Local Recurrent Problems in the SUPPORTED Model

Akanksha Agrawal
Indian Institute of Technology Madras, India

John Augustine
Indian Institute of Technology Madras, India

David Peleg
Weizmann Institute of Science, Rehovot, Israel

Srikkanth Ramachandran
Indian Institute of Technology Madras, India

Abstract

We study the SUPPORTED model of distributed computing introduced by Schmid and Suomela [27], which generalizes the LOCAL and CONGEST models. In this framework, multiple instances of the same problem, differing from each other by the subnetwork to which they apply. recur over time, and need to be solved efficiently online. To do that, one may rely on an initial preprocessing phase for computing some useful information. This preprocessing phase makes it possible, in some cases, to obtain improved distributed algorithms, overcoming locality-based time lower bounds.

Our main contribution is to expand the class of problems to which the SUPPORTED model applies, by handling also multiple recurring instances of the same problem that differ from each other by some problem specific input, and not only the subnetwork to which they apply. We illustrate this by considering two extended problem classes. The first class, denoted PCS, concerns problems where client nodes of the network need to be served, and each recurring instance applies to some Partial Client Set. The second class, denoted PFO, concerns situations where each recurrent instance of the problem includes a partially fixed output, which needs to be completed to a full consistent solution.

Specifically, we propose some natural recurrent variants of the dominating set problem and the coloring problem that are of interest particularly in the distributed setting. For these problems, we show that information about the topology can be used to overcome locality-based lower bounds. We also categorize the round complexity of Locally Checkable Labellings in the SUPPORTED model for the simple case of paths. Finally we present some interesting open problems and some partial results towards resolving them.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Distributed algorithms

Keywords and phrases Distributed Algorithms, LOCAL Model, SUPPORTED Model

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2023.22

Related Version *Full Version*: `https://arxiv.org/abs/2212.14542`

Funding John Augustine and Srikkanth Ramachandran are supported by the Centre of Excellence in Cryptography, Cybersecurity, and Distributed Trust (CCD) and by an IITM-Accenture project (SB/22-23/007/JOHN/ACC).

Acknowledgements David Peleg holds the Venky Harinarayanan and Anand Rajaraman (VHAR) Visiting Chair Professorship at IIT Madras. This work was carried out in part during mutual visits that were supported by the VHAR Visiting Chair funds.

1 Introduction

1.1 Background and motivation

The area of distributed network algorithms concerns the development and analysis of distributed algorithms operating on a network of processors interconnected by communication links. In particular, a substantial body of research has been dedicated to the development of various graph algorithms for problems whose input consists of the network topology. Examples for such problems are finding a maximal independent set (MIS) for the network, finding a maximal or maximum matching (MM), a minimum dominating set (MDS), a proper coloring with few colors, and so on, and considerable efforts were invested in developing sophisticated and highly efficient algorithms for these problems. Such algorithms are particularly significant in settings where the distributed network at hand is *dynamic*, and its topology keeps changing at a high rate.

The observation motivating the current study is that in many practical settings, the network itself may be static, or change relatively infrequently. In such settings, problems depending solely on the graph structure need be solved only once. In contrast, there are a variety of other problems, related to computational processes that occur repeatedly in the network, which need to be solved at a much higher frequency, and whose input consists of the network topology together with some other (varying) elements. For such problems, the traditional model might not provide a satisfactory solution, in the sense that it may be unnecessarily expensive to solve the entire problem afresh for each instance. Rather, it may be possible to derive improved algorithmic solutions that take advantage of the fact that the network topology is static. We refer to such problems as *recurrent problems*.

We envision that depending on the desired problems that the network needs to support, one can compute and store additional information about the topology of the network within each node, to enable recurrent problems to be solved faster. Inherently this captures an aspect of network design. When a network is built, it maybe useful to compute useful information about its topology keeping in mind the recurrent problems that it must support during its lifetime.

This framework has been studied in the literature as the SUPPORTED model [27]. However, the recurrent problems studied so far within this framework were mostly limited to instances of the original problem defined on an edge induced subgraph of the original graph[1].

We believe that this limited class of problems does not fully capture all recurrent problems of interest that can arise in the SUPPORTED model. To demonstrate this, we introduce two new classes of recurring problems that can occur in the SUPPORTED model, namely, problems with *partial client set (PCS)* and problems with *partially fixed output (PFO)*, defined below. We illustrate these classes by focusing on natural extensions of the classical local problems of coloring and dominating set, and developing algorithms and lower bounds for them in the SUPPORTED model.

