Efficient CONGEST Algorithms for the Lovász Local Lemma

Yannic Maus · yannic.maus@cs.technion.ac.il · Technion
Jara Uitto · jara.uitto@aalto.fi · Aalto University

Abstract. We present a poly log log n time randomized CONGEST algorithm for a natural class of Lovász Local Lemma (LLL) instances on constant degree graphs. This implies, among other things, that there are no LCL problems with randomized complexity between log n and poly log log n. Furthermore, we provide extensions to the network decomposition algorithms given in the recent breakthrough by Rozhoň and Ghaffari [STOC2020] and the follow up by Ghaffari, Grunau, and Rozhoň [SODA2021]. In particular, we show how to obtain a large distance separated weak network decomposition with a negligible dependency on the range of unique identifiers.
1 Introduction

Our main contribution is a poly log log $n$ round randomized distributed CONGEST algorithm to solve a natural class of Lovász Local Lemma (LLL) instances on constant degree graphs. Among several other applications, e.g., various defective graph coloring variants, this implies that there is no LCL problem (locally checkable labeling problem) with randomized complexity strictly between poly log log $n$ and $\Omega(\log n)$, which together with the results of [BCHM+21] implies that the known world of complexity classes in the sublogarithmic regime in the CONGEST and LOCAL model are almost identical. As a side effect of our techniques we extend the understanding of the computation of network decompositions in the CONGEST model. We now explain our contributions on LLL and LCLs in more detail; in the second half of the introduction we explain our techniques, contributions on network decomposition and how they relate to results in the LOCAL model.

We work in the CONGEST model of distributed computing, where a network of computational devices is abstracted as an $n$-node graph, where each node corresponds to a computational unit. In each synchronous round, the nodes can send messages of size $b = O(\log n)$ to their neighbors. In the end of the computation, each node is responsible of outputting its own part of the output/solution, e.g., its color in a graph coloring problem. The LOCAL model is otherwise the same, except that there is no bound on the message size $b$.

Background on LLL and LCLs in LOCAL. An instance of the Lovász Local Lemma problem is formed by a set of variables and a set of bad event $E_1, \ldots, E_n$ that depend on the variables. The famous Lovász Local Lemma [EL74] states that if the probability of each event is upper bounded by $p$, each event only shares variables with at most $d$ other events and $epd < 1$ holds, then there exists an assignment to the variables that avoids all bad events. An example of a problem that can be solved via LLL is the sinkless orientation problem on graphs with minimum degree 4. In the sinkless orientation problem the objective is to orient the edges of a graph such that every vertex has at least one outgoing edge. This can be modeled by an LLL as follows: Orient each edge randomly —each edge represents a variable— and introduce a bad event for each vertex that holds if and only if all edges are oriented towards it. One obtains $p = 2^{-d} = \frac{1}{2}$ and $d \geq 4$ such that $epd < 1$ holds. The Lovász Local Lemma has had a huge success in theory of computation. One highlight is the beautiful and simple parallel algorithm by Moser and Tardos [MT10]. In the distributed version of the problem each event corresponds to a node in the communication network and the assignment of the random variables, that are potentially shared by many events, is done by the corresponding nodes. The algorithm by Moser and Tardos immediately yields a randomized $O(\log^2 n)$ LLL algorithm in the LOCAL model. Often one has stronger guarantees on the relation of $p$ and $d$, e.g., a polynomial criterion $epd^2 < 1$ (or even larger exponents) instead of only $epd < 1$, and can obtain simpler and faster algorithms. E.g., in the same model, Chung, Pettie, and Su obtained a randomized algorithm that runs in time $O(\log_{epd} n)$ whenever the criterion $epd^2 < 1$ holds [CPS17].

LCLs through LLL: The main recent interest in distributed LLL are constant degree graphs motivated by the study of LCLs. LCL problems are defined on constant degree graphs and in an LCL problem each node is given an input from a constant sized set of labels and must output a label from a constant sized set of labels. The problem is characterized by a set of feasible constant-radius neighborhoods (for a formal definition see Section 4). Many classic problems are LCL problems, e.g., the problem of finding a vertex coloring with $\Delta$ colors in a graph with maximum degree $\Delta$. The systematic study of LCLs in the LOCAL model initiated by Naor & Stockmeyer [NS95] has picked up speed over the last years leading to an almost complete classification of the complexity landscape of LCL problems in the LOCAL model [CKP16, CP19, BHK+18, BBOS18, RG20]. One of the most important results in this line of research is by Chang and Pettie [CP19] who showed that any $o(\log n)$-round randomized LOCAL
algorithm for an LCL problem $P$ implies the existence of a $T_{LLL}$-round algorithm for $P$ where $T_{LLL}$ is the runtime of an LLL algorithm with an (arbitrary) polynomial criterion, e.g., for $p(ed)^{100}$. Among other implications, the breakthrough result by Rozhoň and Ghaffari provided an $(\text{poly log log } n)$-round LLL algorithm, in the LOCAL model, for the case that the dependency degree $d$ is bounded. This implies a gap in the randomized complexity landscape for LCLs. There is no LCL problem with a complexity strictly between $\text{poly log log } n$ and $\Omega(\log n)$ in the LOCAL model.

1.1 Our Results on the Distributed Lovász Local Lemma and LCLs

Motivated by the progress in LOCAL using the LCLs through LLL application, we aim to design CONGEST algorithms for the LLL problem. The first observation is that one wants to limit the range of the variables, as any reasonable algorithm should be able to send the value of a variable in a CONGEST message, and one also wants to limit the number of variables at a node such that nodes can learn assignments of all variables associated with their bad event(s) efficiently. We call an LLL instance with dependency degree $d$ range bounded if each event only depends on $\text{poly } d$ variables and each variable uses values from a range of size $\text{poly } d$. At first sight, this seems like a very restrictive setting, but in fact most instances of LLL satisfy these requirements. Our main result is contained in the following theorem and meets the poly log log $n$ round state of the art in the LOCAL model [RG20].

**Theorem 1.1.** There is a randomized CONGEST algorithm that with high probability solves any range bounded Lovász Local Lemma instance that has at most $n$ bad events, dependency degree $d = O(1)$ and satisfies the LLL criterion $p(ed)^8 < 1$ where $p$ is an upper bound on the probability that a bad event occurs, in poly log log $n$ rounds.

On the negative side, it is known that a double logarithmic dependency cannot be avoided. There is an $\Omega(\log \log n)$-round lower bound for solving LLL instances with randomized algorithms in the LOCAL model [BFH16], that holds even for constant degree graphs, and it naturally applies to the CONGEST model as well. It stems from a lower bound on the aforementioned sinkless orientation problem.

**Implications for the Theory of LCLs.** As the reduction from [CP19], that reduces sublogarithmic time randomized algorithms to LLL algorithms, immediately works in CONGEST, our fast LLL algorithm implies a gap in the complexity landscape of LCLs. A more precise definition of LCLs and a proof for the following corollary will be presented in the end of Section 4.

**Corollary 1.2.** There is no LCL problem with randomized complexity strictly between poly log log $n$ and $\Omega(\log n)$ in the CONGEST model.

In fact, Corollary 1.2 together with the results of [BCHM+21] implies that the known world of complexity classes in the sublogarithmic regime in the CONGEST and LOCAL model are almost identical. In fact, a difference can only appear in the extremely small complexity regime between $\Omega(\log^* n)$ and $O(\log^* n)$ and in the important regime of complexities that lie between $\Omega(\log \log n)$ and poly log log $n$ where complexity classes are not understood in the LOCAL model. An immediate implication of Corollary 1.2 is a poly log log $n$-round randomized CONGEST algorithm for $\Delta$-coloring when $\Delta$ is constant, a result that was previously only known in the LOCAL model [GHKM18]. Chang and Pettie [CP19] conjecture that the runtime of LLL in the LOCAL model is $O(\log log n)$ on general bounded degree graphs which would further simplify the complexity landscape for LCLs. On trees this complexity can be obtained [CHL+18] in the LOCAL model and for the specific LLL instances that arise in the study of LCLs on trees an $O(\log log n)$-round algorithm has also been found in the CONGEST model [BCHM+21].
1.2 Technical Overview and Background on our Methods

Our core technical contribution to obtain Theorem 1.1 is a bandwidth efficient derandomization of the LLL algorithm by Chung, Pettie, and Su [CPS17] that we combine with the shattering framework of Fischer and Ghaffari [FG17]. To explain these ingredients and how we slightly advance our understanding of network decompositions in the CONGEST model on the way, we begin with explaining the background on distributed derandomization and network decompositions and the relation to results in the LOCAL model.

Background on Network Decompositions and Distributed Derandomization. Network decompositions are powerful tools with a range of applications in the area of distributed graph algorithms and were introduced by Awerbuch, Luby, Goldberg, and Plotkin [AGLP89]. A \((C,D)-network decomposition\) (ND) is a partition of the vertices of a graph into \(C\) collections of clusters (or color classes of clusters) such that each cluster has diameter\(^1\) at most \(D\). Further, it is required that the clusters in the same collection are independent, i.e., are not connected by an edge. This is extremely helpful in the LOCAL model, e.g., to compute a \((\Delta + 1)\)-coloring of the network graph one can iterate through the \(C\) collections, and within each collection process all clusters in parallel (due to their independence). Due the unbounded message size dealing with a single cluster is trivial. A cluster leader can learn all information about the cluster in time that is proportional to the cluster diameter \(D\), solve the problem locally and disseminate the solution to the vertices of the cluster. Thus the runtime scales as \(O(C \cdot D)\). Hence, the objective has been to compute such decompositions as fast as possible and with \(C\) and \(D\) as small as possible, optimally, all values should be polylogarithmic in \(n\). Awerbuch et al. gave a deterministic LOCAL algorithm with round complexity and \(C\) and \(D\) equal to \(2^{O(\sqrt{\log n \log \log n})} \gg \text{poly log } n\). Panconesi and Srinivasan improved both parameters and the runtime to \(2^{O(\sqrt{\log n})}\) [PS95]. Linial and Saks showed that the optimal trade-off between diameter and the number of colors is \(C = D = O(\log n)\) and they provided an \(O(\log^2 n)\)-round randomized algorithm to compute such decompositions [LS93]. These algorithms remained the state of the art for almost three decades even though the need for an efficient deterministic algorithm for the problem has been highlighted in many papers, e.g., [BE13, GKM17a, GHK18a, CP19, GK19].

A few years ago, Ghaffari, Kuhn, and Maus [GKM17b] and Ghaffari, Harris, and Kuhn [GHK18b] highlighted the importance of network decompositions by showing that an efficient deterministic LOCAL algorithm for network decompositions with \(C = D = \text{poly log } n\) would immediately show that an efficient randomized LOCAL algorithm for any efficiently verifiable graph problem would yield an efficient deterministic algorithm. Here, all occurrences of efficient mean polylogarithmic in the number of nodes of the network. Then, in the aforementioned breakthrough Rozhoň and Ghaffari [RG20] devised such an efficient deterministic algorithm for network decompositions, yielding efficient deterministic algorithms for many problems. The result also had an immediate impact on randomized algorithms. Many randomized algorithms in the area use the shattering technique that at least goes back to Beck [Bec91] who used the technique to solve LLL instances in centralized settings. It has been introduced to distributed computing by Barenboim, Elkin, Pettie and Schneider in [BEPS16]. The shattering technique usually implements the following schematic: First, the nodes use a randomized process and the guarantee is that with high probability almost all nodes find a satisfactory output. The remainder of the graph is shattered into small components, that is, after this so called preshattering phase the unsolved parts of the graph induce small—think of \(N = \text{poly log } n\) size—connected components. In the post-shattering phase one wishes to use an efficient deterministic

\(^1\)For the sake of this exposition it is enough to assume that a cluster \(C\) has diameter \(D\), that is, any two vertices in the cluster are connected with a path within the cluster of length at most \(D\). Actually, often these short paths are allowed to leave the cluster which may cause congestion when one uses these paths for communication in two clusters in parallel. The details of the standard way to model this congestion and its impact are discussed in Section 3.
algorithm, e.g., a deterministic algorithm with complexity $T(n) = \text{poly log } n$ applied to each small component results in a complexity of $T(N) = T(\text{poly log } \log n)$. Combining the shattering technique, e.g., [BEPS16, FG17, Gha16], the network decomposition algorithm from [RG20] and the derandomization from [GKM17a, GHK18a] the randomized complexities for many graph problems in the \textit{LOCAL} model are \text{poly log } \log n.$

The Challenges in the \textit{CONGEST} Model. The holy grail would be to obtain a similar derandomization result in the \textit{CONGEST} model. But, even though the network decomposition algorithm from [RG20] immediately works in \textit{CONGEST}, we are probably far from obtaining such a result. Even for simple problems like computing a maximal independent set or a $(\Delta + 1)$-coloring one has to work much harder to even get \text{poly log } n round deterministic algorithms in the \textit{CONGEST} model, even if a $(\log n, \log n)$-network decomposition is given for free [CPS20, BKM20]. For LLL obtaining such a bandwidth efficient algorithm seems much harder. Even in the \textit{LOCAL} model one either has to go through the aforementioned derandomization result or one can solve an LLL instance with criterion $p(\text{ed})^\lambda$, e.g., think of $\lambda = 10$, if the network decomposition only has $\lambda$ color classes. This immediately implies that the cluster diameter is $\Omega(n^{1/\lambda})$ and algorithms that rely on aggregating all information about a cluster in a cluster leader must use at least $\Omega(n^{1/\lambda})$ rounds [LS93]. In the \textit{LOCAL} model there are black box reductions [GKM17a, RG20, BEG18] to compute such decompositions. We provide an analysis of the algorithm by Rozhoň and Ghashami (and of another algorithm by Ghashami, Grumau and Rozhoň [GGR21]) where we carefully study the trade-off between number of colors, the diameter of the clusters, and the runtime of the algorithm to obtain such decompositions in \textit{CONGEST}.

