Sublogarithmic Distributed Algorithms for Lovász Local Lemma, and the Complexity Hierarchy*

Manuela Fischer[1] **and Mohsen Ghaffari**[2]

1 **ETH Zürich, Switzerland**
`manuela.fischer@inf.ethz.ch`
2 **ETH Zürich, Switzerland**
`ghaffari@inf.ethz.ch`

Abstract

Locally Checkable Labeling (LCL) problems include essentially all the classic problems of LOCAL distributed algorithms. In a recent enlightening revelation, Chang and Pettie [FOCS'17] showed that *any* LCL (on bounded degree graphs) that has an $o(\log n)$-round randomized algorithm can be solved in $T_{LLL}(n)$ rounds, which is the randomized complexity of solving (a relaxed variant of) the Lovász Local Lemma (LLL) on bounded degree n-node graphs. Currently, the best known upper bound on $T_{LLL}(n)$ is $O(\log n)$, by Chung, Pettie, and Su [PODC'14], while the best known lower bound is $\Omega(\log \log n)$, by Brandt et al. [STOC'16]. Chang and Pettie conjectured that there should be an $O(\log \log n)$-round algorithm (on bounded degree graphs).

Making the first step of progress towards this conjecture, and providing a significant improvement on the algorithm of Chung et al. [PODC'14], we prove that $T_{LLL}(n) = 2^{O(\sqrt{\log \log n})}$. Thus, any $o(\log n)$-round randomized distributed algorithm for any LCL problem on bounded degree graphs can be automatically sped up to run in $2^{O(\sqrt{\log \log n})}$ rounds.

Using this improvement and a number of other ideas, we also improve the complexity of a number of graph coloring problems (in arbitrary degree graphs) from the $O(\log n)$-round results of Chung, Pettie and Su [PODC'14] to $2^{O(\sqrt{\log \log n})}$. These problems include defective coloring, frugal coloring, and list vertex-coloring.

1998 ACM Subject Classification G.2.2 Graph Algorithms

Keywords and phrases Distributed Graph Algorithms, the Lovász Local Lemma (LLL), Locally Checkable Labeling problems (LCL), Defective Coloring, Frugal Coloring, List Vertex-Coloring

Digital Object Identifier 10.4230/LIPIcs.DISC.2017.18

1 Introduction and Related Work

The Lovász Local Lemma (LLL), introduced by Erdős and Lovász in 1975 [14], is a beautiful result which shows that, for a set of "bad events" in a probability space that have certain sparse dependencies, there is a non-zero probability that none of them happens. This result has become a central tool in the *probabilistic method* [2], when proving that certain combinatorial objects exist. Although the LLL itself does not provide an efficient way for finding these objects, and that remained open for about 15 years, a number of efficient centralized algorithms have been developed for it, starting with Beck's breakthrough in 1991 [7], through [1, 13, 26, 28, 33], and leading to the elegant algorithm of Moser and Tardos in 2010 [29]. See also [9, 19, 21–24] for some of the related work on that track.

* A full version of the paper is available at `https://arxiv.org/abs/1705.04840`.

In contrast, distributed algorithms for LLL and the related complexity are less well-understood. This question has gained an extraordinary significance recently, due to revelations that show that LLL is a "complete" problem for sublogarithmic-time problems. Next, we first overview the concrete statement of the LLL, and then discuss what is known about its distributed complexity, and what its special significance is for distributed algorithms. Then we proceed to presenting our contributions.

1.1 The LLL and its Special Role in Distributed Algorithms

The Lovász Local Lemma. Consider a finite set $\mathcal{V}$ of independent random variables, and a finite family $\mathcal{X}$ of n (bad) events on these variables. Each event $A \in \mathcal{X}$ depends on some subset $\text{vbl}(A) \subseteq \mathcal{V}$ of variables. Define the dependency graph $G_{\mathcal{X}} = (\mathcal{X}, \{(A, B) \mid \text{vbl}(A) \cap \text{vbl}(B) \neq \emptyset\})$ that connects any two events which share at least one variable. Let d be the maximum degree in this graph, i.e., each event $A \in \mathcal{X}$ shares variables with at most d other events $B \in \mathcal{X}$. Finally, define $p = \max_{A \in \mathcal{X}} \Pr[A]$. The Lovász Local Lemma [14] shows that $\Pr[\cap_{A \in \mathcal{X}} \bar{A}] > 0$, under the *LLL criterion* that $epd \leq 1$. Intuitively, if a local union bound is satisfied around each node in $G_{\mathcal{X}}$, with some slack, then there is a positive probability to avoid all bad events.

What's Known about Distributed LLL? In the standard distributed formulation of LLL, we consider LOCAL-model [25, 32] algorithms that work on the n-node dependency graph $G_{\mathcal{X}}$, where per round each node can send a message to each of its neighbors[1].

Moser and Tardos [29] provided an $O(\log^2 n)$-round randomized distributed algorithm. Chung, Pettie, and Su [12] presented an $O(\log n \cdot \log^2 d)$-round algorithm, which was later improved slightly to $O(\log n \cdot \log d)$ [17]. Perhaps more importantly, under a modestly stronger criterion that $epd^2 < 1$, which is satisfied in most of the standard applications, they gave an $O(\log n)$-round algorithm [12]. This remains the best known distributed algorithm. On the other hand, Brandt et al. [8] showed a lower bound of $\Omega(\log \log n)$ rounds, which holds even if a much stronger LLL criterion of $p2^d < 1$ is satisfied. Even under this exponentially stronger criterion, the best known upper bound changed only slightly to $O(\log n / \log \log n)$ [12].

Completeness of LLL for Sublogarithmic Distributed Algorithms. Chang and Pettie [11] showed that any $o(\log n)$-round randomized algorithm $\mathcal{A}$ for any Locally Checkable Labeling (LCL) problem $\mathcal{P}$ – a problem whose solution can be checked in $O(1)$ rounds [30], which includes all the classic local problems – on bounded degree graphs can be transformed to an algorithm with complexity $O(T_{LLL}(n))$. Here, $T_{LLL}(n)$ denotes the randomized complexity for solving LLL on n-node bounded-degree graphs, with high probability.

In a nutshell, their idea is to "lie" to the algorithm $\mathcal{A}$ and say that the network size is some much smaller value $n^* \ll n$. This deceived algorithm $\mathcal{A}$ may have a substantial probability to fail, creating an output that violates the requirements of the LCL problem $\mathcal{P}$ somewhere. However, the probability of failure in each local neighborhood is at most $1/n^*$. Choosing n^* a large enough constant, depending on the complexity of $\mathcal{A}$, the algorithm $\mathcal{A}$ provides an LLL system – where we have one bad event for violation of each local requirement of $\mathcal{P}$ – that satisfies the criterion $pd^c < 1$ for a desirably large constant c. Hence, we can solve this LLL system and thus obtain a solution for the original LCL problem $\mathcal{P}$ in $O(T_{LLL}(n))$ time.

[1] One can imagine a few alternative graph formulations, all of which turn out to be essentially equivalent in the LOCAL model, up to an $O(1)$ overhead in complexity.

