

# Overcoming Congestion in Distributed Coloring\*

Magnús M. Halldórsson<sup>1</sup>, Alexandre Nolin<sup>1</sup>, and Tigran Tonoyan<sup>2</sup>

<sup>1</sup>ICE-TCS, Department of Computer Science, Reykjavik University, Iceland

<sup>2</sup>Krisp Technologies Inc., Armenia

[mmh@ru.is](mailto:mmh@ru.is); [alexandren@ru.is](mailto:alexandren@ru.is); [ttonoyan@gmail.com](mailto:ttonoyan@gmail.com)

## Abstract

We present a new technique to efficiently sample and communicate a large number of elements from a distributed sampling space. When used in the context of a recent LOCAL algorithm for (degree +1)-list-coloring (DILC), this allows us to solve DILC in  $O(\log^5 \log n)$  CONGEST rounds, and in only  $O(\log^* n)$  rounds when the graph has minimum degree  $\Omega(\log^7 n)$ , w.h.p.

The technique also has immediate applications in testing some graph properties locally, and for estimating the sparsity/density of local subgraphs in  $O(1)$  CONGEST rounds, w.h.p.

## 1 Introduction and Related Work

We explore ways to reduce bandwidth, particularly for the fundamental vertex coloring problem. Bandwidth is the key difference between the otherwise similar models of locality that are LOCAL and CONGEST: while nodes can send messages of arbitrary size in LOCAL (and thus, the round complexity of a problem only depends on how far in the graph its nodes need to see), in CONGEST the nodes are restricted to messages of size  $O(\log n)$ , where  $n$  is the number of nodes in the graph. While some classical algorithms designed without bandwidth in mind nonetheless immediately work in both models, this is not true of many recent algorithms for distributed coloring.

In this paper, we introduce a technique to implement some general sampling and estimation tasks in CONGEST. At its heart are families of hash functions we call *representative* for having certain statistical properties. The technique improves on the bandwidth cost of a naïve approach by optimizing the use of randomness, taking ideas from the pseudorandomness literature. In the context of distributed coloring, the technique allows us to adapt crucial parts of recent randomized LOCAL algorithms to CONGEST. Through this and some additional ideas, we construct randomized CONGEST algorithms for some of the most common distributed coloring problems that almost match the complexity of the current best LOCAL algorithms for the same problems. Aside from our results for vertex coloring, our technique has direct applications of independent interest related to testing for the presence of some graph structure, and constructing structural decompositions known as *almost-clique decompositions* in CONGEST.

The key idea behind our technique is a combination of using hashing and optimizing the amount of randomness we use when selecting a hash function. In bandwidth-constrained contexts, hashing is a natural hammer for a number of nails. It allows to exchange information about sparse data living in a very large space by exchanging much smaller images. Random hash functions are also useful tools to sample elements, by taking as sample the elements that

---

\*This paper incorporates results from the technical report [HNT21] on adapting LOCAL algorithms to CONGEST. This excludes the other results in [HNT21], which were refactored in [HKNT21].

hash to certain values. However, the full description of an arbitrary function requires a lot of bits, necessarily more than the data we want to hash. Only by making the hash functions we use less random can they become useful tools in CONGEST. Overall, using hashing in CONGEST is a balancing act between using enough randomness for the random hash functions to have the statistical properties we need, and using randomness efficiently and sparingly so as to make the hash functions communicable within our bandwidth constraints.

When analyzing recent LOCAL algorithms for vertex coloring focusing on the bandwidth use of each of their steps, two steps in particular stand out. One is related to the computation of so-called almost-clique decompositions. In such decompositions, nodes decide to join their neighbors in so-called *almost-cliques* depending on how similar their neighborhood is to that of their neighbors. Since in a graph of maximum degree  $\Delta$ , the description of a node's neighborhood requires up to  $\Delta \log n$  bits, a naïve approach would require  $\Delta$  CONGEST rounds. The other seemingly high-bandwidth step has to do with a procedure named MULTITRIAL, which has nodes send multiple colors (up to  $\Theta(\log n)$ ) to their neighbors. With colors living in a set  $\mathcal{C}$ , describing an arbitrary set of this many colors takes  $\log |\mathcal{C}| \log n$  bits, i.e.,  $\log |\mathcal{C}|$  rounds.

Both steps have the shared quality of essentially reducing to some sampling task. In the case of the almost-clique decomposition, two nodes can get a sense of how similar their neighborhoods are by sending each other random elements from their respective neighborhoods. In the case of MULTITRIAL, the goal of each node is to sample random colors jointly with its neighbors such that each color has a good chance of being both a valid color for it and not simultaneously sampled by one of its neighbors. This and the fact that some sort of random sampling also appears in a variety of other randomized algorithms motivates looking for an efficient implementation in CONGEST.

The cost of communicating a random sample is intrinsically linked to how random it is: for example, it requires 10 times less communication to communicate 10 random elements if they are guaranteed to be all equal instead of being all independent. Thus, it might be tempting to use less randomness to save on communication in a distributed setting. But modifying a working algorithm by making it use less randomness runs the risk of skewing the probabilities to a point where the algorithm no longer works. The field of pseudorandomness has come up with techniques to save on randomness while keeping an algorithm functional, for example in the context of repeating a randomized algorithm to boost the probability of computing the correct answer. Ideas related to pseudorandomness have also previously made their way in fields focusing on the communication cost of algorithms, notably in the form of a seminal result in 2-party communication complexity known as Newman's theorem [New91]. In our setting where we use hash functions as tools for sampling elements, techniques from pseudorandomness allow us to construct families of hash functions that strike a balance between being random enough to produce useful samples, but small enough for our communication constraints.

One of the most general versions of the vertex coloring problem is the *(degree+1)-list-coloring* problem (D1LC). In this version, each vertex  $v$  is given a list of  $d_v + 1$  colors at the beginning of the algorithm, where  $d_v$  is the degree of  $v$ . Each node must then color itself with a color from its list that is distinct from the colors adopted by its neighbors. Compared to versions of the problem where nodes all receive a list of  $\Delta + 1$  colors, where  $\Delta$  is the maximum degree in the graph, D1LC requires to find procedures that can be parameterized to work with nodes that differ greatly in the number of colors they can choose from. Our techniques using pseudorandom hash functions allows such parameterization, by using different sets of hash functions depending on the size of the space we are trying to sample from. Once all steps reducible to a sampling task are implemented using our pseudorandom family of hash functions, few steps of recent randomized LOCAL algorithms remain to be adapted to work in CONGEST. We give a complete CONGEST adaptation of a recent LOCAL algorithm for D1LC. On graphs of large minimum degree, the resulting algorithm matches the ultrafast complexity of the LOCAL algorithm it draws from. On graphs containing nodes of lower degree, our algorithm has a higher complexity

but remains of order polynomial in  $\log \log n$ .

As our main tool for tackling D1LC in CONGEST is a general technique for different types of sampling tasks, and sampling is extensively used in randomized algorithms, it might find uses in a variety of other problems. In fact, we give two simple direct applications in the context of subgraph detection. Our  $O(1)$  CONGEST algorithm for computing an almost-clique decomposition might also prove useful in problems other than vertex coloring.

As our main tool is only shown to exist through an existential proof, our algorithms are not uniform in their default form, in the sense that they require that the nodes either perform massive computation exploring the set of all hash functions, or are given some common advice bits that only depends on the size of the input. We provide explicit, uniform implementations of the subroutines crucial to our vertex coloring results. These implementations are ad hoc, but can be taken as indications that explicit constructions of our general tool of representative hash functions might be possible.

## 1.1 Related Work

The literature of distributed graph coloring is vast and we only mention those with direct implications for our work. In the paper introducing the LOCAL model [Lin92], Linial showed that  $(\Delta + 1)$ -coloring constant-degree graphs requires  $\Omega(\log^* n)$  rounds, and gave a matching deterministic algorithm. This remains the only lower bound known when this many colors are available. The best upper bounds known on deterministic algorithms in general graphs are  $O(\log^2 \Delta \log n)$  [GK21] and  $\tilde{O}(\sqrt{\Delta}) + O(\log^* n)$  [Bar16].

The MultiTrial technique was introduced in [SW10] and further developed in [EPS15] and [CLP20]. Slack generation via sparsity was introduced in [EPS15] (though it traces back to [Ree98] in graph theory), where it was used to give ultrafast algorithms for edge coloring graphs of high degree. The shattering framework for distributed algorithms was proposed in [BEPS16]. The almost-clique decomposition was introduced to distributed computing in [HSS18] (while a similar one was known in graph theory [MR98]), leading to an  $O(\sqrt{\log n})$ -round algorithm for  $(\Delta + 1)$ -coloring. An implementation of ACD with random sampling was proposed in [ACK19]. The best randomized  $(\Delta + 1)$ -coloring algorithm known, given in [CLP20], has complexity  $O(\log^3 \log n)$ .

For the  $(\deg + 1)$ -list coloring (D1LC) problem, the best bound until recently was  $O(\log n)$  of the early algorithms of [Joh99, ABI86, Lub86] and the more refined bound of  $O(\log \Delta + \text{poly}(\log \log n))$  [BEPS16]. This year, the bound was improved to  $O(\log^3 \log n)$ , which drops to  $O(\log^* n)$  when all nodes have degree  $\Omega(\log^7 n)$  [HKNT21].

All the works above (except the recent [GK21]) were stated for the LOCAL model, while the  $O(\log n)$ -round algorithms also run in CONGEST. The complexity of  $(\Delta + 1)$ -list-coloring in CONGEST was recently improved to  $O(\log^5 \log n)$  rounds in [HKMT21].

For the Congested Clique,  $O(1)$ -round algorithms are known for  $(\Delta + 1)$ -coloring, both randomized [CFG<sup>+</sup>19] and deterministic [CDP20]. An earlier  $O(1)$ -round algorithm was given for semi-linear MPC [ACK19].

$(1 + \varepsilon)\Delta$  coloring,  $(2\Delta - 1)$ -edge coloring, and  $(1 + \varepsilon)\Delta^2$  distance-2 coloring were recently shown to respectively admit an  $O(\log^3 \log n)$ ,  $O(\log^4 \log n)$ , and  $O(\log^4 \log n)$  rounds algorithm in CONGEST [HN21]. When  $\Delta \geq \log^{1+1/\log^* n} n$ , the complexities drop to  $O(\log^* n)$  and the edge-coloring algorithm can restrict itself to  $(1 + \varepsilon)\Delta$  colors. The efficiency of these algorithms at high degrees comes from a pseudorandom construct called representative set families. Such families are built so that a random member is likely to intersect any large fraction of the space. This allows nodes to efficiently sample up to  $\Theta(\log n)$  colors in  $O(1)$  rounds when given a constant fraction of their degree as extra colors, making  $O(\log^* n)$  algorithms possible. The technique has an important limit, however: it does not work when the colors live in a color space much larger than the nodes' degrees. This prevents the technique from being useful in list-coloring settings, and when nodes have an amount of slack that is comparable to their

degree late in the algorithm, but much smaller than their original degree. We build on the technique in this paper, and tackle scenarios in which representative sets could not be applied. Obtaining this more general result requires a significant leap, as exchanging information about and sampling from elements living in a very large universe efficiently requires a high level of succinctness in our communication.

Distributed property testing was introduced in [BPS11] and formalized in [CHFSV19]. The usual model is to distinguish graphs that satisfy a given property (e.g., triangle-freeness), from those for which an  $\varepsilon$ -fraction of the edges must be removed for the property to hold. Distributed testing algorithms were given for detecting triangles in [CHFSV19] and  $C_4$  in [FRST16], with the best known round complexity of  $O(1/\varepsilon)$  obtained for both problems (and other cycles) in [FO19].

## 2 Results

**Efficient sampling and estimation.** We give a communication-efficient procedure for two parties each possessing a set to estimate how similar their sets are, and sample an element in their intersection or their difference (Sec. 3.2). The technique also works with more parties, allowing, e.g., a party to sample an element in the difference between her set and the union of all her neighbors' sets. The technique is quite general, and might find applications in problems other than those studied in this paper.

**Coloring.** We bring to CONGEST recent LOCAL  $\text{poly}(\log \log n)$ -round randomized algorithms for coloring, at the cost of a moderate increase in complexity – our algorithm uses  $O(\log^5 \log n)$  CONGEST rounds in general, up from  $O(\log^3 \log n)$  LOCAL rounds. Our algorithm is an adaptation of a recent LOCAL algorithm for D1LC of [HKNT21].

**Theorem 1.** *The D1LC problem can be solved w.h.p. in  $O(\log^5 \log n)$  rounds in the CONGEST model with  $\log n$  bandwidth. When all nodes have degree at least  $\log^7 n$ , the algorithm uses only  $O(\log^* n)$  rounds.*

This coloring algorithm uses only polynomial local computation.

The round complexity reduces to  $O(\log^3 \log n)$  when the size of the color space  $\mathcal{C}$  (and therefore, the degrees) is of order  $\text{poly}(\log n)$ . This, in combination with the  $O(\log^* n)$  complexity on large degree nodes, immediately yields an  $O(\log^3 \log n)$  algorithm for (degree+1)-coloring (D1C). Note that this also improves on the state of the art for the  $(\Delta + 1)$ -coloring ( $\Delta$ 1C) problem in CONGEST.

**Corollary 1.** *D1C can be solved w.h.p. in  $O(\log^3 \log n)$  rounds in the CONGEST model.*

**Uniform implementations.** The implementation that follows from our main technique is non-uniform in the sense that it requires that the nodes perform very large computations locally, or that they have access to some advice only dependent on the size of the input (similar to how the complexity class  $\mathbf{P}/\text{poly}$  is enhanced compared to  $\mathbf{P}$ ). To reduce the total computational demand to polynomial, we provide alternative uniform implementations of our main procedures in Sec. 5.

**Other results.** On our way to proving our results for vertex coloring, we give an algorithm for computing a (degree+1) almost-clique decomposition (Sec. 4.2). Our general technique for sampling and estimation also has some immediate applications in testing for the presence certain graph structures, e.g., triangle-rich neighborhoods (Sec. 3.3 to 3.5).

### 3 Congestion-Reducing Techniques

#### 3.1 Representative Hash Functions

The crux of our results is a procedure for communicating parties to estimate the intersection or difference of sets they keep and/or sample elements in that intersection or difference. We do so through hashing, using a family of hash functions we call *representative* due to their statistical properties. For a given parameter  $b$ , the family is engineered to distort the probabilities of some events by at most  $\exp(-\Omega(b))$  compared to fully random hash functions, while being of small enough size  $\exp(O(b))$ . With  $b$  chosen within a constant multiplicative factor of the available bandwidth, i.e.,  $b \in \Theta(\log n)$ , this enables sending the index of a function in the family in a constant number of CONGEST rounds, while only introducing a manageable distortion compared to a fully random hash function.

**Intuition.** Suppose two parties have access to a shared source of randomness to pick a fully random hash function  $h$  from a universe  $\mathcal{U}$  to  $[\lambda]$ , without communicating. Having access to such a hash function offers several possibilities. In particular, the parties may now communicate about an element  $x \in \mathcal{U}$  through its image  $h(x)$ . Suppose the communicating parties each possess a subset of  $\mathcal{U}$ , respectively  $X$  and  $Y$ . To pick a random element in  $X$ , the node possessing  $X$  can rely on the randomness of the hash function for the selection process: set a threshold  $\sigma$ , consider all the elements of  $X$  that hash to a value  $\leq \sigma$ , and pick one of those low-hashing elements at random. When sampling elements jointly, the parties can ensure that they choose two distinct elements  $(x, y) \in X \times Y, x \neq y$  by ensuring that  $h(x) \neq h(y)$ . To sample an element in  $X \setminus Y$ , it suffices to pick an element in  $X \setminus h^{-1}(h(Y))$ , i.e., an element of  $X$  that hashes to a value that no element of  $Y$  hashes to. To pick an element in the intersection  $X \cap Y$ , the parties may look at the intersection  $h(X) \cap h(Y)$  and take the preimages of a hash in the intersection. When bandwidth is limited, the parties can adapt to this constraint by adjusting the threshold  $\sigma$ :  $\sigma$  bits suffice for each party to encode, for each value  $\leq \sigma$ , whether it has an element hashing to it.

The hash function introduces errors: the set  $h(X) \setminus h(Y)$  can be empty though  $X \setminus Y$  is not, due to collisions; the set  $h(X) \cap h(Y)$  might be non-empty even when  $X \cap Y$  is; and our elements of interest might hash to values  $> \sigma$ , resulting in the parties missing them. But with the right ratios between the sizes of the sets, the size of the output space of the hash function ( $\lambda$ ), and the size of the observation window ( $\sigma$ ), it can be argued that only some amount of errors occurs with the needed probability.