1.2 Recurrent Problems

We consider graph-related optimization problems each of whose *instances* $\langle G, \mathsf{S}\rangle$ consists of a network topology $G = (V, E)$, on which the distributed algorithm is to be run, and some problem-specific input S. The term "recurrent problem" refers to a setting where the

[1] One recent exception [19] introduced a different type of recurrent problem concerning matrix multiplication, where the structure of the input matrices is fixed but the specific values of the nonzero elements are different in each recurring instance. The problem is studied in the (supported) node-congested clique model of distributed computing.

network G is fixed, and is the same in all instances (hence we often omit it). Formally, there is a stream of instances that arrive from time to time and must be solved efficiently. The optimization problem itself may be a variant of some classical graph optimization problem, except it has some additional restrictions, specified in each instance S. Two concrete types of restrictions that are of particular interest are *partial client set (PCS)* and *partially fixed output (PFO)*.

Partial client set (PCS)

An instance S restricted in this manner specifies a subset $C \subseteq V$ of *client vertices* to which the problem applies. The rest of the vertices are not involved (except in their capacity as part of the network). For example, consider the maximum matching problem. In the PCS variant of this problem, a PCS-restricted instance will specify a vertex subset C such that the matching is only allowed (and required) to connect vertex pairs of C.

Partially fixed output (PFO)

An instance S restricted in this manner specifies a part of the output. The rest of the output must be determined by the algorithm. For example, consider the k-centers problem (where the goal is to select a subset C of k vertices serving as centers, so as to minimize the maximum distance from any vertex of V to C). In the PFO variant of the k-centers problem, a PFO-restricted instance will specify a vertex subset C_{pre} of k' vertices that were already pre-selected as centers, and the task left to the algorithm is to select the remaining $k - k'$ centers.

Naturally, some recurrent problems may involve other novel restrictions as well as hybrids, thereby opening up the possibility for rich theory to be developed.

1.2.1 Two representative examples: CDS and PCC

In this paper, we will focus on two concrete examples for recurrent problems of practical significance, and use them for illustration. The first of these two example problems, named CDS, serves to illustrate a recurrent problem with PCS-restricted instances (where the set of clients changes in each instance). The second problem, named CC, illustrates a recurrent problem with PFO-restricted instances (where parts of the output are fixed in advance in each instance).

Minimum client-dominating set (CDS)

In certain contexts, a dominating set D in a network G (i.e., such that every vertex $v \in V$ either belongs to D or has a neighbor in D) is used for placing *servers* providing some service to all the vertices in the network (interpreted as *clients*), in settings where it is required that each vertex is served by a server located either locally or at one of its neighbors. The *minimum dominating set (MDS)* problem requires finding the smallest possible dominating set for G.

We consider the recurrent variant of the CDS problem with PCS-restricted instances. This problem arises in settings where the set of clients in need of service does not include all the vertices of G, but rather varies from one instance to the next. In such settings, the network G is static, and from time to time, a set of clients $C \subseteq V$, formed in an ad-hoc manner due to incoming user requests, requests to select and establish a (preferably small) subset D of vertices from among their neighbors, which will provide them some service. In other words, the set D is required to dominate the vertices in C. On the face of it, solving the minimum dominating set problem once on G may be useful, but not guarantee optimal results for each recurrent instance S; rather, for each instance S, it may be necessary to solve the specialized problem once the request appears in the network. Hereafter, we refer to this problem as *minimum client-dominating set (*MCDS*)*.

Note that one may also consider a generalized problem that includes also a PFO component, by specifying in each instance S also a partial set D' of vertices that were pre-selected as servers (or dominators). Our results are presented for the MCDS problem (without PFO restrictions), but it should be clear that they can easily be extended to the generalized problem with PFO restrictions[2].

Color Completion (CC)

In certain contexts, a proper coloring of a distributed network is used for purposes of scheduling various mutually exclusive tasks over the processors of the network. For example, suppose that performing a certain task by a processor requires it to temporarily lock all its adjacent links for its exclusive use, preventing their use by the processor at the other end. Employing a proper coloring as the schedule (in which all the processors colored by color t operate simultaneously at round t) enables such mutually exclusive operation. Naturally, it is desirable to use as few colors as possible, in order to maximize parallelism.