This result implies that LLL is important not only for a few special problems, but in fact for essentially *all* sublogarithmic-time distributed problems. Due to this remarkable role, Chang and Pettie state that "*understanding the distributed complexity of the LLL is a significant open problem.*" Furthermore, although a wide gap between the best upper bound $O(\log n)$ [12] and lower bound $\Omega(\log \log n)$ [8] persists, they conjecture the latter to be tight:

▶ **Conjecture** (Chang, Pettie [11]). *There exists a sufficiently large constant c such that the distributed LLL problem can be solved in $O(\log \log n)$ time on bounded degree graphs, under the symmetric LLL criterion $pd^c < 1$.*[2]

1.2 Our Contributions

Faster Distributed LLL. We make a significant step of progress towards this conjecture:

▶ **Theorem 1.** *There is a $2^{O(\sqrt{\log \log n})}$-round randomized distributed algorithm that, with high probability*[3]*, solves the LLL problem with degree at most $d = O(\log^{1/5} \log n)$, under a symmetric polynomial LLL criterion $p(ed)^{32} < 1$.*[4]

This improves over the $O(\log n)$-round algorithm of Chung et al. [12]. We note that even under a significantly stronger exponential LLL criterion – formally requiring $4ep2^d d^4 < 1$ – the best known round complexity was $O(\log n / \log \log n)$ [12]. Furthermore, we note that a key ingredient in developing Theorem 1 is a deterministic distributed algorithm for LLL, which we present in Theorem 11. To the best of our knowledge, this is the first (non-trivial) deterministic distributed LLL algorithm. In fact, we believe that any conceivable future improvements on Theorem 1 may have to improve on this deterministic part.

Moreover, our method provides some further supporting evidence for the conjecture of Chang and Pettie. In particular, if one finds a $\operatorname{poly} \log n$-round deterministic algorithm for $\big(O(\log n), O(\log n)\big)$ network decomposition [31] – a central problem that has remained open for a quarter century, but is often perceived as likely to be true – then, combining that with our method would prove $T_{LLL}(n) = \operatorname{poly}(\log \log n)$.

A Gap in the Randomized Distributed Complexity Hierarchy. Putting Theorem 1 with [11, Theorem 6], we get the following automatic speedup result:

▶ **Corollary 2.** *Let $\mathcal{A}$ be a randomized* LOCAL *algorithm that solves some LCL problem $\mathcal{P}$ on bounded degree graphs, w.h.p., in $o(\log n)$ rounds. Then, it is possible to transform $\mathcal{A}$ into a new randomized* LOCAL *algorithm $\mathcal{A}'$ that solves $\mathcal{P}$, w.h.p., in $2^{O(\sqrt{\log \log n})}$ rounds.*

Using a similar method, and our deterministic LLL algorithm (Theorem 11), we obtain the following corollary, the proof of which is deferred to the full version [15]. This corollary shows that any $o(\log \log n)$-round randomized algorithm for an LCL problem on bounded degree graphs can be improved to a deterministic $O(\log^* n)$-round algorithm. This result seems to be implicit in the recent work of Chang, Kopelowitz, and Pettie [10], though with a quite different proof, and it can be derived from [10, Corollary 3] and [10, Theorem 3].

---

[2] This statement, as is, has a small imprecision: one should assume either that $d \geq 2$, in which case $pd^c < 1$ can be replaced with $p(ed)^{c'} < 1$ for some other constant c', or that $pd^c < 1/2$. Otherwise, two events of head or tail for a fair coin have $p = 1/2$ and $d = 1$, thus $pd^c < 1$, but one cannot avoid both.

[3] As standard, the phrase *with high probability* (w.h.p.) indicates that an event has probability at least $1 - n^{-c}$, for a sufficiently large constant c.

[4] We remark that we did not try to optimize the constants.

▶ **Corollary 3.** *Let $\mathcal{A}$ be a randomized* LOCAL *algorithm that solves some LCL problem $\mathcal{P}$ on bounded degree graphs, w.h.p., in $o(\log \log n)$ rounds. Then, it is possible to transform $\mathcal{A}$ into a new deterministic* LOCAL *algorithm $\mathcal{A}'$ that solves $\mathcal{P}$ in $O(\log^* n)$ rounds.*

Faster Distributed Algorithms for Graph Colorings via LLL. For some distributed graph problems on bounded degree graphs, we can immediately get faster algorithms by applying Theorem 1. However, there are two quantifiers which appear to limit the applicability of Theorem 1: (L1) it requires a stronger form of the LLL criterion, concretely needing $p(ed)^{32} < 1$ instead of $epd \leq 1$; (L2) it applies mainly to bounded degree graphs.

We explain how to overcome these two limitations in most of the LLL-based problems studied by Chung, Pettie, and Su [12]. Regarding limitation (L1), we show that even though in many coloring problems the direct LLL formulation would not satisfy the polynomial criterion $p(ed)^{32} < 1$, we can still solve the problem, through a number of iterations of partial colorings, each satisfying this stronger LLL criterion. Regarding limitation (L2), we explain how in many problems, the first step of our LLL algorithm, which is its only part that relies on bounded degrees, can be replaced by a faster randomized step for that coloring.

The end results of our method include algorithms with round complexity $2^{O(\sqrt{\log \log n})}$ for a number of coloring problems, improving on the corresponding $O(\log n)$-round algorithms of Chung, Pettie, and Su [12]: defective coloring, frugal coloring, and list vertex-coloring. The first two are presented respectively, in Section 4, Section 5. The third coloring result, as well as some of the proofs, are deferred to the full version of this article [15].

2 Preliminaries

2.1 Network Decompositions

Roughly speaking, a *network decomposition* [4, 31] partitions the nodes into a few blocks, each of which is made of a number of low-diameter connected components. More formally, the definition is as follows:

▶ **Definition 4** (Network Decomposition)**.** Given a graph $G = (V, E)$, a partition of the nodes V into C vertex-disjoint blocks $V_1, V_2, \ldots, V_C$ is a (C, D) network decomposition if in each block's induced subgraph $G[V_i]$ each connected component has diameter at most D.

▶ **Lemma 5** (The Network Decomposition Algorithm)**.** *Given an n-node network $G = (V, E)$, there is a deterministic distributed algorithm that computes a $(\lambda, n^{1/\lambda} \cdot \log n)$ network decomposition of G in $\lambda \cdot n^{1/\lambda} \cdot 2^{O(\sqrt{\log n})}$ rounds.*

The proof of Lemma 5 is deferred to the full version; it works mainly by putting together some ideas of Awerbuch and Peleg [5], Panconesi and Srinivasan [31], and Awerbuch et al. [3]. However, we are not aware of this result appearing in prior work.

2.2 Shattering

In a number of our algorithms, we make use of the following lemma which, roughly speaking, shows that if each node of the graph remains with some small probability and we have certain independence between these events, the remaining connected components are "small". We remark that this lemma or its variants are key ingredients in Beck's LLL method [7], sometimes referred to as the *shattering lemma*, and analogues of it appear in the literature [1, 6, 17, 18, 20, 26, 27]. The proof of this lemma is deferred to the full version.

