The Complexity of $(\Delta + 1)$ Coloring in Congested Clique, Massively Parallel Computation, and Centralized Local Computation*

Yi-Jun Chang†
University of Michigan
cyijun@umich.edu

Manuela Fischer
ETH Zurich
manuela.fischer@inf.ethz.ch

Mohsen Ghaffari
ETH Zurich
ghaffari@inf.ethz.ch

Jara Uitto
ETH Zurich & University of Freiburg
jara.uitto@inf.ethz.ch

Yufan Zheng
University of Michigan
lwins.lights@gmail.com

ABSTRACT

In this paper, we present new randomized algorithms that improve the complexity of the classic $(\Delta + 1)$-coloring problem, and its generalization $(\Delta + 1)$-list-coloring, in three well-studied models of distributed, parallel, and centralized computation:

Distributed Congested Clique: We present an $O(1)$-round randomized algorithm for $(\Delta + 1)$-list-coloring in the congested clique model of distributed computing. This settles the asymptotic complexity of this problem. It moreover improves upon the $O(\log^* \Delta)$-round randomized algorithms of Parter and Su [DISC'18] and $O((\log\log\Delta)\cdot\log^*\Delta)$-round randomized algorithm of Parter [ICALP'18].

Massively Parallel Computation: We present a randomized $(\Delta + 1)$-list-coloring algorithm with round complexity $O(\sqrt{\log\log n})$ in the Massively Parallel Computation (MPC) model with strongly sublinear memory per machine. This algorithm uses a memory of $O(n^\alpha)$ per machine, for any desirable constant $\alpha > 0$, and a total memory of $\widetilde{O}(m)$, where m is the number of edges in the graph. Notably, this is the first coloring algorithm with sublogarithmic round complexity, in the sublinear memory regime of MPC. For the quasilinear memory regime of MPC, an $O(1)$-round algorithm was given very recently by Assadi et al. [SODA'19].

Centralized Local Computation: We show that $(\Delta + 1)$-list-coloring can be solved by a randomized algorithm with query complexity $\Delta^{O(1)} \cdot O(\log n)$, in the centralized local computation model. The previous state of the art for $(\Delta + 1)$-list-coloring in the centralized local computation model are based on simulation of known LOCAL algorithms. The deterministic $O(\sqrt{\Delta}\text{poly}\log\Delta + \log^* n)$-round LOCAL algorithm of Fraigniaud et al. [FOCS'16] can be implemented in the centralized local computation model with query complexity $\Delta^{O(\sqrt{\Delta}\text{poly}\log\Delta)} \cdot O(\log^* n)$; the randomized $O(\log^*\Delta) + 2^{O(\sqrt{\log\log n})}$-round LOCAL algorithm of Chang et al. [STOC'18] can be implemented in the centralized local computation model with query complexity $\Delta^{O(\log^*\Delta)} \cdot O(\log n)$.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**.

KEYWORDS

coloring, congested clique, centralized local computation, massively parallel computation

ACM Reference Format:
Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. 2019. The Complexity of $(\Delta + 1)$ Coloring in Congested Clique, Massively Parallel Computation, and Centralized Local Computation. In *2019 ACM Symposium on Principles of Distributed Computing (PODC '19), July 29-August 2, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3293611.3331607

1 INTRODUCTION

In this paper, we present improved randomized algorithms for vertex coloring in three models of distributed, parallel, and centralized computation: the *congested clique* model of distributed computing, the *massively parallel computation* model, and the *centralized local computation* model. We next overview these results in three different subsections, while putting them in the context of the state of the art. The next section provides a technical overview of the known algorithmic tools as well as the novel ingredients that lead to our results.

$(\Delta + 1)$-coloring and $(\Delta + 1)$-list Coloring. Our focus is on the standard $(\Delta + 1)$ vertex coloring problem, where Δ denotes the maximum degree in the graph. All our results work for the generalization of the problem to $(\Delta + 1)$-list coloring problem, defined as follows: each vertex v in the graph $G = (V, E)$ is initially equipped with a set of colors $\Psi(v)$ such that $|\Psi(v)| = \Delta + 1$. The goal is to

*The full version of the paper is available at arXiv [20].
†Supported by NSF grants CCF-1514383, CCF-1637546, and CCF-1815316.

PODC '19, July 29-August 2, 2019, Toronto, ON, Canada

ACM ISBN 978-1-4503-6217-7/19/07…$15.00
https://doi.org/10.1145/3293611.3331607

find a proper vertex coloring where each vertex $v \in V$ is assigned a color in $\Psi(v)$ such that no two adjacent vertices are colored the same.

1.1 Congested Clique Model of Distributed Computing

Models of Distributed Computation. There are three major models for distributed graph algorithms, namely LOCAL, CONGEST, and CONGESTED-CLIQUE. In the LOCAL model [48, 56], the input graph $G = (V, E)$ is identical to the communication network and each $v \in V$ hosts a processor that initially knows $\deg(v)$, a unique $\Theta(\log n)$-bit ID(v), and global graph parameters $n = |V|$ and $\Delta = \max_{v \in V} \deg(v)$. Each processor is allowed unbounded computation and has access to a stream of private random bits. *Time* is partitioned into synchronized *rounds* of communication, in which each processor sends one unbounded message to each neighbor. At the end of the algorithm, each v declares its output label, e.g., its own color. The CONGEST model [56] is a variant of LOCAL where there is an $O(\log n)$-bit message size constraint. The CONGESTED-CLIQUE model, introduced in [49], is a variant of CONGEST that allows all-to-all communication: Each vertex initially knows its adjacent edges of the input graph $G = (V, E)$. In each round, each vertex is allowed to transmit $n - 1$ many $O(\log n)$-bit messages, one addressed to each other vertex.

In this paper, our new distributed result is an improvement for coloring in CONGESTED-CLIQUE. It is worth noting that the CONGESTED-CLIQUE model has been receiving extensive attention recently, see e.g., [11, 14, 15, 18, 19, 24, 25, 28–31, 36, 37, 39, 45, 50, 53–55].

State of the Art for Coloring in LOCAL *and* CONGEST. Most prior works on distributed coloring focus on the LOCAL model. The current state-of-the-art randomized upper bound for the $(\Delta + 1)$-list coloring problem is $O(\log^* \Delta) + O(\text{Det}_d(\text{poly} \log n)) = O(\text{Det}_d(\text{poly} \log n))$ of [21] (which builds upon the techniques of [34]), where $\text{Det}_d(n') = 2^{O(\sqrt{\log \log n'})}$ is the deterministic complexity of (deg +1)-list coloring on n'-vertex graphs [51]. In the (deg +1)-list coloring problem, each v has a palette of size $\deg(v)+1$. This algorithm follows the *graph shattering* framework [10, 29]. The pre-shattering phase takes $O(\log^* \Delta)$ rounds. After that, the remaining uncolored vertices form connected components of size $O(\text{poly} \log n)$. The post-shattering phase then applies a (deg +1)-list coloring deterministic algorithm to color all these vertices.

State of the Art for Coloring in CONGESTED-CLIQUE. Hegeman and Pemmaraju [37] gave algorithms for $O(\Delta)$-coloring in the CONGESTED-CLIQUE model, which run in $O(1)$ rounds if $\Delta \geq \Theta(\log^4 n)$ and in $O(\log \log n)$ rounds otherwise. It is worth noting that $O(\Delta)$-coloring is a significantly more relaxed problem in comparison to $(\Delta + 1)$-coloring. For instance, we have long known a very simple $O(\Delta)$-coloring algorithm in LOCAL-model algorithm with round complexity $2^{O(\sqrt{\log \log n})}$ [10], but only recently such a round complexity was achieved for $(\Delta + 1)$ coloring [21, 34].

Our focus is on the much more stringent-$(\Delta + 1)$ coloring problem. For this problem, the LOCAL model algorithms of [21, 34] need messages of $O(\Delta^2 \log n)$ bits, and thus do not extend to CONGEST or CONGESTED-CLIQUE. For CONGESTED-CLIQUE model, the main challenge is when $\Delta > \sqrt{n}$, as otherwise, one can simulate the algorithm of [21] by leveraging the all-to-all communication in CONGESTED-CLIQUE which means each vertex in each round is capable of communicating $O(n \log n)$ bits of information. Parter [53] designed the first sublogarithmic-time $(\Delta+1)$ coloring algorithm for CONGESTED-CLIQUE, which runs in $O(\log \log \Delta \log^* \Delta)$ rounds. The algorithm of [53] is able to reduce the maximum degree to $O(\sqrt{n})$ in $O(\log \log \Delta)$ iterations, and each iteration invokes the algorithm of [21] on instances of maximum degree $O(\sqrt{n})$. Once the maximum degree is $O(\sqrt{n})$, the algorithm of [21] can be implemented in $O(\log^* \Delta)$ rounds in CONGESTED-CLIQUE. Subsequent to [53], the upper bound was improved to $O(\log^* \Delta)$ in [54]. Parter and Su [54] observed that the algorithm of [53] only takes $O(1)$ iterations if we only need to reduce the degree to $n^{1/2+\epsilon}$, for some constant $\epsilon > 0$, and they achieved this by modifying the internal details of [21] to reduce the required message size to $O(\Delta^{8/5} \log n)$.

Our Result. For the CONGESTED-CLIQUE model, we present a new algorithm for $(\Delta+1)$-list coloring in the randomized congested clique model running in $O(1)$ rounds. This improves on the previous best known $O(\log^* \Delta)$-round algorithm of Parter and Su [54] and settles the asymptotic complexity of the problem.