To apply these ideas in CONGEST, it only remains for the parties to be able to sample and communicate a random hash function, which is achieved by building a small set of hash functions with nearly the same statistical properties as the set of all hash functions from  $\mathcal{U}$  to  $[\lambda]$ .

**Notations.** For a set  $\mathcal{U}$  and a number  $\lambda \in \mathbb{N}$ , let  $[\lambda]^{\mathcal{U}}$  denote the set of all functions from  $\mathcal{U}$  to  $[\lambda] = \{1, \dots, \lambda\}$ . For a function  $h$ , sets  $A, B \subseteq \mathcal{U}$ , and number  $\sigma$ , we define:

- $A|_h^{\leq \sigma} = h^{-1}([\sigma]) \cap A$ ,
- $A \wedge_h^{\leq \sigma} B = \{\psi \in A : h(\psi) \in [\sigma] \cap h(B \setminus \{\psi\})\}$ ,
- $A \neg_h^{\leq \sigma} B = (A|_h^{\leq \sigma}) \setminus (A \wedge_h^{\leq \sigma} B)$ .

Intuitively,  $A|_h^{\leq \sigma}$  are the elements of  $A$  that hash to a value at most  $\sigma$  through  $h$ ,  $A \wedge_h^{\leq \sigma} B$  is the subset of  $A|_h^{\leq \sigma}$  that is in collision with some element of  $B$ , i.e., the elements  $x \in A|_h^{\leq \sigma}$  s.t. there exists a  $x' \in B \setminus \{x\}, h(x') = h(x)$ .  $A \neg_h^{\leq \sigma} B$  are the elements of  $A$  that hash to a value  $\leq \sigma$  through  $h$  that no distinct element in  $B$  hashes to (note that  $B$  may contain  $A$  or a

subset of  $A$ ). The definitions of the sets are most clear when  $B = A$ :  $A \wedge_h^{\leq \sigma} A$  are the elements of  $A$  that hash to at most  $\sigma$  through  $h$  and collide with another element of  $A$ ;  $A \neg_h^{\leq \sigma} A$  are the elements of  $A$  that hash to at most  $\sigma$  through  $h$  and do not collide with another element of  $A$ . Note that  $A|_h^{\leq \sigma}$ ,  $A \wedge_h^{\leq \sigma} B$ , and  $A \neg_h^{\leq \sigma} B$  are subsets of the domain of  $h$ , not its codomain.

If  $h$  were fully random, we would expect the size of  $A|_h^{\leq \sigma}$  to be within a constant factor of  $\sigma|A|/\lambda$  w.p.  $1 - \exp(-\Omega(\sigma|A|/\lambda))$ . Also, with a fully random  $h$  and  $\lambda$  sufficiently large w.r.t.  $A$  and  $B$ , we would expect at most  $|A||B|/\lambda$  elements of  $A$  to be in collision with an element of  $B$ . Our goal is to maintain a relaxed version of these probabilistic guarantees while restricting the space of random hash function we select from, so that communicating the index of a selected function is feasible in  $O(1)$  messages.

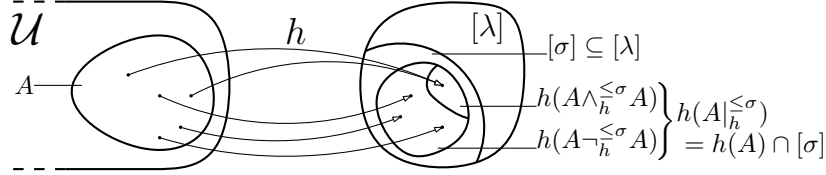


Figure 1: Example of our notation on a symmetric example ( $B = A$ ).  $A|_h^{\leq \sigma}$  is the part of  $A$  hashing to  $\sigma$  or less,  $A \wedge_h^{\leq \sigma} A$  are the elements of  $A|_h^{\leq \sigma}$  that collide through  $h$ , and  $A \neg_h^{\leq \sigma} A$  is the rest of  $A|_h^{\leq \sigma}$ .

For intuition and easier proofs later, a few elementary properties of set operators  $|_h^{\leq \sigma}$ ,  $\wedge_h^{\leq \sigma}$ , and  $\neg_h^{\leq \sigma}$  are given in [Proposition 1](#). When  $\sigma$  is clear from the context, we omit the superscript  $\leq \sigma$  and simply write  $A|_h$ ,  $A \wedge_h B$ , and  $A \neg_h B$ .

**Proposition 1.**

$$\forall A : \quad \left| h(A \wedge_h^{\leq \sigma} A) \right| \leq \left| A \wedge_h^{\leq \sigma} A \right| / 2, \quad (1)$$

$$\forall A, B \text{ s.t. } A \subseteq B : \quad \left| h(A \neg_h^{\leq \sigma} B) \right| = \left| A \neg_h^{\leq \sigma} B \right|, \quad (2)$$

$$\forall A, B, C \text{ s.t. } B \subseteq C : \quad (A \wedge_h^{\leq \sigma} B) \subseteq (A \wedge_h^{\leq \sigma} C) \text{ and thus } (A \neg_h^{\leq \sigma} C) \subseteq (A \neg_h^{\leq \sigma} B). \quad (3)$$

*Proof.* [Equation \(1\)](#) follows from the fact that for each element  $x \in A \wedge_h^{\leq \sigma} A$ , there exists an element  $x' \in A \setminus \{x\}$ ,  $h(x') = h(x)$ , which by definition of  $A \wedge_h^{\leq \sigma} A$  implies that  $x' \in A \wedge_h^{\leq \sigma} A$ . So every hash  $y \in h(A \wedge_h^{\leq \sigma} A)$  has at least two preimages, giving the result.

[Equation \(2\)](#) follows from every element of  $A \neg_h^{\leq \sigma} B$  having a unique hash value among elements of  $B$ , of which  $A \neg_h^{\leq \sigma} B$  is a subset.

[Equation \(3\)](#) describes the simple fact that the set of elements in collision with elements from a set  $B$  is smaller than (and included in) the set of elements in collision with elements from the set  $C$ , where  $C$  is a superset of  $B$ . Conversely, any element not in collision with all the elements of  $C$  is necessarily not colliding with any element of  $B$ .  $\square$

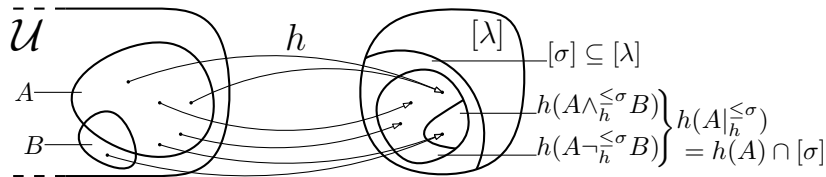


Figure 2: Example of our notation on an asymmetric example ( $B \neq A$ ).  $A \wedge_h^{\leq \sigma} B$  and  $A \neg_h^{\leq \sigma} B$  are the natural non-symmetric generalizations of  $A \wedge_h^{\leq \sigma} A$  and  $A \neg_h^{\leq \sigma} A$  where we focus on collisions between elements of  $A$  and  $B$  instead of within  $A$ .  $A$  and  $B$  can overlap arbitrarily.

We now show the existence of a small (compared to  $[\lambda]^{\mathcal{U}}$ ) family of hash function such that a random element from this family behaves similar to a fully random hash function w.r.t to the sets  $A \neg_h^{\leq \sigma} B$ ,  $A \wedge_h^{\leq \sigma} B$ , and  $A \neg_h^{\leq \sigma} B$ , for all  $A$  and  $B$  in a given size range. The proof is of the probabilistic method type, and bears resemblance to other arguments in the field of pseudorandomness or more generally aimed at reducing the amount of random bits used in a task, such as Newman's theorem [New91] and recent efforts to bring ultrafast distributed coloring algorithms in CONGEST [HN21]. This construction is the basis of our algorithms for efficiently estimating and sampling in CONGEST, ESTIMATESIMILARITY (Alg. 1) and MULTITRIAL (Alg. 4). Note that using simpler objects like  $k$ -wise independent hash functions would not suffice here. Indeed, such hash functions do not ensure the required statistical properties with sufficient probability when  $k$  is low, and increasing  $k$  to the number of elements we generally need to hash ( $\max(\Delta, \log n)$ ) to get the needed probability prohibitively increases the cost of describing a hash function in CONGEST.

**Lemma 1** (Representative hash functions). *Let  $\alpha, \beta, \nu \in (0, 1)$  and  $\lambda \in \mathbb{N}$  be s.t.  $\alpha \leq \beta$  and  $\lambda \geq \max(45\alpha^{-1}, 3\alpha^{-1}\beta^{-2}) \ln(12/\nu)$ , and let  $\mathcal{U}$  be a finite set. There exists a family of  $F = \Theta(\beta\lambda\nu^{-1} \log|\mathcal{U}|)$  hash functions  $\{h_i\}_{i \in [F]} \subseteq [\lambda]^{\mathcal{U}}$  and  $\sigma \leq \lambda$ ,  $\sigma \in \Theta(\beta^{-2}\alpha^{-1} \log(1/\nu))$ , such that for every  $A, B \subseteq \mathcal{U}$  with  $|A|, |B| \in [0, \beta\lambda]$ , at least  $(1 - \nu)F$  of the hash functions  $h$  satisfy*

$$\begin{aligned} \left| A \neg_h^{\leq \sigma} \right| &\in \frac{\sigma|A|}{\lambda} \cdot [1 - \beta, 1 + \beta] & \text{and} & \quad \left| A \wedge_h^{\leq \sigma} B \right| \leq \frac{2\sigma|A|}{\lambda} \beta & \text{when } |A| \geq \alpha\lambda, \\ \left| A \neg_h^{\leq \sigma} \right| &\leq \sigma\alpha \cdot (1 + \beta) & \text{and} & \quad \left| A \wedge_h^{\leq \sigma} B \right| \leq 2\sigma\alpha\beta & \text{when } |A| < \alpha\lambda. \end{aligned}$$

To prove Lemma 1, we first prove the following claim. We only consider sets  $A, B \subseteq \mathcal{U}$  satisfying  $|A|, |B| \in [0, \beta\lambda]$ . A hash function is  $(A, B)$ -good if it satisfies the requirement of the lemma for a given pair  $(A, B)$ . We bound the probability that a random function is  $(A, B)$ -good, for a fixed pair  $(A, B)$ .

**Claim 1.** *Let  $h \in [\lambda]^{\mathcal{U}}$  be chosen uniformly at random. Then  $\Pr[h \text{ is } (A, B)\text{-good}] \geq 1 - \nu/2$ .*

*Proof.* We prove the result when  $|A| \geq \alpha\lambda$ , the case when  $|A| < \alpha\lambda$  being analogous.

For  $x \in A$ , let  $X_x, Y_x$  be indicator r.v.'s such that  $X_x = 1$  iff  $h(x) \in [\sigma]$ , and  $Y_x = 1$  iff  $h(x) \in [\sigma]$  and  $\exists x' \in B \setminus \{x\}, h(x) = h(x')$ . Let  $X = \sum_{x \in A} X_x$  and  $Y = \sum_{x \in A} Y_x$ . Note that  $X = \left| A \neg_h^{\leq \sigma} \right|$  and  $Y = \left| A \wedge_h^{\leq \sigma} B \right|$ . We have:

$$\mathbb{E}[X_x] = \sigma/\lambda \quad \text{and} \quad \mathbb{E}[Y_x] = (\sigma/\lambda) \cdot \left( 1 - (1 - 1/\lambda)^{|B| - \tau} \right),$$

where  $\tau = 1$  if  $x \in B$  and  $\tau = 0$  otherwise. Thus, letting  $\mu = \mathbb{E}[X]$ , we have  $\mu = \sigma|A|/\lambda \geq \alpha\sigma$ . Using the inequality  $1 - kx \leq (1 - x)^k$  (for  $n \in \mathbb{Z}_+$ ,  $x \in [0, 1]$ ), we have:

$$(1 - 1/\lambda)^{|B| - \tau} \geq 1 - (|B| - \tau)/\lambda \geq 1 - \beta,$$

which implies that  $\mathbb{E}[Y_x] \leq \beta \mathbb{E}[X_x]$ , and hence  $\mathbb{E}[Y] \leq \beta\mu$ .

Note that  $h$  being  $(A, B)$ -good corresponds to  $|X - \mathbb{E}[X]| \leq \beta\mu$  and  $|Y| \leq 2\beta\mu$  – where  $2\beta\mu \geq 2\mathbb{E}[Y]$ . We argue that both hold with sufficient probability through concentration inequalities.

For the first inequality, the independence of the  $X_x$  r.v.'s implies that we can apply Chernoff (Lemma 7). Thus:

$$\Pr[|X - \mathbb{E}[X]| > \beta\mu] \leq 2 \exp(-\beta^2\mu/3).$$

It suffices that  $\mu \geq 3\beta^{-2} \ln(8/\nu)$  for  $|X - \mathbb{E}[X]| \geq \beta\mu$  to hold w.p.  $\geq 1 - \nu/4$ . As we assumed  $|A| \geq \alpha\lambda$ , we have  $\mu = \sigma|A|/\lambda \geq \alpha\sigma$ . Therefore,  $\sigma \geq 3\beta^{-2}\alpha^{-1} \ln(8/\nu)$ , i.e.,  $\sigma \in \Theta(\beta^{-2}\alpha^{-1} \log(1/\nu))$ , suffices for the first inequality to hold w.p.  $1 - \nu/4$ .

For the second inequality, as  $|B| \leq \beta\lambda$ , notice that  $h(B)$  covers less than a  $\beta$  fraction of the hash space below  $\sigma$  in expectation.

By the same analysis as the one we did just above with  $A$  (applying [Lemma 7](#)) we obtain that at most  $\sqrt[3]{2}\beta\sigma$  elements of  $B$  hash to a value  $\leq \sigma$ , i.e.,  $|B|_{h \leq \sigma}^{\leq \sigma} \leq \sqrt[3]{2}\beta\sigma$ , w.p.  $1 - \nu/12$ , when  $\sigma \geq 45\beta^{-1} \ln(12/\nu)$  (we use  $(\sqrt[3]{2}-1)^{-2} \leq 15$ ). We condition on this event, as well as on at most  $\sqrt[3]{2}|A|\sigma/\lambda$  elements of  $A$  hashing to  $\leq \sigma$ , which holds w.p.  $1 - \nu/12$ , when  $\sigma \geq 45\alpha^{-1} \ln(12/\nu)$ . More precisely, we fix the subsets  $B' = B|_{h \leq \sigma}^{\leq \sigma} \subseteq B$  and  $A' = A|_{h \leq \sigma}^{\leq \sigma} \subseteq A$  of respective size at most  $\sqrt[3]{2}\beta\sigma$  and  $\sqrt[3]{2}|A|\sigma/\lambda$  containing the elements hashing to values less than  $\sigma$  (and excluding those hashing higher).

We now analyze  $Y = \sum_{x \in A} Y_x$  under this conditioning. For each  $x \in A \setminus A'$ , we now have that  $Y_x = 0$  w.p. 1, so  $Y = \sum_{x \in A'} Y_x$ . Under the conditioning, the value  $h(x)$  of an element  $x \in A'$  is now picked uniformly at random in  $[\sigma]$  independently of other elements' values. Hence, for an element  $x \in A'$ ,  $Y_x = 1$  w.p.  $\leq \sqrt[3]{2}\beta$  even conditioned on arbitrary random choices for the other  $h(y), y \in (B' \cup A') \setminus \{x\}$ . Therefore,  $\mathbb{E}[Y] \leq \sqrt[3]{2}\beta|A'| \leq \sqrt[3]{4}\beta|A|\sigma/\lambda$ , and by the martingale inequality ([Lemma 9](#)),

$$\Pr[Y > 2\beta|A|\sigma/\lambda] \leq \exp(-(\sqrt[3]{2}-1)^2 \sqrt[3]{4}\beta|A|\sigma/(3\lambda)) \leq 1 - \nu/12,$$

where the last step follows from assuming  $\sigma \geq 45\alpha^{-1}\beta^{-1} \ln(12/\nu)$ . Taking into account the previous conditioning of probability  $1 - \nu/6$  (setting  $A'$  and  $A'$ ),  $\Pr[Y > 2\beta|A|\sigma/\lambda]$  holds w.p.  $1 - \nu/4$ .