▶ **Lemma 6** (The Shattering Lemma). *Let $G = (V, E)$ be a graph with maximum degree Δ. Consider a process which generates a random subset $B \subseteq V$ where $P(v \in B) \leq \Delta^{-c_1}$, for some constant $c_1 \geq 1$, and that the random variables $1(v \in B)$ depend only on the randomness of nodes within at most c_2 hops from v, for all $v \in V$, for some constant $c_2 \geq 1$. Moreover, let $H = G^{[2c_2+1, 4c_2+2]}$ be the graph which contains an edge between u and v iff their distance in G is between $2c_2 + 1$ and $4c_2 + 2$. Then with probability at least $1 - n^{-c_3}$, for any constant $c_3 < c_1 - 4c_2 - 2$, we have the following three properties:*

(P1) *$H[B]$ has no connected component U with $|U| \geq \log_\Delta n$.*

(P2) *$G[B]$ has size at most $O\big(\log_\Delta n \cdot \Delta^{2c_2}\big)$.*

(P3) *Each connected component of $G[B]$ admits a $(\lambda, O(\log^{1/\lambda} n \cdot \log^2 \log n))$ network decomposition, for any integer $\lambda \geq 1$, which can be computed in $\lambda \cdot \log^{1/\lambda} n \cdot 2^{O(\sqrt{\log \log n})}$ rounds, deterministically.*

3 Our General Algorithm for Lovász Local Lemma

In this section, we explain our sublogarithmic-time LLL algorithm of Theorem 1, which solves LLL in $2^{O(\sqrt{\log \log n})}$ rounds on bounded degree graphs, given the condition that $p(ed)^{32} < 1$.

The Lovász Local Lemma Setting. We consider a finite set $\mathcal{V}$ of independent random variables, and a finite family $\mathcal{X}$ of n bad events on these variables. Each event $A \in \mathcal{X}$ depends on some subset $\text{vbl}(A) \subseteq \mathcal{V}$ of variables. In fact, essentially without loss of generality, we assume that each random variable in $\mathcal{V}$ is a fair random bit. We note that in practically all settings of interest, we can rewrite the basic random variables as a function of at most $\text{poly}(n)$ independent random bits, hence transitioning from arbitrary set of random variables to another space of random variables with just fair random bits. The number of random bits will impact only the local computations and as such, since we are working in the local model which does not assume a limited computational power, we can allow the number of random bits to be arbitrarily large. The distributed algorithms that we describe work on the dependency graph of the events, defined as $G_{\mathcal{X}} = (\mathcal{X}, \{(A, B) \mid \text{vbl}(A) \cap \text{vbl}(B) \neq \emptyset\})$. That is, this graph has one vertex for each event and that connects any two events which share at least one variable. Then, d denotes the maximum degree in this graph, and $p = \max_{A \in \mathcal{X}} \Pr[A]$.

Our General LLL Algorithm. The algorithm is developed in two stages, as we overview next. In the first stage, presented in Section 3.1, we explain a randomized algorithm with complexity $\lambda \cdot \log^{1/\lambda} n \cdot 2^{O(\sqrt{\log \log n})}$, given that an LLL criterion $p(ed)^{4\lambda} < 1$ is satisfied. In the main regime of interest, the best LLL criterion exponent that we will assume is $\lambda = O(1)$, and thus this $(\lambda \cdot \log^{1/\lambda} n \cdot 2^{O(\sqrt{\log \log n})})$-round algorithm, on its own, would not get us to our target complexity of $2^{O(\sqrt{\log \log n})}$, although still being an improvement on the $O(\log n)$-round algorithm of [12].

In the second stage, presented in Section 3.2, we improve this complexity to $2^{O(\sqrt{\log \log n})}$. That improvement works mainly by viewing the sublogarithmic-time local algorithm of Section 3.1 as setting up a new LLL, with a much larger exponent λ in its LLL criterion, hence allowing us get to a much smaller complexity by (recursively) applying the same scheme. This speed up is inspired by the ideas of Chung and Pettie [11] which showed that LLL can be used to speed up sublogarithmic-time local algorithms.[5]

[5] Though, we find this recursive application of the idea to speed up the complexity of LLL itself, through

3.1 The Base LLL Algorithm

▶ **Theorem 7.** *For any integer $\lambda \geq 8$, there is a randomized distributed algorithm solving the LLL problem under the symmetric criterion $p(ed)^{4\lambda} < 1$, in $O(d^2) + \lambda \cdot \log^{1/\lambda} n \cdot 2^{O(\sqrt{\log \log n})}$ rounds, with high probability.*

This algorithm consists of two parts: (1) a randomized algorithm, explained in Section 3.1.1, which performs some partial sampling in the LLL space, thus setting some of the variables, in a manner that *shatters* the graph, hence leaving small connected components among the unset variables; (2) a deterministic LLL algorithm, explained in Section 3.1.2, which we use to solve the remaining small connected components. To the best of our knowledge, this is the first non-trival deterministic distributed LLL algorithm. In Section 3.1.3, we combine these two parts, concluding the proof of Theorem 7.

This general *shattering* style for randomized algorithms – which first performs some randomized steps to break the graph into small remaining connected components, and then uses deterministically solves these remaining components – is rooted in the breakthrough LLL algorithm of Beck [7], and has been used extensively before [1, 6, 17, 18, 20, 26, 27].

3.1.1 The Randomized Part

We now explain the randomized component of our LLL algorithm for bounded degree graphs, which performs a partial sampling in the LLL space, thus setting some of the variables, in a manner that guarantees the following two properties: (1) the conditional probabilities of the bad events, conditioned on the already set variables, satisfy a polynomial LLL criterion, (2) the connected components of the events on variables that remain unset are "small" (e.g., for bounded degree graphs, they have size at most $O(\log n)$), with high probability.

These two properties together will allow us to invoke the deterministic LLL algorithm that we present later in Section 3.1.2 on the remaining components of variables that remain unset. In particular, (1) means that the bad events $\mathcal{X}$ form another LLL problem on the variables that remain unset, where each new bad event has probability at most $\sqrt{p}$. Furthermore, (2) ensures that the components are small enough to make the deterministic algorithm efficient.

Our partial sampling is inspired by a sequential LLL algorithm of Molloy and Reed [26].

▶ **Lemma 8** (Random Partial Setting for the LLL Variables)**.** *There is a randomized distributed algorithm that computes, w.h.p., in $O(d^2 + \log^* n)$ rounds, a partial assignment of values to variables – setting the values of the variables in a set $\mathcal{V}^* \subseteq \mathcal{V}$, hence leaving the variables in $\mathcal{V}' := \mathcal{V} \setminus \mathcal{V}^*$ unset – of an LLL satisfying $p(ed)^{4\lambda} < 1$, for any integer $\lambda \geq 8$, such that*

- **(i)** *$Pr[A \mid \mathcal{V}^*] \leq \sqrt{p}$ for all $A \in \mathcal{X}$, and*
- **(ii)** *w.h.p. each connected component of $G^2_{\mathcal{X}}[\mathcal{V}']$ admits a $(\lambda, O(\log^{1/\lambda} n \cdot \log^2 \log n))$ network decomposition, which can be computed in $\lambda \cdot \log^{1/\lambda} n \cdot 2^{O(\sqrt{\log \log n})}$ rounds, deterministically.*

Proof. We first compute a (d^2+1)-coloring of the square graph $G^2_{\mathcal{X}}$ on the events, which can be done even deterministically in $\tilde{O}(d) + O(\log^* n)$ rounds [16]. Suppose $\mathcal{X}_i$ is the set of events colored with color i, for $i \in \{1, \ldots, d^2+1\}$. We process the color classes one by one.