Theorem 1.1. *There is an $O(1)$-round algorithm that solves the $(\Delta+1)$-list coloring problem in* CONGESTED-CLIQUE*, with success probability $1 - 1/\text{poly}(n)$.*

The proof is presented in two parts: If $\Delta \geq \log^{4.1} n$, the algorithm of Theorem 3.2 solves the $(\Delta + 1)$-list coloring problem in $O(1)$ rounds; the algorithm for the small degree case is omitted (see the full version of the paper [20]).

1.2 Massively Parallel Computation

Model. The Massively Parallel Computation (MPC) model was introduced by Karloff, Suri, and Vassilvitskii [43], as a theoretical abstraction for practical large-scale parallel processing settings such as MapReduce [23], Hadoop [60], Spark [61], and Dryad [41], and it has been receiving increasing attention over the past few years [1, 3–6, 8, 12, 13, 16, 17, 22, 31, 33, 35, 37, 40, 43, 44, 57]. In the MPC model, the system consists of a number of machines, each with S bits of memory, which can communicate with each other in synchronous rounds through a complete communication network. Per round, each machine can send or receive at most S bits in total. Moreover, it can perform some $\text{poly}(S)$ computation, given the information that it has. In the case of graph problems, we assume that the graph G is partitioned among the machines using a simple and globally known hash function such that each machine holds at most S bits, and moreover, for each vertex or potential edge of the graph, the hash function determines which machines hold that vertex or edge.[1] Thus, the number of machines is $\Omega(m/S)$ and ideally not too much higher, where m denotes the number of edges. At the end, each machine should know the output of the vertices that it holds, e.g., their color.

[1] If $S = o(\Delta \log n)$, then we cannot afford to store all edges incident to a vertex v in a single machine. However, in this case we still need to have a machine holding v that is responsible for storing the output of v (e.g., the color of v for the $(\Delta + 1)$-coloring problem).

State of the Art for Coloring. The CONGESTED-CLIQUE algorithms discussed above can be used to obtain MPC algorithms with the same asymptotic round complexity if machines have memory of $S = \Omega(n \log n)$ bits. In particular, the work of Parter and Su [54] leads to an $O(\log^* \Delta)$-round MPC algorithm for machines with $S = \Omega(n \log n)$ bits. However, this MPC algorithm would have two drawbacks: (A) it uses $\Omega(n^2 \log n)$ global memory, and thus would require $(n^2 \log n)/S$ machines, which may be significantly larger than $\tilde{O}(m)/S$. This is basically because the algorithm makes each vertex of the graph learn some $\widetilde{\Theta}(n)$ bits of information. (B) It is limited to machines with $S = \Omega(n \log n)$ memory, and it does not extend to the machines with strongly sublinear memory, which is gaining more attention recently due to the increase in the size of graphs. We note that for the regime of machines with super-linear memory, very recently, Assadi, Chen, and Khanna [7] gave an $O(1)$-round algorithm which uses only $O(n \log^3 n)$ global memory.[2] However, this algorithm also relies heavily on $S = \Omega(n \log^3 n)$ memory per machine and cannot be run with weaker machines that have strongly sublinear memory.

Our Result. We provide the first sublogarithmic-time algorithm for $(\Delta + 1)$-coloring and $(\Delta + 1)$-list coloring in the MPC model with strongly sublinear memory per machine:

Theorem 1.2. *For any constant $\alpha > 0$, there is an* MPC *algorithm that, in $O(\log^* \Delta + \sqrt{\log \log n}) = O(\sqrt{\log \log n})$ rounds, w.h.p. computes a $(\Delta + 1)$-list coloring of an n-vertex graph with m edges and maximum degree Δ and that uses $O(n^\alpha)$ memory per machine, as well as a total memory of $\widetilde{O}(m)$.*

The proof is presented in Section 3.3.

1.3 Centralized LOCAL Computation

Model. The Local Computation Algorithms (LCA) model is a centralized model of computation that was introduced in [58]; an algorithm in this model is usually called an LCA. In this model, there is a graph $G = (V, E)$, and the algorithm is allowed to make the following queries:

Degree Query: Given $\text{ID}(v)$, the oracle returns $\deg(v)$.

Neighbor Query: Given $\text{ID}(v)$ and an index $i \in [1, \Delta]$, if $\deg(v) \leq i$, the oracle returns $\text{ID}(u)$, where u is the ith neighbor of v; otherwise, the oracle returns $\perp$.

It is sometimes convenient to assume that there is a query that returns the list of all neighbors of v. This query can be implemented using (i) Δ neighbor queries or (ii) one degree query and $\deg(v)$ neighbor queries. For randomized algorithms, we assume that there is an oracle that given $\text{ID}(v)$ returns an infinite-length random sequence associated with the vertex v. Similarly, for problems with input labels (e.g., the color lists in the list coloring problem), the input label of a vertex v can be accessed given $\text{ID}(v)$. Given a graph problem $\mathcal{P}$, an LCA $\mathcal{A}$ accomplishes the following. Given $\text{ID}(v)$, the algorithm $\mathcal{A}$ returns $\mathcal{A}(v)$ = the output of v, after making a small number of queries. It is required that the output of $\mathcal{A}$ at different vertices are consistent with one legal solution of $\mathcal{P}$.

[2]Here "global memory" refers to the memory used for communication. Of course we still need $\tilde{O}(m)$ memory to store the graph.

We emphasize that the outcome $\mathcal{A}(v)$ cannot depend on the results of previous invocations of $\mathcal{A}$. For example, if we invoke $\mathcal{A}$ on $v_1, v_2, \ldots, v_k$ sequentially, the outcome $\mathcal{A}(v_i)$ cannot depend on the previous answers $\mathcal{A}(v_j)$, $1 \leq j < i$.

The complexity measure for an LCA is the number of queries. A well-known technique for obtaining efficient LCA is simulation of known LOCAL algorithms [52]. That is, any τ-round LOCAL algorithm $\mathcal{A}$ can be transformed into an LCA $\mathcal{A}'$ with query complexity Δ^τ. The LCA $\mathcal{A}'$ simply simulates the LOCAL algorithm $\mathcal{A}$ by querying all radius-τ neighborhood of the given vertex v. This is also known as the Parnas-Ron reduction. See [47] for a recent survey of the state of the art in the centralized local model.

State of the Art LCA *for Coloring.* The previous state-of-the-art for $(\Delta + 1)$-list coloring in the centralized local computation model are based on simulation of known LOCAL algorithms. The deterministic $O(\sqrt{\Delta}\text{poly} \log \Delta + \log^* n)$-round LOCAL algorithm of [9, 27][3] can be implemented in the centralized local computation model with query complexity $\Delta^{O(\sqrt{\Delta}\text{poly} \log \Delta)} \cdot O(\log^* n)$; the randomized $O(\log^* \Delta) + 2^{O(\sqrt{\log \log n})}$-round LOCAL algorithm of [21] can be implemented in the centralized local computation model with query complexity $\Delta^{O(\log^* \Delta)} \cdot O(\log n)$.

Our Result. We show that $(\Delta + 1)$-list coloring can be solved with $\Delta^{O(1)} \cdot O(\log n)$ query complexity. Note that $\Delta^{O(1)} \cdot O(\log n)$ matches a "natural barrier" for randomized algorithms based on the graph shattering framework, as each connected component in the post-shattering phase has this size $\Delta^{O(1)} \cdot O(\log n)$.

Theorem 1.3. *There is a centralized local computation algorithm that solves the $(\Delta + 1)$-list coloring problem with query complexity $\Delta^{O(1)} \cdot O(\log n)$, with success probability $1 - 1/\text{poly}(n)$.*

The proof is omitted (see the full version of the paper [20]).

2 TECHNICAL OVERVIEW: TOOLS AND NEW INGREDIENTS

In this section, we first review some of the known technical tools that we will use in our algorithms, and then we overview the two new technical ingredients that lead to our improved results (in combination with the known tools).

Notes and Notations. When talking about randomized algorithms, we require the algorithm to succeed *with high probability (w.h.p.)*, i.e., to have success probability at least $1 - 1/\text{poly}(n)$. For each vertex v, we write $N(v)$ to denote the set of neighbors of v. If there is an edge orientation, $N^{\text{out}}(v)$ refers to the set of out-neighbors of v. We write $N^k(v) = \{u \in V \mid \text{dist}(u, v) \leq k\}$. We use subscript to indicate the graph G under consideration, e.g., $N_G(v)$ or $N_G^{\text{out}}(v)$. In the course of our algorithms, we slightly abuse the notation to also use $\Psi(v)$ to denote the set of *available colors* of v. i.e., the subset of $\Psi(v)$ that excludes the colors already taken by its neighbors in $N(v)$. The number of *excess colors* at a vertex is the number of available colors minus the number of uncolored neighbors. Moreover, we make an assumption that each color can be represented using $O(\log n)$ bits. This is without loss of generality (in all of the models

[3]Precisely, the complexity is $O(\sqrt{\Delta} \log^{2.5} \Delta + \log^* n)$ in [27], and this has been later improved to $O(\sqrt{\Delta \log \Delta} \log^* \Delta + \log^* n)$ in [9].

under consideration in our paper), since otherwise we can hash the colors down to this magnitude, as we allow a failure probability of $1/\text{poly}(n)$ for randomized algorithms.