**Theorem 3.6** (informal). For $k \geq 1$ and $1 \leq \lambda < \log n$ there is an $O(k \cdot n^{1/\lambda} \cdot \text{poly log } n)$-round deterministic \textit{CONGEST} algorithm to compute a $(\lambda, (k \cdot n^{1/\lambda} \cdot \text{poly log } n))$-network decomposition such that any two clusters with the same color have distance strictly more than $k$. The algorithm works with a mild dependence on the ID space$^2$.

For $k = 1$, decompositions with few colors similar to Theorem 3.6 could already be obtained (using randomization) through the early works by Linial and Saks, who also showed that their trade-off of the number of colors and the cluster diameter in Theorem 3.6 is nearly optimal [LS93]. In fact, Theorem 3.6 does not just provide an improved analysis of the algorithm of [RG20] but it also comes with a mild dependence on the identifier space and an arbitrary parameter to increase the distance between clusters. Both ingredients are also present in our second result on network decompositions (Theorem 3.4) where they are crucial for the proof of Theorem 1.1.

**Our Solution:** Range bounded LLLs in \textit{CONGEST}. We use the shattering framework for LLLs of [FG17] whose pre-shattering phase, as we show, works in the \textit{CONGEST} model for range bounded LLLs. As a result we obtain small remaining components of size $N \ll n$, in fact, we obtain $N = O(\log n)$. Furthermore, one can show that the remaining problem that we need to solve on the small components is also an LLL problem. We solve these small instances via a bandwidth efficient derandomization of the LLL algorithm by Chung, Pettie, Su [CPS17]. Note that we cannot solve the small components without derandomizing their algorithm, as running their randomized algorithm on each component for $T(N) = O(\log N) = O(\log \log n)$ rounds would imply an error probability of $1/N$ which is exponentially larger than the desired high probability guarantee of $1/n$. Thus, we desire to find good random bits for all nodes with which we can execute the algorithm from [CPS17] without any error. The goal is to use a network decomposition algorithm to partition the small components into $O(\log N) = O(\log \log n)$ collections of independent clusters. Then, we iterate through the collections and want to obtain good random bits for the vertices inside each cluster. Since the randomized runtime of [CPS17] on a small component would be $T(N) = O(\log N) = O(\log \log n)$, we observe that the random bits

\footnote{To be precise, the dependency on an ID space $S$ space is a $\log^* |S|$ factor.}
of a node \( v \) cannot influence the correctness at a node \( u \) if \( u \) and \( v \) are much further than \( T(N) \) hops apart. Thus, similar to Theorem 3.6 we devise the following theorem to compute network decompositions with large distances between the clusters. In fact, if we apply the theorem with \( k > 2 \cdot T(N) \), we obtain independent clusters in each color class of the decomposition.

**Theorem 3.4** (informal). For \( k \geq 1 \) there is an \( O(k \cdot \text{poly log } n) \)-round deterministic CONGEST algorithm to compute a \((\log n, \text{poly log } n)\)-network decomposition such that any two clusters with the same color have distance strictly more than \( k \). The algorithm works with a mild dependence on the ID space.

While [RG20] provided a modification of their algorithm that provides a larger distance between clusters its runtime and cluster diameter depend polylogarithmically not only on the number of nodes in the network but also on the ID space. In [GGR21] Ghaffari, Grunau and Rozhon have reduced the ID space in the special case in which the cluster distance \( k \) equals one. However, our bandwidth efficient LLL algorithm requires \( k \gg 1 \) and identifier independence at the same time. Thus, one can either say we add the ID space independence to [RG20] or we extend the results of [GGR21] to \( k > 1 \).

We already explained why we require that we obtain a network decomposition with a large cluster distance to derandomize an algorithm. The mild dependence on the identifier space in Theorem 3.4 is also essential as the small components with \( N \) nodes on which we want to use the network decomposition algorithm, actually live in the original communication network \( G \). Thus they are equipped with identifiers that are polynomial in \( n \), that is, exponential in \( N \). The fact that the small components live in the large graph makes our life harder when computing a network decomposition but it helps us when designing efficient CONGEST algorithms. The bandwidth when executing an algorithm on the small components is still \( O(\log n) \) bits per edge per round while the components are of size \( N \ll n \). Ignoring details for the sake of this exposition, we use the increased bandwidth to gather enough information about a cluster in a single cluster leader such that it can select good random bits for all nodes of the cluster and in parallel with all other clusters of the same color class due to the large distance between clusters.

We emphasize that there have been several other approaches to derandomize algorithms in the CONGEST model and discussing all of them here would be well beyond the scope of this work. However, we still believe that our derandomization technique in the post-shattering phase might be of independent interest as it applies to a more general class of algorithms than just the aforementioned LLL algorithm from [CPS17].

**Other Implications.** As the runtime of Theorem 3.4 is a \( \log n \) factor faster than the result in [RG20], it also improves the complexity of deterministic distance-2 coloring with \((1 + \varepsilon)\Delta^2\) colors in the CONGEST model to \( O(\log^7 n) \) rounds when plugged into the framework of [HKM20]. Similar improvements carry over to the approximation of minimum dominating sets [DKM19] and spanner computations [GK18]. Due to the identifier independence and possibility to increase the distance between clusters Theorem 3.4 yields an improved randomized complexity of distance-2 coloring. Using Theorem 3.4 in the shattering based algorithm from [HKMN20] improves the runtime for \( \Delta^2 + 1 \) colors from \( 2^{O(\log \log n)} \) to \( O(\log^7 \log n) \) rounds.

**Roadmap:** In Section 1.3, we provide pointers for further related work, mainly on network decompositions. In Section 2 we define the models and introduce notation. In Section 3 we formally state the result on network decompositions and indicate the main changes to prior work. The formal proofs of these results appear in Appendices C to E. The main part of the paper deals with proving Theorem 1.1 and appears in Section 4.

### 1.3 Further Related Work

We already mentioned that [GGR21] provides an efficient deterministic CONGEST algorithm with a mild dependence on the ID space. More precisely, they provided a \( (O(\log n, \log^2 n))\)-
network decomposition in $O(\log^5 n + \log^4 n \cdot \log^* b)$ rounds, where $\log^* b$ is the dependency on the identifier length $b$ [GGR21]. One drawback that we have ignored until now—and that also applies to all of our results—is that these network decompositions only have so called \textit{weak diameter}; that is, the diameter of a cluster is only guaranteed to be small if it is seen as a subset of the communication network $G$, that is, the distance between two vertices is measured in $G$ and not only in the subgraph induced by a cluster. In contrast, in the so called \textit{strong network decompositions} the diameter in the subgraph of $G$ that is induced by each cluster has to be small. Very recently, at the cost of increasing the polylogarithmic factors in the runtime the results of [GGR21] were extended to obtain strong ($O(\log n, \log^2 n)$)-network decompositions [CG21]. Earlier works by Elkin and Neiman provided an $O(\log^2 n)$ randomized algorithm for computing strong ($O(\log n), O(\log n)$)-decompositions [EN16].

Most previous works in the \textsc{CONGEST} model do not ensure large distances between clusters. Besides the result in [RG20] that we have already discussed there are two notable exceptions. Ghaffari and Kuhn matched the complexity of the \textsc{LOCAL} model algorithm by Awerbuch et al. by giving a deterministic $k \cdot 2^{O(\sqrt{\log n \log \log n})}$ round algorithm [GK18]. Later, this was improved to $k \cdot 2^{O(\sqrt{\log n})}$ rounds by Ghaffari and Portmann [GP19]. In both cases, the decomposition parameters were identical to the runtimes. None of these results is the state of the art anymore, except for the fact that they compute strong network decompositions with large distances between the clusters.

Barenboim, Elkin, and Gavoille provide various algorithms with different trade-offs between the number of colors and the cluster diameter, most notably a randomized algorithm to compute a strong network decomposition with diameter $O(1)$ and $O(n^\epsilon)$ colors [BEG18]. A similar result with $O(n^{1/2+\epsilon})$ colors was obtained by Barenboim in [Bar12].

Brandt, Maus, and Uitto and Brandt, Grunau, and Rozhnoi have shown that LLL instances with an exponential LLL criterion, that is, $p2^d < 1$ holds, can be solved in $O(\log^* n)$ rounds on bounded degree graphs [BMU19, BGR20]. Furthermore, it is known that $\Omega(\log^* n)$ rounds cannot be beaten under any LLL criterion that is a function of the dependency degree $d$ [CPS17]. For an exponential criterion that satisfies $p2^d \geq 1$, it follows from the works of Brandt et al. and Chang, Kopelowitz, and Pettie [CKP16] that there is a lower bound of $\Omega(\log n)$ rounds for deterministic LLL. Hence [BMU19, BGR20] and [CKP16] show that there is a sharp transition of the distributed complexity of LLL at $2p^d = 1$. Before the result in [RG20] improved the runtime to poly $\log \log n$, Ghaffari, Harris, and Kuhn gave the state of the art randomized LLL algorithm in the \textsc{LOCAL} model. On constant degree graphs its runtime was described by a tower function and lies between poly $\log \log n$ and $2^{O(\sqrt{\log \log n})}$ [GHK18a]. We are not aware of any works that explicitly studied LLL in the \textsc{CONGEST} model before our paper.

## 2 Models, LCLs & Notation

Given a graph $G = (V, E)$ the hop distance in $G$ between two vertices $u, v \in V$ is denoted by $dist_G(u, v)$. For a vertex $v \in V$ and a subset $S \subseteq V$ we define $dist_G(v, S) = \min_{u \in S} \{dist(v, u)\} \in \mathbb{N} \cup \{\infty\}$. For two subsets $S, T \subseteq V$ we define $dist_G(S, T) = \min_{v \in T} dist_G(v, S)$. For an integer $n$ we denote $[n] = \{0, \ldots, n - 1\}$.

The \textsc{LOCAL} and \textsc{CONGEST} Model of distributed computing [Lin92, Pel00]. In both models the graph is abstracted as an $n$-node network $G = (V, E)$ with maximum degree at most $\Delta$. Communication happens in synchronous rounds. Per round, each node can send one message to each of its neighbors. At the end, each node has to know its own part of the output, e.g., its own color. In the \textsc{LOCAL} model there is no bound on the message size and in the \textsc{CONGEST} model messages can contain at most $O(\log n)$ bits. Usually, in both models nodes are equipped with $O(\log n)$ bit IDs (polynomial ID space) and initially, nodes know their own ID or their own color in an input coloring but are unaware of the IDs of their neighbors.
Randomized algorithms do not use IDs (but they can create them from space \(n^c\) for a constant \(c\) and with a \(1/n^c\) additive increase in the error probability), have a fixed runtime and need to be correct with probability strictly more than \(1 − 1/n\) (Monte Carlo algorithm). Actually, algorithms are defined such that a parameter \(n\) is provided to them, where \(n\) represents an upper bound on the number of nodes of the graph and the algorithm’s guarantees have to hold on any graph with at most \(n\) nodes. For technical reasons in our gap results, we assume that the number of random bits used by each node is bounded by some finite function \(h(n)\). We note that the function \(h(n)\) can grow arbitrarily fast and that this assumption is made in previous works as well [CKP16, BCHM+21].

**LCL definition [NS95].** An LCL problem \(\Pi\) is a tuple \((\Sigma_{in}, \Sigma_{out}, F, r)\) satisfying the following.

- Both \(\Sigma_{in}\) and \(\Sigma_{out}\) are constant-size sets of labels,
- the parameter \(r\) is an arbitrary constant, called the *checkability radius* of \(\Pi\),
- \(F\) is a finite set of pairs \((H = (V^H, E^H), v)\), where:
  - \(H\) is a graph, \(v\) is a node of \(H\), and the radius of \(v\) in \(H\) is at most \(r\);
  - Every pair \((v, e) \in V^H \times E^H\) is labeled with a label in \(\Sigma_{in}\) and a label in \(\Sigma_{out}\).

In an instance of an LCL problem \(\Pi = (\Sigma_{in}, \Sigma_{out}, F, r)\) on a graph \(G = (V, E)\) each node receives an *input label* from \(\Sigma_{in}\). An algorithm solves \(\Pi\) if, for each node \(v \in V\), the radius-\(r\) hop neighborhood of \(v\) together with the input labels and computed outputs for the nodes lies in the set of *feasible labelings* \(F\).

## 3 Graph Decompositions

In this section we state our results on network decompositions and provide the respective definitions. At the end of the section we provide a short overview of the techniques that we use to prove the results. The formal proofs appear in Appendices C to E.

### 3.1 Cluster Collections and Network Decompositions with Congestion

Classically, one defines a *(weak) \((c,d)\)-network decomposition* as a coloring of the vertices of a graph \(G\) with \(c\) colors such that the connected components of each color class have weak diameter at most \(d\). The *weak diameter* of a subset \(C \subseteq V\) is the maximum distance in \(G\) that any two vertices of \(C\) have.

We use the following definition that augments one color class of a network decomposition with a communication backbone for each cluster. A *cluster* \(C\) of a graph is a subset of nodes. A *Steiner tree* is a rooted tree with nodes labeled as *terminal* and *nonterminal*.

**Definition 3.1 (cluster collection).** A collection of clusters of a graph \(G = (V, E)\) consists of subsets of vertices \(C_1, \ldots, C_p \subseteq V\) and has Steiner radius \(\beta\) and unique \(b\)-bit cluster identifiers if it comes with associated Steiner subtrees \(T_1, \ldots, T_p\) of \(G\) such that clusters are disjoint, i.e., \(C_i \cap C_j = \emptyset\) for \(i \neq j\) and for each \(i \in 1, 2, \ldots, p\) we have

1. cluster \(C_i\) has a unique \(b\)-bit identifier \(id_{C_i}\),
2. the terminal nodes of Steiner tree \(T_i\) of cluster \(C_i\) are formed by \(C_i\) (\(T_i\) might contain nodes \(\notin C_i\) as non terminal nodes),
3. the edges of Steiner Tree \(T_i\) are oriented towards a cluster leader \(\ell_{C_i}\),
4. Steiner tree \(T_i\) has diameter at most \(\beta\) (Steiner tree diameter).
The collection has congestion $\kappa$ if each edge in $E$ is contained in at most $\kappa$ Steiner trees. When we compute a cluster collection in the CONGEST model, we require that each node of a cluster knows the cluster identifier. Furthermore, we require that for each edge of $T_i$ that is incident to some $v \in V$, node $v$ knows $id_C$, and additionally $v$ knows the direction of the edge towards the root $r_C$.