During the process, some variables may get frozen, as we discuss soon. The process assigns values for all non-frozen variables, as follows: For each color $i \in \{1, \ldots, d^2+1\}$, and for each node $A \in \mathcal{X}_i$ in parallel, we make node A sample values for its (so far non-frozen)

strengthening the corresponding LLL criterion, somewhat amusing.

variables locally, one by one. Notice that since we are using a coloring of $G_{\mathcal{X}}^2$, for each color i, each event $B \in \mathcal{X}$ shares variables with at most one event $A \in \mathcal{X}_i$. Hence, during this iteration, at most one node A is sampling variables of event B. Consider a node A that is choosing values for some random variables. Each time, when A is choosing a value for a variable $v \in \text{vbl}(A)$, it checks whether this setting makes one of the events $B \in \mathcal{X}$ involving variable v *dangerous*. We call an event B *dangerous* if $\Pr[B|\mathcal{V}_B^*] \geq \sqrt{p}/2$, where $\mathcal{V}_B^*$ denotes the already set variables of B up to this point in the sampling process. If the recently set variable v leads to a dangerous event B, then node A *freezes* variable v as well as all the remaining variables of event B. We do not assign values to frozen variables in the remainder of the randomized sampling process. We have two key observations regarding this process:

▶ **Observation 9.** *At the end of each iteration, for each event $A \in \mathcal{X}$, the conditional probability of event A, conditioned on the already made assignments $\mathcal{V}_A^*$, is at most $\sqrt{p}$.*

Proof Sketch. The first time that an event A becomes dangerous, all of its remaining variables get frozen and no other assignment gets made for its variables. By definition, before A becoming dangerous, the conditional probability of event A, conditioned on the already made assignments $\mathcal{V}_A^*$, is at most $\sqrt{p}/2$. If A becomes dangerous, that's because of setting of one last random variable. Since we have assumed that the random variables are all fair random bits, at the time of setting one last bit, the conditional probability of event A can increase by at most a 2 factor. Hence, once A becomes dangerous and all of its remaining variables get frozen, its conditional probability is at most $\sqrt{p}$. ◀

▶ **Observation 10.** *For each event $A \in \mathcal{X}$, the probability of A having at least one unset variable is at most $2(d+1)\sqrt{p}$. Furthermore, this is independent of events that are further than 2 hops from A.*

Proof Sketch. We first claim that for each $B \in \mathcal{X}$, the probability that B ever becomes dangerous is at most $2\sqrt{p}$. This is because otherwise the total probability of B happening would exceed p. Notice that during the process, some variables get a value assigned to them, and some variables get frozen, because of B or some other adjacent event becoming dangerous. More concretely, to focus on just one event B, let us consider two processes for revealing the samples values for variables of B. In the first process, we sample all the variables in one shot. Clearly, in this process, $Pr[B] \leq p$. The second process has two phases: in the first phase, we examine the variables of B sequentially, one by one, and each time sample a value for each variable, with one exception: at each time, an adversary might call some variables that have not been revealed so-far "frozen"and moves them to the second phase; any such frozen variable will not be sampled in the first phase. This adversary can take into account all the possibilities of neighboring LLL-events making a random variable of B become frozen. Now in this process, if the conditional probability of event B given the already assigned variables exceeds $\sqrt{p}/2$, or if we run out of non-frozen variables, the first phase ends. In the second phase, we simultaneously sample all the variables that have been moved to the second phase. Now, notice that the probability of event B happening is exactly the same in the two processes; the second process is just a different order of revealing the sampled values of the first process. Hence, also in the second process, we have $Pr[B] \leq p$. Now, in the second process, let B' be the event that the conditional probability of event B at the end of the first phase (given its assignments variables) exceeds $\sqrt{p}/2$. By definition, $Pr[B'|B] \geq \sqrt{p}/2$. Since $Pr[B] \leq p$, we have $Pr[B'] \leq \frac{p}{\sqrt{p}/2} = 2\sqrt{p}$. Thus, in our LLL sampling process, the probability that each event $B \in \mathcal{X}$ ever becomes dangerous is at most $2\sqrt{p}$.

Now, an event $A \in \mathcal{X}$ can have frozen variables only if at least one of its neighboring events B, or event A itself, becomes dangerous. Since A has at most d neighboring events, by a union bound, the latter has probability at most $2(d+1)\sqrt{p}$. ◀

Observation 9 implies property (i) of Lemma 8. We use Observation 10 to conclude that the events with at least one unset variable comprise "small" connected components. In particular, we apply Lemma 6 to $G^2_{\mathcal{X}}$ with the random partial setting process generating a set $B \subseteq \mathcal{X}$ of the events that have at least one variable unset. By Observation 10, each event remains with probability at most $2(d+1)\sqrt{p} \leq 2(d+1) \cdot e^{-2\lambda} \cdot d^{-2\lambda} \leq d^{-15}$. These events depend only on events within at most 1 hop in $G^2_{\mathcal{X}}$ and hence 2 hops in $G_{\mathcal{X}}$. Thus, Lemma 6 (P3) shows that with probability at least $1 - n^{-3}$ property (ii) holds. ◀

3.1.2 The Deterministic Part

▶ **Theorem 11.** *For any integer $\lambda \geq 1$, the distributed LLL problem can be solved deterministically in $\lambda \cdot n^{1/\lambda} \cdot 2^{O(\sqrt{\log n})}$ rounds, under the symmetric LLL criterion $p(ed)^\lambda < 1$. If the algorithm is provided a (λ, γ) network decomposition of the square graph $G^2_{\mathcal{X}}$, then the LLL algorithm runs in just $O(\lambda \cdot (\gamma + 1))$ rounds.*

We make a black-box invocation to the distributed algorithm stated in Lemma 5 for computing a $(\lambda, n^{1/\lambda} \cdot \log n)$ network decomposition, and then solve the LLL problem on top of this decomposition, by going through its blocks one by one.

The running time of our deterministic LLL algorithm hence directly depends on the network decomposition it works with. In particular, if there is a $\operatorname{poly}\log n$-round deterministic distributed algorithm that computes a $(\operatorname{poly}\log n, \operatorname{poly}\log n)$ network decomposition, then this algorithm solves any LLL problem satisfying the criterion $p(ed)^\lambda < 1$ (with $\lambda = \operatorname{poly}\log n$) deterministically in $\operatorname{poly}\log n$ rounds. This would then directly improve the running time of the randomized LLL algorithm of Theorem 7 to $\operatorname{poly}(\log\log n)$, proving that $T_{LLL}(n) = \operatorname{poly}(\log\log n)$, thus almost confirming the conjecture of Chang and Pettie [11].

In fact, we believe that a conceivable future improvement of our LLL algorithm may need to improve this deterministic component, ideally to complexity $O(\log n)$ for proving the $T_{LLL}(n) = O(\log\log n)$ conjecture of Chang and Pettie [11].