2.1 Tools

Lenzen's Routing. The routing algorithm of Lenzen [45] for CONGESTED-CLIQUE allows us to deliver all messages in $O(1)$ rounds, as long as each vertex v is the source and the destination of at most $O(n)$ messages. This is a very useful (and frequently used) communication primitive for designing CONGESTED-CLIQUE algorithms.

Lemma 2.1 (Lenzen's Routing). *Consider a graph $G = (V, E)$ and a set of point-to-point routing requests, each given by the* IDs *of the corresponding source-destination pair. As long as each vertex v is the source and the destination of at most $O(n)$ messages, namely $O(n \log n)$ bits of information, we can deliver all messages in $O(1)$ rounds in the* CONGESTED-CLIQUE *model.*

The Shattering Framework. Our algorithm follows the *graph shattering* framework [10], which first performs some randomized process (known as *pre-shattering*) to solve "most" of the problem, and then performs some clean-up steps (known as *post-shattering*) to solve the remaining part of the problem. Typically, the remaining graph is simpler in the sense of having small components and having a small number of edges. Roughly speaking, at each step of the algorithm, we specify an invariant that all vertices must satisfy in order to continue to participate. Those *bad vertices* that violate the invariant are removed from consideration, and postponed to the post-shattering phase. We argue that the bad vertices form connected components of size $\Delta^{O(1)} \cdot O(\log n)$ with probability $1 - 1/\text{poly}(n)$; we use this in designing LCA. Also, the total number of edges induced by the bad vertices is $O(n)$. Therefore, using Lenzen's routing, in CONGESTED-CLIQUE we can gather all information about the bad vertices to one distinguished vertex $v^\star$, and then $v^\star$ can color them locally. More precisely, we have the following lemma [10, 26].

Lemma 2.2 (The Shattering Lemma). *Let $c \geq 1$. Consider a randomized procedure that generates a subset of vertices $B \subseteq V$. Suppose that for each $v \in V$, we have $\Pr[v \in B] \leq \Delta^{-3c}$, and this holds even if the random bits not in $N^c(v)$ are determined adversarially. Then, the following is true.*

1. *With probability $1 - n^{-\Omega(c')}$, each connected component in the graph induced by B has size at most $(c'/c)\Delta^{2c} \log_\Delta n$.*
2. *With probability $1 - O(\Delta^c) \cdot \exp(-\Omega(n\Delta^{-c}))$, the number of edges induced by B is $O(n)$.*

Round Compression in CONGESTED-CLIQUE *and* MPC *by Information Gathering.* Suppose we are given a τ-round LOCAL algorithm $\mathcal{A}$ on a graph of maximum degree Δ. A direct simulation of $\mathcal{A}$ on CONGESTED-CLIQUE costs also τ rounds. However, if each vertex v already knows all information in its radius-τ neighborhood, then v can locally compute its output in zero rounds. In general, this amount of information can be as high as $\Theta(n^2)$, since there could be $\Theta(n^2)$ edges in the radius-τ neighborhood of v. For the case of $\Delta^\tau = O(n)$, it is possible to achieve an exponential speed-up in the round complexity in the CONGESTED-CLIQUE, compared to that of LOCAL. In particular, in this case, each vertex v can learn its radius-τ neighborhood in just $O(\log \tau)$ rounds in CONGESTED-CLIQUE. Roughly speaking, after k rounds, we are able to simulate the product graph G^{2^k}, which is the graph where any two vertices with distance at most 2^k in graph G are adjacent. This method is known as *graph exponentiation* [46], and it has been applied before in the design of algorithms in CONGESTED-CLIQUE and MPC models, see e.g., [4, 30, 32, 53, 54].

Round Compression via Opportunistic Information Gathering. Our goal is to achieve the $O(1)$ round complexity in CONGESTED-CLIQUE, so an exponential speed-up compared to the LOCAL model will not be enough. Consider the following "opportunisitc" way of simulating a LOCAL algorithm $\mathcal{A}$ in the CONGESTED-CLIQUE model. Each vertex u sends its local information (which has $O(\Delta \log n)$ bits) to each vertex $v \in V$ with some fixed probability $p = O(1/\Delta)$, independently, and it hopes that there exists a vertex $v \in V$ that gathers all the required information to calculate the outcome of $\mathcal{A}$ at u. To ensure that for each u, there exists such a vertex v w.h.p., it suffices that $p^{\Delta^\tau} \gg \frac{\log n}{n}$. We note that a somewhat similar idea was key to the $O(1)$-round MST algorithm of [42] for CONGESTED-CLIQUE.

Lemma 2.3, presented below, summarizes the criteria for this method to work; see the full version of the paper [20] for the proof of the lemma. Denote ℓ_{in} as the number of bits needed to represent the random bits and the input for executing $\mathcal{A}$ at a vertex. Denote ℓ_{out} as the number of bits needed to represent the output of $\mathcal{A}$ at a vertex. We assume that each vertex v initially knows a set $N_*(v) \subseteq N(v)$ such that throughout the algorithm $\mathcal{A}$, each vertex v only receives information from vertices in $N_*(v)$. We write $\Delta_* = \max_{v \in V} |N_*(v)|$.

Lemma 2.3 (Opportunistic Speed-up). *Let $\mathcal{A}$ be a τ-round* LOCAL *algorithm on $G = (V, E)$. There is an $O(1)$-round simulation of $\mathcal{A}$ in* CONGESTED-CLIQUE*, given that (i) $\Delta_*^\tau \log(\Delta_* + \ell_{\text{in}}/\log n) = O(\log n)$, (ii) $\ell_{\text{in}} = O(n)$, and (iii) $\ell_{\text{out}} = O(\log n)$.*

2.2 Our New Technical Ingredients, In a Nutshell

The results in our paper are based on the following two novel technical ingredients, which are used in combination with the known tools mentioned above: (i) a new graph partitioning algorithm for coloring and (ii) a sparsification of the CLP coloring algorithm [21]. We note that the first ingredient suffices for our CONGESTED-CLIQUE result for graphs with maximum degree at least $\text{poly}(\log n)$, and also for our MPC result. This ingredient is presented in Section 3. The second ingredient, which is also more involved technically, is used for extending our CONGESTED-CLIQUE result to graphs with smaller maximum degree, as well as for our LCA result. This ingredient is presented in the full version of the paper [20]. Here, we provide a brief overview of these ingredients and how they get used in our results.

Ingredient 1 — Graph Partitioning for Coloring. We provide a simple random partitioning that significantly simplifies and extends the one in [53, 54]. The main change will be that, besides partitioning the vertices randomly, we also partition the colors randomly. In particular, this new procedure partitions the vertices and colors in a way that allows us to easily apply CLP in a black box manner.

Concretely, our partitioning breaks the graph as well as the respective palettes randomly into many subgraphs $B_1, \ldots, B_k$ of maximum degree $O(\sqrt{n})$ and size $O(\sqrt{n})$, while ensuring that each vertex in these subgraphs receives a random part of its palette with size close to the maximum degree of the subgraph. The palettes for each part are disjoint, which allows us to color all parts in parallel. There will be one left-over subgraph L, with maximum degree $\tilde{O}(\Delta^{3/4})$, as well as sufficiently large remaining palettes for each vertex in this left-over subgraph.

Application in CONGESTED-CLIQUE**:** Since each subgraph has $O(n)$ edges, all of $B_1, \ldots, B_k$ can be colored, in parallel, in $O(1)$ rounds, using Lenzen's routing (Lemma 2.1). The left-over part L is handled by recursion. We show that when $\Delta > \log^{4.1} n$, we are done after $O(1)$ levels of recursion.

Application in Low-memory MPC: We perform recursive calls not only on L but also on $B_1, \ldots, B_k$. After $O(1)$ levels of recursion, the maximum degree can be made $O(n^\beta)$, for any given constant $\beta > 0$, which enables us to run the CLP algorithm on a low memory MPC.

We note that the previous partitioning approach [53, 54] is unable to reduce the maximum degree to below $\sqrt{n}$; this is a significant limitation that our partitioning overcomes. We also note that the CONGESTED-CLIQUE or MPC coloring algorithms of [35, 38] also use the approach of randomly partitioning the palette. However, their algorithms needs a palette of size that is much higher than $\Delta + 1$ to ensure that the set of colors associated with each vertex set B_i is higher than its maximum degree. We avoid the use of extra colors by having a sufficiently large left-over part L and recursively applying the partitioning algorithm on L.

Ingredient 2 — Sparsification of the CLP Algorithm. In general, to calculate the output of a vertex v in a τ-round LOCAL algorithm $\mathcal{A}$, the output may depend on all of the τ-hop neighborhood of v. In particular, if we transform $\mathcal{A}$ into an LCA, the query complexity can be as high as Δ^τ. To efficiently simulate $\mathcal{A}$ in CONGESTED-CLIQUE or to transform $\mathcal{A}$ to an LCA, a strategy is to "sparsify" the algorithm $\mathcal{A}$ so that the number of vertices a vertex has to explore to decide its output is sufficiently small. This notion of sparsification is a key idea behind some recent algorithms [30, 32]. In the present paper, a key technical ingredient is providing such a sparsification for the $(\Delta + 1)$ coloring algorithm of CLP [21].