We now define our notion of a network decomposition with a communication backbone.

**Definition 3.2** (network decomposition with congestion). Let $k \geq 1$. A $(C, \beta)$-network decomposition with cluster distance $k$ and congestion $\kappa$ of a graph $G = (V, E)$ is a partition of $V$ into $C$ cluster collections with Steiner radius $\beta$ and congestion $\kappa$ such that any two clusters $C \neq C'$ in the same cluster collection have distance strictly more than $k$, i.e., $\text{dist}_G(C, C') > k$.

The $C$ cluster collections are also called the $C$ color classes of the decomposition.

Instead of a network decomposition with cluster distance $k$, we often just speak of a network decomposition of $G^k$. In all cases the Steiner trees are given by vertices and edges in $G$ and not by edges in $G^k$, which would correspond to paths in $G$. The default value for $k$ is 1 which we use whenever we do not specify its value. In this case the definition corresponds with the classic network decomposition.

### 3.2 Network Decomposition and Ball Carving Algorithms

At the core of our network decomposition algorithm is the following ball carving result to form one color class of the decomposition, i.e., to form one cluster collection. The name ball carving stems from the original existential proof for network decompositions [AGLP89].

**Lemma 3.3** (ball carving, $k \geq 1$). Let $k \geq 1$ be an integer and $x \geq 1$ (both potentially functions of $n$). Then there is a deterministic distributed CONGEST (bandwidth $b$ and $b$-bit identifiers) algorithm that, given a graph $G = (V, E)$ with at most $n$ nodes and a subset $S \subseteq V$, computes a cluster collection $C_1, \ldots, C_p \subseteq S$ with

- $|C_1 \cup \ldots \cup C_p| \geq (1 - 1/x) \cdot |S|$,  
- Steiner radius $\beta = O(k \cdot x \cdot \log^3 n)$ and congestion $\kappa = O(\log n \cdot \min\{k, x \cdot \log^2 n\})$,  
- the pairwise distance in $G$ between $C_i$ and $C_j$ for $1 \leq i \neq j \leq p$ is strictly more than $k$.

The runtime is $O(k \cdot x \cdot \log^4 n \cdot \log^b b) + O(k \cdot x^2 \cdot \log^6 n \cdot \min\{k, x \cdot \log^2 n\})$ rounds.

By using the aggregation tools for overlapping Steiner trees of [GGR21] Lemma 3.3 is by a log $n$ factor faster than the corresponding result in [RG20]. The ball carving result immediately imply the following network decomposition results.

**Theorem 3.4** ($O(\log n)$ colors). For any (potentially non constant) $k \geq 1$, there is a deterministic CONGEST algorithm with bandwidth $b$ that, given a graph $G$ with at most $n$ nodes and unique $b$-bit IDs from an exponential ID space, computes a (weak) $(O(\log n), O(k \cdot \log^3 n))$-network decomposition with cluster distance $k$ and with congestion $O(\log^2 n \cdot \min\{k, \log^2 n\})$ in $O(k \cdot \log^7 n \cdot \min\{k, \log^2 n\})$ rounds.

**Proof.** Set $x = 2$ and apply Lemma 3.3 for $\log_2 n$ iterations, always with the set of nodes that have not been added to a cluster in one of the previous iterations. Notice that the bandwidth demand by Lemma 3.3 is satisfied since the IDs have at most $b$ bits. Each iteration forms a separate color class and as the number of remaining vertices is reduced by a factor 2 in each iteration each vertex is contained in one cluster at the end. \[\square\]
We also analyze a more involved ball carving algorithm of [GGR21] for different parameters in order to obtain decompositions with fewer colors. Then Lemmas 3.3 and 3.5 imply the following result for network decompositions with few colors.

**Lemma 3.5 (faster ball carving, \(k = 1\)).** Let \(x \geq 1\) (potentially a function of \(n\)). Then there is a deterministic distributed CONGEST (bandwidth \(b\) and \(b\)-bit identifiers) algorithm that, given a graph \(G = (V, E)\) with at most \(n\) nodes and a subset \(S \subseteq V\), computes a cluster collection \(C_1, \ldots, C_p \subseteq S\) with

- \(|C_1 \cup \ldots \cup C_p| \geq (1 - 1/x) \cdot |S|\),
- Steiner radius \(\beta = O(x \cdot \log^2 n)\) and congestion \(\kappa = O(\log n)\),
- the pairwise distance in \(G\) between \(C_i\) and \(C_j\) for \(1 \leq i \neq j \leq p\) is strictly more than 1.

The runtime of the algorithm is \(O(x^2 \log^4 n)\) rounds.

**Theorem 3.6 (few colors).** For any \(\lambda \leq \log n\) there is a deterministic CONGEST algorithm with bandwidth \(b\) that, given a graph \(G\) with at most \(n\) nodes and unique \(b\)-bit IDs from an exponential ID space, computes a weak \((\lambda, n^{1/\lambda} \log^3 n)\)-network decomposition of \(G\) in \(O(\lambda \cdot n^{2/\lambda} \cdot \log^4 n)\) rounds with congestion \(\kappa = O(\log n)\).

For any (possibly non constant) \(k \geq 1\) and any \(\lambda \leq \log n\) there is a deterministic CONGEST algorithm that, given a graph \(G\) with at most \(n\) nodes and unique IDs from an exponential ID space, computes a weak \((\lambda, k \cdot n^{1/\lambda} \log^3 n)\)-network decomposition of \(G^k\) in \(O(k \cdot n^{2/\lambda} \cdot \log^6 n \cdot \min\{k, x \log^2 n\})\) rounds.

**Proof.** For the first result apply Lemma 3.5 with \(x = n^{1/\lambda}\) and for \(\lambda\) iterations, always with the set of nodes that have not been added to a cluster in one of the previous iterations. Notice that the bandwidth demand by Lemma 3.5 is satisfied since the IDs have at most \(b\) bits. For the second result do the same with Lemma 3.3. In both cases the clusters created in each iteration form a separate color class. The number of remaining vertices decreases by a factor \(x\) in each iteration and after \(\lambda\) iterations all vertices have been clustered. \(\square\)

Theorems 3.4 and 3.6 also work with more general ID spaces, as long as the IDs fit into a CONGEST message. To simplify the statements and because exponential IDs are (currently) the main application we do not explicitly state their mild dependence on the size of the ID space \(S\) in theorems. It can be quantified by \(O(x \cdot \text{poly log } n \cdot \log^* |S|)\) where the polynomial logarithmic terms are strictly dominated by the ones in the theorems.

**Summary of Our Ball Carving Contributions.** Both, the ball carving results in [RG20] and [GGR21] begin with each vertex of the to be partitioned set \(S\) (see Lemmas 3.3 and 3.5) being its own cluster inheriting its node ID as the cluster ID. Then they implement a distributed ball growing approach in which vertices leave their cluster, join other clusters, or become dead, i.e., are disregarded; even whole clusters can dissolve, e.g., if all their vertices join a different cluster. The goal is to ensure that at the end no two alive clusters are neighboring, that is, in distance \(k\), that the fraction of vertices of \(S\) that are declared dead is small and that the diameter of the clusters does not become too large. In fact, as vertices can leave clusters one only obtains guarantees on the weak diameter of clusters. To this end the algorithm of [RG20] iterates through the bits of the cluster IDs and in phase \(i\), vertices change their clusters or become dead such that at the end of the \(i\)-th phase no two neighboring clusters have the same \(i\)-th bit in their cluster ID. If cluster IDs have \(b\) bits, then after the \(b\)-th phase each remaining cluster is not connected to any other cluster. Thus, the runtime crucially depends on the size of the cluster IDs. The main change in [GGR21] to remove this ID space dependence is to replace the bits (0 or 1) used in phase \(i\) with a coloring of the clusters with two colors, red and blue. A
new coloring is computed in every phase and the crucial property of the coloring that ensures progress towards the separation of the clusters is that each connected component of clusters has roughly the same number of red and blue clusters. Thus, intuitively, at the end of a phase red and blue clusters are separated and the sizes of all connected components decrease.

In our algorithms we follow the same high level approach but extend the techniques (for identifier “independence”) to work with a larger separation between the clusters, and also for the computation of decompositions with fewer colors. A crucial difficulty is that we cannot quickly disregard all dead vertices and reason that each connected components of clusters in $G$ can be treated independently. It might even be that two clusters are connected by a path with $\leq k$ hops and the path might contain dead vertices. Instead, we consider connected components in $G^k$ and always keep all vertices of $G$ in mind. As two clusters that are adjacent in such a component might not be adjacent in the original graph we ensure that no congestion appears in between the clusters. Careful algorithm design and reasoning is needed when computing a balanced coloring of clusters in such components.

To obtain the network decompositions with fewer colors, e.g., with $\lambda = 10$ colors, we analyze our algorithm for a suitable choice of parameters and show that all congestion parameters are not affected by the choice of $\lambda$.

4 Distributed Range Bounded LLL and its Implications

The objective of this section is to prove the following theorem.

**Theorem 1.1.** There is a randomized CONGEST algorithm that with high probability solves any range bounded Lovász Local Lemma instance that has at most $n$ bad events, dependency degree $d = O(1)$ and satisfies the LLL criterion $p(ed)^8 < 1$ where $p$ is an upper bound on the probability that a bad event occurs, in poly log log $n$ rounds.

**Distributed Lovász Local Lemma (LLL).** In a distributed Lovász Local Lemma instance we are given a set of independent random variables $V$ and a family $X$ of (bad) events $E_1, \ldots, E_n$ on these variables. Each bad event $E_i \in X$ depends on some subset $vbl(E_i) \subseteq V$ of the variables. Define the dependency graph $H_X = (X, \{(E, E') | vbl(E) \cap vbl(E') \neq \emptyset\})$ that connects any two events which share at least one variable. Let $\Delta_H$ be the maximum degree in this graph, i.e., each event $E \in X$ shares variables with at most $d = \Delta_H$ other events $E' \in X$. The Lovász Local Lemma [EL74] states that $Pr(\bigcap_{E \in X} E) > 0$ if $ed < 1$. In the distributed LLL problem each bad event and each variable is assigned to a vertex of the communication network such that all variables that influence a bad event $E$ that is assigned to a vertex $v$ are either assigned to $v$ or a neighbor of $v$. The objective is to compute an assignment for the variables such that no bad event occurs. At the end of the computation each vertex $v$ has to know the values of all variables that influence one of its bad events (by definition these variables are assigned to $v$ or one of its neighbors). In all results on distributed LLL in the LOCAL model that are stated below the communication network is identical to the dependency graph $H$. Most result in the distributed setting require stronger LLL criteria.

Despite its simplicity—the algorithm simply iteratively resamples local ID minima of violated events, i.e., bad events that hold under the current variable assignment —Algorithm 1 results in the following theorem.

**Theorem 4.1** ([CPS17], Algorithm 1). Given an LLL instance with condition $ed^2 < 1$, the LOCAL Algorithm 1 run for $O(\log_{ed^2} n)$ rounds and has error probability $< 1/n$ on any LLL instance with at most $n$ bad events.

Next, we define the notion of a range bounded LLL instances.
Initialize a random assignment to the variables

Let $F$ be the set of bad events under the current variable assignment

while $F \neq \emptyset$ do

Let $I = \{ A \in F \mid ID(A) = \min\{ID(B) \mid B \in N_F(A)\} \}$

Resample $vbl(I) = \cup A \in I vbl(A)$

end

Algorithm 1: The algorithm from [CPS17] iteratively resamples all variables of local ID minima of violated events. IDs can be assigned in an adversarial manner. The runtime is with high probability $O(\log_{epd^2} n)$ rounds under LLL criterion $epd^2 < 1$.

Definition 4.2 (Range bounded Lovász Local Lemma). An instance $(V, X)$ of the Lovász Local Lemma with bad events $E_1, \ldots, E_n$ and dependency graph $H$ is range bounded if $|vbl(E_i)| = \text{poly } \Delta_H$ for each $1 \leq i \leq n$ and the value of each random variable $x \in V$ can be expressed by at most $O(\log \Delta_H)$ bits.

Note that most applications of the Lovász Local Lemma in the distributed form are range bounded instances. Algorithm 1 can be executed with the same asymptotic runtime for range bounded LLL instances and uses few random bits while doing so.

Lemma 4.3. For a range bounded LLL instance with condition $epd^2 < 1$ with constant dependency degree there is a randomized CONGEST (constant bandwidth) algorithm that runs in $O(\log_{epd^2} n)$ rounds and has error probability $< 1/n$ on any (dependency) graph with at most $n$ nodes. Furthermore, in the algorithm each node only requires access to $O(\log n)$ random values from a bounded range. The unique IDs can be replaced by an acyclic orientation of the edges of the graph.

Proof. Let $\Delta$ be the degree of the dependency graph. In Algorithm 1 local minima with regard to an arbitrary ID assignment resample their variables. The algorithm is oblivious to the values of the IDs and the IDs can be replaced with an arbitrary acyclic orientation on the edges. To check which events need to be resampled nodes only need to learn the currently sampled values of adjacent nodes (and the orientation of the edges). As the LLL is range bounded the values of variables fit into $O(1)$ bits and can be communicated in one round. In each iteration of the algorithm, each vertex resamples only its own random variables (or none). To resample one variable it requires a random value from the bounded range poly $\Delta = O(1)$, and in each iteration it resamples at most poly $\Delta = O(1)$ variables.

The currently fastest randomized algorithm for LLL on bounded degree graphs with polynomial criterion in the LOCAL model is based on two main ingredients. The first ingredient is a deterministic poly log $n$-round LLL algorithm obtained via a derandomization (cf. [GKM17a, GHK18a]) of the $O(\log^2 n)$-round randomized algorithm [MT10] using the breakthrough efficient network decomposition algorithm [RG20]. The second ingredient is the (randomized) shattering framework for LLL instances by [FG17], which shatters the graph into small unsolved components of logarithmic size. Then, these components are solved independently and in parallel with the deterministic algorithm.