Proof of Theorem 11. We first compute a $(\lambda, n^{1/\lambda} \cdot \log n)$ network decomposition of $G^2_{\mathcal{X}}$, which decomposes its nodes into λ disjoint blocks $\mathcal{X}_1, \ldots, \mathcal{X}_\lambda$, such that each connected component of $G^2_{\mathcal{X}}[\mathcal{X}_i]$ has diameter at most $n^{1/\lambda} \cdot \log n$. This decomposition can be computed in $\lambda \cdot n^{1/\lambda} \cdot 2^{O(\sqrt{\log n})}$ rounds, using Lemma 5.

Then, iteratively for $i = 1, \ldots, \lambda$, we assign values to all variables of events in $\mathcal{X}_i$ that have remained unset. The values are chosen is such a way that, after i steps, the conditional probability of *any* event in $\mathcal{X}$, conditioned on all the assignments in variables of events in $\bigcup_{j=1}^{i} \mathcal{X}_j$, is at most $p(ed)^i < 1$. Once $i = \lambda$, since the conditional failure probability is $p(ed)^\lambda < 1$ but all the variables are already assigned, we know that none of the events occurs.

The base case $i = 0$ is trivial. In the following, we explain how to set the values for variables involved in events of $\mathcal{X}_i$ in $n^{1/\lambda} \cdot \log n$ rounds. Let $\mathcal{V}_i$ be the set of variables in events of $\mathcal{X}_i$ that remain with no assigned value. We form a new LLL problem, as follows: For each bad event $A \in \mathcal{X}$, we introduce an event $B_{A,i}$ on the space of values of $\mathcal{V}_i$. This is the event that the values of $\mathcal{V}_i$ get chosen such that the conditional probability of the event A, conditioned on the variables in $\bigcup_{j=1}^{i} \mathcal{V}_j$, is larger than $p(ed)^i$. Notice that $Pr[B_{A,i} \mid \bigcup_{j=1}^{i-1} \mathcal{V}_j] \leq \frac{p(ed)^{i-1}}{p(ed)^i} = \frac{1}{ed}$. This is because, the variables $\bigcup_{j=1}^{i-2} \mathcal{V}_j$ are set such that

the conditional probability of $B_{A,i}$ given these set values is at most $p(ed)^{i-1}$, and thus, the probability that the values of $\mathcal{V}_{i-1}$ get chosen that the conditional probability given the set values in $\bigcup_{j=1}^{i-1}\mathcal{V}_j$ exceeds $p(ed)^i$ is at most $\frac{p(ed)^{i-1}}{p(ed)^i} = \frac{1}{ed}$. Moreover, each event $B_{A,i}$ depends on at most d other events $B_{A',i}$. Hence, the family of events $B_{A,i}$ on the variable set $\mathcal{V}_i$ satisfies the conditions of the tight (symmetric) LLL. Therefore, by the Lovász local lemma, we know that there exists an assignment to variables of $\mathcal{V}_i$ which makes no event $B_{A,i}$ happen. That is, an assignment such that the conditional probability of each event A, conditioned on the assignments in $\bigcup_{j=1}^{i}\mathcal{V}_j$, is bounded by at most $p(ed)^i$.

Given the existence, we find such an assignment in $n^{1/\lambda}\cdot\log n$ rounds, as follows: each component of $G^2_{\mathcal{X}}[\mathcal{X}_i]$ first gathers the whole topology of this component (as well as its incident events and the current assignments to any of their variables), in $O(n^{1/\lambda}\log n)$ rounds. Then, it decides about an assignment for its own variables in $\mathcal{V}_i$, by locally brute-forcing all possibilities. Different components can decide independently as there is no event that shares variables with two of them, since they are non-adjacent in $G^2_{\mathcal{X}}$. ◀

3.1.3 Wrap-Up: Base LLL Algorithm

Proof of Theorem 7. We run the randomized algorithm of Lemma 8 for computing a partial setting of the variables, in $O(d^2+\log^* n)$ rounds. Then, by Lemma 8 (i), the remaining events $\mathcal{X}'$ (those which have at least one unset variable) form a new LLL system on the unset variables, where each bad event has probability at most $\sqrt{p}$.

Moreover, by Lemma 8 (ii), each connected component of the square graph $G^2_{\mathcal{X}}[\mathcal{X}']$ of these remaining events $\mathcal{X}'$ has a $(\lambda, O(\log^{1/\lambda} n\cdot\log^2\log n))$ network decomposition, which we can compute in $\lambda\cdot\log^{1/\lambda} n\cdot 2^{O(\sqrt{\log\log n})}$ rounds, deterministically. From now on, we handle the remaining events in different connected components of $G^2_{\mathcal{X}}[\mathcal{X}']$ independently.

Since $\sqrt{p}(ed)^\lambda < 1$, we can now invoke the deterministic LLL algorithm of Theorem 11 on top of the network decomposition of each component. Our deterministic LLL then runs in $\lambda\cdot\log^{1/\lambda} n\cdot\log^2\log n$ additional rounds, and finds assignments for these remaining variables, without any of the events occurring, hence solving the overall LLL problem. The overall round complexity is $O(d^2)+\lambda\cdot\log^{1/\lambda} n\cdot 2^{O(\sqrt{\log\log n})}$. ◀

3.2 Improving the Base LLL Algorithm via Bootstrapping

Proof of Theorem 1. In Theorem 7, we saw an algorithm $\mathcal{A}$ that solves any n-event LLL under the criterion $p(ed)^{32}<1$ in $T_{n,d}=O(d^2+\log^{1/4} n)$ rounds. We now explain how to bootstrap this algorithm to run in $2^{O(\sqrt{\log\log n})}$ rounds, on bounded degree graphs.

Inspired by the idea of Chang and Pettie [11], we will lie to $\mathcal{A}$ and say that the LLL graph has $n^* \ll n$ nodes, for a value of n^* to be fixed later. Then, $\mathcal{A}_{n^*}$ runs in $T_{n^*,d} = O(d^2+\log^{1/4} n^*)$ rounds. In this algorithm, the probability of any local failure (i.e., a bad event of LLL happening) is at most $1/n^*$. We can view this as a new system of bad events which satisfies a much stronger LLL criterion. In particular, we consider each of the previous bad LLL events as a bad event of the new LLL system, on the space of the random values used by $\mathcal{A}_{n^*}$, but now we connect two bad events if their distance is at most $2T_{n^*,d}+1$. Notice that if two events are not connected in this new LLL, then in algorithm $A_{n^*,d}$, they depend on disjoint sets of random variables and thus they are independent.

The degree of the new LLL system is $d' = d^{2T_{n^*,d}+1} = d^{O(d^2+\log^{1/4} n^*)}$. On the other hand, the probability of the bad events of the new system is at most $p' = 1/n^*$. Hence, the polynomial LLL criterion is satisfied with exponent $\lambda' = \frac{\log_d n^*}{O(d^2+\log^{1/4} n^*)}$. We choose

$n^* = \log n$, which, for $d = O((\log\log n)^{1/5})$, means $\lambda' = \Omega(\sqrt{\log\log n})$. Hence, this new LLL system can be solved using the LLL algorithm of Theorem 7 in time

$$(d')^2 + \lambda' \cdot \log^{1/\lambda'} n \cdot 2^{O(\sqrt{\log\log n})} =$$
$$d^{O(d^2+(\log\log n)^{1/4})} + \sqrt{\log\log n} \cdot (\log n)^{1/\Omega(\sqrt{\log\log n})} \cdot 2^{O(\sqrt{\log\log n})} = 2^{O(\sqrt{\log\log n})}.$$

We should note that these are rounds on the new LLL system, but each of them can be performed in $2T_{n^*,d} + 1 = O(d^2 + \log^{1/4} n^*) = O(\sqrt{\log\log n})$ rounds on the original graph. Hence, the overall complexity is still $2^{O(\sqrt{\log\log n})}$. ◀

We next state another result obtained via this speedup method, targeting higher degree graphs, which we will use in our coloring algorithms. The proof is deferred to the full version.