The pre-shattering phase of the CLP algorithm [21] consists of three parts: (i) initial coloring, (ii) dense coloring, and (iii) color bidding. Parts (i) and (ii) take $O(1)$ rounds;[4] part (iii) takes $\tau = O(\log^* \Delta)$ rounds. In this paper, we sparsify the color bidding part of the CLP algorithm. We let each vertex v sample $O(\text{poly} \log \Delta)$ colors from its palette at the beginning of this procedure, and we show that with probability $1 - 1/\text{poly}(\Delta)$, these colors are enough for v to correctly execute the algorithm. Based on the sampled colors, we can do an $O(1)$-round pre-processing step to let each vertex v identify a subset of neighbors $N_*(v) \subseteq N(v)$ of size $\Delta_* = O(\text{poly} \log \Delta)$ neighbors $N_*(v) \subseteq N(v)$, and v only needs to receive messages from neighbors in $N_*(v)$ in the subsequent steps of the algorithm.

[4] In the preliminary versions (arXiv:1711.01361v1 and STOC'18) of [21], dense coloring takes $O(\log^* \Delta)$ time. This time complexity has been later improved to $O(1)$ in a revised full version of [21].

Application in CONGESTED-CLIQUE**:** For the case $\Delta = O(\text{poly} \log n)$, the parameters $\tau = O(\log^* \Delta)$ and $\Delta_* = O(\text{poly} \log \Delta) = O(\text{poly}(\log \log n))$ satisfy the condition for applying the opportunistic speedup lemma (Lemma 2.3), and so the pre-shattering phase of the CLP algorithm can be simulated in $O(1)$ rounds in CONGESTED-CLIQUE.

Application in Centralized Local Computation: With sparsification, the pre-shattering phase of the CLP algorithm can be transformed into an LCA with $\Delta^{O(1)} \cdot \Delta_*^\tau = \Delta^{O(1)}$ queries.

The recent work [7] on $(\Delta + 1)$-coloring in MPC is also based on some form of palette sparsification, as follows. They showed that if each vertex samples $O(\log n)$ colors uniformly at random, then w.h.p., the graph still admits a proper coloring using the sampled colors. Since we only need to consider the edges $\{u, v\}$ where u and v share a sampled color, this effectively reduces the degree to $O(\log^2 n)$. For an MPC algorithm with $\tilde{O}(n)$ memory per processor, the entire sparsified graph can be sent to one processor, and a coloring can be computed there, using any coloring algorithm, local or not. This sparsification is not applicable for our setting. In particular, in our sparsified CLP algorithm, we need to ensure that the coloring can be computed by a LOCAL algorithm with a small locality volume; this is because the final coloring is constructed distributedly via the opportunistic speedup lemma (Lemma 2.3).

3 COLORING OF HIGH-DEGREE GRAPHS VIA GRAPH PARTITIONING

In this section, we describe our graph partitioning algorithm, which is the first new technical ingredient in our results. As mentioned in Section 2.2, this ingredient on its own leads to our CONGESTED-CLIQUE result for graphs with $\Delta = \Omega(\text{poly}(\log n))$ and also our MPC result, as we will explain in Section 3.2 and Section 3.3, respectively. The algorithm will be applied recursively, but it is required that the failure probability is at most $1 - 1/\text{poly}(n)$ in all recursive calls, where n is the number of vertices in the original graph.

In this section, the parameter n refers to the number of vertices in the original graph, not the number of vertices in the current subgraph $G = (V, E)$ under consideration.

3.1 Graph Partitioning

We consider a graph partitioning algorithm parameterized by two constants γ and λ satisfying $\gamma \geq 2$ and $\lambda = \frac{1}{2} + \frac{2}{3\gamma+2}$. Consider a graph $G = (V, E)$ with maximum degree Δ. Recall that in this section G is assumed to be a subgraph of the n-vertex original graph, and so $n \geq |V|$. Each vertex $v \in V$ has a palette $\Psi(v)$ of size $|\Psi(v)| \geq \max\{\deg_G(v), \Delta'\} + 1$, where $\Delta' = \Delta - \Delta^\lambda$. Denote $G[S]$ as the subgraph induced by the vertices $S \subseteq V$. For each vertex $v \in V$, denote $\deg_S(v)$ as $|N(v) \cap S|$. The algorithm is as follows, where we set $k = \sqrt{\Delta}$.

Vertex Set: The partition $V = B_1 \cup \cdots \cup B_k \cup L$ is defined by the following procedure. Including each $v \in V$ to the set L with probability $q = \Theta\left(\frac{\sqrt{\log n}}{\Delta^{1/4}}\right)$. Each remaining vertex joins one

of $B_1, \ldots, B_k$ uniformly at random. Note that $\Pr[v \in B_i] = p(1-q)$, where $p = 1/k = 1/\sqrt{\Delta}$.

Palette: Denote $C = \bigcup_{v \in V} \Psi(v)$ as the set of all colors. The partition $C = C_1 \cup \cdots \cup C_k$ is defined by having each color $c \in C$ join one of $C_1, \ldots, C_k$ uniformly at random. Note that $\Pr[c \in C_i] = p = 1/k$.

We require that with probability $1 - 1/\text{poly}(n)$, the output of the partitioning algorithm satisfies the following properties, assuming that $\Delta = \omega(\log^\gamma n)$.

i) Size of Each Part: It is required that $|E(G[B_i])| = O(|V|)$, for each $i \in [k]$. Also, it is required that $|L| = O(q|V|) = O(\frac{\sqrt{\log n}}{\Delta^{1/4}}) \cdot |V|$.

ii) Available Colors in B_i: For each $i \in \{1, \ldots, k\}$ and $v \in B_i$, the number of available colors in v in the subgraph B_i is $g_i(v) := |\Psi(v) \cap C_i|$. It is required that $g_i(v) \geq \max\{\deg_{B_i}(v), \Delta_i - \Delta_i^\lambda\} + 1$, where $\Delta_i := \max_{v \in B_i} \deg_{B_i}(v)$.

iii) Available Colors in L: For each $v \in L$, define $g_L(v) := |\Psi(v)| - (\deg_G(v) - \deg_L(v))$. It is required that $g_L(v) \geq \max\{\deg_L(v), \Delta_L - \Delta_L^\lambda\} + 1$ for each $v \in L$, where $\Delta_L := \max_{v \in L} \deg_L(v)$. Note that $g_L(v)$ represents a lower bound on the number of available colors in v *after* all of $B_1, \ldots, B_k$ have been colored.

iv) Remaining Degrees: The maximum degrees of B_i and L are $\max_{v \in B_i} \deg_{B_i}(v) \leq \Delta_i = O(\sqrt{\Delta})$ and $\max_{v \in L} \deg_L(v) \leq \Delta_L = O(q\Delta) = O(\frac{\sqrt{\log n}}{\Delta^{1/4}}) \cdot \Delta$. For each vertex v, we have $\deg_{B_i}(v) \leq \max\{O(\log n), O(1/\sqrt{\Delta}) \cdot \deg(v)\}$ if $v \in B_i$, and $\deg_L(v) \leq \max\{O(\log n), O(q) \cdot \deg(v)\}$ if $v \in L$.

Intuitively, we will use this graph partitioning in the following way. First compute the decomposition of the vertex set and the palette, and then color each B_i using colors in C_i. Since $|E(G[B_i])| = O(|V|) = O(n)$, in the CONGESTED-CLIQUE model we are able to send the entire graph $G[B_i]$ to a single distinguished vertex $v_i^\star$ using Lenzen's routing (Lemma 2.1), and then $v_i^\star$ can compute a proper coloring of $G[B_i]$ locally. This procedure can be done in parallel for all i. If $|E(G[L])| = O(n)$, then similarly we can let a vertex compute a proper coloring of $G[L]$; otherwise we apply the graph partitioning recursively on $G[L]$, with the *same* parameter n. Later we will see that it suffices to set the recursion depth *constant*. This graph partitioning will be used as well in our MPC algorithm in the similar way.

Lemma 3.1. *Suppose $|\Psi(v)| \geq \max\{\deg_G(v), \Delta'\} + 1$ with $\Delta' = \Delta - \Delta^\lambda$, and $|V| > \Delta = \omega(\log^\gamma n)$, where γ and λ are two constants satisfying $\gamma \geq 2$ and $\lambda = \frac{1}{2} + \frac{2}{3\gamma+2}$. The two partitions $V = B_1 \cup \cdots \cup B_k \cup L$ and $C = \bigcup_{v \in V} \Psi(v) = C_1 \cup \cdots \cup C_k$ satisfy the required properties, with probability $1 - 1/\text{poly}(n)$.*

Proof. We prove that the properties i), ii), iii), and iv) hold with high probability.

i) Size of Each Part: We first show that $|E(G[B_i])| = O(|V|)$, for each $i \in [k]$, with probability $1 - 1/\text{poly}(n)$. To have $|E(G[B_i])| = O(|V|)$, it suffices to have $\deg_{B_i}(v) = O(p\Delta)$ for each v, and $|B_i| = O(p|V|)$, since $p = 1/\sqrt{\Delta}$. Note that we already have $\mathrm{E}[\deg_{B_i}(v)] \leq (1-q)p\Delta < p\Delta$ and $\mathrm{E}[|B_i|] = (1-q)p|V| < p|V|$, so we only need to show that these parameters concentrate at their expected values with high probability. This can be established by a Chernoff bound, as follows. Note that we have $\epsilon_1 < 1$ and $\epsilon_2 < 1$ in the following calculation. In particular, the inequality $\epsilon_1 < 1$ holds because of the assumption $\Delta = \omega(\log^\gamma n) \geq \omega(\log^2 n)$.