Putting everything together, the two inequalities hold simultaneously, i.e.,  $h$  is  $(A, B)$ -good, w.p.  $\geq 1 - \nu/2$ . Note that throughout the analysis,  $\sigma \leq \lambda$  was assumed, which constrains  $\lambda$  in terms of  $\alpha, \beta$ , and  $\nu$ , as in the statement of the lemma.  $\square$

*Proof of [Lemma 1](#).* Let  $h_1, \dots, h_F \in [\lambda]^{\mathcal{U}}$  be  $F$  functions, chosen independently and uniformly at random. For fixed sets  $A, B$ , let  $X_i = 1$  if  $h_i$  is not  $(A, B)$ -good, otherwise  $X_i = 0$ ; by the claim above,  $\Pr[X_i] \leq \nu/2$ . By Chernoff ([Lemma 7](#)), the probability that more than  $\nu F$  of them fail to be  $(A, B)$ -good is  $\Pr\left[\sum_{i \in [F]} X_i \geq \nu F\right] \leq e^{-\nu F/6}$ . There are at most  $|\mathcal{U}|^{\beta\lambda+1}$  choices for each of the subsets  $A$  and  $B$ , so at most  $|\mathcal{U}|^{2\beta\lambda+2}$  choices for the pair  $(A, B)$ . By the union bound, the probability that there are  $\nu F$  functions that are not  $(A, B)$ -good for some pair  $(A, B)$  is at most  $|\mathcal{U}|^{4\beta\lambda} e^{-\nu F/6} < 1$ , assuming  $F > (24\beta\lambda/\nu) \log |\mathcal{U}|$ . Thus, there is a family of  $F$  hash functions such that for every pair  $(A, B)$ , at least  $(1 - \nu)F$  hash functions from the family are  $(A, B)$ -good.  $\square$

### 3.2 Estimation and Sampling of Set Intersection, Union, Difference

Representative hash function immediately give an efficient way for two nodes  $u$  and  $v$  possessing two sets  $S_u$  and  $S_v$  to estimate the size of the intersection  $|S_u \cap S_v|$  with an accuracy  $\varepsilon|S_u \cap S_v|$ , as long as  $S_u \cap S_v$  is a large enough fraction of  $S_u \cup S_v$ . At a high level, the idea is quite natural: estimate the size of the intersection  $|S_u \cap S_v|$  through the size of the intersection  $|h(S_u) \cap h(S_v)|$ , which itself is approximated by the subset  $|h(S_u) \cap h(S_v) \cap [\sigma]|$ . This, of course, omits a few details, and we give the formal statement in [Lemma 2](#) and its proof. The same idea can be used to sample elements in the intersection rather than estimating its size, by having nodes pick as elements the preimages of a random element in  $h(S_u) \cap h(S_v) \cap [\sigma]$  (see [Lemma 3](#)).



---

**Algorithm 1** ESTIMATESIMILARITY( $\varepsilon$ ), for edge  $e = uv$ , with sets  $S_u, S_v$

---

- 1: **if**  $\min(|S_u|, |S_v|) = 0$  **then return** 0
  - 2: Let  $k = \lceil 96\varepsilon^{-3} \ln(12/\nu) / \max(|S_u|, |S_v|) \rceil$ .
  - 3: **if**  $k > 1$  **then** replace  $S_u$  and  $S_v$  by their scaled up versions  $S_u \times [k]$  and  $S_v \times [k]$ .
  - 4: Let  $\mathcal{H}$  be a representative family of hash functions with parameters  $\lambda = 8 \max(|S_u|, |S_v|) / \varepsilon$ ,  $\beta = \varepsilon/4, \alpha = \varepsilon^2/8$ . Let  $F = |\mathcal{H}|$  be its size, and let us index its elements:  $\mathcal{H} = (h_i)_{i \in [F]}$ .
  - 5:  $u$  and  $v$  jointly pick a random number  $i_e \in [F]$ , use  $h = h_{i_e} \in \mathcal{H}^\lambda$  as shared hash function.
  - 6:  $u$  sends  $h(T_u)$  to  $v$ ,  $v$  sends  $h(T_v)$  to  $u$ , where  $T_u = S_u \neg_h S_u$  and  $T_v = S_v \neg_h S_v$ .
  - 7: **return**  $|h(T_u) \cap h(T_v)| \lambda / (\sigma \cdot k)$
- 

**Lemma 2.** ESTIMATESIMILARITY( $\varepsilon$ ) outputs a value within  $\varepsilon \max(|S_u|, |S_v|)$  of  $|S_u \cap S_v|$ , w.p.  $1 - \nu$ . It uses  $O(1)$  messages of  $O(\varepsilon^{-4} \log(1/\nu) + \log \log |\mathcal{U}| + \log \max(|S_u|, |S_v|))$  bits.

*Proof.* Step 3 ensures that the parameters we set in step 4 satisfy the hypotheses of Lemma 1, i.e.,  $\lambda \in \Omega(\varepsilon^{-4} \ln(1/\nu))$ , by making the sets artificially bigger if needed. This is done by replacing the original sets by their Cartesian products with a simple set of size  $k$ :  $S'_u = S_u \times [k]$  and  $S'_v = S_v \times [k]$ , living in the bigger universe  $\mathcal{U} \times [k]$ . Clearly,  $|S'_u \cap S'_v| = k \cdot |S_u \cap S_v|$  and  $\max(|S'_u|, |S'_v|) = k \cdot \max(|S_u|, |S_v|)$ , so if  $s$  is an estimate for  $|S'_u \cap S'_v|$  accurate up to  $\varepsilon \max(|S'_u|, |S'_v|)$  then  $s/k$  is an estimate for  $|S_u \cap S_v|$  accurate up to  $\varepsilon \max(|S_u|, |S_v|)$ . As  $k$  is at most  $O(\varepsilon^{-3} \ln(1/\nu))$ , messages describing an element from the representative hash function family remain of order  $O(\log(1/\varepsilon) + \log(1/\nu) + \log \log |\mathcal{U}| + \log(\max(|S_u|, |S_v|)))$  even as we scale up the sets. From now on, we ignore  $k$ , i.e., assume its value to be 1.

With  $\lambda = 8 \max(|S_u|, |S_v|) / \varepsilon$ ,  $\beta = \varepsilon/4$ ,  $\alpha = \varepsilon^2/8$ , we have that  $|S_u \cup S_v| \leq 2 \max(|S_u|, |S_v|) \leq \beta \lambda$ .

We first show that the estimate we get is a good approximate lower bound on the intersection, and then show it is a good approximate upper bound.

Suppose  $|S_u \cap S_v| \geq \alpha \lambda = \varepsilon \max(|S_u|, |S_v|)$ . Then, by Lemma 1,  $|(S_u \cap S_v) \neg_h (S_u \cup S_v)| \geq (1 - \beta) \sigma |S_u \cap S_v| / \lambda$  w.p.  $1 - \nu$ . Since (by Eq. (3)):

$$(S_u \cap S_v) \neg_h (S_u \cup S_v) \subseteq (S_u \cap S_v) \neg_h S_u \subseteq S_u \neg_h S_u \quad (4)$$

$$\text{and } (S_u \cap S_v) \neg_h (S_u \cup S_v) \subseteq (S_u \cap S_v) \neg_h S_v \subseteq S_v \neg_h S_v \quad (5)$$

it holds that

$$(S_u \cap S_v) \neg_h (S_u \cup S_v) \subseteq (S_u \neg_h S_u) \cap (S_v \neg_h S_v) = T_u \cap T_v,$$

and  $|h(T_u) \cap h(T_v)| \geq |T_u \cap T_v|$ , so  $|h(T_u) \cap h(T_v)| \geq (1 - \beta) \sigma |S_u \cap S_v| / \lambda$ . Hence, when  $|S_u \cap S_v| \geq \alpha \lambda$ , the estimate we give is at most  $\beta |S_u \cap S_v| \leq \varepsilon \max(|S_u|, |S_v|) / 4$  lower than the true value, w.p.  $1 - \nu$ . When  $|S_u \cap S_v| \leq \alpha \lambda$ , the estimate cannot be lower than  $|S_u \cap S_v| - \varepsilon \max(|S_u|, |S_v|) \leq 0$ , since the estimate is always positive, so is within  $\varepsilon \max(|S_u|, |S_v|)$  of the true value.

For the other direction, notice that there are at most as many elements in the intersection of  $h(T_u)$  and  $h(T_v)$  as there are elements in  $S_u \cap S_v$  plus elements in  $S_u \cup S_v$  in collision with another element of  $S_u \cup S_v$ :

$$|h(T_u) \cap h(T_v)| \leq |(S_u \cap S_v)|_h + |(S_u \cup S_v) \wedge_h (S_u \cup S_v)|$$

When  $|S_u \cap S_v| \geq \alpha \lambda$ , this gives that  $|h(T_u) \cap h(T_v)| \leq (1 + \beta) |S_u \cap S_v| \sigma / \lambda + \beta |S_u \cup S_v| \sigma / \lambda$ , hence that the estimate overestimates the result by at most  $2\beta |S_u \cup S_v| \leq 4\beta \max(|S_u|, |S_v|) \leq \varepsilon \max(|S_u|, |S_v|)$ . The same holds when  $|S_u \cap S_v| \leq \alpha \lambda$ .

The communication cost is obtained by adding the cost  $\sigma$  of sending  $h(T_u)$  and  $h(T_v)$  (as they are subsets of  $[\sigma]$ ), and the cost  $\log F$  of sending the index of a representative hash function in the family.  $\square$

As evoked before, almost the same algorithm can be used by the nodes  $u$  and  $v$  to jointly sample elements from the intersection of their sets by selecting a random element  $y$  in  $h(T_u) \cap h(T_v)$  and (respectively) outputting the single element in  $S_u \cap h^{-1}(y)$  and  $S_v \cap h^{-1}(y)$ . Since when  $S_u \cap S_v$  is large – at least  $\alpha\lambda$  – a large fraction of the elements of  $h(T_u) \cap h(T_v)$  are images of elements of  $S_u \cap S_v$ ,  $u$  and  $v$  are likely to sample elements from  $S_u \cap S_v$  this way.

---

**Algorithm 2** JOINTSAMPLE( $\varepsilon$ ), for edge  $e = uv$ , with sets  $S_u, S_v$

---

- 1: **if**  $\min(|S_u|, |S_v|) = 0$  **then return** 0
  - 2: Let  $k = \lceil 96\varepsilon^{-3} \ln(12/\nu) / \max(|S_u|, |S_v|) \rceil$ .
  - 3: **if**  $k > 1$  **then** replace  $S_u$  and  $S_v$  by their scaled up versions  $S_u \times [k]$  and  $S_v \times [k]$ .
  - 4: Let  $\mathcal{H}$  be a representative family of hash functions with parameters  $\lambda = 8 \max(|S_u|, |S_v|)/\varepsilon$ ,  $\beta = \varepsilon/4, \alpha = \varepsilon^2/8$ . Let  $F = |\mathcal{H}|$  be its size, and let us index its elements:  $\mathcal{H} = (h_i)_{i \in [F]}$ .
  - 5:  $u$  and  $v$  jointly pick a random number  $i_e \in [F]$ , use  $h = h_{i_e} \in \mathcal{H}^\lambda$  as shared hash function.
  - 6: Let  $J = |h(T_u) \cap h(T_v)|$ . **if**  $J = 0$  **then return** nothing.
  - 7:  $u$  and  $v$  jointly pick a random number  $j_e \in [J]$ .
  - 8: **return**  $h^{-1}(j_e) \cap T_u$  on  $u$ 's side,  $h^{-1}(j_e) \cap T_v$  on  $v$ 's side.
- 

**Lemma 3.** *When  $|S_u \cap S_v| \geq \varepsilon \max(|S_u|, |S_v|)$ , two nodes  $u$  and  $v$  running JOINTSAMPLE( $\varepsilon$ ) output the same element at the end of the algorithm, w.p.  $1 - 5\varepsilon/4 - \nu$ .*

*Proof.* The result follows naturally from  $h(T_u) \cap h(T_v)$  containing at least  $(1 - \varepsilon/4)|S_u \cap S_v|$  elements of  $h(S_u \cap S_v)$ , and  $h(T_u) \cap h(T_v)$  being of size at most  $(1 + \varepsilon)|S_u \cap S_v|$ , w.p.  $1 - \nu$ , as argued in the proof of Lemma 2.  $\square$

The nodes can even sample multiple elements from the intersection of their sets by picking multiple indices instead of a single one in step 7. This takes the same number of CONGEST rounds. The only caveat is that some sampled elements might be duplicates when  $k = \lceil 96\varepsilon^{-3} \ln(12/\nu) / \max(|S_u|, |S_v|) \rceil > 1$ .

While JOINTSAMPLE shows some of the ideas we will use later to try multiple colors in a single round, the fact that it only involves two parties means that the procedure may have been designed in a simpler manner, invoking Newman's theorem [New91]. The way we use representative hash functions later to sample random colors is however very multiparty, and may not be derived from statements about public vs private randomness in 2-party communication complexity.

### 3.3 Application: Sparsity

A number of recent algorithms for distributed coloring and other problems treat nodes differently depending on a measure called *sparsity*. Intuitively, sparsity measures the number of missing edges in a node's neighborhood. Depending on the problem, two definitions of sparsity are in use.

**Definition 1.** *For any subset of the nodes  $S \subseteq V$ , let  $m(S) := |E[S]|$  be the number of edges between nodes of  $S$ . The global sparsity of a node  $v$  is defined as:*

$$\zeta_v^{[\Delta]} = \frac{1}{\Delta} \left( \binom{\Delta}{2} - m(N(v)) \right) = \frac{\Delta - 1}{2} - \frac{1}{2\Delta} \sum_{u \in N(v)} |N(u) \cap N(v)|$$

*The local sparsity of a node  $v$  is defined as:*

$$\zeta_v^{[d]} = \frac{1}{d_v} \left( \binom{d_v}{2} - m(N(v)) \right) = \frac{d_v - 1}{2} - \frac{1}{2d_v} \sum_{u \in N(v)} |N(u) \cap N(v)|$$

The global sparsity is the definition of sparsity generally used in algorithms that solve a coloring problem in which each node can select its own color from  $\Delta + 1$  colors. The local sparsity is the definition generally used when nodes have only  $\deg + 1$  colors to choose from.

ESTIMATESIMILARITY immediately gives an efficient way to estimate both definitions of sparsity – under an assumption for the local sparsity. We give the analysis for the global sparsity, and later explain the caveat with local sparsity.

---

**Algorithm 3** ESTIMATESPARSITY( $\varepsilon$ ), for  $v$  (for global sparsity)

---

- 1: **for all**  $u \in N(v)$  **do**
  - 2:    $v$  runs ESTIMATESIMILARITY( $\varepsilon/2$ ) with  $u$  to get an estimate of  $s_u = |N(u) \cap N(v)|$ .
  - 3: **end for**
  - 4:  $v$  outputs  $\frac{\Delta-1}{2} - \frac{1}{2\Delta} \sum_u s_u$  as estimate for its sparsity.
- 

**Lemma 4.** ESTIMATESPARSITY( $\varepsilon$ ) outputs an estimate of  $\zeta_v^{[\Delta]}$  that is  $\varepsilon\Delta$ -close to the true value, w.p.  $1 - (\nu\Delta)^{\varepsilon\Delta/2}$ .

*Proof.* For the estimate to be off by  $\varepsilon\Delta$  or more, at least  $\varepsilon\Delta/2$  neighbors of  $v$  must give an estimate  $s_u$  that is at least  $\varepsilon\Delta/2$  off. For a subset of  $\varepsilon\Delta/2$  neighbors of  $v$ , the probability that they all give a bad estimate is at most  $\nu^{\varepsilon\Delta/2}$ . So the probability that such an all-failing subset exists is at most:

$$\binom{\Delta}{\varepsilon\Delta/2} \nu^{\varepsilon\Delta/2} \leq (\nu\Delta)^{\varepsilon\Delta/2}. \quad \square$$

Note that this means ESTIMATESPARSITY works w.h.p. when  $\nu \in 1/\text{poly}(n)$ , as well as when  $\nu \in 1/\text{poly}(\Delta)$  and  $\varepsilon\Delta \in \Omega(\log n / \log \log n)$ .