▶ **Lemma 12.** *Let $\mathcal{A}$ be a randomized* LOCAL *algorithm that solves some LCL problem $\mathcal{P}$ on n-node graphs with maximum degree $d \leq 2^{O(\log^{1/4}\log n)}$ in $O(\log^{1/4} n)$ rounds. Then, it is possible to transform $\mathcal{A}$ into a new randomized* LOCAL *algorithm $\mathcal{A}'$ that solves $\mathcal{P}$, w.h.p., in $2^{O(\sqrt{\log\log n})}$ rounds.*

4 Defective Coloring

An f-defective coloring is a (not necessarily proper) coloring of nodes, where each node has at most f neighbors with the same color. In other words, in an f-defective coloring, each color class induces a subgraph with maximum degree f. Chung, Pettie, and Su [12] gave an $O(\log n)$-round distributed algorithm for computing an f-defective coloring with $O(\Delta/f)$ colors. We here improve this complexity to $2^{O\left(\sqrt{\log\log n}\right)}$ rounds.

▶ **Theorem 13.** *There is a $2^{O\left(\sqrt{\log\log n}\right)}$-round randomized distributed algorithm that computes an f-defective $O(\Delta/f)$-coloring in an n-node graph with maximum degree Δ, w.h.p., for any integer $f \geq 0$.*

Direct LLL Formulation of Defective Coloring. Chung, Pettie, and Su [12] give a formulation of f-defective $\lceil 2\Delta/f \rceil$-coloring as LLL as follows. Each node picks a color uniformly at random. For each node v, there is a bad event D_v that v has more than f neighbors assigned the same color as v. The probability of a neighbor u having the same color as v is $f/(2\Delta)$. Hence, the expected number of neighbors of v with the same color as v is at most $f/2$. By a Chernoff bound, the probability of v having more than f neighbors with the same color is at most $e^{-f/6}$. Moreover, the dependency degree between the bad events D_v is $d \leq \Delta^2$. Therefore, $p(ed)^{32} \leq e^{-f/6+32+64\log\Delta} < 1$ for $f = \Omega(\log\Delta)$.

We are unable to directly apply our LLL algorithm of Theorem 1 to this formulation, because: (A) For $f = o(\log\Delta)$, this LLL formulation does not satisfy the polynomial criterion $p(ed)^{32} < 1$, (B) even if this criterion is satisfied, the dependency degree d may be larger than what Theorem 1 can handle.

Iterative LLL Formulation of Defective Coloring via Bucketing. Instead of directly finding an f-defective $O(\Delta/f)$-coloring with one LLL problem – i.e., a partition of G into $O(\Delta/f)$ buckets with maximum degree f each – we gradually approach this goal by iteratively partitioning the graph into buckets, until they have maximum degree f. In other words, we slow down the process of partitioning. We gradually decrease the degree, moving from maximum degree x to $\log^5 x$ in one iteration. We can see each of these bucketing steps – that

is, the partitioning into subgraphs – as a partial coloring, which fixes some bits of the final color. Each of these slower partitioning steps can be formulated as an LLL. The function $x \mapsto \log^5 x$ is chosen large enough for the corresponding LLL to satisfy the polynomial criterion, and small enough so that decreasing the degree from Δ to f does not take too many iterations, namely $O(\log^* \Delta)$ iterations only.

We now explain how a defective coloring problem can be solved using iterated bucketing. We first formulate the bucketing as an LLL problem satisfying the polynomial LLL criterion, and present ways for solving this LLL for different ranges of Δ. Then, we explain how iterated application of solving these bucketing LLLs leads to a partition of the graph into $O(\Delta/f)$ many degree-f buckets.

One Iteration of Bucketing. In one bucketing step, we would like to partition our graph with degree Δ into roughly Δ/Δ' buckets, each with maximum degree Δ', for a $\Delta' = \Omega(\log^5 \Delta)$. Notice that we can achieve the defective coloring of Theorem 13, by repeating this bucketing procedure, iteratively. See the proof of Theorem 13, which appears in the full version, for details of iterative bucketing. Each iteration of bucketing can be formulated as an LLL as follows.

LLL Formulation of Bucketing. Let $k = (1+\varepsilon)\Delta/\Delta'$ for $\varepsilon = \log^2 \Delta/\sqrt{\Delta'}$. We consider the random variables assigning each node a bucket number in $[k]$. Then, we introduce a bad event D_v for node v if more than Δ' neighbors of v are assigned the same number as v. In expectation, the number of neighbors of a node in the same bucket is at most $\Delta'/(1+\varepsilon)$. By a Chernoff bound, the probability of having more than Δ' neighbors in the same bucket is at most $p = e^{-\Omega(\varepsilon^2\Delta')} = e^{-\Omega(\log^4 \Delta)}$. Moreover, the dependency degree between these bad events is $d \leq \Delta^2$. Hence, this LLL satisfies the polynomial criterion.

If $\Delta \leq O(\log^{1/10} \log n)$, then $d = O(\log^{1/5} \log n)$, and thus we can directly apply the LLL algorithm of Theorem 1 to compute such a bucketing in $2^{O(\sqrt{\log\log n})}$ rounds. For larger values of Δ, however, we cannot apply Theorem 1. The following lemma discusses how we handle this range by sacrificing a 2-factor in the number of buckets. In a nutshell, the idea is to just perform one sampling step of bucketing, and then to deal with nodes with too large degree separately, by setting up another bucketing LLL. While the first LLL on the whole graph could not be solved directly, the second LLL is formulated only for a "small" subset of nodes, which allows an efficient solution. Because of the two trials of solving an LLL, we lose a 2-factor in the total number of buckets.

▶ **Lemma 14.** *For $\Delta \geq \Omega(\log^{1/10} \log n)$, there is a $2^{O(\sqrt{\log\log n})}$-round randomized distributed algorithm that computes a bucketing into $2k$ buckets with maximum degree Δ' each, for $\Delta' = \Omega(\log^5 \Delta)$, $\varepsilon = \log\Delta/\sqrt{\Delta'}$, and $k = (1+\varepsilon)\Delta/\Delta'$, with high probability.*

5 Frugal coloring

A *β-frugal coloring* is a proper coloring in which no color appears more than β times in the neighborhood of any node. We improve the complexity of β-frugal $O(\Delta^{1+1/\beta})$-coloring from $O(\log n)$ by Chung, Pettie, and Su [12] to $2^{O(\sqrt{\log\log n})}$.

