM algorithms developed for the LLL, including [18], [37]; these are often similar to distributed LLL algorithms but are not directly comparable.

B. Graph Shattering and Bootstrapping

The LLL algorithms use the graph shattering technique. In the first phase, there is some random process which satisfies most vertices. In the second phase, we let $R \subseteq V$ denote the unsatisfied vertices. A deterministic algorithm (which can in turn be derived from a randomized algorithm) can then be used to solve the residual problem on each component of $G[R]$. We summarize this here:

Theorem V.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 $c \geq 1$.*

Then w.h.p each connected component of the residual graph has size at most $O(\Delta^{2c} \log n)$.

There is another important subroutine used in our LLL algorithm, referred to as *bootstrapping*. This is a technique wherein LLL algorithms can be used to "partially" solve a problem, generating a new LLL problem instance whose slack is "amplified." This technique was first introduced by [16], and extended to more general parameters in [10].

When developing our LLL algorithms, there are two ways of measuring success. First, there is the global guarantee: what is the probability that all $\mathcal{B}$ is avoided? Second, there is a local guarantee: for any fixed bad-event B, what is the probability that the variable assignment generated by our algorithm makes B be true? We say that the algorithm has *local failure probability* ρ if every $B \in \mathcal{B}$ has a probability at most ρ of being false.

Bootstrapping is a method of converting an r-round LLL algorithm to solve $\mathcal{B}$, which has some guaranteed local failure probability ρ, into a new LLL instance $\mathcal{C}$ which represents $\mathcal{B}$. The new instance $\mathcal{C}$ has parameters $p' = \rho$ and $d' = d^{2r}$.

The local success probability of an algorithm is closely linked to the network size parameter n. It is possible to run a LOCAL algorithm A running in time $r(n)$ with an alternate choice of n; we say in this case that we *bootstrap* A with parameter n' and write the resulting algorithm as $A^{[n']}$. This is often referred to as running A with a "fake" value of n.

Proposition V.2. *If $\Delta^{r(w)} \leq w$, then $A^{[w]}$ has local failure probability at most $2/w$.*

C. The LLL for Low-Degree Graphs

We now describe our main LLL algorithm, which is targeted at graphs of bounded degree; specifically, $d \leq 2^{O(\sqrt{\log \log n})}$. In analyzing this algorithm, 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 $\overline{D} = D \cup \{?\}$.

Given any vector $x \in \overline{D}^v$, and an event E on the space Ω, we define the *marginal probability* of E with respect to x as the probability that E holds, if all variables i with $X(i) = ?$ are resampled from the original distribution.

Our algorithm will use the same main subroutine of [10], which is inspired by sequential LLL algorithms of Molloy & Reed [38] and Pach & Tardos [39]. 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-events whose marginal probability under X' is non-zero. We use the following key facts about this algorithm:

Theorem V.3 ([10])**.** *The algorithm of [10] 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) *Every $B \in \mathcal{B}$ has $\mathbb{P}(B \in \mathcal{A}) \leq (ed)^{-8}$; furthermore, this probability bound holds even if the random bits made outside a two-hop radius of B are chosen adversarially.*

Lemma V.4. *If $20000d^8 p \leq 1$, then a configuration avoiding $\mathcal{B}$ can be found in $O(d^2) + 2^{O(\sqrt{\log \log n})}$ rounds w.h.p.*

Proof: Run the algorithm of [10] to obtain a partial solution $X' \in \overline{D}$. 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 V.3, each $B \in \mathcal{A}$ has marginal probability at most $2(ed)^8 p$. Our condition $20000d^8 p \leq 1$ ensures that $\mathcal{A}$ satisfies the the symmetric LLL criterion. The algorithm of [18] 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.

By Theorem V.3, each $B \in \mathcal{B}$ survives to the residual with probability $(ed)^{-8}$. By Theorem V.1 with $c = 2$, each connected component in the residual has size at most $N = O(\Delta^{2c} \log n)$. The algorithm of [18] gives a Las Vegas algorithm on each component in $O(\log^2 N) = O((\log \log n)^2)$ rounds. By Theorem V.1, therefore, the residual problem can be solved in $2^{O(\sqrt{\log \log n})}$ rounds. ∎

Lemma V.5. *Suppose* $20000d^{10}p \leq 1$ *and* $d \geq \log^* n$. *Then there is randomized* LOCAL *algorithm running in* $O(d^2)$ *rounds and with local failure probability at most* $e^{-\frac{1}{10000d^{10}p}}$.