$$\begin{aligned}&\Pr[\deg_{B_i}(v) \leq (1+\epsilon_1)(1-q)p\Delta]\\ &= 1 - \exp(-\Omega(\epsilon_1^2(1-q)p\Delta)) = 1 - O(1/\text{poly}(n)),\\ &\text{where } \epsilon_1 = \Theta\left(\sqrt{\frac{\log n}{(1-q)p\Delta}}\right) = \Theta\left(\sqrt{\frac{\log n}{p\Delta}}\right).\end{aligned}$$

$$\begin{aligned}&\Pr[|B_i| \leq (1+\epsilon_2)(1-q)p|V|]\\ &= 1 - \exp(-\Omega(\epsilon_2^2(1-q)p|V|)) = 1 - O(1/\text{poly}(n)),\\ &\text{where } \epsilon_2 = \Theta\left(\sqrt{\frac{\log n}{(1-q)p|V|}}\right) = \Theta\left(\sqrt{\frac{\log n}{p|V|}}\right).\end{aligned}$$

Next, we show the analogous results for L, i.e., with probability $1 - 1/\text{poly}(n)$, both $|L|/|V|$ and Δ_L/Δ are $O(q) = O\left(\frac{\sqrt{\log n}}{\Delta^{1/4}}\right)$, where $\Delta_L = \max_{v \in L} \deg_L(v)$. Similarly, we already have $\mathrm{E}[\deg_L(v)] \leq q\Delta$ and $\mathrm{E}[|L|] = q|V|$, and remember that $q = O(\frac{\sqrt{\log n}}{\Delta^{1/4}})$, so we only need to show that these parameters concentrate at their expected values with high probability, by a Chernoff bound.

$$\begin{aligned}&\Pr[\deg_L(v) \leq (1+\epsilon_3)q\Delta]\\ &= 1 - \exp(-\Omega(\epsilon_3^2 q\Delta)) = 1 - O(1/\text{poly}(n)),\\ &\text{where } \epsilon_3 = \Theta\left(\sqrt{\frac{\log n}{q\Delta}}\right).\end{aligned}$$

$$\begin{aligned}&\Pr[|L| \leq (1+\epsilon_4)q|V|]\\ &= 1 - \exp(-\Omega(\epsilon_4^2 q|V|)) = 1 - O(1/\text{poly}(n)),\\ &\text{where } \epsilon_4 = \Theta\left(\sqrt{\frac{\log n}{q|V|}}\right).\end{aligned}$$

Similarly, we have $\epsilon_3 < 1$ and $\epsilon_4 < 1$. In particular, $\epsilon_3 < 1$ because $\Delta = \omega(\log^\gamma n) \geq \omega(\log^2 n)$.

ii) Available Colors in B_i: Now we analyze the number of available color for each set B_i. Recall that for each $v \in B_i$, the number of available colors in v in the subgraph B_i is $g_i(v) := |\Psi(v) \cap C_i|$. We need to prove the following holds with probability $1 - 1/\text{poly}(n)$: (i) $|\Psi(v) \cap C_i| \geq \deg_{B_i}(v) + 1$, and (ii) $|\Psi(v) \cap C_i| \geq \Delta_i - \Delta_i^\lambda + 1$, where $\Delta_i := \max_{v \in B_i} \deg_{B_i}(v)$. We will show that with probability $1 - 1/\text{poly}(n)$, we have $|\Psi(v) \cap C_i| \geq \Delta_i + 1$ for each B_i and each $v \in B_i$, and this implies the above (i) and (ii).

Recall that $\Delta' = \Delta\left(1 - \Delta^{-(1-\lambda)}\right)$, $q = \Theta\left(\frac{\sqrt{\log n}}{\Delta^{1/4}}\right) \gg \Delta^{-(1-\lambda)}$,[5] and $\epsilon_1 = \Theta\left(\frac{\sqrt{\log n}}{\Delta^{1/4}}\right)$. By selecting $q \geq 3\epsilon_1 = \Theta\left(\frac{\sqrt{\log n}}{\Delta^{1/4}}\right)$, we have

$$(1-\epsilon_1)p\Delta' = (1-\epsilon_1)\left(1 - \Delta^{-(1-\lambda)}\right)p\Delta \geq (1+\epsilon_1)(1-q)p\Delta + 1.$$

[5] The assumptions $\gamma \geq 2$ and $\lambda = \frac{1}{2} + \frac{2}{3\gamma+2}$ imply that $\lambda \in (1/2, 3/4]$, and so $\Delta^{-(1-\lambda)} \leq \Delta^{-1/4} \ll q$.

We already know that $\Delta_i \le (1+\epsilon_1)(1-q)p\Delta$ with probability $1-1/\text{poly}(n)$. In order to have $|\Psi(v)\cap C_i| \ge \Delta_i + 1$, we only need to show that $|\Psi(v)\cap C_i| \ge (1-\epsilon_1)p\Delta'$ with probability $1-1/\text{poly}(n)$. For the expected value, we know that $\mathrm{E}[|\Psi(v)\cap C_i|] = p|\Psi(v)| \ge p\Delta'$. By a Chernoff bound, we have

$$\begin{aligned}\Pr[|\Psi(v)\cap C_i| \ge (1-\epsilon_1)p\Delta'] &= 1-\exp(-\Omega(\epsilon_1^2 p\Delta')) \\ &= 1-O(1/\text{poly}(n)).\end{aligned}$$

iii) Available Colors in L: Next, we consider the number of available colors in L. We show that with probability $1-1/\text{poly}(n)$, for each $v \in L$, we have $g_L(v) \ge \max\{\deg_L(v), \Delta_L - \Delta_L^\lambda\} + 1$, where $g_L(v) = |\Psi(v)| - (\deg_G(v) - \deg_L(v))$. It is straightforward to see that $g_L(v) \ge \deg_L(v) + 1$, since $g_L(v) = (|\Psi(v)| - \deg_G(v)) + \deg_L(v) \ge 1 + \deg_L(v)$. Thus, we only need to show that $g_L(v) \ge \Delta_L - \Delta_L^\lambda + 1$.

We first calculate the expected value of g_L. In the last inequality we use the assumption that $\deg_G(v) \le |\Psi(v)|-1$ and $\Delta' \le |\Psi(v)|-1$.

$$\begin{aligned}\mathrm{E}[g_L] &= (|\Psi(v)| - \deg_G(v)) + \mathrm{E}[\deg_L(v)] \\ &= (|\Psi(v)| - \deg_G(v)) + q\deg_G(v) \\ &= |\Psi(v)| - (1-q)\deg_G(v) \\ &\ge q\Delta'.\end{aligned}$$

We prove that $g_L(v) \ge (1-\epsilon_3)q\Delta'$ with probability $1-1/\text{poly}(n)$. Here we use the following variant of a Chernoff bound. If it is known that $\mathrm{E}[X] \le \beta$, then the probability that X deviates from $\mathrm{E}[X]$ by an additive term of more than $\epsilon\beta$ is $\exp(-\Omega(\epsilon^2\beta))$. We set $X = \deg_L(v)$, $\epsilon = \epsilon_3(\Delta'/\Delta) = \Theta(\epsilon_3)$, and $\beta = q\Delta \ge \mathrm{E}[\deg_L(v)]$.

$$\begin{aligned}\Pr[g_L(v) \ge (1-\epsilon_3)q\Delta'] &= \Pr[\deg_L(v) \ge q\deg_G(v) - \epsilon_3 q\Delta'] \\ &= 1-\exp(-\Omega(\epsilon_3^2 q\Delta')) \\ &= 1-O(1/\text{poly}(n))\end{aligned}$$

Remember that $\epsilon_3 = \Theta\left(\sqrt{\frac{\log n}{q\Delta}}\right) = \Theta\left(\sqrt{\frac{\log n}{q\Delta'}}\right)$, and we already know that $\epsilon_3 < 1$. Using the above concentration bound, we infer that $g_L(v) \ge q\Delta - q\Delta^\lambda - O\left(\sqrt{q\Delta\log n}\right)$ holds with probability $1-1/\text{poly}(n)$.

$$\begin{aligned}g_L(v) &\ge (1-\epsilon_3)q\Delta' \\ &\ge q\Delta' - O\left(\sqrt{q\Delta'\log n}\right) \\ &\ge q\Delta - q\Delta^\lambda - O\left(\sqrt{q\Delta\log n}\right).\end{aligned}$$

Combining this with $\Delta_L \le (1+\epsilon_3)q\Delta = q\Delta + O(\sqrt{q\Delta\log n})$, we obtain $g_L(v) \ge \Delta_L - q\Delta^\lambda - O(\sqrt{q\Delta\log n})$. Note that $q\Delta^\lambda + O(\sqrt{q\Delta\log n}) = o\left((q\Delta)^\lambda\right) = o\left(\Delta_L^\lambda\right)$,[6] and so we finally obtain $g_L(v) \ge \Delta_L - \Delta_L^\lambda + 1$.

iv) Remaining Degrees: The degree upper bounds of Δ_i and Δ_L follow immediately from the concentration bounds on $\deg_{B_i}(v)$ and $\deg_L(v)$ calculated in the proof of i). The bounds $\deg_{B_i}(v) \le \max\{O(\log n), O(1/\sqrt{\Delta})\cdot\deg(v)\}$ and $\deg_L(v) \le \max\{O(\log n), O(q)\cdot\deg(v)\}$ can be derived by a straightforward application of Chernoff bound. □

[6] The bound $\sqrt{q\Delta\log n} \ll (q\Delta)^\lambda$ can be derived from the assumptions $\lambda = \frac{1}{2} + \frac{2}{3\gamma+2}$ and $\Delta = \omega(\log^\gamma n)$, as follows: $q\Delta = \Theta(\Delta^{\frac{3}{4}}\log^{\frac{1}{2}} n) = \omega(\log^{\frac{3}{4}\gamma+\frac{1}{2}} n) \implies \sqrt{q\Delta\log n} = (q\Delta)^{\frac{1}{2}}\log^{1/2} n \ll (q\Delta)^{\frac{1}{2}}(q\Delta)^{\frac{1}{2}\left(\frac{3}{4}\gamma+\frac{1}{2}\right)^{-1}} = (q\Delta)^\lambda$.