▶ **Theorem 15.** *There is a $2^{O(\sqrt{\log\log n})}$-round randomized distributed algorithm that computes a β-frugal $(120 \cdot \Delta^{1+1/\beta})$-coloring[6] in a n-node graph with maximum degree Δ, w.h.p., for any integer $\beta \geq 1$.*

[6] We remark that we have not tried to optimize this constant 120.

Direct LLL Formulation of Frugal Coloring. Molloy and Reed [27, Theorem 19.3] formulated frugal coloring as an LLL problem in the following straight-forward way: Each node picks a color uniformly at random. There are two types of bad events: On the one hand, we have the *properness* condition, i.e., a bad event $M_{u,v}$, for each $\{u, v\} \in E$, which happens if u and v have the same color. On the other hand, the *frugality* condition – requiring that no node has more than β neighbors of the same color. That is, we have one bad event $F_{u_1,\dots,u_{\beta+1}}$ for each set $u_1, \dots, u_{\beta+1} \in N(v)$ of nodes in the neighborhood of some node v, which happens if all these nodes $u_1, \dots, u_{\beta+1}$ are assigned the same color. For palettes of size C, the probability of a bad event is at most $1/C$ for type 1 and at most $1/C^\beta$ for type 2. Each event depends on at most $(\beta+1)\Delta$ type 1 and at most $(\beta+1)\Delta\binom{\Delta}{\beta}$ type 2 events.

Iterated LLL Formulation of Frugal Coloring via Partial Frugal Coloring. While the above formulation is enough to satisfy the asymmetric tight LLL criterion for $C = O(\Delta^{1+1/\beta})$, it does not satisfy the (symmetric) polynomial LLL. Therefore, the algorithm of Theorem 1 is not directly applicable. We show how to break down the frugal coloring problem into a sequence of few partial coloring problems, coloring only some of the nodes that have remained uncolored, each of them satisfying the polynomial LLL criterion.

Roadmap. In Section 5.1, we formalize our notion of partial frugal colorings and present a method for sampling them. Then, in Section 5.2, we show how to use this sampling to formulate the problem of finding a partial frugal coloring guaranteeing progress (to be made precise) as a polynomial LLL and how to solve it. In Section 5.3, we explain how – after several iterations of setting up and solving these "progress-guaranteeing" LLLs, gradually extending the partial frugal coloring – we can set up and solve one final polynomial LLL for completing the partial coloring, also based on the sampling method presented in Section 5.1.

5.1 Sampling a Partial Frugal Coloring

▶ **Definition 16** (Partial Frugal Coloring)**.** A partial β-frugal coloring of $G = (V, E)$ is an assignment of colors to a subset $V^* \subseteq V$ such that it is proper in $G[V^*]$ and no node in V has more than β neighbors with the same color. In other words, it is a β-frugal coloring of $G[V^*]$ with the additional condition that no uncolored node in $V' := V \setminus V^*$ has more than β neighbors in V^* with the same color.

A partial coloring naturally splits the base graph G into two parts: $G[V^*]$ induced by colored nodes and $G[V']$ induced by uncolored nodes. However, the problem of extending or completing a partial frugal coloring does not only depend on $G[V']$, but also on the base graph G. That is why we introduce the notion of base-graph degree, a property of the uncolored set V' with respect to the base graph G.

▶ **Definition 17** (Base-Graph Degree of a Partial (Frugal) Coloring)**.** Given a partial coloring, we call the number $d(v, V')$ of neighboring uncolored nodes of a node $v \in V$ its *base-graph degree* into the uncolored set V'. Moreover, we call the maximum base-graph degree Δ' of a node $v \in V$ into V' the base-graph degree of V'.

In the following, we show how one can sample a partial frugal coloring, thus randomly assign some of the nodes in a set V' of uncolored nodes a color. The main idea of our sampling process is to pick a color uniformly at random, and then discard it if this choice would lead to a violation (in terms of properness and frugality). In order to increase the chances of a node being colored, instead of just sampling one color, each node v samples x different

colors from x different palettes at the same time, for some parameter $x \geq 1$, and then picks the first color that does not lead to a violation. If v has no such violation-free among its x choices, then v remains uncolored.

The next lemma, the proof of which is deferred to the full version, analyzes the probability of two kinds of events: Event (E1) that a node is uncolored. This event is important if we aim to color all the nodes in V'. Event (E2) that the base-graph degree of a node into the set of uncolored nodes in V' is too large. This event is important if we do not aim at a full coloring of all the nodes in V', but we want to ensure that we make enough progress in decreasing the base-graph degree of the uncolored set.

▶ **Lemma 18.** *Let $G = (V, E)$ be a graph with maximum degree Δ, $V' \subseteq V$ an uncolored set with base-graph degree Δ', $\beta \in [\Delta]$, and $x \geq 1$. Then there is an $O(1)$-round randomized distributed algorithm that computes a partial β-frugal $(20 \cdot x \cdot \Delta' \cdot \Delta^{1/\beta})$-coloring of some of the nodes in V' such that*

- **(i)** *the probability that a node in V' is uncolored is at most 10^{-x},*
- **(ii)** *the probability that the base-graph degree of a node $v \in V$ into the uncolored subset of V' is larger than $5^{-x} \cdot \Delta'$ is at most $e^{-\Omega(5^{-x} \cdot \Delta')}$.*

5.2 Iterated Partial Frugal Coloring

In the following, we first show how a "progress-guaranteeing" partial coloring – that is, a coloring that decreases the base-graph degree of every node quickly enough – can be found based on the sampling process presented in Section 5.1. Then, we prove that by iterating this algorithm for $O(\log^* \Delta)$ repetitions, using different palettes in each iteration, the base-graph degree reduces to $O(\sqrt{\Delta})$.

In one iteration, given a set V' of uncolored nodes, we want to color a subset $V^* \subseteq V'$ such that the uncolored nodes $V'' := V' \setminus V^*$ have a base-graph degree Δ'' that is sufficiently smaller than the base-graph degree Δ' of V'. Note that the sampling of Section 5.1 only provides us with a partial coloring where every node is likely to have a decrease in the base-graph degree. Here, however, we want to enforce that for every node in V there is such a decrease. To this end, we set up an LLL as follows.

LLL Formulation for "Progress-Guaranteeing" Coloring. Performing the sampling of Lemma 18, we have a bad event D_v for every node $v \in V$ that its base-graph degree into V'' is larger than $\Delta'' = 5^{-x} \cdot \Delta'$. By Lemma 18 (ii), we know that the probability of D_v is at most $e^{-\Omega(5^{-x} \cdot \Delta')}$. Moreover, the dependency degree is at most $d \leq \Delta^2$. This LLL thus satisfies the polynomial criterion.

However, as d might be large, we cannot directly apply the LLL algorithm of Theorem 1. In the following, we present an alternative way of finding a partial coloring ensuring a drop in the base-graph degree of every node. In a nutshell, the idea is to just perform one sampling step of a partial frugal coloring, as described in Section 5.1, and then deal with nodes associated with bad events (to be made precise) separately, by setting up another "progress-guaranteeing" LLL. While the first LLL on the whole graph could not be solved directly, the second LLL is formulated only for a "small" subset of nodes, which allows an efficient solution. Because of the two trials of solving an LLL, we lose a 2-factor in the total number of colors. The proof of the next lemma appears in the full version.