Proof: Let $\lambda = \frac{1}{20000d^{10}p} \geq 1$. We run the algorithm of [10] 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. By Theorem V.3, each $B \in \mathcal{A}$ has marginal probability at most $q = 2(ed)^8 p$, and this depends only on the two-hop neighborhood of B.

Let $\mathcal{R}$ be the connected component containing B in the residual graph and let $N = |\mathcal{R}|$. The trivial 1-round randomized algorithm for the residual problem (drawing from Ω and checking B), fails with probability at most Nq. Therefore, if $Nq < 1$, this gives a Las Vegas algorithm for the component of B. By Proposition III.2, the component can be solved deterministically in $O(d^2 + \log^* n) = O(d^2)$ rounds. Thus, B can only be true if $N \geq 1/q$. As we show in the full paper, this event has probability at most $e^{-\frac{1}{10000d^{10}p}}$. ∎

We can reduce the local failure probability further by repeatedly applying Lemma V.5:

Proposition V.6. *Suppose* $d^{15}p \leq 1$ *and* $d \geq \log^* n$. *Then for* n *sufficiently large and any* $i \geq 0$, *one can transform* $\mathcal{B}$ *into a new problem instance* $\mathcal{C}$ *with parameters*

$$p' = \frac{1}{\exp^{(i)}(\frac{1}{d^{15}p})}, d' = \exp^{(i)}(d^3)$$

The communication graph for $\mathcal{C}$ *can be simulated in* $\exp^{(i-1)}(d^3)$ *rounds with respect to* G.

Proof sketch of Theorem I.6*:* For the randomized part, let A be the algorithm of Lemma V.4. We begin by bootstrapping $A^{[w]}$ for $w = (\log^{(i-1)} n)^{20}$. This runs in time $r = e^{O(\log d + \sqrt{\log \log w})} \leq e^{O(\sqrt{\log^{(i+1)} n})}$, and generates a new problem instance $\mathcal{C}$ with $p' = 2/w = 2(\log^{(i-1)} n)^{-20}$ and $d' = d^{2r} = \exp^{(2)}\big(O(\sqrt{\log^{(i+1)} n})\big)$.

Finally, we use Lemma V.6 to further amplify $\mathcal{C}$, getting a new problem instance $\mathcal{C}_{i-1}$, with parameters

$$p'' = 1/\exp^{(i-1)}(\frac{1}{(d')^{15}p'}) \leq \frac{1}{\exp^{(i-1)}((\log^{(i-1)} n)^{19})}.$$

and this is at most n^{-20} for n sufficiently large.

Since the local failure probability of $\mathcal{C}_{i-1}$ is so small, we can find a satisfying assignment w.h.p. by sampling $\mathcal{C}_{i-1}$ a single time. So, the run-time for our algorithm is simply the cost of simulating a single round of $\mathcal{C}_{i-1}$, i.e. it is $\exp^{(i-2)} O(d'^3) = \exp^{(i)}\big(O(\sqrt{\log^{(i+1)} n})\big)$.

To get the deterministic algorithm, we apply our generic derandomization framework; the running time has its dependence on n stepped up by an exponential factor. ∎

VI. The LLL for High-Degree Graphs

There are two main shortcomings with the distributed LLL algorithm of Section V. 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 V to get distributed algorithms running in time $2^{O(\sqrt{\log \log n})}$, without any condition on Δ, 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 [10]; for example, in defective vertex coloring, certain vertices are re-colored using a secondary set of colors, instead of resampling a bad-event directly.

In this section, we describe an alternative LLL algorithm, whose run-time does not depend on Δ. This matches the run-time of [10]; however, the algorithms can be described in a much more generic, high-level way. Our algorithm is a slightly modified version of an LLL algorithm of [19]; the key definition underlying the algorithms is that of a *dangerous* event.