We consider the recurrent variant of the coloring problem with PFO-restricted instances. From time to time we may receive a partial (collision-free) coloring assignment to some subset $C \subseteq V$ of the vertices, representing processors constrained to operate on some particular time slots. We are required to color all the remaining vertices in $V \setminus C$ properly and consistently with the initial coloring. Typically, using colors already occurring in the precoloring (i.e., used by some vertices in the set C) is fine, since these time slots are already committed for the task at hand. However, it is desirable to use as few new time slots (or new colors), to minimize the overall time spent on the task.

Note that one may also consider a generalized problem that includes also a PCS component, by specifying in each instance S also a partial set V' of vertices that are interested in being scheduled, and hence need to be colored. Our results are presented for the CC problem (without PCS restrictions), but it should be clear that they can easily be extended to the generalized problem with PCS restrictions[3].

1.3 The SUPPORTED model

The SUPPORTED model is an extension of the well studied LOCAL and CONGEST models with an additional preprocessing phase. Specifically the solution to a problem in the SUPPORTED model consists of two stages, (i) a *preprocessing* stage and (ii) an *online* stage.

- In the preprocessing stage, run an algorithm $\mathcal{A}_{\text{pre}}(G)$ on the topology of the network G and obtain information $\mathsf{Inf}(G)$ to be stored at the network vertices (different vertices may of course store different information).
- During runtime, a stream of instances arrive. Whenever a new instance S arrives, run a distributed algorithm $\mathcal{A}(\mathsf{S}, \mathsf{Inf}(G))$ to solve this problem instance.

In view of the fact that the preprocessing stage takes place only once, the particulars of the preprocessing algorithm are less important to us, and we allow it to be arbitrary (even oracular). For the scope of this paper, in the upper bounds that we show, all our preprocessing phases are explicitly computable, whereas the lower bounds hold for any arbitrary preprocessing.

[2] Essentially, for this problem, the pre-selected vertices of D' can be used to satisfy all the clients that neighbor them, leaving us with a smaller set C' of unsatisfied clients that need to be covered.

[3] Essentially, for this problem, the vertices of $V \setminus V'$, which do not require coloring, can simply avoid participating in the coloring process.

In the online stage, we insist that the computation performed by each node in a single round must be polynomial in the size of the graph. Therefore even knowledge of the complete network for each node might not be sufficient, as underlying information about the topology (such as chromatic number) might not be computable in polynomial time.

For a given problem Π on a graph G, one may seek to optimize several parameters. For the scope of this paper, we consider only two, (i) the round complexity of the online algorithm, i.e., the number of synchronous rounds required to solve each recurrent instance and (ii) the size of the output to each node in the preprocessing phase, i.e., the amount of additional information that needs to be stored in each node of the graph from the preprocessing phase.

1.4 Our Contributions

Client Dominating Set (Section 2). We first show that even on a path of n nodes, it is not possible to (i) optimally solve CDS in $o(n)$ time and (ii) compute a $1+\epsilon$ approximation in $\Omega(\epsilon^{-1})$ rounds (whenever $\epsilon = \Omega(1/D)$). On the other hand, we show that for trees and planar graphs, one can obtain a $1+\epsilon$ approximation in $O(\epsilon^{-1})$ and $O(\text{poly}(\epsilon^{-1}))$ rounds respectively. The algorithm for trees decomposes it into blocks of depth $\Theta(\epsilon^{-1})$ and executes an optimal algorithm within each block. The algorithm for planar graphs is an adaptation of the $O(\text{poly}(\epsilon^{-1}) \log^* n)$ round LOCAL algorithm of [14]. We look at the most time consuming part of their algorithm and we speedup a particular sub-routine. In order to do that, we make use of a combinatorial object called *non-repetitive* coloring, first proposed by [2] who conjectured that it is bounded in planar graphs and recently [15] showed an upper bound of 768. To the best of our knowledge, this is the first application of such a coloring in the distributed setting.

Color Completion (Section 3). We provide an algorithm to complete a given coloring using at most $\chi(\Delta+1)/k$ new colors in k rounds, for any $1 \leq k \leq \chi$. We show that for $k = 1$, the number of colors used is asymptotically tight in the worst case. Tighter analysis of the same algorithm when $k = \chi$ shows that only χ new colors are needed when $k = \chi$.