We follow the same general approach of shattering the graph via the methods of [FG17], but need to be more careful to obtain a bandwidth efficient algorithm when dealing with the small components.

We begin with a lemma that describes how fast we can gather information in a cluster leader with limited bandwidth. Its proof uses standard pipelining techniques (see e.g. [Pel00, Chapter 3] and fills each $b$-bit messages with as much information as possible.

Lemma 4.4 (Token learning). Let $G$ be a communication graph on $n$ vertices in which each node can send $b$ bits per round on each edge. Assume a cluster collection with Steiner radius $\beta$
and congestion $\kappa$ in which each cluster $\mathcal{C}$ is of size at most $N$ and each vertex holds at most $x$ bits of information. Then, in parallel each cluster leader $\ell_{\mathcal{C}}$ can learn the information of each vertex of $\mathcal{C}$ in $O(\kappa \cdot (\beta + N \cdot x/b))$ rounds. In the same runtime the leader $\ell_{\mathcal{C}}$ can disseminate $x$ bits of distinct information to each vertex of $\mathcal{C}$.

Next, we prove our core derandomization result that we use to obtain efficient deterministic algorithms for the small components that arise in the post-shattering phase. It might be of independent interest.

**Lemma 4.5 (Derandomization).** Consider an LCL problem $P$, possibly with promises on the inputs. Assume a $T(n)$-round randomized CONGEST (bandwidth $b = \Theta(\log n)$) algorithm $A$ for $P$ with error probability $< 1/n$ on any graph with at most $n$ nodes that uses at most $O(\log n)$ random bits per node.

Then for any graph on at most $N$ nodes there is a deterministic $T(N) \cdot \text{poly} \log N$ round CONGEST (with bandwidth $b = \Theta(N)$) algorithm $B$ to solve $P$ under the same promises, even if the ID space is exponential in $N$.

**Proof.** For an execution of algorithm $A$ on a graph with $N$ nodes define for each $v \in V$ an indicator variable $X_v$ that equals to 1 if the verification of the solution fails at node $v$ and 0 otherwise. The value of $X_v$ depends on the randomness of nodes in the $(T(N) + r)$ hop neighborhood of $v$, where $r = O(1)$ is defined by the LCL problem.

We design an efficient algorithm $B$ that deterministically fixes the (random bits) of each node $v \in V$ such that $X_v = 0$ for all nodes. Thus, executing $A$ with these random bits solves problem $P$. During the execution of $B$ we fix the random bits of more and more nodes. At some point during the execution of algorithm $B$ let $X \subseteq V$ denote the nodes whose randomness is already fixed and let $\phi_X$ be their randomness (we formally define $\phi_X$ later). The crucial invariant that we maintain for each cluster $\mathcal{C}$ at all times is the following

$$\text{Invariant: } E\left[\sum_{v \in V} X_v | \phi_X\right] < 1. \quad (1)$$

The invariant will be made formal in the rest of the proof. Initially (when $X = \emptyset$), it holds with the linearity of expectation and as the error probability of the algorithm implies that $E[X_v] = P(X_v = 1) < 1/n$ holds.

**Algorithm $B$:** Compute a weak network decomposition with cluster distance strictly more than $4(T(N) + r)$ with $O(\log N)$ color classes, $\beta = T(N) \cdot \text{poly} \log N$ Steiner tree cluster radius and congestion $\kappa = O(\log N)$. Iterate through the color classes of the decomposition and consider each cluster separately. We describe the process for one cluster $\mathcal{C}$ when processing the $i$-th color class of the network decomposition. We define three layers according to the distance to vertices in $\mathcal{C}$. Denote the vertices in $\mathcal{C}$ as $W^0_{\mathcal{C}}$, the vertices in $V \setminus W^0_{\mathcal{C}}$ in distance at most $r + T(N)$ from $\mathcal{C}$ as $W^1_{\mathcal{C}}$ and the vertices in $V \setminus (W^0_{\mathcal{C}} \cup W^1_{\mathcal{C}})$ with distance at most $2(r + T(N))$ from $\mathcal{C}$ as $W^2_{\mathcal{C}}$. Define $W'_{\mathcal{C}} = W^0_{\mathcal{C}} \cup W^1_{\mathcal{C}} \cup W^2_{\mathcal{C}}$. Note that we have $W_{\mathcal{C}} \cap W'_{\mathcal{C}} = \emptyset$ for two distinct clusters $\mathcal{C}, \mathcal{C}'$ in the same color class of the decomposition due to the cluster distance. Further, even if Lemma 4.5 is applied to a subgraph $H$ (with size at most $N$) of a communication network $G$ we have $|W_{\mathcal{C}}| \leq N$, as $W_{\mathcal{C}}$ only contains nodes of $H$.

**Dealing with one cluster $\mathcal{C}$:** Extend the Steiner tree of $\mathcal{C}$ to $W_{\mathcal{C}}$ using a BFS, ties broken arbitrarily. Notice that no two BFS trees interfere with each other since we handle different color classes separately and two clusters of the same color are in large enough distance. Use the Steiner tree to assign new IDs (unique only within the nodes in $W_{\mathcal{C}}$) from range $[N]$ to all nodes in $W_{\mathcal{C}}$. Different clusters use the same set of IDs and each ID can be represented with $O(\log N)$ bits. Each node of $W_{\mathcal{C}}$ learns about the new ID of each of its (at most) $\Delta$ neighbors in $W_{\mathcal{C}}$ in one round and uses $O(\Delta \cdot \log N) = O(\log N)$ bits to store its adjacent edges. Using Lemma 4.4,
the cluster leader $\ell_C$ learns the whole topology, inputs and already determined random bits of $G[W_C]$ in $O(\kappa \cdot (\beta + (N \cdot \text{poly log } N)/b))$ rounds. In this step, the input also includes an acyclic orientation of the edges between vertices of $W_C$, where an edge is oriented from $u \in W_C$ to $v \in W_C$ if the original ID of $v$ is larger than the original ID of $v$.

For a node $v$ let $R_v$ describe its randomness. When we process cluster $C$ we determine values $\{r_v \mid v \in C\}$ for the random variables $\{R_v \mid v \in C\}$. Let $X$ be a set of vertices $v$ for which we have already determined values $r_v$. Then we denote $\phi_X = \bigwedge_{v \in X} (R_v = r_v)$.

**Claim 4.6.** Assume that Invariant (1) holds for $X = \bigcup_{C \text{ has color } < i} C$ before processing the clusters of color $i$. Then the cluster leaders $\ell_C$ of all clusters $C$ of color $i$, can in parallel, find values $\{r_v \mid v \in C\}$ such that Invariant (1) holds afterwards for $X' = \bigcup_{C \text{ has color } \leq i} C$.

If we considered just a single cluster $C$, independently from all other clusters with color $i$, then there are choices for the values of $r_v$ satisfying the claim due to the law of total probability, e.g., as used in the method of conditional expectation, and as $E\left[ \sum_{v \in V} X_v \mid \phi_X \right] < 1$ holds before we fix any randomness of vertices inside $C$.

**Proof.** The only information that cluster leaders need to perform the necessary calculations is information on the topology of $G[W_C]$, inputs to nodes in $W_C$, the relative order of adjacent IDs of nodes in $G[W_C]$ and already determined randomness of nodes in $X \cap W_C$.

We show that the randomness of all vertices in clusters with color $i$ can be fixed with this knowledge. To this end the cluster leader $r_C$ fixes the randomness $\phi_C$ such that

$$E\left[ \sum_{v \in W_C} X_v \mid \phi_C \land \phi_{X \cap W_C} \right] \leq E\left[ \sum_{v \in W_C} X_v \mid \phi_{X \cap W_C} \right]$$

holds. Such a choice for $\phi_C$ exists, again due to the law of total probability, and $\ell_C$ has full information to compute such values as all values in the inequality only depend on information that vertices in $W_C$ sent to $\ell_C$. In particular, the randomness of nodes in $C$ only influences the random variables $X_v$ of nodes in $W_C^0 \cup W_C^i$ and computing this influence only requires knowledge from $W_C^0 \cup W_C^i \cup W_C^3$.

After all cluster leaders of clusters with color $i$ fix the randomness of the vertices in their clusters according to the above method the invariant is still satisfied since the randomness $R_v$ for $v \in C$ does not influence the random variable $X_u$ for $u \in V \setminus W_C$. More formally, let $X$ be the set of vertices with fixed randomness before we process the $i$-th color class of clusters and let $Y$ be the set of vertices whose randomness we fix when processing the $i$-th color class. Let $X' = X \cup Y$. Let $W = \bigcup_{C \text{ has color } i} W_C$. Recall that $W_C \cap W_C' = \emptyset$ for $C \neq C'$. Due to the linearity of expectation and the aforementioned reasons we obtain

$$E\left[ \sum_{v \in V} X_v \mid \phi_X \right] = E\left[ \sum_{v \in V \setminus W} X_v \mid \phi_X \right] + \sum_{C \text{ has color } i} E\left[ \sum_{v \in W_C} X_v \mid \phi_{X \cap W_C} \right]$$

$$\leq E\left[ \sum_{v \in V \setminus W} X_v \mid \phi_X \right] + \sum_{C \text{ has color } i} E\left[ \sum_{v \in W_C} X_v \mid \phi_{X \cap W_C} \right]$$

$$= E\left[ \sum_{v \in V} X_v \mid \phi_X \right] < 1.$$  

All clusters can be processed in parallel as their distance is strictly more than $4((N) + r)$ and the choices in cluster $C$ do not change the probability for $X_v = 1$ for $v \in C' \neq C$. □

Once the cluster leader $\ell_C$ has fixed the randomness for the vertices in $C$, it disseminates them to all vertices of its cluster via Lemma 4.4 and we continue with the next color class of the
network decomposition. At the end of the algorithm each node knows the values of its random bits and one can execute \( T(N) \) with it, as the initial algorithm \( A \) also works with bandwidth \( b \).

Due to the Invariant (1) we obtain that the distributed random bits are such that \( E[\sum_{v \in V} X_v | \phi_v] < 1 \) at the end of the algorithm, but as all \( X_v \)'s are in \( \{0, 1\} \) and there is no randomness involved (each vertex is processed in at least one cluster and thus we fix \( R_v = r_v \) for each \( v \in V \)) we obtain that \( X_v = 0 \) for all \( v \in V \), that is, the algorithm does not fail at any vertex.

**Runtime:** Computing the network decomposition takes \( T(N) \cdot \text{poly} \log N \) rounds via Theorem 3.4. The runtime for processing one color class of the network decomposition is bounded as follows. Collecting the topology and all other information takes \( T(N) \cdot \text{poly} \log N \) rounds due to Lemma 4.4, fixing randomness locally does not require communication. Disseminating the random bits to the vertices of each cluster takes \( \text{poly} \log N \) rounds, again due to Lemma 4.4. As there are only \( \log N \) color classes the runtime for computing good random bits for all nodes is \( T(N) \cdot \text{poly} \log N \). Executing algorithm \( A \) with these random bits takes \( T(N) \) rounds.

**Lemma 4.7.** There is a deterministic LLL algorithm for range bounded LLL instances with constant dependency degree \( d \) and LLL criterion \( epd^2 < 1 \) that runs in \( O(\text{poly} \log \log n) \) rounds on any graph with at most \( n \) nodes if the communication bandwidth is \( b = \Theta(n) \).

**Proof.** Plug the randomized algorithm of Lemma 4.3 into the derandomization result of Lemma 4.5. We can apply the lemma because the feasibility of a computed solution can be checked in 1 round in **CONGEST**.

**Proof of Theorem 1.1.** The goal is to apply the shattering framework of [FG17]. The pre-shattering phase takes \( \text{poly} \Delta + O(\log^* n) \) rounds and afterwards all events with an unset variable induce small components of size \( N = \text{poly}(\Delta, \log n) \). Furthermore, each of the small components also forms an LLL instance, but with a slightly worse LLL criterion. Then we apply Lemma 4.7 to solve all small components in parallel. Next, we describe these steps in detail.

The pre-shattering phase begins with computing a distance-2 coloring of the dependency graph \( H \) with \( O(\Delta^2) \) colors, i.e., a coloring in which each color appears only once in each inclusive neighborhood. In the **LOCAL** model this takes \( O(\log^* n) \) rounds with Linial’s algorithm [Lin92]. In the **CONGEST** model we can compute such a coloring in \( O(\text{poly} \Delta_H \cdot \log^* n) = O(\log^* n) \) rounds.

Next, we iterate through the \( O(\Delta^2) \) color classes to set some of the variables. The unset variables either have the status *frozen* or *non-frozen*. At the beginning all variables are unset and non-frozen. We iterate through the \( O(\Delta_H^2) \) color classes and process all vertices (events) of the same color class in parallel. Each non-frozen variable of a processed event \( E \) is sampled; then the node checks how the conditional probabilities of neighboring events have changed. As the LLL is range bounded this step can be implemented in \( O(1) \) rounds. If there is an event \( E' \) (possibly \( = E \)) whose probability has increased to at least \( p' = \sqrt{p} \), its variables are unset and all variables of event \( E' \) are frozen. The next three observations are proven in [FG17] and capture the properties of this pre-shattering phase.

**Observation 4.8.** For each event \( E \in X \), the probability of \( E \) having at least one unset variable is at most \( (d + 1)\sqrt{p} \). Furthermore, this is independent of events that are further than 2 hops from \( E \).

The following result follows with Observation 4.8 and the by now standard shattering lemma (see Lemma B.1). We do not discuss the details as it has been done in [FG17].

**Observation 4.9 (Small components).** The connected components in \( H \) induced by all events with at least one unset variable are w.h.p. in \( n \) of size \( N = \text{poly}(\Delta_H) \cdot \log n = O(\log n) \).

The following observation holds as each event with an unset variable fails at most with probability \( p' = \sqrt{p} \) when its variables are frozen. It is proven in [FG17].
**Observation 4.10.** The problem on each connected component induced by events with at least one unset variable is an LLL problem with criterion \( p'(d + 1) < 1 \).