Definition VI.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 the algorithm of [19], will be to bound the probability that an event is q-dangerous. In [19], the following bound was provided:

Proposition VI.2 ([19])**.** *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 much 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 additional local information beyond just p and d. We will define a statistic we

refer to as *fragility* and use this to compute the probability of a bad-event becoming dangerous. No further information about $\mathcal{B}$ will be used.

Definition VI.3. *For any vector $a \in \{0,1\}^v$, define a configuration $Z_a \in D^v$ by $Z_a(i) = X_{a(i)}(i)$, and define the event E_B by*

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

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

Proposition VI.4. *For any bad-event B and any $q \in [0,1]$ we have $\mathbb{P}(B \text{ is } q\text{-dangerous}) \leq f(B)/q$.*

Theorem I.7. *Suppose that $f(B) \leq e^{-10}d^{-12}$ for every $B \in \mathcal{B}$. Then there is an algorithm to find a satisfying assignment in $2^{O(\sqrt{\log\log n})}$ rounds w.h.p.*

Proof sketch: We run the same randomized algorithm as [19] for the pre-shattering phase: to begin, we draw X from the distribution Ω; if any bad-event B is q-dangerous we revert it, where we set the parameter $q = (ed)^{-3}$. We then use a deterministic LLL algorithm for the residual problem $\mathcal{R}$ in $2^{O(\sqrt{\log\log n})}$ rounds. ∎

As an application, let us consider a boolean formula Φ, 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. Theorem I.7 gives a qualitatively similar bound.

Proposition VI.5. *If Φ has m clauses and every clause intersects at most d others where $d \leq e^{-10}(4/3)^{k/12} \approx 0.00005 \times 1.02426^k$, there is a distributed algorithm to find a satisfying solution to Φ is $2^{O(\sqrt{\log\log m})}$ rounds.*

Proof: The probability space Ω is defined by selecting each variable $X(i)$ to be true or false with probability $1/2$. Consider some clause C, wlg $C = x_1 \vee x_2 \vee \cdots \vee x_k$; let B denote the bad-event that C is false. Now let X_0, X_1 be drawn independently from Ω; a necessary event for the B to fail on some Z_a is for $X_0(i) = F$ or $X_1(i) = F$. This has probability $3/4$ for each i and by independence we have $f(B) \leq (3/4)^k$. So by Theorem I.7, we need $e^{10}(3/4)^k d^{12} \leq 1$, i.e. $d \leq e^{-10}(4/3)^{k/12}$. ∎

VII. Obstacles to Derandomizing Local Algorithms

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

A. An Exponential Separation in the SLOCAL Model

A key consequence of Theorem III.1 is that in the SLOCAL model, for locally checkable problems, randomized algorithms are no more powerful than deterministic algorithms up to logarithmic factors. In this section, we show an exponential separation for sublogarithmic complexities. More concretely, we show that the sinkless orientation problem, 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. [17] showed that randomized sinkless orientation requires $\Omega(\log\log n)$ round complexity and Chang et al. [20] 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 [9]. In contrast, in the SLOCAL model, the tight complexities are an exponential lower: we show that sinkless orientation has deterministic SLOCAL complexity $\Theta(\log\log n)$ and randomized SLOCAL complexity $\Theta(\log\log\log n)$.

Theorem VII.1. *Any* SLOCAL *(respectively,* RSLOCAL*) algorithm for sinkless orientation on d-regular graphs has locality $\Omega(\log_d \log n)$ (respectively, $\Omega(\log_d \log\log n)$).*

Proof: Suppose there is a SLOCAL(t) sinkless orientation algorithm. By Proposition III.2, this yields a deterministic LOCAL$(td^t + t\log^* n)$ algorithm. As shown by Chang et al.[20], deterministic LOCAL sinkless orientation algorithms must have round complexity $\Omega(\log n)$. So $td^t + t\log^* n \geq \Omega(\log n)$, which implies that $t \geq \Omega(\log_d \log n)$.

The RSLOCAL proof is similar. Here, we use the $\Omega(\log\log n)$ lower bound of Brandt et al.[17] for RLOCAL sinkless orientation algorithms. ∎

Theorem VII.2. *There is an* SLOCAL$(O(\log\log n))$ *algorithm and a* RSLOCAL$(O(\log\log\log n))$ *algorithm to compute a sinkless orientation of any graph with minimum degree at least* 3*.*

Proof: For the SLOCAL algorithm, we apply Theorem III.1 to the randomized LOCAL algorithm of Ghaffari and Su [9] which has round complexity $O(\log\log n)$.