**Estimating local sparsity.** Local sparsity can be similarly estimated, with a caveat. The accuracy of ESTIMATESIMILARITY depends on the sizes of the sets we are dealing with. For global sparsity, i.e., in the  $\Delta + 1$  setting, the global bound of  $\Delta$  on the degrees of all nodes implies that ESTIMATESIMILARITY( $\varepsilon$ ) gives an estimate of  $|N(u) \cap N(v)|$  within  $\varepsilon\Delta$  of the true value.

The difficulty with local sparsity comes from higher degree neighbors. If we could guarantee that each estimate  $s_u$  of  $|N(u) \cap N(v)|$  for  $u \in N(v)$  was accurate up to  $\varepsilon d_v$  w.p.  $1 - \nu$ , we would only need to replace  $\Delta$  by  $d_v$  in the formula at the end of Alg. 3, i.e., output  $\frac{d_v-1}{2} - \frac{1}{2d_v} \sum_u s_u$ , to get an estimate of the local sparsity within  $\varepsilon d_v$  w.p.  $(\nu d_v)^{\varepsilon d_v/2}$ . Unfortunately, ESTIMATESIMILARITY( $\varepsilon$ ) only gives an estimate of  $|N(u) \cap N(v)|$  within  $\varepsilon \max(d_u, d_v)$  of the true value, which might be completely off if  $d_u \gg d_v$ , e.g., with  $d_u \geq d_v/\varepsilon$ . We hence only claim that we are able to estimate the local sparsity of nodes which do not have too many neighbors of much higher degree.

**Lemma 5.** Let a node  $v$  have less than  $\varepsilon d_v/3$  neighbors of degree  $\geq 2d_v$ . ESTIMATESPARSITY( $\varepsilon$ ) can be tweaked to output an estimate of  $\zeta_v^{[d]}$  that is  $\varepsilon d_v$ -close to the true value, w.p.  $1 - (\nu d_v)^{\varepsilon d_v/3}$ .

*Proof.*  $\varepsilon d_v/3$  nodes can contribute at most  $\varepsilon d_v/3$  to the sparsity, so estimating the number of missing edges within the rest of  $v$ 's neighborhood with precision  $2\varepsilon d_v/3$  suffices to get a  $\varepsilon d_v$ -accurate estimate of  $v$ 's local sparsity. Since the rest of  $v$ 's neighborhood has degree at most  $2d_v$ , we can run ESTIMATESPARSITY( $\varepsilon/3$ ) on the subgraph it induces to get an  $2\varepsilon d_v/3$  estimate of  $v$ 's sparsity in this subgraph, giving the result.  $\square$

### 3.4 Application: Local Triangle Finding

In standard property testing, the goal is to detect with constant probability if a graph is far from satisfying a property. For example, the task may be to distinguish with constant probability

whether a graph contains no triangle vs whether an  $\varepsilon$ -fraction of the edges needs to be deleted for the graph to contain no triangle. The task is solved distributedly but is global in several ways: the notion of distance between graphs takes into account the whole graph and the goal is only for one node of the graph to detect the property. Our ESTIMATESIMILARITY primitive allows us to solve a related but more local task: make every edge involved in many triangles detect that it is so.

**Theorem 2.** *There exists an  $O(\varepsilon^{-4})$ -round randomized CONGEST algorithm that, for each edge, detects w.h.p. when it is part of  $\varepsilon\Delta$  triangles.*

*Proof.* On each edge  $uv$ , estimate the size of the intersection  $|N(u) \cap N(v)|$ .  $\square$

Compared to the usual property testing setting, we solve a harder problem in that we solve the problem with high probability instead of constant probability, and solve it on each edge instead of globally. However our condition for detection is incomparable with that of the property testing setting: our algorithm works whenever a single edge is part of  $\varepsilon\Delta$  triangles, while property testing typically assumes  $\varepsilon$ -farness, i.e., that  $\varepsilon|E|$  edges have to be deleted from the graph to make it triangle-free. The two are incomparable, since our condition being satisfied on some edge only implies that the graph is  $(\varepsilon\Delta/|E|)$ -far from being triangle-free, while in an  $\varepsilon$ -far graph, it can be the case that each edge is only part of at most one triangle while  $\Delta \in \Theta(n)$ .

### 3.5 Application: Local 4-Cycle Finding

Our technique also allows us to detect 4-cycle locally in CONGEST, with the same tradeoff as in the detection of triangles compared to standard property testing. Our result is stronger in that we detect occurrence of the pattern locally instead of globally, and we have a higher success probability, but on the other end the two settings are incomparable in that an  $\varepsilon$ -far graph might not satisfy our local threshold for detection anywhere in any meaningful sense and vice-versa.

**Theorem 3.** *There exists an  $O(\varepsilon^{-4})$ -round CONGEST algorithm that, for each pair of edges incident on the same vertex, detects w.h.p. when they are part of  $\varepsilon\Delta$  4-cycles.*

*Proof.* Let  $v$  be a vertex of the graph.  $v$  picks a random representative hash function  $h$  and sends it to all its neighbors  $u \in N(v)$ , who answer with  $N(u) \rightarrow_h^{\leq \sigma} N(u)$ . For each pair of neighbors  $u, u'$  of  $v$ ,  $v$  then estimates  $|N(u) \cap N(u')|$  with these hashes, as is done in ESTIMATESIMILARITY.  $\square$

## 4 Ultrafast Coloring in Congest

The techniques we presented allow us to implement all steps of a recent D1LC algorithm in CONGEST. The correctness of the algorithm is in [HKNT21]. We give a succinct but complete description of the algorithm in Appendix E for reference.

**Theorem 1.** *The D1LC problem can be solved w.h.p. in  $O(\log^5 \log n)$  rounds in the CONGEST model with  $\log n$  bandwidth. When all nodes have degree at least  $\log^7 n$ , the algorithm uses only  $O(\log^* n)$  rounds.*

The algorithm as a whole is bandwidth-efficient, but a larger bandwidth is assumed in four places. Two particularly stand out, and are the focus of the upcoming Sec. 4.1 and 4.2. We sketch how to adapt the rest of the algorithm in Sec. 4.3, with the rigorous treatment of these last minor modifications deferred to Appendix D.

The most challenging step to implement in CONGEST is a method for sampling and “trying” a set of colors, called “MultiTrial”. We detail its implementation in Sec. 4.1. Another non-trivial step of computing an almost-clique decomposition is dealt with in Sec. 4.2.

## 4.1 MultiTrial

When breaking down recent ultrafast ( $O(\log^* n)$  rounds for graphs with large enough degrees) algorithms, all of them contain a step that stands out in the amount of information it requires. Intuitively, in those algorithms, some nodes try up to  $\Theta(\log n)$  colors over the course of the algorithm, with the idea that if each color succeeds with constant probability, then the nodes get colored w.h.p. by trying that many colors. However,  $\log n$  arbitrary colors take at least  $\Theta(\log \Delta \log n)$  bits to describe (and possibly much more when nodes are given lists of colors instead of using  $[\deg + 1]$  or  $[\Delta + 1]$ ), which would require  $\Omega(\log \Delta)$  rounds in CONGEST. A more communication-efficient procedure following the same idea needs to compromise on some front, which we do here by compromising on the randomness and accuracy of the colors that nodes try, using representative hash functions.

We give a procedure – MULTITRIAL – that within bandwidth  $b$  allows a node  $v$  to try  $x$  random colors from its palette, where  $x$  can be as large as  $\Theta(b)$ . While trying  $\Theta(b)$  colors in a single round is straightforward in LOCAL, a naïve implementation in CONGEST would take  $\Omega(\log |\mathcal{C}|)$  rounds for a color space  $\mathcal{C}$ . We achieve similar results in CONGEST by replacing the random sampling of colors by a pseudorandom one. While the  $x$  colors tried are not as random as  $x$  independent random samples, enough randomness is used so that one of those colors succeeds w.p.  $1 - \exp(-\Omega(x)) - \exp(-\Omega(b))$ , essentially the same probability as if they were independent. With bandwidth  $b \in \Theta(\log n)$ , this allows up to  $\Theta(\log n)$  colors to be tried in a single round, and for a node to be colored with probability  $1 - 1/\text{poly}(n)$ . Previously, this was only known to be possible in the very restricted setting of locally sparse graphs [HN21].

To get an intuitive understanding of our approach, let us assume that each node  $v$  can sample and communicate to its neighbors a random hash function  $h_v : \mathcal{C} \rightarrow [\lambda] = \{1, \dots, \lambda\}$  for a number  $\lambda$  of its choice. To have all nodes try  $x$  colors, on each edge  $uv$ , node  $v$  sends to  $u$  the hash values of the color it tries through  $h_u$  (and vice versa). If  $v$  tries a color  $\psi$  that hashes to a value different from all the hash values it received,  $v$  can safely color itself with  $\psi$ . To make the procedure more efficient, we have  $v$  pick random colors among those with a hash value  $\leq \sigma = O(\log n)$  through  $h_v$ . With this restriction, the neighbors of  $v$  only need to tell  $v$  about the colors they try that hash to a value  $\leq \sigma$  through  $h_v$ . This uses  $\sigma = O(\log n)$  bits of communication using a  $\sigma$ -size bitstring.

For this to work, the hash function must satisfy three properties: first, enough colors must hash to a value  $\leq \sigma = O(\log n)$ ; second, collisions must be rare enough for a unique hash to be sampled; and third, it should be possible to communicate a hash function in  $O(\log n)$  bits so the process takes  $O(1)$  rounds. Increasing  $\lambda$  reduces the number of collisions, but reduces how many elements hash to a value  $\leq \sigma = O(\log n)$ , so a balance must be found. This balance is found at  $\lambda \in \Theta(|\Psi_v|)$ .

Using families of representative hash functions, whose existence we proved in Lemma 1, we show how to implement MULTITRIAL efficiently in CONGEST (Alg. 4 and Lemma 6).

The pseudocode of MULTITRIAL is presented in Alg. 4. Let  $\alpha = 1/12$ ,  $\beta = 1/3$ , and for each  $\lambda \in \mathbb{N}$ , let  $\nu_\lambda = \max(n^{-c}, 12 \exp(-\alpha\lambda/45))$  and  $\sigma_\lambda \in \Theta(\beta^{-2}\alpha^{-1} \log(1/\nu_\lambda))$ , for a constant  $c > 3$  (hence, even for  $n^2$  events of probability  $\nu_\lambda$ , when  $\lambda \in \omega(\log n)$ , none occurs w.h.p.). We assume that all the nodes know, for each  $\lambda \in [2\beta^{-1}\Delta] = [6\Delta]$ , a common family of hash functions  $\mathcal{H}^\lambda = (h_i^{(\lambda)})_{i \in [F]} \subseteq [\lambda]^{\mathcal{C}}$  and value  $\sigma_\lambda$  with the properties of Lemma 1. This could be achieved, e.g., by having each node compute the lexicographically first such pair of family and parameter, for each  $\lambda$ . Note that  $\sigma_\lambda \in O(\log n)$  for all  $\lambda$ , and that this parameter can be chosen to be the same  $\sigma = \Theta(\log n)$  for all values of  $\lambda \in \omega(\log n)$ .

---

**Algorithm 4** MULTITRIAL( $x$ ), for node  $v$ 

---

- 1: Let  $\lambda_v \leftarrow 6|\Psi_v|$ , pick a random  $h_v = h_{i_v}^{(\lambda_v)} \in \mathcal{H}^{\lambda_v}$ , broadcast  $\lambda_v, i_v$  to  $N(v)$ .
  - 2:  $X_v \leftarrow x$  independently chosen random colors in  $\Psi_v \neg_{h_v} \Psi_v$ .
  - 3: **for all**  $u \in N(v)$  and all  $i \in [\sigma_{\lambda_u}]$  **do**
  - 4:   **if**  $\exists \psi \in X_v, h_u(\psi) = i$  **then**  $b_{v \rightarrow u}[i] \leftarrow 1$  **else**  $b_{v \rightarrow u}[i] \leftarrow 0$
  - 5: **end for**
  - 6: Send  $b_{v \rightarrow u}$  and receive  $b_{u \rightarrow v}$  to/from  $u$ , for all  $u \in N(v)$ .
  - 7: **if**  $\exists \psi \in X_v$  s.t.  $\forall u \in N(v), b_{u \rightarrow v}[h_v(\psi)] = 0$  **then**
  - 8:   Adopt some such  $\psi$  as permanent color and broadcast to  $N(v)$ .
  - 9: **end if**
- 

**Lemma 6.** *For every node  $v$ , if  $x \leq |\Psi_v|/2|N(v)|$ , then an execution of MULTITRIAL( $x$ ) colors  $v$  with probability  $1 - (7/8)^x - 2\nu$ , where  $\nu \leq e^{-\Theta(|\Psi_v|)} + n^{-\Theta(1)}$ , even when conditioned on any particular combination of random choices of the other nodes.*

*Proof.* Consider  $Y_v = \bigcup_{u \in N(v)} X_u$ , the set of colors tried by neighbors of  $v$ . Note that  $|Y_v| \leq x|N(v)| \leq |\Psi_v|/2 \leq \lambda_v/12$  (recall  $\lambda_v = 6|\Psi_v|$ ), and its composition is independent from  $v$ 's choice of random colors. Letting  $T_v = \Psi_v \setminus Y_v$  and  $P_v = \Psi_v \cup Y_v$ , we have  $|P_v|, |T_v|, |\Psi_v| \in [\lambda_v/12, \lambda_v/3]$ , and so, the triplets  $(\lambda_v, P_v, T_v)$  and  $(\lambda_v, P_v, \Psi_v)$  satisfy Lemma 1 with our parameters  $\alpha, \beta, \nu$ . Let  $\sigma = \sigma_{\lambda_v}$ . The lemma implies that w.p.  $1 - \nu$ ,  $|\Psi_v \neg_{h_v} \Psi_v| \leq (1 + \beta) \cdot \sigma |\Psi_v| / \lambda_v \leq 2\sigma/9$ , and similarly, w.p.  $1 - \nu$ ,  $|T_v \neg_{h_v} P_v| \geq (1 - 2\beta) \cdot \sigma |T_v| / \lambda_v \geq \sigma/36$ . Since additionally  $(T_v \neg_{h_v} P_v) \subseteq (\Psi_v \neg_{h_v} P_v) \subseteq (\Psi_v \neg_{h_v} \Psi_v)$ , we conclude that  $T_v \neg_{h_v} P_v$  forms a  $(\sigma/36)/(2\sigma/9) = 1/8$  fraction of  $\Psi_v \neg_{h_v} \Psi_v$ , and any color randomly picked in  $\Psi_v \neg_{h_v} \Psi_v$  is in  $T_v \neg_{h_v} P_v$  w.p. at least  $1/8$ . Hence, conditioned on the  $1 - 2\nu$  probability event that  $|T_v \neg_{h_v} P_v| \geq |\Psi_v \neg_{h_v} \Psi_v|/8$ , the  $x$  colors randomly picked by  $v$  in  $\Psi_v \neg_{h_v} \Psi_v$  all miss  $T_v \neg_{h_v} P_v$  w.p. at most  $(7/8)^x$ . As any color found in  $T_v \neg_{h_v} P_v$  will be successful for  $v$ ,  $v$  gets colored w.p.  $1 - (7/8)^x$ , conditioned on an event of probability  $1 - 2\nu$ .  $\square$

## 4.2 Almost-Clique Decomposition

Almost-clique decompositions are commonly defined and computed according to a relation that classifies two connected nodes as *friends* if they share most of their neighborhoods. Nodes whose neighborhood is almost all friends have dense neighborhoods, i.e., most pairs of nodes in their neighborhood are connected by an edge, and are mostly adjacent to nodes of similar degree. Reciprocally, nodes with few friends are either *uneven*, i.e., adjacent to many nodes of much higher degree, or have a sparse neighborhood, i.e., a large fraction of their neighbors are not directly connected.

**Definition 2** ([AA20]). *Let  $\varepsilon \in [0, 1]$ . An edge  $uv$  is*

- $\varepsilon$ -balanced *iff*  $\min(d_u, d_v) \geq (1 - \varepsilon) \max(d_u, d_v)$ ,
- $\varepsilon$ -friend *iff it is  $\varepsilon$ -balanced and  $|N(u) \cap N(v)| \geq (1 - \varepsilon) \min(d_u, d_v)$ .*

Computing  $\varepsilon$ -FRIEND predicates exactly would be too costly in many models of computation where almost-clique decompositions are relevant. Fortunately, this much accuracy is not needed: in CONGEST, it suffices to have access to a procedure  $\varepsilon$ -BUDDY that distinguishes between an edge being  $\varepsilon$ -friend and it being far from it, i.e., not  $c \cdot \varepsilon$ -friends for some constant  $c > 1$  (see, e.g., Appendix B in [HKMN20] for details). The idea of computing almost-clique decompositions using a sampling-based approach originated in [ACK19]. This can be done easily by testing whether  $d_u$  and  $d_v$  are approximately equal and then running ESTIMATESIMILARITY if this is the case, which works w.h.p. with bandwidth  $\log n$ .