To complete our proof we apply the deterministic algorithm of Lemma 4.7 on each component in parallel. Let \( U \) be the set of nodes that have its random bits not yet determined. Note that any vertex in \( U \) is part of one component. For each \( u \in U \) the values of already determined random bits in the \( r \)-hop ball around \( u \) are included in \( u \)'s input—many nodes already determine their random bits in the pre-shattering phase. Even conditioned on the random bits determined in the pre-shattering phase (formally this provides a promise to the inputs), each instance is a range bounded LLL on a bounded degree graph with at most \( N \) nodes and the standard \textsc{congest} bandwidth is \( b = \Omega(\log n) = \Omega(N) \), the runtime is \( \text{poly } \log N = \text{poly } \log \log n \). All instances can be dealt with independently since by definition, the connected unsolved components share no events nor variables.

The celebrated result by \cite{CP19} says that an LCL problem either cannot be solved faster than in \( \Omega(\log n) \) rounds or can be solved with an LLL algorithm.

**Lemma 4.11** (\cite{CP19}). Any LCL problem that can be solved with a randomized \( o(\log n) \)-round \textsc{local} algorithm with error probability < \( 1/n \) on any graph with at most \( n \) nodes can be solved via the following procedure: Create a bounded range LLL instance with LLL criterion \( p(\varepsilon d)^{100} < 1 \), solve the instance, and run a constant time deterministic \textsc{local} algorithm that uses the solution of the LLL instance as input.

**Proof.** If we omitted the requirement that the LLL instance has to be range bounded the lemma is directly proven in \cite{CP19}. Thus, we focus on showing that the obtained LLL instance is range bounded. We first explain the setup of the proof of \cite{CP19}.

To solve an LCL with an \( o(\log n) \) round algorithm \( A \) in the \textsc{local} model, they set up an LLL. The random variables of a node in this LLL consist of the random bits that are needed for an execution of algorithm \( A \) for a carefully chosen constant number of rounds \( t_0 = \tilde{O}(1) \). There is a bad event for each node \( v \in V \) of the communication network which holds if \( A \) executed with the random bits violates the LCL constraint of \( v \). Thus, the goal of the LLL is to find an assignment of good random bits such that the execution of \( A \) for \( t_0 \) rounds does not fail (in solving the LCL problem) at any node.

The core of \cite{CP19} is to show that due to the assumptions on the runtime function one can lie to the algorithm \( A \) about the number of nodes in the input graph, run the algorithm with a 'fake' small value for the number of nodes at the cost of increasing the error probability. They show that there exists an absolute constant \((!\) value \( n_0 = O(1) \) that only depends on the LCL and the runtime function \( t_A \) of algorithm \( A \) such that the problem of finding good random bits for \( A(n_0) \), that is \( A \) executed for \( t_0 = t_A(n_0) \) rounds under the belief that the graph has at most \( n_0 \) vertices, is indeed an LLL problem satisfying the LLL criterion \( p(\varepsilon d)^{100} < 1 \). Actually, they show that for any constant \( c > 1 \) there is a choice of \( n_0 \) such that the criterion \( \varepsilon p d^c < 1 \) holds.

The only part of Lemma 4.11 that is not explicitly proven in \cite{CP19} is the fact that the obtained LLL is range bounded. We next reason that this merely follows as \( n_0 \) is an absolute constant. We emphasize that this is the part of the proof that is not argued in the previous work but follows as a simple observation from the proof details. The dependency radius (in the communication network) of this LLL is \( 2(t_A(n_0) + r) \) where \( r \) is the checking radius of the LCL. Recall, that we assume that, in a randomized algorithm, the number of random bits used by a node on a graph on at most \( n_0 \) nodes can be upper bounded by an arbitrarily fast growing function \( h_A(n) \). Thus, the number of random bits (or number of variables) of a node in the LLL is bounded by \( h_A(n_0) = O(1) \). The LLL is range bounded as any variable, i.e., any random bit, can only take one out of two values.

Once the LLL is solved one can solve the original LCL problem by executing \( A(n_0) \) in \( t_0 = t_A(n_0) = O(1) \) \textsc{local} rounds (this execution is deterministic given the random bits from
the LLL solution). An $O(1)$-round LOCAL algorithm can be run in CONGEST in $\text{poly } \Delta = O(1)$ rounds.

The influential result of Lemma 4.11 of [CP19] has been used in several other works, e.g., in [BCHM+21, BGR21]. Note that [BCHM+21] sets up the same LLL with the purpose of designing an efficient CONGEST algorithm for it; however, [BCHM+21] is restricted to the setting where the input graph is a tree which allows for very different methods of solving the respective LLL and admits, as they show, even an $O(\log \log n)$-round algorithm. We combine Lemma 4.11 with our LLL algorithm in Theorem 1.1 to prove Corollary 1.2.

**Corollary 1.2.** There is no LCL problem with randomized complexity strictly between $\text{poly log log } n$ and $\Omega(\log n)$ in the CONGEST model.

**Proof of Corollary 1.2.** Let $P$ be an LCL problem with randomized CONGEST complexity $o(\log n)$, which also implies a LOCAL algorithm with the same complexity. Now, we use the CONGEST LLL algorithm from Theorem 1.1 in the framework of Lemma 4.11 to obtain a CONGEST algorithm with runtime $O(\text{poly log log } n)$ for $P$. Observe that even though the LLL construction requires running the LLL algorithm from Theorem 1.1 in a powergraph, since we are dealing with a bounded degree input graph, the dependency degree of the LLL remains a constant. This approach also requires that we can execute the constant time LOCAL algorithm at the end in the CONGEST model, which follows as the maximum degree is constant for LCLs, the solution to the LLL is constant as the LLL is range bounded and also the output for the LCL problem is constant.

The results in this section imply randomized poly log log $n$-round algorithms for classic problems such as $\Delta$-coloring on constant degree graphs (as $\Delta$-coloring is an LCL which has an $o(\log n)$-round LOCAL algorithm [GHKM18] the result follows along the same lines as Corollary 1.2) and various defective coloring variants for constant degree graphs by modeling them as range bounded LLLs and applying Theorem 1.1, see [CPS17] for various such problems and how they can be modeled as LLLs.

Our network decomposition algorithm with few colors in combination with the LOCAL model LLL algorithm from [FG17] provides the following theorem. Due to the unconstrained message size in the LOCAL model, it does not require the LLL instances to be range bounded.

**Theorem 4.12.** Let $\lambda \in \mathbb{N}$ be constant. There exists a deterministic $n^{2/\lambda} \text{poly log } n$ round LOCAL algorithm for LLL instances with criterion $p(\text{ed})^\lambda < 1$.

**Proof.** Let $H$ be the dependency graph of the LLL. Use Theorem 3.6 to compute a network decomposition of $H^2$ with $\lambda$ colors and cluster diameter $n^{1/\lambda} \text{poly log } n$ in $n^{2/\lambda} \text{poly log } n$ rounds; in the LOCAL model we can ignore any congestion between the clusters. From [FG17, arxiv version, Theorem 3.5], we get that under the LLL criterion $p(\text{ed})^\lambda$ and given a $(\lambda, \gamma)$ network decomposition, we can solve the LLL in $O(\lambda \cdot (\gamma + 1) \text{ rounds})$. By plugging in $\gamma = n^{1/\lambda} \text{poly log } n$, we observe that the computation of the network decomposition dominates the runtime and we obtain the result.

**Acknowledgments**

This project was partially supported by the European Union’s Horizon 2020 Research and Innovation Programme under grant agreement no. 755839 (Yannic Maus).
References

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

[Bar12] Leonid Barenboim. On the locality of some np-complete problems. In Automata, Languages, and Programming - 39th International Colloquium (ICALP), pages 403–415, 2012.

[BBOS18] Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. Almost global problems in the LOCAL model. In 32nd International Symposium on Distributed Computing, DISC, pages 9:1–9:16, 2018.

[BCHM+21] Alkida Balliu, Keren Censor-Hillel, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Locally checkable labelings with small messages. In International Symposium on Distributed Computing DISC, 2021.

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

[Bec91] József Beck. An algorithmic approach to the Lovász local lemma. Random Structures & Algorithms, 2(4):343–365, 1991.

[BEG18] Leonid Barenboim, Michael Elkin, and Cyril Gavoille. A fast network-decomposition algorithm and its applications to constant-time distributed computation. Theor. Comput. Sci., 751:2–23, 2018.

[BEPS16] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. Journal of the ACM, 63(3):20:1–20:45, 2016.

[BFH+16] Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiainen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A Lower Bound for the Distributed Lovász Local Lemma. In ACM Symposium on Theory of Computing (STOC), 2016.

[BGR20] Sebastian Brandt, Christoph Grunau, and Václav Rozhon. Generalizing the sharp threshold phenomenon for the distributed complexity of the lovász local lemma. In ACM Symposium on Principles of Distributed Computing (PODC), pages 329–338, 2020.

[BGR21] Sebastian Brandt, Christoph Grunau, and Václav Rozhon. The randomized local computation complexity of the lovász local lemma. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, ACM Symposium on Principles of Distributed Computing (PODC), pages 307–317. ACM, 2021.

[BHK+18] Alkida Balliu, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiainen, Dennis Olivetti, and Jukka Suomela. New classes of distributed time complexity. In Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC), pages 1307–1318, 2018.

[BKM20] Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Efficient deterministic distributed coloring with small bandwidth. In ACM Symposium on Principles of Distributed Computing (PODC), pages 243–252, 2020.
[BMU19] Sebastian Brandt, Yannic Maus, and Jara Uitto. A sharp threshold phenomenon for the distributed complexity of the Lovász local lemma. In Proceedings of the 2019 ACM Symposium on Principles of Distributed (PODC), pages 389–398, 2019.

[CG21] Yi-Jun Chang and Mohsen Ghaffari. Strong-diameter network decomposition. the Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC), abs/2102.09820, 2021.

[CHL+18] Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. The complexity of distributed edge coloring with small palettes. In Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 2633–2652, 2018.

[CKP16] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An Exponential Separation between Randomized and Deterministic Complexity in the LOCAL Model. In the Proceedings of the Symposium on Foundations of Computer Science (FOCS), pages 615–624, 2016.

[CP19] Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the LOCAL model. SIAM J. Comput., 48(1):33–69, 2019.

[CPS17] Kai-Min Chung, Seth Pettie, and Hsin-Hao Su. Distributed Algorithms for the Lovász Local Lemma and Graph Coloring. Distributed Computing, 30(4):261–280, 2017.

[CPS20] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. Distributed Comput., 33(3-4):349–366, 2020.

[DKM19] Janosch Deurer, Fabian Kuhn, and Yannic Maus. Deterministic distributed dominating set approximation in the CONGEST model. In Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC), pages 94–103, 2019.

[EL74] Paul Erdős and László Lovász. Problems and Results on 3-chromatic Hypergraphs and some Related Questions. Colloquia Mathematica Societatis János Bolyai, pages 609–627, 1974.

[EN16] Michael Elkin and Ofer Neiman. Distributed strong diameter network decomposition. In Proc. 35th ACM Symp. on Principles of Distributed Computing (PODC), pages 211–216, 2016.

[FG17] Manuela Fischer and Mohsen Ghaffari. Sublogarithmic Distributed Algorithms for Lovász Local Lemma, and the Complexity Hierarchy. In the Proceedings of the 31st International Symposium on Distributed Computing (DISC), pages 18:1–18:16, 2017.

[GGR21] Mohsen Ghaffari, Christoph Grunau, and Václav Rozhoň. Improved deterministic network decomposition. In the Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), 2021.

[Gha16] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In Proc. 27th ACM-SIAM Symp. on Discrete Algorithms (SODA), pages 270–277, 2016.
[GHK18a] Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. On derandomizing local
distributed algorithms. In 59th IEEE Annual Symposium on Foundations of
Computer Science, FOCS 2018, Paris, France, October 7-9, 2018, pages 662–673,
2018.

[GHK18b] Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. On Derandomizing Local
Distributed Algorithms. In the Proceedings of the Symposium on Foundations of
Computer Science (FOCS), pages 662–673, 2018.

[GHKM18] Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, and Yannic Maus. Improved
distributed $\delta$-coloring. In Proceedings of the 2018 ACM Symposium on Principles
of Distributed Computing (PODC), pages 427–436, 2018.

[GK18] Mohsen Ghaffari and Fabian Kuhn. Derandomizing distributed algorithms with
small messages: Spanners and dominating set. In 32nd International Symposium on
Distributed Computing (DISC), pages 29:1–29:17, 2018.

[GK19] Mohsen Ghaffari and Fabian Kuhn. On the use of randomness in local distributed
graph algorithms. In Proceedings of the 2019 ACM Symposium on Principles of
Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2,
2019, pages 290–299, 2019.

[GKM17a] Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the complexity of local
distributed graph problems. In ACM Symposium on Theory of Computing (STOC),
pages 784–797, 2017.

[GKM17b] Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the complexity of local
distributed graph problems. In ACM Symposium on Theory of Computing (STOC),
pages 784–797. ACM, 2017.

[GP19] Mohsen Ghaffari and Julian Portmann. Improved network decompositions using
small messages with applications on mis, neighborhood covers, and beyond. In
33rd International Symposium on Distributed Computing (DISC), pages 18:1–18:16,
2019.

[HKM20] Magnús M. Halldórsson, Fabian Kuhn, and Yannic Maus. Distance-2 coloring in
the CONGEST model. In ACM Symposium on Principles of Distributed Computing
(PODC), pages 233–242, 2020.

[HKMN20] Magnús M. Halldórsson, Fabian Kuhn, Yannic Maus, and Alexandre Nolin. Coloring
fast without learning your neighbors’ colors. In 34th International Symposium on
Distributed Computing (DISC), pages 39:1–39:17, 2020.

[Lin92] Nati Linial. Locality in distributed graph algorithms. SIAM Journal on Computing,
21(1):193–201, 1992.

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

[MT10] Robin A. Moser and Gábor Tardos. A Constructive Proof of the General Lovász
Local Lemma. J. ACM, pages 11:1–11:15, 2010.

[NS95] M. Naor and L. Stockmeyer. What can be computed locally? SIAM Journal on
Computing, 24(6):1259–1277, 1995.