The RSLOCAL algorithm is based on shattering, and it is also borrowed from Ghaffari and Su [9] almost verbatim. In the first phase, we mark each edge with probability $1/4$ and orient each marked edge randomly. There are a number of bad-events, corresponding to vertices which have too many marked edges or no outgoing marked edges. This first pass has locality $O(1)$. In the second phase, we use the SLOCAL algorithm on the induced subgraphs on bad vertices. As shown in [9], each connected component of bad nodes has size at most $N = O(\log n)$, and therefore this second pass has locality $O(\log\log N) = O(\log\log\log n)$.

As shown in Ghaffari, Kuhn, and Maus [13], this two-pass algorithm can then be transformed into a single-pass

RSLOCAL($O(\log\log\log n)$) algorithm. ∎

B. Impossibility of Derandomizing Non-Locally-Checkable Problems

The local checkability condition is an important property in derandomization, and it is not there just for technicalities. Specifically, let us consider the following simple toy problem: 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 $1/\sqrt{n}$, succeeds with high probability. However, any deterministic SLOCAL algorithm needs $\Omega(\sqrt{n})$ rounds.

C. Complete Problems

We next prove completeness results for several natural and widely-studied distributed graph problems. 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 $\mathcal{B}$ [13]. In addition, a distributed graph problem $\mathcal{A}$ is called P-SLOCAL-complete if $\mathcal{A}$ is in the class P-SLOCAL and 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, then every problem in P-SLOCAL (and thus by Theorem III.3 also all problems in P-ZLOCAL) could also be solved in $\operatorname{polylog} n$ deterministic time. The following completeness reductions can therefore be understood as conditional lower bounds.

Theorem VII.3. *Approximating distributed set cover by any factor $\alpha = \mathit{polylog}(n)$ is* P-SLOCAL*-complete.*

Given a graph $G = (V, E)$, the minimum dominating set (MDS) problem asks for a smallest possible vertex set $D \subseteq V$ such that for all $u \in V$, either $u \in D$ or some neighbor v of u is in D. MDS is one of the most widely studied problems in the LOCAL model and is essentially equivalent to minimum set cover. Polylogarithmic-time randomized distributed approximation algorithms have been known for a long time, e.g., [24], [25], [26], [27].

Theorem VII.4. *Approximating MDS by a polylogarithmic factor is* P-SLOCAL*-complete.*

We next consider a greedy packing problem similar to MIS, one of the four classic symmetry breaking problems. (We do now know whether the MIS problem is P-SLOCAL-complete.) Consider a bipartite graph $G = (L \cup R, E)$. For a positive integer k, we define a k-star of G to be a node $u \in L$ together with k neighbors neighbors $v_1 \ldots, v_k \in R$.

Theorem VII.5. *Let $G = (L \cup R, E)$ be an n-node bipartite graph, where every node in L has degree at most Δ. 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.*

The last problem we consider in this section is *sparse neighborhood cover* of a graph [29], which is a fundamental structure with a large number of applications in distributed systems [12].

Theorem VII.6. *The problem of computing a sparse neighborhood cover of a graph for radius $r \leq \operatorname{polylog} n$ is* P-SLOCAL*-complete.*

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.

ACKNOWLEDGMENT

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

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

[2] N. Alon, L. Babai, and A. Itai, "A fast and simple randomized parallel algorithm for the maximal independent set problem," *Journal of Algorithms*, vol. 7, no. 4, pp. 567–583, 1986.

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

[4] A. Panconesi and A. Srinivasan, "On the complexity of distributed network decomposition," *Journal of Algorithms*, vol. 20, no. 2, pp. 581–592, 1995.

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

[6] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider, "The locality of distributed symmetry breaking," *Journal of the ACM*, vol. 63, no. 3, p. 20, 2016.

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

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

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

[10] M. Fischer and M. Ghaffari, "Sublogarithmic distributed algorithms for Lovász local lemma, and the complexity hierarchy," in *Proc. 31st Symp. on Distributed Computing (DISC)*, 2017, pp. 18:1–18:16.