3.2 Congested Clique Algorithm for High-Degree Graphs

In this section, we show that the $(\Delta+1)$-list coloring problem can be solved in $O(1)$ rounds in the CONGESTED-CLIQUE model when the degrees are assumed to be sufficiently high. The formal statement is captured in Theorem 3.2. First, we show that the partitioning algorithm can indeed be implemented in the CONGESTED-CLIQUE model. Then, we show how to color the parts resulting from the graph partitioning efficiently. The proof of Theorem 3.2 is completed by showing that only $O(1)$ recursive applications of the partitioning are required.

Implementation of the Graph Partitioning. The partitions can be computed in $O(1)$ rounds on CONGESTED-CLIQUE. Partitioning the vertex set V is straightforward, as every vertex can make the decision independently and locally, whereas it is not obvious how to partition C to make all vertices agree on the same partition. Note that we can assume $|C| \le (\Delta+1)|V|$; if $|C|$ is greater than $(\Delta+1)|V|$ initially, then we can let each vertex decrease its palette size to $\Delta+1$ by removing some colors in its palette, and we will have $|C| \le (\Delta+1)|V|$ after removing these colors.

A straightforward way of partitioning C is to generate $\Theta(|C|\log n)$ random bits at a vertex v locally, and then v broadcasts this information to all other vertices. Note that it takes $O(\log k) = O(\log|V|) = O(\log n)$ bits to encode which part of $C_1 \cup \cdots \cup C_k$ each $c \in C$ is in. A direct implementation of the approach cannot be done in $O(1)$ rounds, due to the message size constraint of CONGESTED-CLIQUE, as each vertex can send at most $\Theta(n\log n)$ bits in each round.

To solve this issue, observe that it is not necessary to use total independent random bits for each $c \in C$, and $\Theta(\log n)$-wise independence suffices. More precisely, suppose X is the summation of n K-wise independent 0-1 random variables with mean p, and so $\mu = \mathrm{E}[X] = np$. A Chernoff bound with K-wise Independence [59] guarantees that

$$\Pr[X \ge (1+q)\mu] \le \exp\left(-\min\{K, q^2\mu\}\right).$$

In order to guarantee a failure probability of $1/\text{poly}(n)$ in all applications of Chernoff bound in Lemma 3.1, it suffices that $K = \Theta(\log n)$. Therefore, to compute the decomposition $C = C_1 \cup \cdots \cup C_k$ with K-wise independent random bits, we only need $O(K\cdot\log(|C|\log k)) = O(\log^2 n)$ total independent random bits.[7] Broadcasting $O(\log^2 n)$ bits of information to all vertices can be done in $O(1)$ rounds via Lenzen's routing (Lemma 2.1).

The Algorithm of $(\Delta+1)$-list coloring on High-degree Graphs. We next present our CONGESTED-CLIQUE-model coloring algorithm for high-degree graphs, using the partitioning explained above.

[7] It is well-known that n Bernoulli random variables with $p = 1/2$ with d-wise independence can be constructed from $O(d\log n)$ Bernoulli random variables with $p = 1/2$ with total independence [2, Section 16.2].

Theorem 3.2. *Suppose $\Delta = \Omega(\log^{4+\epsilon} n)$ for some constant $\epsilon > 0$. There is an algorithm that solves $(\Delta+1)$-list coloring in* CONGESTED-CLIQUE *in $O(1)$ rounds.*

Proof. We show that a constant-depth recursive applications of Lemma 3.1 suffices to give an $O(1)$-round CONGESTED-CLIQUE $(\Delta+1)$-list coloring algorithm for graphs with $\Delta = \Omega(\log^{4+\epsilon} n)$, for any constant $\epsilon > 0$. Consider the graph $G = (V, E)$. First, we apply the graph partitioning algorithm of Lemma 3.1 to partition vertices V into subsets $B_1, \ldots, B_k, L$ with parameter $n = |V|$, and $k = \sqrt{\Delta}$. After that, let arbitrary $k = \sqrt{\Delta}$ vertices to be responsible for coloring each $G[B_i]$. Each of these k vertices, in parallel, gathers all information of $G[B_i]$ from vertices B_i, and then computes a proper coloring of $G[B_i]$, where each vertex $v \in B_i$ uses only the palette $\Psi(v) \cap C_i$. The existence of such a proper coloring is guaranteed by Property (ii). Using this approach, we can color all vertices in $V \setminus L$ in $O(1)$ rounds using Lenzen's routing. Note that Property (i) guarantees that $|E(G[B_i])| = O(n)$. Finally, each vertex $v \in L$ removes the colors that have been taken by its neighbors in $V \setminus L$ from its palette $\Psi(v)$. In view of Property (iii), after this operation, the number of available colors for each $v \in L$ is at least $g_L(v) \geq \max\{\deg_L(v), \Delta_L - \Delta_L^{\lambda}\} + 1$. Now the subgraph $G[L]$ satisfies all conditions required to apply Lemma 3.1, so long as $\Delta_L = \omega(\log^{\gamma} n)$. We will see that this condition is always met in our application.

We then recursively apply the algorithm of the lemma on the subgraph induced by vertices L *with the same parameter* n. The recursion stops once we reach a point that $|E(G[L])| = O(n)$, and so we can apply Lenzen's routing to let one vertex v gather all information of $G[L]$ and compute its proper coloring.

Now we analyze the number of iterations needed to reach a point that $|E(G[L])| = O(n)$. Here we use $\gamma = 2$ and $\lambda = 3/4$.[8] Define $V_1 = V$ and $\Delta_1 = \Delta$ as the vertex set and the maximum degree for the first iteration. Let $V = B_1 \cup \cdots \cup B_k \cup L$ be the outcome of the first iteration, and define $V_2 = L$ and $\Delta_2 = \Delta_L$. Similarly, for $i > 2$, we define V_i and Δ_i based on the set L in the outcome of the graph partitioning algorithm for the $(i-1)$th iteration. We have the following formulas.

$$\Delta_1 = \Delta$$

$$\Delta_i = \Delta_{i-1} \cdot O\left(\frac{\sqrt{\log n}}{\Delta_{i-1}^{1/4}}\right) \qquad \text{by Property iv)}$$

$$|V_1| = n$$

$$|V_i| = |V_{i-1}| \cdot O\left(\frac{\sqrt{\log n}}{\Delta_{i-1}^{1/4}}\right) \qquad \text{by Property i)}$$

Let $\alpha > 0$ be chosen such that $\Delta = \Delta_1 = (\log n)^{2+\alpha}$, and assume $\alpha = \Omega(1)$ and $i = O(1)$. We can calculate the value of Δ_i and $|V_i|$ as follows.

$$\Delta_i = O\left((\log n)^{2+\alpha\cdot(3/4)^{i-1}}\right)$$

$$|V_i| = O(n/\Delta) \cdot \Delta_i = n \cdot O\left((\log n)^{\alpha\left((3/4)^{i-1}-1\right)}\right)$$

[8] We choose $\gamma = 2$ (the smallest possible) to minimize the degree requirement in Theorem 3.2.

Thus, given that $\alpha = \Omega(1)$ and $i = O(1)$, the condition of $\Delta_i = \omega(\log^{\gamma} n) = \omega(\log^2 n)$ for applying Lemma 3.1 must be met.

Next, we analyze the number of iterations it takes to make $\Delta_i |V_i|$ sufficiently small. In the CONGESTED-CLIQUE model, if $\Delta_i |V_i| = O(n)$, then we are able to compute a proper coloring of V_i in $O(1)$ rounds by information gathering. Let us write $\Delta = \log^{2+\alpha} n$, where $\alpha = 2 + \beta$. The lemma statement implies that $\beta = \Omega(1)$. Note that the condition for $\Delta_i |V_i| = O(n)$ can be re-written as

$$(2 - \alpha) + 2\alpha(3/4)^{i-1} \leq 0.$$

Combining this with $\alpha = 2 + \beta$, we obtain the formula $-\beta + 2(2 + \beta)(3/4)^{i-1} \leq 0$, and this can be re-written as $(4/3)^{i-1} \geq \frac{2(2+\beta)}{\beta}$. Now we can calculate the minimum i needed so that the condition for $\Delta_i |V_i| = O(n)$ is met:

$$i \geq 1 + \log_{4/3} 2(2 + \beta)/\beta.$$

Since $\beta = \Omega(1)$, we have $1 + \log_{4/3} 2(2 + \beta)/\beta = O(1)$, and so our algorithm takes only $O(1)$ iterations. In particular, when $\beta \geq 10.8$, i.e., $\Delta = \Omega(\log^{12.8} n)$, we have $i \geq 1 + \log_{4/3} 2(2 + \beta)/\beta$ for $i \geq 4$. Therefore, $\Delta_4 |V_4| = O(n)$, and so 3 iterations suffice. Since each iteration can be implemented in CONGESTED-CLIQUE in $O(1)$ rounds, overall we get an algorithm with round complexity $O(1)$. □

Similar to the proof of Theorem 3.2, the graph partitioning algorithm also leads to an $O(1)$-round MPC coloring algorithm with $S = \widetilde{O}(n)$ memory per processor and $\widetilde{O}(m)$ total memory. This gives a simple alternate proof of a result of [7] that $(\Delta+1)$-coloring can be solved with $S = \widetilde{O}(n)$ memory per processor. However, the coloring algorithm of [7] takes only one round of communication, and it uses only $\widetilde{O}(n)$ bits in total.