[Pel00] David Peleg. Distributed Computing: A Locality-Sensitive Approach. SIAM, 2000.
A Tree aggregation of [GGR21]

The following result was essentially proven in [GGR21, Corollary 5.3, arxiv version].

**Corollary A.1.** Let $G$ be a communication graph on $n$ vertices. Suppose that each vertex of $G$ is part of some cluster $C$ such that each such cluster has a rooted Steiner tree $T_C$ of diameter at most $\beta$ and each node of $G$ is contained in at most $\kappa$ such trees. Then, in $O(\max\{1, \kappa/b\} \cdot (\beta + \kappa))$ rounds of the CONGEST model with $b$-bit messages, we can perform the following operations for all clusters in parallel on all clusters:

1. **Broadcast:** The root of $T_C$ sends a $b$-bit message to all nodes in $C$;
2. **Convergecast:** We have $O(1)$ special nodes $u \in C$, where each special node starts with a separate $b$-bit message. At the end, the root of $T_C$ knows all messages;
3. **Minimum:** Each node $u \in C$ starts with a non negative $b$-bit number $x_u$. At the end, the root of $T_C$ knows the value of $\min_{u \in C} x_u$;
4. **Summation:** Each node $u \in C$ starts with a non negative $b$-bit number $x_u$. At the end, the root of $T_C$ knows the value of $(\sum_{u \in C} x_u) \mod 2^{O(b)}$.

**Proof.** Use $x = \max\{1, \kappa/b\}$ CONGEST $b$-bit rounds to simulate one CONGEST $x \cdot b$-bit round. Then, apply [GGR21, Corollary 5.3, arxiv version] with a slowdown of $x$. \hfill \Box

B Shattering

We use the following by now standard shattering result for our LLL algorithm in Section 4. This lemma and its variants have been applied in a vast amount of results in the area of distributed computing and its application is not limited to solving LLL instances.

**Lemma B.1** (The Shattering Lemma [FG17] cf. [BEPS16]). Let $H = (V, E)$ be a graph with maximum degree $\Delta_H$. Consider a process which generates a random subset $B \subseteq V$ where $\Pr(v \in B) \leq \Delta^{-c_1}$, for some constant $c_1 \geq 1$, and where 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 $Z = H[2c_2 + 1, 4c_2 + 2]$ be the graph which contains an edge between $u$ and $v$ if and only if their distance in $H$ is between $2c_2 + 1$ and $4c_2 + 2$. Then with probability at least $1 - n^{-c_3}$, for any constant $c_3$ satisfying $c_1 > c_3 + 4c_2 + 2$, we have the following two properties:

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

(P2) Each connected component of $H[B]$ has size at most $O(\log_\Delta n \cdot \Delta^{2c_2})$. 

20
C Faster Distance-1 Color Class Carving with Levels

In this section we prove the following result.

Lemma 3.5 (faster ball carving, $k = 1$). Let $x \geq 1$ (potentially a function of $n$). Then there is a deterministic distributed CONGEST (bandwidth $b$ and $b$-bit identifiers) algorithm that, given a graph $G = (V, E)$ with at most $n$ nodes and a subset $S \subseteq V$, computes a cluster collection $C_1, \ldots, C_p \subseteq S$ with

- $|C_1 \cup \ldots \cup C_p| \geq (1 - 1/x) \cdot |S|$,
- Steiner radius $\beta = O(x \cdot \log^2 n)$ and congestion $\kappa = O(\log n)$,
- the pairwise distance in $G$ between $C_i$ and $C_j$ for $1 \leq i \neq j \leq p$ is strictly more than 1.

The runtime of the algorithm is $O(x^2 \log^4 n)$ rounds.

Throughout this section fix the parameter $x > 0$ (possibly depending on $n$). In this section we analyze the trade-off between the number of colors in a resulting network decomposition, the cluster diameter and the runtime in the network decomposition algorithm from [GGR21]. Hence, some of the proofs are along the lines of their counterparts in [GGR21] or streamlined versions of their proofs. Here, we focus on the ball carving for a single color class.

C.1 Algorithm: Color Class Carving with Levels

Parameters: The algorithm and its analysis uses the following parameters.

- Levels = $\log_{4/3} n + 1$
- Phases = $2 \cdot$ Levels + $2 \log_2 n$
- totalTokens = $4 \cdot$ Phases (upper bound on tokens created per node)
- PayPerKill = totalTokens $\cdot$ $x$
- ProposalParameter = $1/(2 \cdot$ PayPerKill$)$
- Steps = $1/$ProposalParameter = $2$PayPerKill = $8 \cdot x \cdot$ Phases

In the description of the algorithm, we did purposely not describe how the red/blue coloring is performed, how a proposal is implemented and also not how the Steiner Trees are formed as none of these internals is influenced by the choice of $x$. Each step can be implemented similar to the ones in the previous section and have been detailed on in [GGR21].

Algorithm Initialization: Each node is its own cluster and possesses one token, we run the red/blue coloring with a large distance, each cluster is at level 0. Clusters can either be stalled or alive. We define $t_i(C)$ as the number of tokens of $C$ at the beginning of the $i$-th phase and set $t_1(C) = 1$.

Algorithm: The algorithm has Phases phases and each phase has Steps steps. At the beginning of the phase each cluster wakes up if it was stalling in the previous phase. Then all clusters that have changed their level from the previous phase forget about their previous red/blue color and get a new red/blue color via [GGR21, arxiv, proposition 4.3]. Clusters that have not changed their levels keep their red/blue color.

3This red/blue coloring provides stronger properties than the red/blue coloring in Appendix E. These properties are used to prove the separation of clusters. As the runtime of computing the coloring is dominated by the other steps (also for $x = \omega(1)$) we do not detail on the involved specifics.
One Step:

Proposing: In each step every (!) vertex $v$ in each cluster $C$, including vertices in stalling clusters, determines whether there is an adjacent cluster $C' \neq C$ with $(\text{lev}_C, \text{color}_C) < (\text{lev}_{C'}, \text{color}_{C'})$ with priority on the level. If the answer is yes node $v$ creates a new token and sends a proposal to an arbitrary such cluster.

Accepting/Rejecting: For a cluster $C$ let $P_C$ be the set of vertices that propose to $C$ in a fixed step. if $|P_C| \geq t(C)/\text{ProposalParameter}$ holds, the cluster accepts all proposals and it increases its token counter by one for each vertex in $P_C$, otherwise the cluster kills all vertices in $P_C$ and pays PayPerKill for each killed vertex, that is, it decreases its token counter $t(C)$ by $\text{PayPerKill} \cdot |P_C|$ tokens. If the cluster rejects proposals it becomes stalling until the end of the phase. If a cluster accepts the proposals its token counter increases by 1 for each accepted vertex. At the end of the phase each stalling cluster increments its level by one.

Steiner Tree Forming: At the end of each step accepted proposals join the cluster that they proposed to and Steiner trees are build. This is the end of the algorithm description.

Useful Clarifications. It is essential that also vertices in stalled clusters keep proposing. The level of a cluster does not change within a phase and never decreases; it can only be incremented by one in the transition between two consecutive phases. As proven in [GGR21] a cluster that has reached the highest level does not receive proposals anymore as it is non adjacent to any other cluster. In a phase a vertex that initially is in a cluster of level 10 can go to a cluster of level 9, then 8, then 7 ... and so on.

C.2 Analysis: Correctness, Steiner Trees and Runtime

The proof of the following observation uses exactly the same ideas as the corresponding proof in [GGR21] but we measure the potential of vertices instead of the potential of clusters.

**Observation C.1** (small total number of tokens). Each vertex only changes its cluster $3 \cdot \text{Phases} + 1$ times and we create at most $\text{totalTokens} := 4 \cdot \text{Phases} \cdot |S|$ tokens in total.

**Proof.** We assign a potential to a node, depending on the phase, the level of its current cluster and the color of its cluster. We denote with $\text{lev}_i(C)$ the level of $C$ during phase $i$. Let $v$ be a node, consider phase $i$ and let $C$ be the current cluster of node $v$ with color $\text{color}_i(C) \in \{0,1\} = \{\text{red}, \text{blue}\}$, where red = 0 and blue = 1. Note that the phase does not uniquely determine the cluster of node $v$ as nodes change their cluster during a phase. But, for a fixed phase $i$ and cluster $C$ its color $\text{color}_i(C)$ is uniquely determined. Then, let the potential of $v$ be $\Phi(v) = 3i - 2\text{lev}_i(C) + \text{color}_i(C)$

Initially, the potential of a node is $\Phi(v) = O(1)$. The potential of a vertex does not decrease when we go to phase $i + 1$, as the phase counter increases by 1 (which increases the potential by 3), $\text{lev}_{i+1}(C) \leq \text{lev}_i(C) + 1$ (which decreases the potential at most by 2) and $\text{color}_{i+1}(C) \leq \text{color}_i(C) + 1$ (which decreases the potential at most by 1). When a vertex changes from cluster $C$ to some cluster $C'$ in a step during in some phase the potential increases at least by 1, as the phase counter remains the same and either the level remains the same as well and $\text{color}_i(C') = \text{color}_i(C) + 1$, or $\text{lev}_i(C') \leq \text{lev}_i(C) - 1$ and $\text{color}_i(C') = \text{color}_i(C) + 1$.

The first claim follows as the potential of a vertex is upper bounded by $3\text{Phases} + 1$. The second part of the claim holds as after the initialization there are $|S|$ tokens in the system and each vertex in $S$ can create at most $3\text{Phases} + 1$ tokens, totaling in $(3\text{Phases} + 1)|S| + |S| \leq 4\text{Phases} \cdot |S|$ tokens, as $\text{Phases} \geq 2$. \qed

**Lemma C.2** (Few vertices die). Total number of vertices that die is $|S|/x$.

**Proof.** This requires that you do not pay too much per vertex. Due to Observation C.1, each vertex creates at most $\text{Phases}$ tokens, that is, there are $4 \cdot |S| \cdot \text{Phases}$ tokens in total. Whenever a
vertex is killed we remove \( \text{PayPerKill} \) tokens permanently from the algorithm. Thus, the number of vertices that die is upper bounded by

\[
\frac{\text{totalTokens}}{\text{PayPerKill}} \leq \frac{|S|}{x}
\]

While, the previous claims did not depend on the number of steps, the choice of \( \text{Steps} \) is crucial for the next lemma. In Appendix E each (blue) cluster became stalling in each phase. As stalling blue clusters are separated from red clusters in that algorithm, there is synchronous progress towards separation. For the algorithm in this section the separation implied by a stalling cluster is weaker and also the measure of progress is asynchronous. Blue clusters do not have to get stalled in every phase, but we show that a cluster either becomes stalling in a phase or it increases its cluster counter by a factor \( \geq 2 \). A cluster is finished once it has reached the last level \( \text{Levels} \).

**Lemma C.3** (Progress on Token counter). In each phase, each non-finished cluster either doubles its token counter, or it levels up and keeps at least half of its tokens. Further, the token counter of a cluster is always at least 1.

**Proof.** Let \( t \) be the token counter of a cluster \( C \) at the beginning of some phase.

- If \( C \) is not finished and does not level up after the phase its token counter increases by a \((1 + \text{ProposalParameter})\) factor in each step. Using the number of steps in a phase we obtain that its token counter at the end of the phase is at least

\[
(1 + 1/\text{ProposalParameter})^{\text{Steps}} \cdot t \geq (1 + \text{Steps}/\text{ProposalParameter}) \cdot t \geq 2t.
\]

- Let \( t' \geq t \) be the token counter of cluster \( C \) at the beginning of the step in which it becomes stalling. As the cluster becomes stalling in this step the number of vertices that propose in the step is at most \( t'/\text{ProposalParameter} \). Thus, the number of tokens removed by \( \text{paying PayPerKill} \) for each rejected proposal is upper bounded by

\[
\text{PayPerKill} \cdot t'/\text{ProposalParameter} = \text{PayPerKill} \cdot t'/(2 \text{PayPerKill}) \geq t'/2.
\]

Thus, the cluster keeps at least \( t' - t'/2 = t'/2 \geq t/2 \) tokens when it becomes stalling in a phase.

- The fact that at most \( t/2 \) tokens are removed from the tokens of a cluster in each step together with the fact that a cluster never kills a vertex if it only has one token (if a cluster has only one token it even accepts if it only gets a single proposal) implies that the token counter is always at least 1.

With Lemma C.3 and the total limit on the number of tokens given by Observation C.1 it is straightforward to show that every cluster has to reach the largest level by the last phase.

**Lemma C.4** (Last level is reached). At the end of the algorithm each cluster is in the largest level.

**Proof.** Due to Lemma C.3, whenever a cluster does not level up in a phase, it doubles its number of tokens. When it increases its level (at most by one per phase) it keeps at least half of its tokens. If a cluster only leveled up for \( \text{Levels} - 1 \) times, it never finishes and it doubled its value in \( \text{Phases} - (\text{Levels} - 1) \) phases, while always having at least one token (Lemma C.3). That is, it would posses

\[
(1/2)^{(\text{Levels}-1)2\text{Phases}-(\text{Levels}-1)} = 2^{\text{Phases} - 2(\text{Levels}-1)} = 2^{2 \log_2 n} = n^2,
\]

a contradiction to the total number of tokens that exist in the system (due to Observation C.1 this is upper bounded by \(|S| \cdot 4\text{Phases} < n^2 \) (this uses that \( n \) is larger than a sufficiently large constant, e.g., 300)).
One of the most involved parts in the analysis of the algorithm is to prove that clusters are separated at the end of the execution. None of the steps in [GGR21] is influenced by the choice of $x$ and thus we only state the following lemma. The most involved part is to prove that clusters are actually separated at the end of the execution. The proof is along similar lines in [GGR21]. The following lemma is very local in time and only holds for the phase in which a cluster is stalling. It also crucially requires that vertices of stalling clusters keep proposing.

**Lemma C.5.** At the end of the algorithm every cluster separated.

The fact that Steiner trees are formed properly is proven in [GGR21]. Here, we only restate their observations that influence the runtime of the algorithm and the parameters of the computed decomposition.