### 4.3 Final Minor Modifications

While the bulk of adapting the LOCAL algorithm of [HKNT21] for D1LC to CONGEST is figuring out how to efficiently try up to  $\Theta(\log n)$  colors (MULTITRIAL) and compute an almost-clique decomposition (COMPUTEACD) within the  $O(\log n)$  bandwidth constraint, a few additional minor modifications are required. We defer their rigorous treatment to Appendix D, and sketch here the essence of those changes.

**Leader selection.** In the original LOCAL algorithm, a node is chosen as leader in each almost-clique based on a quantity called *slackability* (see Appendix C for its definition). As the slackability of a node is entirely determined by its neighborhood (palettes included), finding the node of minimum slackability within each almost-clique is trivial in LOCAL, and only takes  $O(1)$  rounds. This is no longer the case in CONGEST. This is circumvented by arguing that it suffices to identify a node of low but not necessarily minimal slackability within each almost-clique, and that the slackability can be estimated with the needed accuracy efficiently in CONGEST. The details of these two arguments are given in Appendix D.1.

**Coloring put-aside sets.** In the D1LC algorithm we are adapting to CONGEST, almost-cliques and their nodes are dealt with differently depending on whether their sparsity is above or below some threshold. Very dense almost-cliques are dealt with by putting aside a subset of its nodes to color later, in order to provide temporary slack to the rest of the almost-cliques. To color those nodes at the end of the algorithm, information about their palettes and how they are connected is centralized. How all this information can be centralized in CONGEST is not as simple as in LOCAL, and requires in particular more control on the sizes of the put-aside sets. The adaptation of this part of the algorithm is detailed in Appendix D.2.

**Handling large colors.** An important aspect of list-coloring problems in models with a bandwidth constraint is that colors may live in a color space bigger than  $2^{\Theta(b)}$ , i.e., too big for the nodes to send a color in a constant number of rounds. Our procedure MULTITRIAL circumvents this, in the case of trying multiple colors, by hashing. This is fortunately possible in other parts of the algorithm, e.g., whenever nodes need to inform their neighbors of their newly adopted color, or need to inform another node of the color it should try. We show that color spaces of size up to  $\exp(n^{\Theta(1)})$  can be handled without increasing the complexity of the algorithm. How this is done is explained in Appendix D.3.

## 5 Uniform Implementation

**Lemma 1** – on the existence of representative hash functions – does not give an explicit construction. Hence, an algorithm relying on their existence either needs to assume that the nodes receive a common family of such hash functions “for free” at the beginning, or have the nodes find a common family of representative hash functions themselves. This second option requires extensive computational resources, as it involves exploring the space of  $F$ -element subsets of  $[\lambda]^c$ , performing expensive statistical tests on each subset.

In this section, we show how our subroutines that use representative hash functions can be modified to not rely on them. With these new implementations, the nodes only have to perform computations polynomial in  $n$  and  $\Delta$  in our algorithm. We leave as an open question the explicit construction of families of representative hash functions.

In our new uniform implementations of MULTITRIAL and BUDDY, a key idea is to introduce and exploit some asymmetry between the parties. By having one of the parties partially *choose* a hash function instead of taking it fully at random, this party can ensure that not too many collisions occur between the elements it knows of. To remove the reliance on representative hash

functions, we make use of other objects with explicit constructions: pairwise-independent hash functions, representative multisets (constructed from averaging samplers), and error-correcting codes. Subroutines other than MULTITRIAL and BUDDY do not rely on representative hash functions, and as such do not need to be modified for the algorithm to be uniform.

## 5.1 MultiTrial

The core properties of MULTITRIAL are twofold: first, MULTITRIAL is able to describe up to  $\Theta(b)$  colors in a single  $b$ -sized CONGEST message; second, those colors are sufficiently random that it is as if they each had a constant probability of success, i.e., when trying  $k$  colors, at least one succeeds w.p.  $1 - \exp(-\Omega(k))$ .

Let  $\mathcal{H}_{\text{pwi}}^\lambda$  be a set of  $\varepsilon$ -almost pairwise-independent hash functions from  $\mathcal{C}$  to  $[\lambda]$  (see, e.g., Problem 3.4 in [Vad12]). When selecting a hash function from such a family,

$$\Pr_{h \leftarrow \mathcal{H}_{\text{pwi}}^\lambda} [h(x_1) = y_1 \text{ and } h(x_2) = y_2] \leq \frac{1 + \varepsilon}{\lambda^2}, \quad \forall x_1, x_2 \in \mathcal{C}, y_1, y_2 \in [\lambda] \text{ with } x_1 \neq x_2.$$

There exist explicit such families of size  $\text{poly}(\lambda, \log|C|, 1/\varepsilon)$ , such that sampling an element from the family only requires to pick  $(\log \lambda + \log \log|C| + \log(1/\varepsilon))$  random bits.

We sketch the argument showing that a MULTITRIAL procedure that only relies on explicit constructs (and thus, does not rely on representative hash functions at the moment) is possible. We give pseudocode of this procedure below. Its core idea is to have each node  $v$  select a hash function that has few collisions in its palette. As a result, the image of  $v$ 's palette through the hash function it picked is almost of the same size as the palette itself. Let us now consider the images of the colors tried by  $v$ 's neighbor through this hash function. If the total number of colors tried by the neighbors of  $v$  is less than half the number of colors in  $v$ 's palette, then a constant fraction of the image of  $v$ 's palette is necessarily not the image of any color tried by a neighbor of  $v$ . Therefore, by sampling  $x$  hashes from the image of its palette using an explicit representative multiset over the space of hashes ([HN21], and Appendix B),  $v$  can succeed in securing a random color w.p.  $1 - \exp(-\Omega(x)) - \exp(-\Omega(b))$ .

---

**Algorithm 5** Uniform MULTITRIAL( $x$ ), for node  $v$

---

- 1: Let  $\lambda_v \leftarrow 6|\Psi_v|$ , pick a random  $h_v = h_{i_v}^{(\lambda_v)} \in \mathcal{H}_{\text{pwi}}^{\lambda_v}$  with at most  $\lambda_v/3$  collisions in  $\Psi_v$ , broadcast  $\lambda_v, i_v$  to  $N(v)$ .
  - 2: Let  $\sigma_v = \min(b, \lambda_v)$ , pick a random representative multiset  $S_v$  of size  $\sigma_v$ , subset of  $[\lambda_v]$ . Let  $S_v = \{s_1^{(v)}, \dots, s_{\sigma_v}^{(v)}\}$ .
  - 3:  $X_v \leftarrow x$  random elements of  $\Psi_v \cap h_v^{-1}(S_v)$ , picked uniformly at random.
  - 4: **for all**  $u \in N(v)$  and all  $i \in [\sigma_u]$  **do**
  - 5:     **if**  $\exists \psi \in X_v, h_u(\psi) = s_i^{(u)}$  **then**  $b_{v \rightarrow u}[i] \leftarrow 1$  **else**  $b_{v \rightarrow u}[i] \leftarrow 0$
  - 6:     **end for**
  - 7: Send  $b_{v \rightarrow u}$  and receive  $b_{u \rightarrow v}$  to/from  $u$ , for all  $u \in N(v)$ .
  - 8: **if**  $\exists \psi \in X_v$  s.t.  $\forall u \in N(v), b_{u \rightarrow v}[h_v(\psi)] = 0$  **then**
  - 9:     Adopt some such  $\psi$  as permanent color and broadcast to  $N(v)$ .
  - 10: **end if**
- 

## 5.2 Almost-Clique Decomposition

Similar ideas to those that enable a uniform implementation of MULTITRIAL allow for a uniform implementation of  $\varepsilon$ -BUDDY. Again, we provide pseudocode of the procedure below, and sketch the argument for its correctness.

The algorithm takes place between two nodes  $u$  and  $v$ . They first test whether their degrees differ significantly. If they do, the algorithm stops: the edge is not  $\varepsilon$ -BUDDY. Otherwise, one of



the nodes chooses an almost pairwise-independent hash function with few collisions between the IDs of its neighborhood. Then, the nodes pick a random representative multiset over the space of hashes, and compute the sampled hashes that are the image of a single of their neighbors.

If the nodes have few hashes in common, they declare the edge non- $\varepsilon$ -BUDDY, as having few hashes in common is only likely when the nodes' neighborhoods do not mostly intersect.

When the nodes share a lot of hashes, however, two causes are possible: they either share a large part of their neighborhoods, or the hash function that was picked has many collisions between the two neighborhoods. The rest of the algorithm is devoted to distinguishing the two.

To do so, the nodes apply an error-correcting code to the ID of each of their neighbors. As a result, distinct IDs now differ in a constant fraction of their bits. The nodes then each build a bitstring by concatenating the preimages of the hashes they found to have in common. The two resulting bitstrings are guaranteed to be of small Hamming distance if the nodes genuinely share many neighbors, but must differ in a large fraction of indices if the hashes that the nodes found in common were due to collisions. The nodes sample random indices of these bitstrings using representative multisets, exchange the bits at those indices, estimate the Hamming distance between their bitstrings from those bits, and conclude. The idea of using an error-correcting code to increase the Hamming distance between distinct bitstrings has been used previously in communication-focused models, for example [Amb96].

In the following pseudocode, “\_” represents an empty bitstring, **enc** is the encoder of an error correcting code,  $x_u.\mathbf{enc}(w)$  is the concatenation of bitstrings  $x_u$  and  $\mathbf{enc}(w)$ . The error correcting code is chosen to have parameter, e.g.,  $[3b, b, b/2]$ , where  $b \in \Theta(\log n)$  is the number of bits used to write IDs in the graph, meaning that IDs initially written on  $b$  get expanded to  $3b$  bits, and that two distinct IDs differ by at least  $b/2$  bits after the encoding.

---

**Algorithm 6** Uniform  $\varepsilon$ -BUDDY, for edge  $uv$

---

- 1: **if**  $d_u > d_v/(1 - \varepsilon)$  **or**  $d_v > d_u/(1 - \varepsilon)$  **then return false**
  - 2: Let  $\lambda \leftarrow 6 \max(d_u, d_v)/\varepsilon$ .  $v$  chooses an hash function  $h = h_i^{(\lambda)} \in \mathcal{H}_{\text{pwi}}^\lambda$  such that at most  $\varepsilon d_v/3$  elements in  $N(v)$  are involved in a collision, and sends  $(\lambda, i)$  to  $u$ .
  - 3: Let  $\sigma = \min(b, \lambda)$ ,  $u$  and  $v$  pick a random representative multiset  $S$  of size  $\sigma$ , subset of  $[\lambda]$ .  
Let  $S = \{s_1, \dots, s_\sigma\}$ .
  - 4: **for all**  $i \in [\sigma]$  **do**
  - 5: **if**  $\exists! w \in N(u), h(w) = s_i$  **then**  $b_u[i] \leftarrow 1$  **else**  $b_u[i] \leftarrow 0$
  - 6: **if**  $\exists! w \in N(v), h(w) = s_i$  **then**  $b_v[i] \leftarrow 1$  **else**  $b_v[i] \leftarrow 0$
  - 7: **end for**
  - 8:  $u$  and  $v$  exchange  $b_u$  and  $b_v$
  - 9: **if**  $|\{i : b_u[i] \cdot b_v[i] = 1\}| \leq (1 - 3\varepsilon)\sigma$  **then return false**
  - 10:  $x_u \leftarrow \text{"_"}, x_v \leftarrow \text{"_"}$
  - 11: **for all**  $i \in [\sigma]$  s.t.  $b_u[i] \cdot b_v[i] = 1$  **do**
  - 12:  $x_u \leftarrow x_u.\mathbf{enc}(w)$  where  $w$  is the unique  $w \in N(u)$  s.t.  $h(w) = s_i$
  - 13:  $x_v \leftarrow x_v.\mathbf{enc}(w)$  where  $w$  is the unique  $w \in N(v)$  s.t.  $h(w) = s_i$
  - 14: **end for**
  - 15: Let  $\ell = \text{length}(x_u)$ ,  $\sigma' = \min(b, \ell)$ ,  $u$  and  $v$  pick a random representative multiset  $S$  of size  $\sigma'$ , subset of  $[\ell]$ .
  - 16: **if**  $|\{i \in [\sigma'] : x_u[i] \neq x_v[i]\}| \geq \varepsilon\sigma'$  **then return false else return true**
- 

## Acknowledgements

This project was supported by the European Union's Horizon 2020 Research and Innovation Programme under grant agreement no. 755839 and by Icelandic Research Fund grants no. 174484 and 217965. Part of the work was done while T. Tonoyan was with the CS Department of the

Technion, Israel.

## References

- [AA20] Noga Alon and Sepehr Assadi. Palette sparsification beyond  $(\Delta + 1)$  vertex coloring. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 176 of *LIPICs*, pages 6:1–6:22, 2020.
- [ABI86] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986.
- [ACK19] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for  $(\Delta + 1)$  vertex coloring. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 767–786, 2019.
- [Amb96] Andris Ambainis. Communication complexity in a 3-computer model. *Algorithmica*, 16(3):298–301, 1996.
- [Bar16] Leonid Barenboim. Deterministic  $(\Delta + 1)$ -coloring in sublinear (in  $\Delta$ ) time in static, dynamic, and faulty networks. *Journal of the ACM*, 63(5):47:1–47:22, 2016.
- [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.
- [BJKS93] Jürgen Bierbrauer, Thomas Johansson, Gregory Kabatianskii, and Ben J. M. Smeets. On families of hash functions via geometric codes and concatenation. In *Advances in Cryptology - CRYPTO*, volume 773 of *LNCS*, pages 331–342, 1993.
- [BPS11] Zvika Brakerski and Boaz Patt-Shamir. Distributed discovery of large near-cliques. *Distributed Computing*, 24(2):79–89, 2011.
- [CDP20] Artur Czumaj, Peter Davies, and Merav Parter. Simple, deterministic, constant-round coloring in the congested clique. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, page 309–318, 2020.
- [CFG<sup>+</sup>19] Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of  $(\Delta + 1)$  coloring in congested clique, massively parallel computation, and centralized local computation. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 471–480, 2019.
- [CHFSV19] Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. *Distributed Computing*, 32(1):41–57, 2019.
- [CLP20] Yi-Jun Chang, Wenzheng Li, and Seth Pettie. Distributed  $(\Delta + 1)$ -coloring via ultrafast graph shattering. *SIAM Journal on Computing*, 49(3):497–539, 2020.
- [Doe20] Benjamin Doerr. *Probabilistic Tools for the Analysis of Randomized Optimization Heuristics*, pages 1–87. Springer International Publishing, Cham, 2020.
- [DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

- [EPS15] Michael Elkin, Seth Pettie, and Hsin-Hao Su.  $(2\Delta - 1)$ -edge-coloring is much easier than maximal matching in the distributed setting. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 355–370, 2015.
- [FO19] Pierre Fraigniaud and Dennis Olivetti. Distributed detection of cycles. *ACM Transactions on Parallel Computing (TOPC)*, 6(3):1–20, 2019.
- [FRST16] Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. Distributed testing of excluded subgraphs. In *International Symposium on Distributed Computing*, pages 342–356. Springer, 2016.
- [GGR21] Mohsen Ghaffari, Christoph Grunau, and Václav Rozhoň. Improved deterministic network decomposition. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2021.
- [GK21] Mohsen Ghaffari and Fabian Kuhn. Deterministic distributed vertex coloring: Simpler, faster, and without network decomposition. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, 2021.
- [HKMN20] Magnús M. Halldórsson, Fabian Kuhn, Yannic Maus, and Alexandre Nolin. Coloring fast without learning your neighbors’ colors. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 39:1–39:17, 2020.
- [HKMT21] Magnús M. Halldórsson, Fabian Kuhn, Yannic Maus, and Tigran Tonoyan. Efficient randomized distributed coloring in CONGEST. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2021.
- [HKNT21] Magnús M. Halldórsson, Fabian Kuhn, Alexandre Nolin, and Tigran Tonoyan. Near-optimal distributed degree+1 coloring. *CoRR*, abs/2112.00604, 2021.
- [HN21] Magnús M. Halldórsson and Alexandre Nolin. Superfast coloring in CONGEST via efficient color sampling. In *Proceedings of the International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2021.
- [HNT21] Magnús M. Halldórsson, Alexandre Nolin, and Tigran Tonoyan. Ultrafast distributed coloring of high degree graphs. *CoRR*, abs/2105.04700, 2021.
- [Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
- [HSS18] David G. Harris, Johannes Schneider, and Hsin-Hao Su. Distributed  $(\Delta + 1)$ -coloring in sublogarithmic rounds. *Journal of the ACM*, 65:19:1–19:21, 2018.
- [Joh99] Öjvind Johansson. Simple distributed  $\Delta + 1$ -coloring of graphs. *Inf. Process. Lett.*, 70(5):229–232, 1999.
- [Lin92] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.
- [Lub86] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036–1053, 1986.
- [MR98] Michael Molloy and Bruce A. Reed. Colouring graphs whose chromatic number is almost their maximum degree. In *Proceedings of the Latin American Symposium on Theoretical Informatics (LATIN)*, volume 1380 of *LNCS*, pages 216–225, 1998.
- [New91] Ilan Newman. Private vs. common random bits in communication complexity. *Inf. Process. Lett.*, 39(2):67–71, 1991.