▶ **Lemma 19.** *Given a partial β-frugal coloring with uncolored set V' with base-graph degree Δ' and a parameter $x \geq 1$ such that $5^{-x} \cdot \Delta' = \Omega(\sqrt{\Delta})$, there is a $2^{O(\sqrt{\log \log n})}$-round randomized distributed algorithm that computes a partial β-frugal $(40 \cdot x \cdot \Delta' \cdot \Delta^{1/\beta})$-coloring such that the uncolored set has base-graph degree at most $\Delta'' = 5^{-x} \cdot \Delta'$.*

The next lemma describes how through iterated application of finding partial colorings, as supplied by Lemma 19, the base-graph degree of the uncolored set decreases to $O(\sqrt{\Delta})$ after $O(\log^* \Delta)$ rounds and using $O(\Delta^{1+1/\beta})$ colors. The proof appears in the full version.

▶ **Lemma 20.** *There is a $2^{O(\sqrt{\log\log n})}$-round randomized algorithm that computes a partial β-frugal $(80 \cdot \Delta^{1+1/\beta})$-coloring such that the uncolored set V' has base-graph degree $O(\sqrt{\Delta})$.*

5.3 Completing a Partial Frugal Coloring

In this section, we describe how, once the base-graph degree is $O(\sqrt{\Delta})$, all the remaining uncolored nodes can be colored, hence completing the partial frugal coloring. We first give a general formulation for the completion of partial frugal colorings.

LLL Formulation for Completion of Partial Frugal Coloring. Performing the sampling of Lemma 18, we have a bad event U_v for every node $v \in V$ that it is uncolored. By Lemma 18 (i), the probability of U_v is at most 10^{-x}. Moreover, the dependency degree d is at most Δ^2. This LLL satisfies the polynomial criterion if $x = \Omega(\log \Delta)$.

In the following lemma, the proof of which appears in the full version, we show to solve this LLL. The idea is to first perform one sampling step (of Lemma 18), which shatters the graph into "small" components of uncolored nodes, then to set up an LLL for completing the partial coloring, and finally to solve it by employing our deterministic LLL algorithm, on each of the components.

▶ **Lemma 21.** *Given a partial β-frugal coloring and a set V' of uncolored nodes with base degree $\Delta' = O(\sqrt{\Delta})$, there is a $2^{O(\sqrt{\log\log n})}$-round randomized algorithm that completes this β-frugal coloring, by assigning colors to all nodes in V', using $40 \cdot \Delta^{1+1/\beta}$ additional colors.*

A wrap-up of these results about iterated partial colorings and completing a partial coloring immediately leads to a proof of Theorem 15.

Proof of Theorem 15. We first apply the iterated coloring algorithm of Lemma 20 with $80 \cdot \Delta^{1+1/\beta}$ colors, in $2^{O(\sqrt{\log\log n})}$ rounds. Then, we run the algorithms of Lemma 21 to complete this partial coloring with $40 \cdot \Delta^{1+1/\beta}$ additional colors, in $2^{O(\sqrt{\log\log n})}$ rounds. This yields a β-frugal $(120 \cdot \Delta^{1+1/\beta})$-coloring, in $2^{O(\sqrt{\log\log n})}$ rounds. ◀

References

1 Noga Alon. A parallel algorithmic version of the local lemma. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 586–593. IEEE, 1991.

2 Noga Alon and Joel H. Spencer. *The probabilistic method.* John Wiley & Sons, 2004.

3 Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Fast distributed network decompositions and covers. *J. of Parallel and Distributed Comp.*, 39(2):105–114, 1996.

4 Baruch Awerbuch, Michael Luby, Andrew V. Goldberg, and Serge A. Plotkin. Network decomposition and locality in distributed computation. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 364–369, 1989.

5 Baruch Awerbuch and David Peleg. Sparse partitions. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 503–513, 1990.

6 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *Journal of the ACM (JACM)*, 63(3):20, 2016.

7 József Beck. An algorithmic approach to the Lovász Local Lemma. I. *Random Structures & Algorithms*, 2(4):343–365, 1991.

8 Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A lower bound for the distributed Lovász Local Lemma. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 479–488. ACM, 2016.

9 Karthekeyan Chandrasekaran, Navin Goyal, and Bernhard Haeupler. Deterministic algorithms for the Lovász local lemma. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, pages 992–1004, 2010.

10 Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the local model. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, 2016.

11 Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the local model. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, 2017, preprint arXiv:1704.06297.

12 Kai-Min Chung, Seth Pettie, and Hsin-Hao Su. Distributed algorithms for the Lovász Local Lemma and graph coloring. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 134–143, 2014.

13 Artur Czumaj and Christian Scheideler. A new algorithm approach to the general Lovász local lemma with applications to scheduling and satisfiability problems. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 38–47, 2000.

14 Paul Erdős and László Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. *Infinite and finite sets*, 10(2):609–627, 1975.

15 Manuela Fischer and Mohsen Ghaffari. Sublogarithmic distributed algorithms for Lovász local lemma, and the complexity hierarchy. *preprint arXiv:1705.04840*, 2017. URL: `https://arxiv.org/abs/1705.04840`.

16 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 625–634. IEEE, 2016.

17 Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, 2016.

18 Mohsen Ghaffari and Hsin-Hao Su. Distributed degree splitting, edge coloring, and orientations. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, 2017.

19 David G. Harris. Lopsidependency in the Moser-Tardos framework: Beyond the lopsided Lovász local lemma. *ACM Trans. Algorithms*, 13(1):17:1–17:26, December 2016.

20 David G. Harris, Johannes Schneider, and Hsin-Hao Su. Distributed $(\Delta+1)$-coloring in sublogarithmic rounds. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 465–478, 2016.

21 David G. Harris and Aravind Srinivasan. The Moser-Tardos framework with partial resampling. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 469–478. IEEE, 2013.

22 David G. Harris and Aravind Srinivasan. A constructive algorithm for the Lovász local lemma on permutations. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, SODA'14, pages 907–925, 2014.

23 David G. Harris and Aravind Srinivasan. Algorithmic and enumerative aspects of the Moser-Tardos distribution. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, pages 2004–2023, 2016.

24 Kashyap Babu Rao Kolipaka and Mario Szegedy. Moser and Tardos meet Lovász. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 235–244, 2011.

25 Nathan Linial. Distributive graph algorithms – global solutions from local data. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 331–335. IEEE, 1987.

26 Michael Molloy and Bruce Reed. Further algorithmic aspects of the local lemma. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 524–529. ACM, 1998.

27 Michael Molloy and Bruce Reed. Graph coloring and the probabilistic method, 2002.

28 Robin A Moser. A constructive proof of the Lovász local lemma. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 343–350. ACM, 2009.

29 Robin A Moser and Gábor Tardos. A constructive proof of the general Lovász Local Lemma. *Journal of the ACM (JACM)*, 57(2):11, 2010.

30 Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.

31 Alessandro Panconesi and Aravind Srinivasan. Improved distributed algorithms for coloring and network decomposition problems. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 581–592. ACM, 1992.

32 David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

33 Aravind Srinivasan. Improved algorithmic versions of the Lovász local lemma. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, pages 611–620. Society for Industrial and Applied Mathematics, 2008.