**Observation C.6 (Steiner tree growth).** In each step, the radius of the Steiner tree of each cluster grows by at most 1.

**Observation C.7.** Over the whole course of the algorithm each edge is added to at most $\kappa = O(\text{Phases})$ Steiner trees.

**Proof.** As we consider $k = 1$ an edge is only added to a Steiner tree if one of its endpoints proposes to the other endpoint. Due to Observation C.1 this can happen at most $O(\text{Phases})$ times for each endpoint. \hfill $\square$

**Proof of Lemma 3.5.** Due to Lemma C.2 at most $|S|/x$ vertices die in the process and due to Lemma C.5 each cluster is separated at the end of the process. Due to Observation C.6 each Steiner tree grows by at most 1 per step, that is, the weak diameter of the computed decomposition is upper bounded by $\beta = O(\text{PhasesSteps}) = O(x \cdot \log^2 n)$. Due to Observation C.7 each edge is in at most $O(\text{Phases}) = O(\log n)$ Steiner trees.

The runtime is dominated by the aggregation (to determine the number of proposals for each cluster) in each of the steps. Using the aggregation result from Corollary A.1 one aggregation step can be performed in $O(\beta + \kappa) = O(x \cdot \log^2 n)$ rounds. There are $\text{Phases} \cdot \text{Steps}$ of these aggregations, which results in a runtime of

$$O(\text{Phases} \cdot \text{Steps} \cdot x \cdot \log^2 n)) = O(x^2 \log^4 n)$$

\hfill $\square$

## D Balanced Coloring Algorithms for Cluster Graphs

The goal of this section is to devise algorithms that color cluster graphs in a balanced way. These colorings are used in the network decomposition algorithms in Appendix E and Appendix C.

In a cluster collection $C_1, \ldots, C_p$ a cluster $C_i$ is distance-$k$ isolated if for all nodes $v \in C_i$ and each $C_j, i \neq j$ and each $w \in C_j$ the distance of $v$ and $w$ in $G$ is strictly larger than $k$. A component is said to be distance-$k$ connected if each pair of nodes in the component are connected by a path in the power graph $G^k$. In this section we prove the following lemma.

**Lemma D.1.** Let $k$ be an integer. Consider a graph $G = (V, E)$ and a cluster collection $C_1, \ldots, C_p$ with congestion $\kappa$, Steiner radius $\beta$ and unique $b$-bit cluster identifiers. Let the bandwidth on the communication graph $G$ be $B \geq b$. There is a deterministic $O(k + (\beta + \kappa) \log^* b + \beta \cdot \kappa)$ round algorithm to color each cluster red or blue such that in each distance-$k$ connected component of clusters with at least 2 clusters, the fraction of vertices colored blue lies within the interval $[1/2, 3/4]$.

The high level idea of the following algorithm stems from [GGR21, arxiv version Lemma 3.1 and 3.2]. However, to deal with $k > 1$ we develop careful whitebox implementations of each step. The most important implementation step is captured by Lemma D.3.
Algorithmic steps:

- **Step 0 (color isolated clusters):** Each distance-$k$ isolated cluster is colored red.

- **Step 1 (path selection):** Each cluster $C$ that is not distance-$k$ isolated selects a (directed) path of length at most $k$ to some other cluster $C' \neq C$.

- **Step 2 (determine heavy/light clusters):** A cluster is heavy if it has at least 11 incoming paths, otherwise a cluster is light. Each cluster determines whether it is heavy or light.

- **Step 3 (color children of heavy clusters):** Heavy clusters color the clusters that selected them as balanced as possible favoring blue (largest possible imbalance $\pm 1$). The heavy cluster does not color itself in this step.

- **Step 4 (color heavy clusters):** Color each heavy cluster that has not been colored in step 3, blue.

- **Step 5 (color remaining light clusters):** The graph of remaining light vertices has maximum degree 11, compute a cover of it with stars of size at least 2, e.g., by computing a maximal independent set of the square of the graph. Then, the leader of the star colors each star as balanced as possible favoring blue.

**Observation D.2.** In any distance-$k$ connected component of clusters the fraction of vertices that is colored blue lies within the interval $[1/2, 3/4]$.

**Proof.** Consider a distance-$k$ connected component with $p$ clusters. The largest relative imbalance in Step 3 occurs when a heavy cluster has 13 children and 6 children are colored red and 7 children are colored blue. This relative imbalance is dominated by the imbalance in step 5 where the largest relative imbalance between red and blue clusters occurs if each star consists of three clusters. In this case a 2/3 fraction is colored bluer and a 1/3 fraction is colored red. As there are $p$ selected (outgoing) paths in total and each heavy vertex has at least 12 incoming paths there are at most $|p|/12$ heavy clusters. Thus, in step 4, at most $p/12$ clusters are colored blue. In total in step 3 at most a 7/13 fraction of the clusters is colored blue, in step 5 at most a 2/3 fraction of clusters is colored blue and in step 4 a $p/12$ fraction of all clusters in the component is colored blue. Thus we can upper bound the number of blue clusters by $\max\{7/13, 2/3\} \cdot p + p/12 = (3/4) \cdot p$. As each step colors at least as many clusters in the component blue as red the claim follows. □

**Implementation and Communication:** The implementation of each step crucially relies on the communication structure between the clusters that we build with the next lemma.

**Lemma D.3 (distance-$k$ cluster connecting structure).** Let $k \geq 1$. Given a cluster collection with $b$-bit cluster identifiers in a communication network with bandwidth $B \geq b$ there is an $O(k + \beta + \kappa/B)$ round algorithm that builds BFS trees in $G$ with depth at most $k$ such that

1. the root and all leaves of each BFS tree are nodes of a cluster,
2. each non distance-$k$ isolated cluster $C$ has exactly one vertex $w$ that is a leaf of a BFS tree $T_v$ with root $v$ where $v \in C' \neq C$,
3. each edge is contained in at most four distinct BFS trees.
We emphasize that all token forwardings happen in the communication network $G$ which implies that whole execution of the algorithm, $k$ then, tokens are forwarded in $G$. This ends the description of the algorithm.

Proof of Properties: We begin with Property 1. By definition, the root $v$ of BFS tree $T_v$ is part of a cluster. Also a node $u$ can only be a leaf of a BFS tree $T_v$ if it initiated a convergecast, which implies that $u$ is part of a cluster.

For Property 2, first observe that a cluster leader chooses at most one node of its cluster to become a BFS leaf. Thus, we only need to show that if a cluster $C$ has another cluster in distance at most $k$, at least one vertex of $C$ receives at least one token. We show this by induction over the following invariant: For every non distance-$k$ isolated cluster $C$ after iteration $i$ there is a node in distance at most $k - i$ from $C$ that has seen a token from a cluster $C' \neq C$.

Induction start: As $C$ is non distance-$k$ isolated there is some cluster $C'$ in distance at most $k$. As each vertex of $C'$ creates a token the invariant holds before the first iteration.

Induction step: Apply the induction hypothesis for iteration $i$ and among the nodes satisfying it let $v_i$ be the node that is closest to $C$ at the end of iteration $i$, that is, the distance of $v_i$ to $C$ is $\alpha \leq k - i$ and let $t$ be the token with $t_{ID} \neq ID_C$ that $v_i$ has seen. If $\alpha < k - i$ we have shown the induction step. So, consider $\alpha = k - i$ and let $v_{i+1}$ be a neighbor of $v_i$ that lies on a shortest path from $v_i$ to $C$. By the choice of $v_i$, node $v_{i+1}$ has not seen a token from a cluster $C' \neq C$. In particular, $v_i$ has forwarded at most one token to $v_{i+1}$ so far (possibly a token with $ID_C$). Thus $t_{ID}$ is not equal to the cluster ID of $v_i$’s cluster, $t$ does not get dropped by $v_i$, and $v_i$ forwards $t$ to $v_{i+1}$ in iteration $i + 1$, that is, the hypothesis holds for $i + 1$.

To show Property 3, notice that by definition, at most 2 tokens are forwarded over any edge by both endpoints. Hence, at most 4 different clusters can choose this edge to the BFS with the convergecast.

We next explain how each step can be implemented, given Lemma D.3. An efficient implementation of several of these steps uses the third property in Lemma D.3.

- **Step 0/Step 1:** We first apply Lemma D.3. Let $C$ be a cluster and $T_v$ be the BFS tree rooted at $v$ such that $w \in C$ is a leaf of $T_v$. Then, cluster $C$ selects the cluster of $v$ and the path between $w$ and $v$ in the BFS tree $T_v$. As there is exactly one such leaf for each non distance-$k$ isolated cluster we obtain the desired path selection. Note that the constant overlap allows us to perform aggregation on the BFS trees efficiently.

- **Step 2:** Each cluster $C$ needs to count the number of leaves of BFS trees $T_v$ for which the root $v$ is in $C$. As the BFS trees have constant overlap, $O(k)$ rounds are sufficient to for
each such root to learn the number of leaves in the BFS tree rooted at it. Then, within each cluster we only need to determine whether the total number of leaves in all BFS trees \( \{T_v \mid v \in C\} \) is at least 12, which can be implemented via an constant bit aggregation within the vertices of the cluster in \( O(k + \beta + \kappa) \) rounds (Corollary A.1).

- **Step 3:** See Claim D.4.
- **Step 4:** This step requires no communication, as after step 2 each cluster knows whether it is heavy or light.
- **Step 5:** See Claim D.5.

**Claim D.4.** A heavy cluster can color the clusters that selected it with a \( \pm 1 \) imbalance in \( O(k + \beta \cdot \kappa) \) rounds.

**Proof.** First the leader \( \ell_C \) of each heavy cluster \( C \) uses \( O(k + \beta \cdot \kappa) \) to count the number of clusters that selected it. Further, each node of the extended Steiner tree of cluster \( C \) receives for each incoming edge of the extended Steiner tree how many clusters (that selected \( C \)) can be reached through this edge. This is enough to inform each such cluster about its color such that the total imbalance is \( \pm 1 \).

**Claim D.5.** Step 5 can be implemented in \( O((k + \beta) \log^* b) \) rounds in the original communication network.

**Proof.** The dominating step is to color a maximal independent set in the square of the (cluster) graph \( H \) induced by light clusters. Due to the definition of light clusters the graph \( H \) is of constant degree. In particular, each BFS tree \( T_v \) that is used to connect two clusters in graph \( H \) has a constant number of leaves. Thus, via Corollary A.1 (part 2) one round of communication in graph \( H \) can be simulated in \( (\beta + \kappa) \) rounds in the communication network \( G \); in one round only a constant number of connecting vertices of a cluster need to send a message to the root. A maximal independent of \( H^2 \) set can be computed in \( O(\log^* b) \) rounds on \( H^2 \) which results in \( O((k + \beta) \log^* b) \) rounds in the original communication network.

**Proof of Lemma D.1.** Due to the explanation just before this proof all steps can be implemented in \( O(k + (\beta + \kappa) \log^* b + \beta \cdot \kappa) \) rounds. The balanced guarantee of the computed coloring follows with Observation D.2.

## E Simple Distance-\( k \) Color Class Carving

In this section we prove the following ball carving result.

**Lemma 3.3** (ball carving, \( k \geq 1 \)). Let \( k \geq 1 \) be an integer and \( x \geq 1 \) (both potentially functions of \( n \)). Then there is a deterministic distributed CONGEST (bandwidth \( b \) and \( b \)-bit identifiers) algorithm that, given a graph \( G = (V,E) \) with at most \( n \) nodes and a subset \( S \subseteq V \), computes a cluster collection \( C_1, \ldots, C_p \subseteq S \) with

- \( |C_1 \cup \ldots \cup C_p| \geq (1 - 1/x) \cdot |S| \),
- Steiner radius \( \beta = O(k \cdot x \cdot \log^3 n) \) and congestion \( \kappa = O(\log n \cdot \min\{k, x \cdot \log^2 n\}) \),
- the pairwise distance in \( G \) between \( C_i \) and \( C_j \) for \( 1 \leq i \neq j \leq p \) is strictly more than \( k \).

The runtime is \( O(k \cdot x \cdot \log^4 n \cdot \log^* b) + O(k \cdot x^2 \cdot \log^6 n \cdot \min\{k, x \cdot \log^2 n\}) \) rounds.
Fix parameters \( k \geq 1 \) and \( x > 0 \). Define the following parameters for the algorithm:

\[
\begin{align*}
\text{Phases} &= \log_{4/3} n + 1, \quad (2) \\
\text{ProposalParameter} &= x \cdot \text{Phases} = O(x \cdot \log n), \quad (3) \\
\text{Steps} &= (\text{ProposalParameter} + 1) \cdot \log n = O(x \cdot \log^2 n), \quad (4) \\
\beta &= k \cdot \text{Phases} \cdot \text{Steps} = O(k \cdot x \cdot \log^3 n), \quad (5) \\
\kappa &= 2 \cdot \text{Phases} \cdot \min\{k, \text{Steps}\} = O(\log n \cdot \min\{k, x \cdot \log^2 n\}). \quad (6)
\end{align*}
\]

### E.1 Simple Algorithm: Distance-\( k \) Color Class Carving

**Algorithm (for one color class, Lemma 3.3):** The algorithm has \( \text{Phases} \) phases and each phase consists of \( \text{Steps} \) steps.