- [Ree98] Bruce A. Reed.  $\omega$ ,  $\Delta$ , and  $\chi$ . *J. Graph Theory*, 27(4):177–212, 1998.
- [SW10] Johannes Schneider and Roger Wattenhofer. A new technique for distributed symmetry breaking. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 257–266. ACM, 2010.
- [Vad12] Salil P. Vadhan. Pseudorandomness. *Found. Trends Theor. Comput. Sci.*, 7(1-3):1–336, 2012.

## A Concentration Bounds

**Lemma 7** (Chernoff bounds). *Let  $\{X_i\}_{i=1}^r$  be a family of independent binary random variables with  $\Pr[X_i = 1] = q_i$ , and let  $X = \sum_{i=1}^r X_i$ . For any  $\delta > 0$ ,  $\Pr[|X - \mathbb{E}[X]| \geq \delta \mathbb{E}[X]] \leq 2 \exp(-\min(\delta, \delta^2) \mathbb{E}[X]/3)$ .*

**Lemma 8** (Hoeffding’s inequality [Hoe63]). *Let  $X_1 \dots X_n$  be  $n$  independent random variables distributed in  $[a_i, b_i]$ ,  $X := \sum_{i=1}^n X_i$  their sum. For  $t > 0$ :*

$$\Pr[|X - \mathbb{E}[X]| > t] \leq 2 \exp\left(-\frac{2 \cdot t^2}{\sum_i (b_i - a_i)^2}\right).$$

We use the following variants of Chernoff bounds for dependent random variables. The first one is obtained, e.g., as a corollary of Lemma 1.8.7 and Thms. 1.10.1 and 1.10.5 in [Doe20].

**Lemma 9** (Martingales [Doe20]). *Let  $\{X_i\}_{i=1}^r$  be binary random variables, and  $X = \sum_i X_i$ . If  $\Pr[X_i = 1 \mid X_1 = x_1, \dots, X_{i-1} = x_{i-1}] \leq q_i \leq 1$ , for all  $i \in [r]$  and  $x_1, \dots, x_{i-1} \in \{0, 1\}$  with  $\Pr[X_1 = x_1, \dots, X_r = x_{i-1}] > 0$ , then for any  $\delta > 0$ ,*

$$\Pr[X \geq (1 + \delta) \sum_{i=1}^r q_i] \leq \exp\left(-\frac{\min(\delta, \delta^2)}{3} \sum_{i=1}^r q_i\right).$$

*If  $\Pr[X_i = 1 \mid X_1 = x_1, \dots, X_{i-1} = x_{i-1}] \geq q_i$ ,  $q_i \in (0, 1)$ , for all  $i \in [r]$  and  $x_1, \dots, x_{i-1} \in \{0, 1\}$  with  $\Pr[X_1 = x_1, \dots, X_r = x_{i-1}] > 0$ , then for any  $\delta \in [0, 1]$ ,*

$$\Pr[X \leq (1 - \delta) \sum_{i=1}^r q_i] \leq \exp\left(-\frac{\delta^2}{2} \sum_{i=1}^r q_i\right).$$

A function  $f(x_1, \dots, x_n)$  is  $c$ -Lipschitz iff changing any single  $x_i$  affects the value of  $f$  by at most  $c$ , and  $f$  is  $r$ -certifiable iff whenever  $f(x_1, \dots, x_n) \geq s$  for some value  $s$ , there exist  $r \cdot s$  inputs  $x_{i_1}, \dots, x_{i_{r \cdot s}}$  such that knowing the values of these inputs certifies  $f \geq s$  (i.e.,  $f \geq s$  whatever the values of  $x_i$  for  $i \notin \{i_1, \dots, i_{r \cdot s}\}$ ).

**Lemma 10** (Talagrand’s inequality [DP09]). *Let  $\{X_i\}_{i=1}^n$  be  $n$  independent random variables and  $f(X_1, \dots, X_n)$  be a  $c$ -Lipschitz  $r$ -certifiable function; then for  $t \geq 1$ ,*

$$\Pr[|f - \mathbb{E}[f]| > t + 30c\sqrt{r \cdot \mathbb{E}[f]}] \leq 4 \cdot \exp\left(-\frac{t^2}{8c^2 r \mathbb{E}[f]}\right).$$

**Lemma 11** (Lemma 24 in [HKNT21]). *Let  $\{X_i\}_{i=1}^n$  be  $n$  independent random variables. Let  $\{A_j\}_{j=1}^k$  and  $\{B_j\}_{j=1}^k$  be two families of events that are functions of the  $X_i$ ’s. Let  $f = \sum_{j \in [k]} \mathbb{I}_{A_j}$ ,  $g = \sum_{j \in [k]} \mathbb{I}_{A_j \cap \bar{B}_j}$ <sup>1</sup> and  $h = f - g$  be such that  $f$  and  $g$  are  $c$ -Lipschitz and  $r$ -certifiable w.r.t. the  $X_i$ ’s, and  $\mathbb{E}[h] \geq \alpha \mathbb{E}[f]$  for some constant  $\alpha \in (0, 1)$ . Let  $\delta \in (0, 1)$ . Then for  $\mathbb{E}[h]$  large enough:*

$$\Pr[|h - \mathbb{E}[h]| > \delta \mathbb{E}[h]] \leq \exp(-\Omega(\mathbb{E}[h])).$$

<sup>1</sup> $\mathbb{I}$  denotes the indicator random variable of an event.

## B Explicit Representative Multisets

For completeness, we give an explicit construction of representative multisets in this section.

Intuitively, a *sampler* for a domain of size  $M$  is a function that takes some number  $N$  of perfect random bits as input and outputs  $t$  elements  $z_1, \dots, z_t$  in  $[M]$ . A *hitting sampler* gives the guarantee that the sampled outputs hit any large enough subset of  $[M]$  with some probability, while an *averaging sampler* gives the guarantee that for any function  $f$  with output in  $[0, 1]$ , the average value of  $f$  on the sampled elements is close to the average value of  $f$  over the whole domain  $[M]$ .

**Definition 3** (Averaging Samplers). *A function  $\text{SAMP} : [N] \rightarrow [M]^t$  is a  $(\delta, \varepsilon)$ -averaging sampler if for every function  $f : [M] \rightarrow [0, 1]$ , we have:*

$$\Pr_{(z_i)_{i=1}^t \leftarrow \text{SAMP}(U_{[N]})} \left[ \frac{1}{t} \sum_{i=1}^t f(z_i) - \frac{1}{M} \sum_{x \in [M]} f(x) > \varepsilon \right] \leq \delta .$$

Averaging samplers are relevant to our setting in the following way: let a node  $v$  of palette  $\Psi_v$  in a color space  $\mathcal{C}$  of size  $|\mathcal{C}| \in [\Delta, 2\Delta]$  have slack at least  $4\varepsilon|\mathcal{C}|$  for some constant  $\varepsilon \in (0, 1)$ . Let its uncolored neighbors try a total of at most  $2\varepsilon|\mathcal{C}|$  colors in a given round. Let  $S$  be the set of colors in  $\Psi_v$  that is not tried by any of its neighbors, of size at least  $2\varepsilon|\mathcal{C}|$ . Let  $f$  be the indicator function for  $S$ . Then, if we use a  $(\delta, \varepsilon)$ -averaging sampler over  $\mathcal{C}$  to sample  $t$  elements, then with probability at least  $1 - \delta$ , at least  $\varepsilon t$  of the sampled elements are in  $S$ . This means that  $v$  trying any of the sampled colors succeeds with probability at least  $1 - \varepsilon$ , conditioned on an event of probability  $1 - \delta$ . If we furthermore assume that the palette of  $v$  has size comparable to its slack, e.g., at most  $8\varepsilon|\mathcal{C}|$ , and condition on the event that the random sampler does not over-sample elements from  $\Psi_v$  by more than  $\varepsilon$ , then having  $v$  try a random sampled color that is in its palette succeeds with constant  $\Omega(1)$  probability (i.e., independent of  $\varepsilon$ ).

Used that way, an averaging sampler replaces representative sets in most use cases. It may be interpreted as being a family of multisets, by considering the output of an averaging sampler on all possible choices of random bits. Taking  $\delta = 1/\text{poly}(n)$  and  $\varepsilon \in \Theta(1)$ , there exists explicit averaging samplers that use  $N = \Theta(\log n)$  random bits as input and sample  $t = \Theta(\log|\mathcal{C}| + \log n)$  elements.

## C Definitions Related to Almost-Clique Decompositions

**Definition 4** (Sparsity). *The (local) sparsity  $\zeta_v$  of node  $v$  is defined as  $\frac{1}{d_v} \cdot \left[ \binom{d_v}{2} - m(N(v)) \right]$ . Node  $v$  is  $\zeta$ -sparse if  $\zeta_v \geq \zeta$ , and  $\zeta$ -dense if  $\zeta_v \leq \zeta$ .*

**Definition 5** (Disparity, Discrepancy & Unevenness). *The disparity of  $u$  towards  $v$  is defined as  $\bar{\eta}_{u,v} = |\Psi_u \setminus \Psi_v| / |\Psi_u|$ . The discrepancy of node  $v$  is defined as  $\bar{\eta}_v = \sum_{u \in N(v)} \bar{\eta}_{u,v}$ , and its unevenness is defined as  $\eta_v = \sum_{u \in N(v)} \frac{\max(0, d_u - d_v)}{d_u + 1}$ . Node  $v$  is  $\bar{\eta}$ -discrepant if  $\bar{\eta}_v \geq \bar{\eta}$ ,  $\eta$ -uneven if  $\eta_v \geq \eta$ .*

**Definition 6** ( $(\deg + 1)$  ACD [AA20]). *Let  $G = (V, E)$  be a graph and  $\varepsilon_{\text{ac}}, \varepsilon_{\text{sp}} \in (0, 1)$  be parameters. A partition  $V = V^{\text{sparse}} \sqcup V^{\text{uneven}} \sqcup V^{\text{dense}}$  of  $V$ , with  $V^{\text{dense}}$  further partitioned into  $V^{\text{dense}} = \bigsqcup_{C \in \mathcal{S}_{\text{ac}}} C$ , is an almost-clique decomposition (ACD) for  $G$  if:*

1. Every  $v \in V^{\text{sparse}}$  is  $\varepsilon_{\text{sp}} d_v$ -sparse ,
2. Every  $v \in V^{\text{uneven}}$  is  $\varepsilon_{\text{sp}} d_v$ -uneven ,
3. For every  $C \in \mathcal{S}_{\text{ac}}$  and  $v \in C$ ,  $d_v \leq (1 + \varepsilon_{\text{ac}})|C|$  ,
4. For every  $C \in \mathcal{S}_{\text{ac}}$  and  $v \in C$ ,  $(1 + \varepsilon_{\text{ac}})|N_C(v)| \geq |C|$  .

The slackability of an almost-clique is defined as  $\bar{\sigma}_C = \min_{v \in C} \bar{\sigma}_v$ . As we deal with nodes of degree between  $\log^7 \Delta$  and  $\Delta$ , we set a threshold  $\ell = \log^{2.1} \Delta$  and declare almost-clique of lower slackability to be *low-slack*, and almost-clique of higher slackability to be *high-slack*.

## D Final Details of the Algorithm for (Degree+1)-List-Coloring in CONGEST

We give here the remaining details of the implementation of a  $(\deg + 1)$ -list-coloring algorithm in CONGEST. We explain how to find good enough “leaders” of almost-cliques in [Appendix D.1](#), and compute “put-aside” sets in [Appendix D.2](#), and finally how to deal with large color values in [Appendix D.3](#). We first give an informal description of the D1LC algorithm of [\[HKNT21\]](#).

**Algorithm overview.** The D1LC algorithm of [\[HKNT21\]](#) consists of up to  $O(\log^* n)$  phases in which all the nodes whose degree falls within a range of the form  $[\log^7 x, x]$  get colored, w.h.p. Each phase takes at most  $\text{poly}(\log \log n)$  rounds to complete, but a phase dealing only with nodes of degree  $\log^7 n$  or higher can achieve its task in merely  $O(\log^* n)$  rounds, resulting in an  $O(\log^* n)$  algorithm when given a graph of minimum degree  $\log^7 n$ .

Each phase starts with computing an almost-clique decomposition, which partitions the nodes in *uneven*, *sparse*, and *dense* vertices. Dense vertices are themselves partitioned into *almost-cliques*, highly connected (clique-like) subgraphs of diameter at most 2. See [Appendix C](#) for formal definitions of  $V^{\text{uneven}}$ ,  $V^{\text{sparse}}$ ,  $V^{\text{dense}}$ , and of almost-clique decompositions. Our primitive ESTIMATESIMILARITY can be directly used to compute such a decomposition, in a process which we was explained in [Sec. 4.2](#).

Within each phase, the algorithm first deals with all the sparse and uneven nodes, then all the dense nodes. In both cases, slack is generated by first having each node try a random color of its palette with some constant probability. The only possible difficulty in implementing this part in CONGEST is that the nodes may have to communicate colors whose description does not fit in  $O(\log n)$  bits. We explain how we can deal with a color space of size up to  $\exp(n^{\Theta(1)})$  in [Appendix D.3](#). From there, the algorithms differ between the dense and the non-dense case.

When dealing with sparse and uneven nodes, the algorithm identifies a set  $V^{\text{start}}$ , consisting of sparse nodes for which GENERATESLACK might not generate permanent slack but which can get temporary slack by being colored early. Instead of identifying this set before slack generation as in the LOCAL algorithm, which might be hard to do in CONGEST, we simply let the success of the slack generation process guide our partitioning. More precisely, we let each node  $v$  join  $V^{\text{start}}$  if it received less than  $\hat{\epsilon}d_v$  permanent slack but is adjacent to at least  $\hat{\epsilon}d_v$  uncolored nodes that did. A node that neither received permanent slack nor is adjacent to many nodes that did is added to a set BAD, which is either empty or shattered due to the probability of success of slack generation. Indeed, while this process may fail at providing the needed slack to some nodes, [Proposition 2](#) (from [\[HKNT21\]](#)) shows that this happens to a node with probability  $d^{-\omega(1)}$  when we consider nodes of degree in the range  $[\log^7 d, d]$ . This probability is low enough w.r.t.  $d$  that all nodes receive the slack they need w.h.p. when they are all of degree  $\log^7 n$  or more. It is also low enough w.r.t.  $d$  that when considering nodes of lower degree, the subgraph induced by nodes that do not receive the slack they need is *shattered*, i.e., has  $\text{poly}(\log n)$ -sized connected components which can be efficiently colored with by a deterministic algorithm later. Our process may add some extra nodes to  $V^{\text{start}}$  (in the unlikely event that a node supposed to get permanent slack does not obtain it) and remove some nodes from it (in the event that a node of  $V^{\text{start}}$  gets some permanent slack) compared to the original, fixed definition of  $V^{\text{start}}$  as a set of nodes adjacent to many nodes likely to get permanent slack, but it guarantees nonetheless what matters, that every node gets slack (temporary or permanent) with probability  $1 - d^{-\omega(1)}$ .