3.3 Massively Parallel Computation with Strongly Sublinear Memory

We now show how to apply Lemma 3.1 as well as the CLP algorithm of [21], as summarized in the following lemma, to prove Theorem 1.2.

Lemma 3.3 ([21, 53]). *Let G be an n-vertex graph with m edges and maximum degree Δ. Suppose any vertex v has a palette $|\Psi(v)|$ that satisfies $|\Psi(v)| \geq \max\left\{\deg_G(v) + 1, \Delta - \Delta^{3/5}\right\}$. Then the list-coloring problem can be solved w.h.p. in $O(\sqrt{\log \log n})$ rounds of low-memory* MPC *with local memory $O(n^{\alpha})$ for an arbitrary constant $\alpha \in (0, 1)$ and total memory $\widetilde{O}\left(\sum_v \deg_G(v)^2\right)$ if $\Delta^2 = O(n^{\alpha})$.*

The proof of Lemma 3.3 almost immediately follows from [21, 53]; there are only few changes that have to be made in order to turn their CONGESTED-CLIQUE algorithm into a low-memory MPC algorithm. The details are deferred to the full version of the paper [20].

Proof of Theorem 1.2. We present a recursive algorithm based on the randomized partitioning algorithm of Lemma 3.1. If $\Delta = \text{poly}(\log n)$ then the conditions of Lemma 3.3 are satisfied trivially; we can solve the problem in $O(\log^* \Delta + \sqrt{\log \log n}) = O(\sqrt{\log \log n})$ rounds of low-memory MPC with total memory $\widetilde{O}(n \cdot \Delta^2) = \widetilde{O}(m)$. Otherwise, we execute the following algorithm.

Randomized Partitioning: Let G be the graph that we want to color. We apply the randomized partitioning algorithm of Lemma 3.1 to G, which gives us sets $B_1, \ldots, B_k$ and L, as well as color sets $C_1, \ldots, C_k$. The goal is now to first color $B_1, \ldots, B_k$ with colors from $C_1, \ldots, C_k$, respectively. Since the colors in the sets C_i are disjoint, this gives a proper coloring of $B := \bigcup_{i=1}^{k} B_i$. Then, for every vertex in L, we remove all colors already used by neighbors in B from the palettes, leaving us with a list-coloring problem of the graph induced by L with maximum degree Δ_L.

In the following, we first describe how to color each set B_i with colors in C_i, and then how to solve the remaining list-coloring problem in L. For the parameters in Lemma 3.1, we use $\gamma = 6$ and $\lambda = 3/5$.[9]

List-Coloring Problem in B_i: If the maximum degree Δ_i in B_i satisfies $\Delta_i^2 = O(n^\alpha)$, then, by Lemma 3.1 ii), B_i satisfies the conditions of Lemma 3.3 We thus can apply the algorithm of Lemma 3.3 to B_i. Otherwise, we recurse on B_i. Note that this is possible since, by Lemma 3.1 ii) applied to G, B_i satisfies the conditions of Lemma 3.1.

List-Coloring Problem in L: If the maximum degree Δ_L in L satisfies $\Delta_L^2 = O(n^\alpha)$, then, by Lemma 3.1 iii) applied to G, L satisfies the conditions of Lemma 3.3. We thus can apply the algorithm of Lemma 3.3 to L. Otherwise, we recurse on L. Note that this is possible since by Lemma 3.1 iii), L satisfies the conditions of Lemma 3.1.

Number of Iterations: Since the maximum degree in L reduces by a polynomial factor in every step, after at most $O(1/\alpha)$ steps, the resulting graph has maximum degree at most $O(n^{\alpha/2})$, where we satisfy the conditions of Lemma 3.3, and hence do not recurse further. Note that when recursing on sets B_i, the degree drop is even larger, and hence the same reasoning applies to bound the number of iterations.

Memory Requirements: It is obvious that the recursive partitioning of the input graph G does not incur any asymptotic overhead in the memory, neither local nor global. Now, let $\mathcal{H}$ be the set of all graphs H on which we apply the algorithm of Lemma 3.3. As we only apply this algorithm when the maximum degree Δ_H of H is $O(n^{\alpha/2})$ or $\text{poly}(\log n)$, we clearly have $\Delta_H^2 = O(n^\alpha)$, so the algorithm Lemma 3.3 is guaranteed to run with local memory $O(n^\alpha)$.

It remains to show how to guarantee the total memory requirement of $\widetilde{O}(m)$, where m is the number of edges in the input graph G, as promised in Theorem 1.2. First, observe that due to the specifications of Lemma 3.3, we can write the total memory requirement as $\sum_{H\in\mathcal{H}} \sum_{v\in H} (\deg_H(v))^2$. First, assume that the graph G has been partitioned at least three times to get to H. By Lemma 3.1 iv), the degree of any vertex v in H is either $\tilde{O}(1)$ or at most

$$\deg_G(v) \cdot \tilde{O}\left(\Delta^{-\frac{1}{4}}\right) \cdot \tilde{O}\left(\Delta^{-\frac{1}{4}\cdot\frac{3}{4}}\right) \cdot \tilde{O}\left(\Delta^{-\frac{1}{4}\cdot(\frac{3}{4})^2}\right)$$
$$= \deg_G(v) \cdot \tilde{O}\left(\Delta^{-37/64}\right) < \tilde{O}\left(\sqrt{\deg_G(v)}\right).$$

Note that in the above calculation we assume v always goes to the left-over part L in all three iterations. If v goes to B_i, then the degree shrinks faster. Remember that we set $q = \tilde{O}(\Delta^{-1/4})$. Hence, we require a total memory of

$$\widetilde{O}\left(\sum_{H\in\mathcal{H}}\sum_{v\in H}(\deg_H(v))^2\right) = \widetilde{O}\left(\sum_{H\in\mathcal{H}}\sum_{v\in H}\deg_G(v)\right)$$
$$= \widetilde{O}\left(\sum_{v\in G}\deg_G(v)\right) = \widetilde{O}(m).$$

Note that the algorithm can be easily adapted to always perform at least three partitioning steps if Δ_H is bounded from below by a sufficiently large $\text{poly}(\log n)$, because then the conditions of Lemma 3.1 are satisfied. On the other hand, if $\Delta_H = \text{poly}(\log n)$, it is follows immediately that $\widetilde{O}\left(\sum_v (\deg_H(v))^2\right) = \text{poly}(\log n) = \widetilde{O}(1)$. Put together, we have $\sum_{H\in\mathcal{H}}\sum_{v\in H}(\deg_H(v))^2 = \widetilde{O}(m)$. □

[9] The choice $\lambda = 3/5$ is to ensure that the number of available colors for each vertex in each subgraph meets the palette size constraint specified in Lemma 3.3.

REFERENCES

[1] Kook Jin Ahn and Sudipto Guha. 2015. Access to Data and Number of Iterations: Dual Primal Algorithms for Maximum Matching Under Resource Constraints. In *Proc. SPAA*. 202–211.

[2] Noga Alon and Joel H. Spencer. 2016. *The Probabilistic Method* (4th ed.). Wiley Publishing.

[3] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. 2014. Parallel Algorithms for Geometric Graph Problems. In *Proc. Symposium on Theory of Computation (STOC)*. 574–583.

[4] Alexandr Andoni, Clifford Stein, Zhao Song, Zhengyu Wang, and Peilin Zhong. 2018. Parallel Graph Connectivity in Log Diameter Rounds. In *Proceedings 59th IEEE Symposium on Foundations of Computer Science (FOCS)*. 674–685.

[5] Sepehr Assadi. 2017. Simple round compression for parallel vertex cover. *arXiv preprint arXiv:1709.04599* (2017).

[6] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. 2019. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *Proceedings 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

[7] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. 2019. Sublinear Algorithms for $(\Delta + 1)$ Vertex Coloring. In *Proceedings 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

[8] Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. 2018. Massively Parallel Algorithms for Finding Well-Connected Components in Sparse Graphs. *arXiv preprint arXiv:1805.02974* (2018).

[9] Leonid Barenboim, Michael Elkin, and Uri Goldenberg. 2018. Locally-Iterative Distributed $(\Delta + 1)$-Coloring below Szegedy-Vishwanathan Barrier, and Applications to Self-Stabilization and to Restricted-Bandwidth Models. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 437–446.

[10] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. 2016. The Locality of Distributed Symmetry Breaking. *J. ACM* 63, 3, Article 20 (2016), 20:1–20:45 pages.