**Initialization:** Each vertex of \( S \) forms its own cluster with a unique \( b \)-bit cluster identifier.

- **Phase (for reducing the number of clusters in each distance-\( k \) connected component):**
  - **Coloring clusters:** Use Lemma D.1 to color the clusters red and blue in a balanced way.
  - Repeat \( \text{Steps} \) times:
    - **One step:** In the beginning of each step, all nodes in red clusters propose to non-stalling blue clusters within distance at most \( k \) and either get accepted or die. We next explain how to determine which red node proposes to which blue cluster.
    - **Distance-\( k \) proposal implementation:** We have \( k \) iterations where blue nodes in non-stalling clusters send tokens to distance \( k \). Each blue node in a non-stalling cluster creates one token which is forwarded in the original graph \( G \) (including dead nodes and nodes in \( V \setminus S \)) for at most \( k \) hops. A vertex only forwards the first token that it receives, breaking ties arbitrarily. In particular, blue vertices do not forward any tokens. This builds BFS trees of depth at most \( k \) rooted at blue nodes. Each red node that is contained in a BFS tree commits to propose to the blue cluster of the root of the tree. Red nodes that are not in a BFS tree do not propose further.
    - **Accepting/Rejecting Proposals:** Let \( P_\mathcal{C} \) be the (red) vertices that propose to blue cluster \( \mathcal{C} \). Cluster \( \mathcal{C} \) accepts all proposals if its cluster would grow by a \((1 + 1/\text{ProposalParameter})\) factor by adding the proposals, that is, if \(|\mathcal{C} \cup P_\mathcal{C}| \geq (1 + 1/\text{ProposalParameter}) \cdot |\mathcal{C}|\), otherwise, including the case \(|P_\mathcal{C}| = 0\), all proposals in \( P_\mathcal{C} \) are rejected and the vertices in \( P_\mathcal{C} \) die permanently. They will not be part of any cluster during the algorithm for Lemma 3.3 ever again. However, dead nodes still participate in forwarding information and tokens, in particular, they participate in the token forwarding in the proposal phase. If a blue cluster rejects proposals it becomes stalling until the end of the phase. If a cluster accepts a set of proposals \( P_\mathcal{C} \) these vertices are added to the cluster and the Steiner trees are updated as described in the next paragraph.
    - **Steiner tree building:** For accepted proposals Steiner trees are extending along the BFS trees that are created in the proposal phase.
    - **End of Phase:** Stalling blue clusters forget their stalling status, every cluster forgets their color, dead vertices remain dead.

**Clarifications and Observations:** In a phase, there can also be red nodes that for some steps do not propose but then start proposing. In the distance-\( k \) proposal a red node cannot get a token from its own cluster as its own cluster is red, i.e., no token originates from it. In one
step, red nodes in the same cluster might propose to different blue clusters. Notice that in the distance-
\(k\) proposal implementation red nodes are fine with proposing to any blue cluster. Thus, they only need to receive one arbitrary token from an arbitrary blue cluster in distance-
\(k\), in particular, they do not care which token gets forwarded. The identifier is only used to identify clusters, BFS trees, Steiner trees and in the computation of the balanced coloring.

E.2 Analysis: Correctness

**Lemma E.1** (few vertices die). The number of vertices that die during the whole execution of the algorithm is upper bounded by \(|S|/x\).

**Proof.** Let \(S_i\) be the vertices that are alive at the beginning of phase \(i = 0, \ldots, \text{Phases}\). We have \(S_0 = |S|\). In each phase, each killed vertex can be uniquely charged to one stalling cluster such that each stalling cluster is only charged for \(1/\text{ProposalParameter}\) fraction of its vertices. Thus, for \(i \in [\text{phases} - 1]\) we obtain

\[
|S_{i+1}| \geq |S_i| \left(1 - \frac{1}{\text{ProposalParameter}}\right).
\] (7)

We deduce

\[
S_{\text{Phases}} \geq |S| \left(1 - \frac{1}{\text{ProposalParameter}}\right)^{\text{Phases}} \geq |S| \left(1 - \frac{\text{Phases}}{\text{ProposalParameter}}\right) \geq |S| \left(1 - \frac{\text{Phases}}{\text{ProposalParameter}}\right)
\] (8)

Thus there are at most \(\text{Phases} \cdot |S|/\text{ProposalParameter} = |S|/x\) dead vertices.

**Lemma E.2.** Every blue cluster is stalling at the end of a phase.

**Proof.** In each step, in which a cluster does not become stalling, it grows by a factor of \((1 + \frac{1}{\text{ProposalParameter}})^{\text{Steps}}\). If it does not become stalling in a phase, then after \(\text{Steps} = (\text{ProposalParameter} + 1) \log n\) iterations we obtain that it has at least

\[
(1 + \frac{1}{\text{ProposalParameter}})^{\text{Steps}} > e^{\text{Steps} \frac{1}{\text{ProposalParameter}}} = e^{\text{Steps} \frac{1}{\text{ProposalParameter} + 1}} = n
\] (9)

vertices, where we used \(1 + y \geq e^{y/(1+y)}\) for \(y > -1\). This is a contradiction.

We now show that clusters are separated at the end of the algorithm.

**Lemma E.3** (cluster separation 1). When a blue cluster becomes stalling in some phase \(i\) there is no red vertex in distance at most \(k\) from it, and this will be the case until the end of phase \(i\).

**Proof.** Consider a cluster \(C\) that is stalling in some phase \(i\). Consider the step \(j\) in which \(C\) became stalling. After step \(j\) there is no red vertex in distance-
\(k\) of \(C\), as in step \(j\) every vertex in this neighborhood either proposed to \(C\) and got killed, or proposed to some other cluster such that it either also got killed or joined the cluster, i.e., became blue. In particular, there is no red node in distance-
\(k\) of \(C\) that does not propose in step \(j\).

As nodes never change their color to red during a phase no node in the distance-
\(k\) neighborhood of \(C\) is red throughout the phase.

The **cluster graph** \(H\) during some point of the execution of the algorithm has a vertex for each cluster \(C\) and an edge between \(C\) and \(C'\) if there are vertices \(u \in C\) and \(v \in C'\) with \(\text{dist}_{G}(u, v) \leq k\). The connected components of \(H\) are called the distance-
\(k\) connected components of the clustering.

Two clusters \(C\) and \(C'\) are called distance-
\(k\) separated if for all nodes \(u \in C\) and \(v \in C'\) and all paths in \(G\) between \(u\) and \(v\) contain at least \(k\) consecutive nodes that are dead or in \(V \setminus S\).

**Lemma E.4.** At the end of each phase each distance-
\(k\) connected component of blue clusters is distance-
\(k\) separated from each distance-
\(k\) connected component of red clusters.
Proof. We want to show that since each blue cluster becomes stalling, it must be the case that a connected component of stalling clusters is distance-$k$ separated from red clusters. By Lemma E.2, we know that eventually, each blue cluster becomes stalling. Consider now a connected component $D$ of blue clusters and suppose for a contradiction that there is a red cluster that is in distance $k$ from $D$. Then, there must be a red node that is in distance $k$ from some cluster $C \in D$, which is a contradiction by Lemma E.3. Therefore, it must be the case that each distance-$k$ connected component of blue clusters is distance-$k$ separated from red clusters. \qed

Note that Lemmas E.2 and E.3 also imply that each red vertex does not have a blue vertex in distance-$k$ at the end of a phase.

The following lemma does not use any red or blue coloring.

Lemma E.5. If at some point two clusters $C$ and $C'$ are not in the same distance-$k$ connected component of clusters, they will not be in the same component for the rest of the phase.

Proof. Every path in $G$ connecting the clusters contains $k$ consecutive vertices that are dead or in $V \setminus S$. So, no alive cluster can ever contain any of these vertices ever again. Thus, they cannot become connected. \qed

Lemma E.6 (Connected components decrease). Ignoring all colors, at the end of phase $i$, for each cluster $C$ the number of clusters in the distance-$k$ connected component of $C$ is at most $\max\{1, (3/4)^i |S|\}$. 

Proof. We prove the remaining claim by induction over the phases. For $i = 0$, the claim holds as there are initially only $S$ clusters. Fix some phase $i \geq 0$ and assume that the claim holds at the end of phase $i$. Due to Lemma E.2 every blue cluster is stalling at the end of phase $i+1$. Thus, due to Lemma E.3, there is no red vertex in the distance-$k$ neighborhood of any blue cluster, which implies that vice versa also for every red cluster there is no blue vertex in the distance-$k$ neighborhood.

Now, let $C$ be a connected component of clusters in the cluster graph $H$ at the end of phase $i$ (and also at the beginning of phase $i+1$). By the induction hypothesis we have $|C| \leq \max\{1, (3/4)^i |S|\}$. If $|C| = 1$ the claim also holds after phase $i+1$ as due to Lemma E.5 connected components can never increase. If $|C| > 1$, then at the beginning of phase $i+1$ the clusters of $C$ are colored red or blue in a balanced way, that is, at most $3/4$ of the clusters in the component have the same color. Let $B$ be the blue clusters of $C$ and $R$ the red clusters of $C$. Due to Lemma E.4 the clusters in $C$ and $R$ are distance-$k$ separated after the phase. The bound on the size of the connected component at the end of phase $i+1$ follows due to the size of $B$ and $R$ and because Lemma E.5 implies that no cluster can join the component in phase $i+1$. \qed

E.3 Analysis: Steiner Trees

The correct building of Steiner trees for computing the network decomposition of $G^k$ has been analyzed in [RG20] in the case where nodes in the $i$-th phase use the $i$-th bit of their ID to determine whether they are red or blue. As most observations are not affected at all by the red/blue coloring we only re-visit the observations that are crucial to bound the Steiner tree radius, the congestion and thus also have an influence on the runtime.

Observation E.7 (Steiner tree growth). In each step, the radius of the Steiner tree of each blue cluster grows by at most $k$, while the radius of the Steiner tree of each red cluster does not grow.

Observation E.8. In each step an edge is added to at most $2$ Steiner trees and in each phase an edge is added to at most $\kappa = 2 \cdot \min\{k, \text{Steps}\}$ Steiner trees.
Proof. In a step an edge is only added to a Steiner tree if a token traveled through the edge during the distance-\(k\) proposal phase. As vertices only forward the first token that they receive there can at most be one token per direction that travel through the edge, that is, in each step an edge joins at most 2 Steiner trees.

During a phase, an edge can never be added to a Steiner tree anymore once both of its endpoints become blue. If both endpoints of an edge \(e = \{u, v\}\) are red and \(e\) is added to a spanning tree both endpoints are blue afterwards and the edge will not be added to another spanning tree in the phase. Thus, we concentrate on edges \(e = \{u, v\}\) in which at least one vertex is dead or in \(V \setminus S\). Let \(u\) be this vertex. The edge \(e\) has been added to a spanning tree because a token traveled through. If \(e\) is added to the spanning tree again blue has to be closer to \(u\) than in the last iteration as otherwise it does not reach any new red vertex. \(\Box\)

Recall, that the congestion of Steiner trees is the maximum number of trees that contain the same edge.

**Observation E.9.** At all times the radius of the Steiner trees is at most \(\beta\) and each edge of \(G\) is in at most \(\kappa = \text{Phases} \cdot \mathcal{O}(\min\{k, \text{Steps}\})\) Steiner trees.

**Proof.** Due to Observation E.7 the radius of each Steiner tree grows at most by \(\kappa\) per step. Thus, the radius throughout all phases is upper bounded by \(\beta = \mathcal{O}(k \cdot \text{Phases} \cdot \text{Steps}) = \mathcal{O}(k \cdot x \cdot \log^2 n)\).

The congestion bound follows with Observation E.8. \(\Box\)

### E.4 Analysis: Implementation and Runtime

The presented algorithm is a \(\log n\) factor faster than the variant in [RG20] (that uses unique IDs of size \(\text{poly}\,n\)) because we incorporated the faster aggregation methods from [GGR21] for parallel trees when accepting/rejecting proposals.

**Observation E.10 (Runtime per step).** One step can be implemented in \(\mathcal{O}(k + (\beta + \kappa) \cdot \max\{1, \kappa/b\})\).

**Proof.** \(\mathcal{O}(k)\) to send the proposals. For aggregating the number of proposals we require \(Z = (\beta + \kappa) \cdot \max\{1, \kappa/b\}\) via the simultaneous aggregation with overlapping Steiner trees from Corollary A.1. Informing nodes whether their proposal is accepted and extending the Steiner trees can also be done in \(\mathcal{O}(k + Z)\) rounds. \(\Box\)

**Lemma E.11 (Runtime per phase).** The runtime of one phase is \(\mathcal{O}(\beta \cdot \log^* b + \text{Steps} \cdot (\beta + \kappa) \cdot \max\{1, \kappa/b\})\).

**Proof.** The red/blue coloring of clusters takes \(\mathcal{O}(k + (\kappa + \beta) \cdot \log^* b + \kappa \cdot \beta)\) rounds via Lemma D.1. Due to Observation E.10 each step requires \(\mathcal{O}(k + \max\{1, \kappa/b\} \cdot (\beta + \kappa))\) rounds, that is, a phase can be implemented in (using \(\beta \geq k\) and \(\text{Steps} \geq \beta\))

\[
\mathcal{O}(k + (\kappa + \beta) \cdot \log^* b + \beta \cdot \kappa) + \text{Steps} \cdot \mathcal{O}(k + (\beta + \kappa) \cdot \max\{1, \kappa/b\})
\]

\[
= \mathcal{O}(\beta \cdot \log^* b) + k \cdot \text{Steps} + \text{Steps} \cdot (\beta + \kappa) \cdot \max\{1, \kappa/b\}
\]

\[
= \mathcal{O}((\kappa + \beta) \cdot \log^* b + \text{Steps} \cdot (\beta + \kappa) \cdot \max\{1, \kappa/b\})
\] \(\Box\)

**Proof of Lemma 3.3.** Due to to Lemma E.1 at most \(|S|/x\) vertices are not clustered. The distance-\(k\) cluster separation follows with Lemma E.6. The bound on the Steiner tree radius and congestion is stated in Observation E.9. The runtime follows by multiplying the runtime per phase from Lemma E.11 by the number of phases. Using \(b = \log n\). we get \(\max\{1, \kappa/b\} = \max\{1, \log n \cdot \min\{k, x \cdot \log^2 n\}/b\} = \min\{k, x \cdot \log^2 n\}\). We obtain

\[
\mathcal{O}(\text{Phases} \cdot \beta \cdot \log^* b + \text{Phases} \cdot \text{Steps} \cdot (\beta + \kappa) \cdot \min\{k, x \cdot \log^2 n\})
\]

\[
= \mathcal{O}(\log^* n \cdot \log^* b) + \mathcal{O}(k \cdot x^2 \cdot \log^6 n \cdot \min\{k, x \cdot \log^2 n\})
\] \(\Box\)