**Proposition 2.** [*Proposition 1 in [\[HKNT21\]](#)*] Assume all nodes have degree at least  $s \geq C \cdot \ln^2 \Delta$  for some universal constant  $C$ . There is an  $O(1)$ -round procedure that identifies a subset  $V^{\text{start}} \subseteq V^{\text{sparse}}$  such that after running GENERATESLACK in the subgraph induced by  $V^{\text{sparse}} \cup V^{\text{uneven}}$ :

1. Each node  $v$  in  $V^{\text{start}}$  has  $\Omega(d_v)$  uncolored neighbors in  $V \setminus V^{\text{start}}$  w.p.  $1 - \exp(-\Omega(d_v))$ , and
2. Each node  $v$  in  $V^{\text{uneven}} \cup V^{\text{sparse}} \setminus V^{\text{start}}$  has slack  $\Omega(d_v)$ , w.p.  $1 - \exp(-\Omega(\sqrt{s}))$ .

For each node, the probability bounds hold even when conditioned on arbitrary random choices outside its 2-hop neighborhood.

Sparse and uneven nodes then run a procedure SLACKCOLOR that, given nodes each with slack  $s_v \in \Omega(d_v)$  and  $s_v \geq s_{\min}$ , where  $s_{\min}$  is a lower bound of the slack of every participating nodes known by all of them, colors them in  $O(\log^* s_{\min})$  rounds with probability  $1 - \exp(-s_{\min}^{\Omega(1)}) - \Delta \exp(-\Omega(s_{\min}))$ . The procedure directly works in CONGEST if we can give a CONGEST implement of its main subroutine, MULTITRIAL, which was done in [Sec. 4.1](#). In LOCAL, this subroutine simply consists of each node trying  $x$  random colors from its palette, which, when the nodes have the needed slack, results in each node getting colored w.p.  $1 - \exp(-\Omega(x))$  (as if each color had an independent constant success probability). The CONGEST subroutine we give achieves essentially the same performance using representative hash functions, which allow us to (imperfectly) communicate  $x$  colors in less than the naïve  $x \log |\mathcal{C}|$  bits.

Dense nodes follow a more involved algorithm. In the original LOCAL algorithm, each almost-clique elects a leader according to a metric called slackability, which combines two measures: sparsity and discrepancy (a measure of how much one’s palette differs from those of one’s neighbors). To adapt this step to CONGEST, we instead elect the leader through a different process which leverages the relative uniformity of sparsity inside each almost-clique and the fact that discrepancy can be estimated by its contribution to a node’s slack. We give details of the leader selection process in [Appendix D.1](#). The almost-clique is then partitioned into inliers (direct neighbors of the leader, sharing many of its neighbors, and of not too high degree) and outliers (other nodes). This partitioning is easily done in CONGEST, as it only requires nodes to announce whether they are directly connected to the leader, count how many of their neighbors are direct neighbors of the leader, send this count and their degree to the leader, and let the leader pick its inliers according to the  $O(\log \Delta)$ -bit information it received from each of its in-clique neighbors.

Almost-cliques in which the leader has small slackability (below a threshold related to the degree range) compute so-called “put-aside” sets, which provide temporary slack for the remaining nodes of the almost-clique. These put-aside sets are sampled by a simple procedure: each inlier joins the put-aside set of its almost-clique according to a biased coin flip, and leaves it if one of its neighbors in another almost-clique also had a positive coin flip. This is readily implemented in CONGEST.

In addition to using SLACKCOLOR as previously with sparse and uneven nodes, dense nodes use a procedure SYNCHCOLORTRIAL in which the leader randomly distributes colors from its palette to uncolored inliers (not in the put-aside set). The only obstacle in CONGEST is possibly the size of colors, which is treated in [Appendix D.3](#). In the final step of the randomized part of the algorithm, the leader of each almost-clique with a put-aside set learns enough of the palettes of the put-aside elements to color them. Learning enough colors from each palette is done using other nodes of the almost-clique as relays, which we explain in [Appendix D.2](#).

Finally, a part of our randomized algorithm that deals with nodes of degree  $o(\log n)$  will likely fail at coloring some nodes. Such nodes are colored with a deterministic algorithm, following the standard shattering framework [[BEPS16](#)]. To deal with large colors in this last phase (the *post-shattering* phase), we compute a network decomposition and have each component compute a hash function without any collision in each node’s palette to reduce the space of colors. This allows us to apply a deterministic algorithm whose runtime depends on the size of the space of colors. We explain this in [Appendix D.3](#).

A full pseudocode description of the algorithm is available in [Appendix E](#) for completeness.

## D.1 Leader Selection

The original LOCAL algorithm takes as leader of an almost-clique  $C$  the node  $w := \arg \min_{v \in C} \bar{\sigma}_v$  of minimum slackability within  $C$ . In addition, it defines the slackability  $\bar{\sigma}_C$  of an almost-clique

$C$  as the slackability of this minimal node. Both selecting the leader properly and estimating its slackability accurately are important for the algorithm, as the leader has unique duties within the clique that not all nodes of the clique are fitted for, and cliques are assigned different behaviors depending on their slackability.

Computing the slackability of a node exactly would be too expensive in CONGEST, as would computing the slackabilities of all the nodes to find the node with the minimum value. The slackability might also be hard to estimate accurately in low-slackability almost-cliques (in very much the same way that our procedure ESTIMATESPARSITY does not give accurate results for nodes of sublinear sparsity).

Fortunately, taking as leader the node of minimum slackability is not necessary. What is necessary is selecting a leader whose slackability is of the same order of magnitude or less than the amount of slack nodes in the almost-clique later have when running SLACKCOLOR. This suffices as the number of nodes staying uncolored after SYNCHCOLORTRIAL is bounded by the slackability of the leader, in expectation. As nodes get slack from a different source depending on whether they are in a low- or high-slack almost-clique, what constitutes a good leader differs slightly between the two types of almost-cliques. It is also important that almost-cliques estimate their slackability sufficiently well so that a very low-slack almost-clique does not consider itself high-slack, or vice-versa.

In low-slack almost-cliques ( $\bar{\sigma}_C \leq \ell = \log^{2.1} \Delta$ ), it suffices that we select a leader of slackability  $O(\ell)$ . This leader might be significantly worse than the true leader  $w$  at coloring the almost-clique when running SYNCHCOLORTRIAL. However, it will bring down the number of uncolored nodes to  $O(\ell)$ , which is sufficient, as put-aside sets provide  $\Omega(\ell)$  slack in low-slack almost-cliques. Second, in almost-cliques of higher slack, it suffices that the leader we pick has slackability of order  $O(\bar{\sigma}_C)$  instead of exactly  $\bar{\sigma}_C$ . As in the low-slack case, the leader will be worst at coloring the almost-clique during SYNCHCOLORTRIAL, but a leader of slackability  $O(\bar{\sigma}_C)$  preserves that SYNCHCOLORTRIAL likely colors all but  $O(\bar{\sigma}_C)$  nodes of the almost-clique, which guarantees that the nodes have slack linear in their uncolored degree.

We show that good-enough leader can be selected through a combination of three metrics: anti-degree, external-degree, and a quantity we call *chromatic slack*.

**Definition 7** (Chromatic slack). *Let  $v$  be a dense node in an almost-clique  $C$ . Its (in-clique) chromatic slack  $\kappa_v$  is defined as the number of  $v$ 's neighbors that adopted a permanent color outside of  $v$ 's original palette  $\Psi_v$  during GENERATESLACK.*

Our method for selecting a good-enough leader is summarized in [Lemma 12](#).

**Lemma 12.** *For an almost-clique  $C$ , let it pick as leader the node  $x$ :  $x = \arg \min_{v \in C} (e_v + a_v + \kappa_v)$ . Then:*

- *If  $C$  is high-slack,  $x$  has slackability  $\bar{\sigma}_x \in O(\bar{\sigma}_C)$ , w.p.  $1 - \exp(-\Omega(\bar{\sigma}_C))$ .*
- *If  $C$  is low-slack,  $x$  has slackability  $\bar{\sigma}_x \in O(\ell)$ , w.p.  $1 - \exp(-\Omega(\ell))$ .*

We prove [Lemma 12](#) through the combination of two structural results from previous works ([Lemmas 13](#) and [14](#)) and a statement on the distribution of chromatic slack.

**Lemma 13** (Lemma 2 in [[HKNT21](#)]). *There is a constant  $c_e = c_e(\varepsilon_{ac})$  such that  $e_v \leq c_e \cdot \sigma_v$  holds for every node  $v$  in an almost-clique  $C$ .*

**Lemma 14** (Lemma 3 in [[HKNT21](#)]). *There is a constant  $c_a = c_a(\varepsilon_{ac})$  such that  $a_v \leq c_a \cdot \sigma_v$  holds for any dense node  $v$ .*

Let the in-clique discrepancy of a node  $v$  be defined as  $\bar{\eta}_v^C := \sum_{u \in N_C(v)} \bar{\eta}_{u,v} = \sum_{u \in N_C(v)} |\Psi_u \setminus \Psi_v| / |\Psi_u|$ .



**Lemma 15.** Consider a node  $v \in C$  and  $\mu_L, \mu_H$  such that  $\mu_L \leq \bar{\eta}_v^C \leq \mu_H$ . During GENERATESLACK,  $v$  gets chromatic slack  $\kappa_v \in O(\mu_H)$  w.p.  $1 - \exp(-\mu_H)$ , and  $\kappa_v \in \Omega(\mu_L)$  w.p.  $1 - \exp(-\mu_L)$ .

*Proof.* Let  $p_g$  be the (constant) probability that a node tries a color during GENERATESLACK. Let us define the random variables  $X_u, Y_u, Z_\psi$  and their sums  $X, Y, Z$  as follows:

- For each  $u \in N_C(v)$ ,  $X_u$  corresponds to the event that  $u$  tries a color outside  $v$ 's palette.
- For each  $u \in N_C(v)$ ,  $Y_u$  is the same event as  $X_u$ , with the addition that  $u$  gets to keep the color it tries as permanent color.
- For each  $\psi \in \mathcal{C} \setminus \Psi_v$ ,  $Z_\psi$  is the event that a unique node in  $N_C(v)$  tries  $\psi$  and keeps it as permanent color.
- $X = \sum_{u \in N_C(v)} X_u$ ,  $Y = \sum_{u \in N_C(v)} Y_u$ , and  $Z = \sum_{u \in \mathcal{C} \setminus \Psi_v} Z_u$ .

The upper bound follows directly from  $\kappa_v = Y$  and  $\mathbb{E}[Y] \leq \mathbb{E}[X] = p_g \bar{\eta}_v^C \leq p_g \mu_H$ . Since the events  $X_u$  are all independent, by Chernoff,  $\kappa_v \in O(\mu_H)$  w.p.  $1 - \exp(-\Omega(\mu_H))$ .

For the lower bound, we relate the chromatic slack to  $Z$ . First  $\kappa_v = Y \geq Z$  simply by definition, and  $\mathbb{E}[Z] \in \Omega(\mathbb{E}[X])$  as each color try has an  $\Omega(1)$  chance of being successful and unique within  $C$ . Since  $\mathbb{E}[X] = p_g \bar{\eta}_v^C \geq p_g \mu_L$ ,  $\mathbb{E}[Z] \in \Omega(\mu_L)$ . By Lemma 11, as  $Z$  is the difference of two  $O(1)$ -Lipschitz and  $O(1)$ -certifiable random quantities,  $Z \in \Omega(\mathbb{E}[Z])$  w.p.  $1 - \exp(-\Omega(\mathbb{E}[Z]))$ . Hence,  $\kappa_v \in \Omega(\mu_L)$  w.p.  $1 - \exp(-\Omega(\mu_L))$ .  $\square$

*Proof of Lemma 12.* By definition,  $e_x + a_x + \kappa_x \leq e_w + a_w + \kappa_w$ , where  $w$  is the node of minimum slackability within  $C$ . Lemmas 13 and 14 imply  $(e_w + a_w) \in O(\bar{\sigma}_C)$ . By Lemma 15, since  $\bar{\eta}_w^C \leq \bar{\sigma}_w = \bar{\sigma}_C$  by definition,  $\kappa_w \in O(\ell)$  w.p.  $1 - \exp(-\Omega(\ell))$  when  $C$  is low-slack ( $\bar{\sigma}_C \leq \ell$ ), and  $\kappa_w \in O(\bar{\sigma}_C)$  w.p.  $1 - \exp(-\Omega(\bar{\sigma}_C))$  when  $C$  is high-slack. This implies overall that for the leader  $x$  selected:

- $(e_x + a_x + \kappa_x) \in O(\ell)$  w.p.  $1 - \exp(-\Omega(\ell))$  if  $C$  is low-slack,
- $(e_x + a_x + \kappa_x) \in O(\bar{\sigma}_C)$  w.p.  $1 - \exp(-\Omega(\bar{\sigma}_C))$  if  $C$  is high-slack.

From now on, let us focus on the high-slack case, the low-slack case being similar. We have  $(e_x + a_x + e_w + a_w) \in O(\bar{\sigma}_C)$ , which means that  $x$ 's and  $w$ 's neighborhood may not significantly differ, i.e.,  $|N(x) \Delta N(w)| \in O(\bar{\sigma}_C)$ . Their sparsities therefore only differ by  $O(\bar{\sigma}_C)$ . Their discrepancies also only differ by  $O(\bar{\sigma}_C)$ , as  $\kappa_x \in O(\bar{\sigma}_C)$  means that  $\bar{\eta}_x^C \in O(\bar{\sigma}_C)$ , as a larger in-clique discrepancy would have likely resulted in much higher chromatic slack by Lemma 15, and other differences in discrepancy must come from difference in neighborhoods, which we have shown to be bounded by  $O(\bar{\sigma}_C)$ .  $\square$

It only remains for the almost-clique to estimate its slackability with enough accuracy to categorize itself as either high- or low-slack. The aggregate  $(e_x + a_x + \kappa_x)$  we used to pick a leader gives us some idea of the slackability of the almost-clique, in that it is upper-bounded by  $O(\max(\ell, \bar{\sigma}_C))$  as we have seen in the proof of Lemma 12. However, it does not measure in-clique sparsity. For instance, the aggregate could even be 0 in a high-slack almost-clique  $C$ : it suffices that  $C$ 's slackability is mostly due to sparsity, and that it contains a node  $v \in C$  that is connected to all other nodes of  $C$ , has no external neighbor, and has a palette containing the palettes of other nodes in  $C$ .

To estimate the sparsity of the almost-clique, we approximate the sparsity of the leader  $x$  by counting the number of edges in its in-clique neighborhood. This is easily done in CONGEST by having each node tell their neighbors whether they are adjacent to  $x$ , and having each neighbor of  $x$  count and transmit to  $x$  to how many neighbors of  $x$  it is connected.

**Lemma 16.** Let  $\hat{m} = m(N_C(x)) = \frac{1}{2} \sum_{u \in N_C(x)} |N(u) \cap N_C(x)|$  count the number of edges in  $x$ 's in-clique neighborhood. Then  $\hat{\zeta}_x = \frac{1}{d_x} \binom{d_x}{2} - \hat{m}$  satisfies  $\hat{\zeta}_x \in [\zeta_x, \zeta_x + e_x]$ , and so  $(e_x + \hat{\zeta}_x + \kappa_x) \in \Omega(\bar{\sigma}_C)$ , w.p.  $1 - \exp(-\Omega(\bar{\sigma}_C))$ .

*Proof.* The sparsity of  $x$  corresponds to the number of missing edges in its neighborhood, divided by  $d_x$ . As  $\hat{m}$  only counts edges in  $x$ 's in-clique neighborhood, it undercounts the number of edges in  $N(x)$  by  $e_x \cdot d_x$  or less, i.e.,  $\hat{m} \in [m(N(x)) - e_x \cdot d_x, m(N(x))]$ . In turn, this means that the estimate  $\hat{\zeta}_x$  of  $x$ 's sparsity is in the range  $[\zeta_x, \zeta_x + e_x]$ .

The discrepancy of  $x$  is the sum of the contribution of its in-clique neighbors and that of its external neighbors, that is,  $\bar{\eta}_x \in [\bar{\eta}_x^C, \bar{\eta}_x^C + e_x]$ .