[11] Y.-J. Chang, W. Li, and S. Pettie, "An optimal distributed $(\Delta + 1)$-coloring algorithm?" in *Proc. 50th ACM Symp. on Theory of Computing (STOC)*, 2018.

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

[13] 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)*, 2017, pp. 784–797.

[14] 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.

[15] M. Ghaffari, F. Kuhn, Y. Maus, and J. Uitto, "Deterministic distributed edge-coloring with fewer colors," in *Proc. 50th ACM Symp. on Theory of Computing (STOC)*, 2018.

[16] Y.-J. Chang and S. Pettie, "A time hierarchy theorem for the LOCAL model," in *Proc. 58th IEEE Symp. on Foundations of Computer Science (FOCS)*, 2017, pp. 156–167.

[17] 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)*, 2016, pp. 479–488.

[18] R. A. Moser and G. Tardos, "A constructive proof of the general Lovász local lemma," *Journal of the ACM*, vol. 57, no. 2, p. 11, 2010.

[19] K.-M. Chung, S. Pettie, and H.-H. Su, "Distributed algorithms for the Lovász local lemma and graph coloring," *Distributed Computing*, vol. 30, no. 4, pp. 261–280, 2017.

[20] 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.

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

[22] 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*, vol. 49, no. 3, pp. 454–477, 1994.

[23] S. Rajagopalan and V. V. Vazirani, "Primal-dual RNC approximation algorithms for set cover and covering integer programs," *SIAM Journal on Computing*, vol. 28, no. 2, pp. 525–540, 1998.

[24] D. Dubhashi, A. Mei, A. Panconesia, J. Radhakrishnan, and A. Srinivasan, "Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons," *Journal of Computer and System Sciences*, vol. 71, no. 4, pp. 467–479, 2005.

[25] L. Jia, R. Rajaraman, and R. Suel, "An efficient distributed algorithm for constructing small dominating sets," *Distributed Computing*, vol. 15, no. 4, pp. 193–205, 2002.

[26] F. Kuhn and R. Wattenhofer, "Constant-time distributed dominating set approximation," *Distributed Computing*, vol. 17, no. 4, pp. 303–310, 2005.

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

[28] N. Linial and M. Saks, "Low diameter graph decompositions," *Combinatorica*, vol. 13, no. 4, pp. 441–454, 1993.

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

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

[31] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg, "Fast network decompositions and covers," *J. of Parallel and Distributed Computing*, vol. 39, no. 2, pp. 105–114, 1996.

[32] P. Fraigniaud, A. Korman, and D. Peleg, "Towards a complexity theory for local distributed computing," *Journal of the ACM*, vol. 60, no. 5, p. 35, 2013.

[33] L. Barenboim, M. Elkin, and F. Kuhn, "Distributed $(\Delta + 1)$-coloring in linear (in Δ) time," *SIAM Journal on Computing*, vol. 43, no. 1, pp. 72–95, 2015.

[34] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM Journal on computing*, vol. 2, no. 4, pp. 225–231, 1973.

[35] Y.-J. Chang, Q. He, W. Li, S. Pettie, and J. Uitto, "The complexity of distributed edge coloring with small palettes," in *Proc. 29th ACM-SIAM Symp. on Discrete Algorithms*, 2018, pp. 2633–2652.

[36] D. G. Harris, "Oblivious resampling oracles and parallel algorithms for the lopsided lovász local lemma," *arXiv preprint arXiv:1702.02547*, 2017.

[37] B. Haeupler and D. G. Harris, "Parallel algorithms and concentration bounds for the Lovász local lemma via witness DAGs," *ACM Transactions on Algorithms (TALG)*, vol. 13, no. 4, p. 53, 2017.

[38] M. Molloy and B. Reed, "Further algorithmic aspects of the local lemma," in *Proc. 30th ACM Symp. on Theory of Computing (STOC)*, 1998, pp. 524–529.

[39] J. Pach and G. Tardos, "Conflict-free colourings of graphs and hypergraphs," *Combinatorics, Probability and Computing*, vol. 18, no. 5, pp. 819–834, 2009.