[11] Leonid Barenboim and Victor Khazanov. 2018. Distributed Symmetry-Breaking Algorithms for Congested Cliques. In *Computer Science - Theory and Applications - 13th International Computer Science Symposium in Russia, CSR 2018, Moscow, Russia, June 6-10, 2018, Proceedings*. 41–52.

[12] Paul Beame, Paraschos Koutris, and Dan Suciu. 2013. Communication Steps for Parallel Query Processing. In *Proceedings of the 32Nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*. 273–284.

[13] Paul Beame, Paraschos Koutris, and Dan Suciu. 2014. Skew in Parallel Query Processing. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*. 212–223.

[14] Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. 2017. Near-Optimal Approximate Shortest Paths and Transshipment in Distributed and Streaming Models. In *31st International Symposium on Distributed Computing (DISC 2017) (Leibniz International Proceedings in Informatics (LIPIcs))*, Andréa W. Richa (Ed.), Vol. 91. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 7:1–7:16.

[15] Andrew Berns, James Hegeman, and Sriram V Pemmaraju. 2012. Super-fast distributed algorithms for metric facility location. In *Automata, Languages, and Programming*. Springer, 428–439.

[16] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, and Saeed Seddighin. 2018. Approximating edit distance in truly subquadratic time: quantum and MapReduce. In *Proc. Symposium on Discrete Algorithms (SODA)*. 1170–1189.

[17] Sebastian Brandt, Manuela Fischer, and Jara Uitto. 2018. Breaking the Linear-Memory Barrier in MPC: Fast MIS on Trees with n^ϵ Memory per Machine. *arXiv preprint arXiv:1802.06748* (2018).

[18] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. 2016. Algebraic methods in the congested clique. *Distributed Computing* (2016). https://doi.org/10.1007/s00446-016-0270-2

[19] Keren Censor-Hillel, Dean Leitersdorf, and Elia Turner. 2018. Sparse Matrix Multiplication with Bandwidth Restricted All-to-All Communication. *CoRR* abs/1802.04789 (2018).

[20] Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. 2018. The Complexity of $(\Delta+1)$ Coloring in Congested Clique, Massively Parallel Computation, and Centralized Local Computation. *CoRR* abs/1808.08419 (2018). arXiv:1808.08419 http://arxiv.org/abs/1808.08419

[21] Yi-Jun Chang, Wenzheng Li, and Seth Pettie. 2018. An Optimal Distributed $(\Delta+1)$-coloring Algorithm?. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2018)*. ACM, New York, NY, USA, 445–456. https://doi.org/10.1145/3188745.3188964

[22] Artur Czumaj, Jakub Lacki, Aleksander Madry, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. 2018. Round Compression for Parallel Matching Algorithms. In *Proc. Symposium on Theory of Computation (STOC)*. 471–484.

[23] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation (OSDI)*. USENIX Association, Berkeley, CA, USA, 10–10. http://dl.acm.org/citation.cfm?id=1251254.1251264

[24] Danny Dolev, Christoph Lenzen, and Shir Peled. 2012. "Tri, Tri Again": Finding Triangles and Small Subgraphs in a Distributed Setting. In *Distributed Computing*. Springer, 195–209.

[25] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. 2014. On the Power of the Congested Clique Model. In *Proc. Principles of Distributed Computing (PODC)*. ACM, 367–376.

[26] Manuela Fischer and Mohsen Ghaffari. 2017. Sublogarithmic Distributed Algorithms for Lovász Local Lemma with Implications on Complexity Hierarchies. In *Proceedings 31st International Symposium on Distributed Computing (DISC)*. 18:1–18:16.

[27] Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. 2016. Local conflict coloring. In *Proceedings 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 625–634.

[28] Francois Le Gall. 2016. Further Algebraic Algorithms in the Congested Clique Model and Applications to Graph-Theoretic Problems. In *DISC*.

[29] Mohsen Ghaffari. 2016. An improved distributed algorithm for maximal independent set. In *Proceedings 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 270–277.

[30] Mohsen Ghaffari. 2017. Distributed MIS via All-to-All Communication. In *Proc. Principles of Distributed Computing (PODC)*. 141–149.

[31] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. 2018. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *Proc. Principles of Distributed Computing (PODC)*. arXiv:1802.08237.

[32] Mohsen Ghaffari and Jara Uitto. 2019. Sparsifying Distributed Algorithms with Ramifications in Massively Parallel Computation and Centralized Local Computation. In *Proceedings 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

[33] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. 2011. Sorting, searching, and simulation in the MapReduce framework. In *Proc. ISAAC*. Springer, 374–383.

[34] David G Harris, Johannes Schneider, and Hsin-Hao Su. 2018. Distributed $(\Delta+1)$-Coloring in Sublogarithmic Rounds. *J. ACM* 65, 4 (2018), 19:1–19:21. https://doi.org/10.1145/3178120

[35] Nicholas J. A. Harvey, Christopher Liaw, and Paul Liu. 2018. Greedy and Local Ratio Algorithms in the MapReduce Model. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, New York, NY, USA, 43–52. https://doi.org/10.1145/3210377.3210386

[36] James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. 2015. Toward Optimal Bounds in the Congested Clique: Graph Connectivity and MST. In *Proc. Principles of Distributed Computing (PODC)*. ACM, 91–100.

[37] James W Hegeman and Sriram V Pemmaraju. 2015. Lessons from the congested clique applied to MapReduce. *Theoretical Computer Science* 608 (2015), 268–281.

[38] James W. Hegeman and Sriram V. Pemmaraju. 2015. Lessons from the Congested Clique Applied to MapReduce. *Theor. Comput. Sci.* 608, P3 (Dec. 2015), 268–281. https://doi.org/10.1016/j.tcs.2015.09.029

[39] James W Hegeman, Sriram V Pemmaraju, and Vivek B Sardeshmukh. 2014. Near-constant-time distributed algorithms on a congested clique. In *Distributed Computing*. Springer, 514–530.

[40] Sungjin Im, Benjamin Moseley, and Xiaorui Sun. 2017. Efficient Massively Parallel Methods for Dynamic Programming. In *Proc. Symposium on Theory of Computation (STOC)*. 798–811.

[41] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. *SIGOPS Operating Systems Review* 41, 3 (2007), 59–72. https://doi.org/10.1145/1272998.1273005

[42] Tomasz Jurdzinski and Krzysztof Nowicki. 2018. MST in $O(1)$ Rounds of Congested Clique. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*. 2620–2632.

[43] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A Model of Computation for MapReduce. In *Proc. Symposium on Discrete Algorithms (SODA)*. 938–948.

[44] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. 2011. Filtering: a method for solving graph problems in MapReduce. In *Proc. SPAA*. 85–94.

[45] Christoph Lenzen. 2013. Optimal Deterministic Routing and Sorting on the Congested Clique. In *Proceedings 33rd ACM Symposium on Principles of Distributed Computing (PODC)*. 42–50. https://doi.org/10.1145/2484239.2501983

[46] Christoph Lenzen and Roger Wattenhofer. 2010. Brief Announcement: Exponential Speed-up of Local Algorithms Using Non-local Communication. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*. ACM, 295–296. https://doi.org/10.1145/1835698.1835772

[47] Reut Levi and Moti Medina. 2017. A (centralized) local guide. *Bulletin of EATCS* 2, 122 (2017).

[48] Nathan Linial. 1992. Locality in Distributed Graph Algorithms. *SIAM J. Comput.* 21, 1 (1992), 193–201.

[49] Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. 2005. Minimum-weight spanning tree construction in O($\log \log n$) communication rounds. *SIAM J. Comput.* 35, 1 (2005), 120–131.

[50] Danupon Nanongkai. 2014. Distributed Approximation Algorithms for Weighted Shortest Paths. In *Proc. Symposium on Theory of Computation (STOC)*.

[51] Alessandro Panconesi and Aravind Srinivasan. 1996. On the Complexity of Distributed Network Decomposition. *J. Algor.* 20, 2 (1996), 356–374.

[52] Michal Parnas and Dana Ron. 2007. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science* 381, 1 (2007), 183 – 196. https://doi.org/10.1016/j.tcs.2007.04.040

[53] Merav Parter. 2018. $(\Delta+1)$ coloring in the congested clique model.. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*. 160:1–160:14.

[54] Merav Parter and Hsin-Hao Su. 2018. Randomized $(\Delta+1)$ coloring in $O(\log^* \Delta)$ congested clique rounds. In *Proceedings of the International Symposium on Distributed Computing (DISC)*. 39:1–39:18.

[55] Boaz Patt-Shamir and Marat Teplitsky. 2011. The round complexity of distributed sorting. In *Proc. Principles of Distributed Computing (PODC)*. 249–256.

[56] David Peleg. 2000. *Distributed Computing: A Locality-Sensitive Approach*. SIAM.

[57] Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. 2016. Shuffles and Circuits: (On Lower Bounds for Modern Parallel Computation). In *Proc. SPAA*. 1–12.

[58] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. 2011. Fast Local Computation Algorithms. In *Proceedings of the First Symposium on Innovations in Computer Science (ICS)*. 223–238. See also *CoRR* abs/1104.1377.

[59] Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. 1995. Chernoff-Hoeffding Bounds for Applications with Limited Independence. *SIAM Journal on Discrete Mathematics* 8, 2 (1995), 223–250.

[60] Tom White. 2012. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc.

[61] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*. https://www.usenix.org/conference/hotcloud-10/spark-cluster-computing-working-sets