The slackability of  $x$  is defined as  $\bar{\sigma}_x = \zeta_x + \bar{\eta}_x$ . If  $\zeta_x \geq \bar{\sigma}_x/3$  or  $e_x \geq \bar{\sigma}_x/3$ , then  $(e_x + \hat{\zeta}_x + \kappa_x) \in \Omega(\bar{\sigma}_x)$  trivially. Otherwise,  $\bar{\eta}_x^C \geq \bar{\eta}_x - e_x \geq \bar{\sigma}_x/3$ , which implies by [Lemma 15](#) that  $\kappa_x \in \Omega(\bar{\sigma}_x)$  w.p.  $1 - \exp(-\Omega(\bar{\sigma}_x))$ . The statement can be reformulated with  $\bar{\sigma}_C$  instead of  $\bar{\sigma}_x$  as  $\bar{\sigma}_x \geq \bar{\sigma}_C$  by definition of  $\bar{\sigma}_C$ .  $\square$

Putting everything together, we get that  $(e_x + \hat{\zeta}_x + \kappa_x) \in \Theta(\bar{\sigma}_C)$  w.p.  $1 - \exp(-\Omega(\bar{\sigma}_C))$  in high slack almost-cliques, and that  $(e_x + \hat{\zeta}_x + \kappa_x) \in O(\ell)$  w.p.  $1 - \exp(-\Omega(\ell))$  in low-slack almost-cliques, which is sufficient for our purposes.

## D.2 Coloring the Put-Aside Sets

The coloring of the put-aside sets  $P_C$  is the only step of the algorithm for dense nodes ([Alg. 9](#)) that remains to be explained. In this step of the algorithm, the nodes in each put-aside set  $P_C$  transmit the content of their palettes and the topology of  $G[P_C]$  to their leader  $x_C$ . Provided with this information, the leader can then assign each node of  $P_C$  a color from its individual palette that does not conflict with the color of its neighbors. Without any adjustment, this process has each node from a put-aside  $P_C$  send  $\Theta(|P_C|(\log |\mathcal{C}| + \log n))$  bits to its leader, which on a single communication link would be too costly in CONGEST. We adapt this part of the algorithm to CONGEST through three avenues. First, we have nodes send colors to their leader by using a hash function chosen by said leader. This solves the bandwidth requirements that sending very large colors presents, as explained in [Appendix D.3](#). Second, we reduce the size of the put-aside sets to just the size that is needed to get sufficient slack. Finally, we use other nodes of the almost-clique as relays to increase the bandwidth between the leader and each node of the put-aside.

The leader restricts the size of  $P_C$  to  $\Theta(\ell)$ , which is a sufficient amount of slack for the parts of the algorithm that rely on slack from the put-aside sets (invoking SLACKCOLOR). Recall that  $I_C \subseteq N_C(x_C)$ . The leader enumerates the nodes in  $I_C$  and allocates each node  $v \in P_C$  a contiguous interval of  $2|P_C| + 1$  indices, corresponding to a set  $R_v$  of nodes. Since  $|I_C| \geq 2|P_C|^2 + |P_C|$ , the nodes receive disjoint intervals. Each node  $v \in P_C$  has  $a_v = O(\sigma_C) = O(\ell) \leq |P_C|$  non-neighbors in  $C$ , and hence it has at least  $|R_v| - a_v \geq |P_C|$  neighbors in  $R_v$ . Now  $v$  can send  $|N(v) \cap P_C| + 1$  colors from its palette to  $x_C$  in  $O(1)$  rounds, via the relay nodes in  $N(v) \cap R_v$ . The topology of  $P_C$  can similarly be transmitted. The leader can then properly color  $P_C$  locally and forward the colors to the nodes.

## D.3 Large Colors

We have implicitly assumed until now that sending a color over an edge, as nodes do when broadcasting their permanent color to their neighbors, only takes  $O(1)$  rounds. This is possible if the color space  $\mathcal{C}$  is of size  $|\mathcal{C}| \in n^{O(1)}$ . In [Lemma 1](#), the dependency of  $t$  in  $|\mathcal{C}|$  is only  $\log |\mathcal{C}|$ , meaning that sending a representative hash function still takes only  $O(1)$  rounds even when  $|\mathcal{C}| \in \exp(n^{\Theta(1)})$ . Can we tolerate such a large color space in other parts of the algorithm? We resolve this in the affirmative.

**Pre-shattering phase.** For all parts of the algorithm except the post-shattering phase, we achieve this using a family  $\mathcal{H}$  of  $1 + \varepsilon$ -approximately universal hash functions, i.e., a set of hash functions  $h : [N] \rightarrow [M]$  such that for all  $x_1 \neq x_2$ ,  $\Pr_{h \leftarrow \mathcal{H}}[h(x_1) = h(x_2)] \leq (1 + \varepsilon)/M$ . There

exists small enough families of such hash functions so that specifying an element in the family only takes  $O(\log \log N + \log M + \log(1/\varepsilon))$  bits ([BJKS93], or Problem 3.4 in [Vad12]). Set  $\varepsilon = 1$  and let us hash to  $M = \Theta(n^d)$  values, where  $d \in \Theta(1)$ . Under these assumptions, sending a hash value only takes  $O(1)$  rounds, and sending an element of  $\mathcal{H}$  takes  $O(\lceil \log \log \mathcal{C} / \log n \rceil)$  rounds – in particular,  $O(1)$  if colors are written on  $\text{poly}(n)$  bits. Let each node  $v$  pick and broadcast a random  $1 + \varepsilon$ -approximately universal hash function  $h_v$  from  $\mathcal{H}$  at the start of our algorithms. Whenever a node  $u$  was previously sending a color  $\psi$  to a node  $v$  in our algorithms, we now have  $u$  send  $h_v(\psi)$  to  $v$ . Granted no collision occurs in any neighborhood, these hash values perfectly replace the actual colors wherever nodes were previously using the exact colors of their neighbors, such as when updating their palettes, computing their chromatic slack, and when a leader in an almost-clique sends colors to the inliers – each inlier looks for a color that hashes to the hash sent by the leader, and then tries that color by hashing it using its neighbors’ hash functions.

With  $\log n$  bandwidth, we ensure no collision occurs in any neighborhood w.h.p., by taking  $d$  appropriately large. Consider a node  $v$  and its neighborhood. There are at most  $(\Delta + 1)^2$  distinct colors in the palettes of  $v \cup N(v)$ . The probability that a collision occurs in these colors with a random hash function from  $\mathcal{H}$  is bounded by  $\binom{(\Delta+1)^2}{2} \cdot n^{-d} \leq n^{-d+4}$ . So, w.p. at least  $1 - n^{-d+5}$ , there are no collisions in all neighborhoods. Setting  $d \geq 6$ , this holds w.h.p.

**Post-shattering phase.** For the post-shattering phase, unlike in the LOCAL model, in general we may not directly use one of the recent deterministic algorithms of Ghaffari and Kuhn [GK21], as the complexity of their algorithm in CONGEST depends on the size of the color space  $\mathcal{C}$ . Indeed, their two CONGEST algorithms use either  $\log^2 \Delta \log n$  rounds of  $\Delta \log |\mathcal{C}|$  bits or  $\log^2 |\mathcal{C}| \log n$  rounds of  $\log |\mathcal{C}|$  bits (note that  $n$  and  $\Delta$  are the parameters of the shattered connected component, but  $\mathcal{C}$  is the original color space). When  $|\mathcal{C}| \in \text{poly}(\log n)$  (and therefore, all the degrees of the graph as well) we get an  $O(\log^3 \log n)$  algorithm, but handling a larger color space requires additional work.

We handle larger color spaces by computing a network decomposition on the shattered graph in  $O(\log^5 \log n)$  rounds [GGR21], and coloring each cluster of each color class by first computing a color space reduction before using the deterministic algorithm of [GK21]. The color space reduction simply consists of finding a function that maps each color from  $\mathcal{C}$  to a  $\text{poly}(\log n)$  number such that no collision occurs in any node’s palette. This is achieved through derandomizing the random selection of such a function with the method of conditional expectation.

**Lemma 17** (Lemma 3.19 in [HKMN20] (full version)). *Let  $N \in \log^{O(1)} n$  be the size of the subgraph on which we compute a network decomposition of diameter  $D$ . Consider one cluster  $C$  of the network decomposition and let  $\Psi(u)$  be the palette of vertex  $u \in C$  of size  $L \leq N$ . There is a deterministic  $D \cdot \log N$  round algorithm that computes a colorspace reduction  $f : \mathcal{C} \rightarrow N^{10}$  such that  $|f(\Psi(u))| = |\Psi(u)|$  for all  $u \in C$ . The colorspace reduction  $f$  can be described with  $O(\log \log n)$  bits.*

## E Full Statement of (Degree+1)-List-Coloring Algorithm

### E.1 Broad Structure

As explained in the algorithm overview in Appendix D, nodes are dealt with in degree ranges of the form  $[\log^7 x, x]$ . The full algorithm simply consists of  $O(\log^* n)$  call to Alg. 7, which assumes the degrees of nodes in the graph to be within such a degree range.

---

**Algorithm 7** Randomized D1LC Algorithm ( $\forall v, d_v \in [\log^7 \Delta, \Delta]$ )

---

- 1: COMPUTEACD.
  - 2: Apply Alg. 8 to sparse nodes.
  - 3: Apply Alg. 9 to dense nodes.
- 

The algorithm for a given degree range (Alg. 7) itself calls two procedures: one that colors the sparse nodes (Alg. 8), and one that colors the dense nodes (Alg. 9). Before that, it computes an almost-clique decomposition, which we explained how to do in CONGEST in Sec. 4.2.

---

**Algorithm 8** Main Procedure for Coloring Sparse Nodes

---

- 1: Identify the set  $V^{\text{start}} \subset V^{\text{sparse}}$
  - 2: GENERATESLACK in  $G[V^{\text{sparse}} \cup V^{\text{uneven}}]$ .
  - 3: SLACKCOLOR  $V^{\text{start}}$ .
  - 4: SLACKCOLOR  $V^{\text{sparse}} \setminus V^{\text{start}}$  and  $V^{\text{uneven}}$ .
- 

---

**Algorithm 9** Main Procedure for Coloring Dense Nodes

---

- 1: Compute the leader  $x_C$  and outliers  $O_C$  of each almost-clique  $C$ . Let  $O = \cup_C O_C$ .
  - 2: GENERATESLACK.
  - 3:  $P_C \leftarrow \text{PUTASIDE}(C)$  in each low-slack almost-clique  $C$ . Let  $P = \cup_C P_C$ .
  - 4: SLACKCOLOR  $O$ .
  - 5: SYNCHCOLORTRIAL  $V^{\text{dense}} \setminus P$ .
  - 6: SLACKCOLOR  $V^{\text{dense}} \setminus P$ .
  - 7: For each low-slack  $C$ , let  $x_C$  collect the palettes in  $P_C$  and color the nodes locally.
- 

Approaching the D1LC problem by giving two separate algorithms for sparse and dense nodes is natural. Indeed, splitting the problem in that manner results in two D1LC instances of similar degree ranges (more generally, the problem is self-reducible), and the all-sparse and all-dense cases are possible inputs that need to be considered anyway.

## E.2 Subroutines

We now detail the subroutines referred to in Alg. 8 and 9 above.

**Trying colors and slack generation.** GENERATESLACK simply consists of each node trying a random color in its palette, with some constant probability. What is trying a color formally means is described in Alg. 12, in particular what it means to try a color when some nodes have priority over other nodes. How trying a color can be done in CONGEST even when the color space is of order  $\exp(n^{\Theta(1)})$  is explained in Appendix D.3.

---

**Algorithm 10** GENERATESLACK(probability  $p_g$ )

---

- 1:  $S \leftarrow$  sample each  $v \in G$  into  $S$  independently w.p.  $p_g = 1/10$ .
  - 2: **for all**  $v \in S$  **in parallel do** TRYRANDOMCOLOR( $v$ ).
- 

---

**Algorithm 11** TRYRANDOMCOLOR (vertex  $v$ )

---

- 1: Pick  $\psi_v$  u.a.r. from  $\Psi_v$ .
  - 2: TRYCOLOR( $v, \psi_v$ )
-

A more refined version gives priority to some nodes over others: for each node  $v$ , we partition its neighborhood  $N(v)$  into  $N^+(v)$  – the nodes whose colors conflict with  $v$ 's – and  $N^-(v) = N(v) \setminus N^+(v)$ . For correctness of TRYCOLOR,  $u \in N^-(v) \rightarrow v \in N^+(u)$  should hold for each edge  $uv$ .

---

**Algorithm 12** TRYCOLOR (vertex  $v$ , color  $\psi_v$ )

---

- 1: Send  $\psi_v$  to  $N(v)$ , receive the set  $T^+ = \{\psi_u : u \in N^+(v)\}$ .
  - 2: **if**  $\psi_v \notin T^+$  **then** permanently color  $v$  with  $\psi_v$ .
  - 3: Send/receive permanent colors, and remove the received ones from  $\Psi(v)$ .
- 

**Leader, inliers, and outliers of an almost-clique.** Once the leader  $x$  of an almost-clique  $C$  is chosen, the *outliers*  $O_C$  of this almost-clique are chosen to be:

1. the  $\max(d_x, |C|)/3$  nodes in  $C$  with the fewest common neighbors with  $x$ ,
2. the  $|C|/6$  nodes of largest (original) degree, and
3. the anti-neighbors  $A_x$  of  $x$ .

The *inliers*  $I_C$  are the rest of the almost-clique,  $I_C = C \setminus O_C$ . Recall that  $\ell = \log^{2.1} \Delta$ . How the leader selection process is adapted to work in CONGEST is explained in [Appendix D.1](#).

**Put-aside sets.** The construction of put-aside sets is a simple random sample ([Alg. 13](#)). How these sets are colored at the end of the procedure for dense nodes in CONGEST is explained in [Appendix D.2](#).

---

**Algorithm 13** PUTASIDE( $C$ )

---

- 1:  $S_C \leftarrow$  each node  $v \in I_C$  is sampled independently w.p.  $p_s = \ell^2/(48\Delta_C)$ .
  - 2: **return**  $P_C \leftarrow \{v \in S_C : E_v \cap S = \emptyset\}$ , where  $S = \cup_{C'} S_{C'}$
- 

**Synchronized color trials within almost-cliques.** An important subroutine of the algorithm for dense nodes is SYNCHCOLORTRIAL, in which the leader of each almost-clique  $C$  randomly gives a unique color from its palette to the uncolored non-put-aside inliers of  $C$ . The only possible issue in CONGEST is that the colors may be too large to send efficiently. How to overcome this hurdle is explained in [Appendix D.3](#).

---

**Algorithm 14** SYNCHCOLORTRIAL, for almost-clique  $C$

---

- 1:  $x_C$  randomly permutes its palette  $\Psi(x_C)$ , sends each neighbor  $u \in I_C$  a distinct color  $\psi_u$ .
  - 2: Each  $u \in I_C$  calls TRYCOLOR( $u, \psi_u$ ) if  $\psi_u \in \Psi(u)$
- 

**Coloring with slack.** Finally, we give the pseudocode for SLACKCOLOR, an important subroutine in all randomized algorithms that achieve complexity  $O(\log^* n)$  for graphs of large enough degree without increasing the number of colors polynomially as in Linial's algorithm. This subroutine has nodes with slack linear in their degree try increasing numbers of colors through  $O(\log^* n)$  iterations. How to implement in CONGEST its main building block, MULTITRIAL, is explained in [Sec. 4.1](#). The nodes can also readily compute their slack in CONGEST with the techniques for handling large colors described in [Appendix D.3](#), making the whole procedure implementable in CONGEST.

$\kappa \in (1/s_{\min}, 1]$  is a parameter,  $a \uparrow\uparrow b$  denotes tetration ( $a \uparrow\uparrow 0 = 1$ ,  $a \uparrow\uparrow (b + 1) = a^{a \uparrow\uparrow b}$ ).

---

**Algorithm 15** SLACKCOLOR( $s_{\min}$ ), for node  $v$ 

---

- 1: **for**  $O(1)$  rounds **do** TRYRANDOMCOLOR( $v$ ).
- 2: **if**  $s(v) < 2d(v)$  **then** terminate.
- 3: Let  $\rho \leftarrow s_{\min}^{1/(1+\kappa)}$
- 4: **for**  $i$  from 0 to  $\log^* \rho$  **do**
- 5:    $x_i \leftarrow 2 \uparrow \uparrow i$
- 6:   MULTITRIAL( $x_i$ ) 2 times.
- 7:   **if**  $d(v) > s(v) / \min(2^{x_i}, \rho^\kappa)$  **then** terminate.
- 8: **end for**
- 9: **for**  $i$  from 1 to  $\lceil 1/\kappa \rceil$  **do**
- 10:    $x_i \leftarrow \rho^{i \cdot \kappa}$
- 11:   MULTITRIAL( $x_i$ ) 3 times.
- 12:   **if**  $d(v) > s(v) / \min(\rho^{(i+1) \cdot \kappa}, \rho)$  **then** terminate.
- 13: **end for**
- 14: MULTITRIAL( $\rho$ ).

---