Locally Checkable Labellings (Section 4). We study a generic class of problems called Locally Checkable Labellings (LCL) [26]. We show that on a path, every LCL problem either has worst case complexity $O(1)$ or $\Theta(n)$. In the specific case of recurrent problems where the online instances are a specific LCL on a sub-path of the given path, we provide an efficient centralized algorithm to classify the LCL into one of the two cases and also construct the distributed algorithm to solve an LCL given its description.

Miscellaneous Problems (Section 5). Finally, we provide some partial results on sub-graph maximal matching and sub-graph maximal independent set that could potentially be useful in finding optimal solutions for these problems in the SUPPORTED model.

1.5 Related Work

The SUPPORTED model was first proposed by [27]. [16] provide several results including lower bounds for problems such as sinkless orientation and approximating independent set. [20] provide near optimal algorithms for global network optimization problems, such as minimum spanning tree, min-cut etc.

Dominating Set. [14] provided an $O(\text{poly}(\epsilon^{-1}) \log^* n)$ round algorithm for a $1+\epsilon$ approximation for the dominating set problem and it was later extended to bounded genus graphs by [3]. [16] briefly discuss about extending these results to the SUPPORTED model.

Coloring. Color Completion has been one of the methods used for $\Delta + 1$ coloring graphs in $\log^* n + f(\Delta)$ rounds. Existing algorithms decide on a coloring for a sub-graph of the given graph and then recursively complete the chosen coloring. [7] provided the first sub-linear in Δ algorithm. The current best known algorithm has round complexity $\log^* n + O(\sqrt{\Delta \log \Delta})$ (see [24, 8, 17]). [24] also provided a smooth tradeoff between the number of colors and the round complexity, specifically in $k + \log^* n$ rounds, graphs can be properly colored using $O(\Delta^2/k^2)$ colors for any $1 \leq k \leq \sqrt{\Delta}$. We note that Maus's algorithm ([24]) does not provide a $\Delta + 1$ coloring but rather an $O(\Delta)$ coloring.

LCL. Locally Checkable Labellings (LCL) were first proposed by [26]. [11] showed gaps in the deterministic complexity of LCL's. They showed that the worst case deterministic round complexity of LCL's on any hereditary graph class is either $\omega(\log_\Delta n)$ or $O(\log^* n)$. They also show that for paths, there is no LCL with complexity $o(n)$ and $\omega(\log^* n)$. Later [12] and [18] showed that on trees, the deterministic worst case complexities for LCL's is either $O(1), \Theta(\log^* n), \Theta(\log n)$ or $n^{\Theta(1)}$. More recently, [6] showed that for a more restricted class of LCL problems, on rooted trees, there is a centralized algorithm that takes as input the description of the LCL and decides which of the above complexity classes it belongs to. Given the LCL, deciding its distributed complexity class on trees was shown to be EXPTIME hard by [10].

[13] studied the complexity of LCL's on unlabelled paths and cycles. For this easier case, they showed the correspondence between a solution to the LCL and a non-deterministic finite automaton. This observation helps to identify polynomial-time computable properties of the LCL which almost exactly determine its worst round complexity, with the exception of a case that turns out to be co-NP complete. Our characterisation of LCL problems in SUPPORTED in Section 4 follows along the same ideas. [9] study the round complexity of LCL problems in two dimensional grid and toroids. Even though these topologies are simple extensions of paths and cycles, and the complexity classes of LCL problems are exactly the same $(O(1), \Theta(\log^* n), \Theta(n))$, characterizing the round complexity is undecidable.

2 Dominating Sets

2.1 Client Dominating Set

▶ **Definition 1** (Client Dominating Set (CDS)). *Given a graph G and a subset of its vertices $C \subseteq V(G)$, called the client set, we say that a set $D \subseteq V(G)$ dominates C if for every client $c \in C$, there exists $v \in D$ such that either $v = c$ or v is a neighbor of c. D is said to be a client dominating set of G for the clients C.*

The minimum client dominating set problem (MCDS) asks for a CDS of minimum size. The CDS problem is of course a generalization of the classical Dominating Set problem as the dominating set is precisely the case when $C = V(G)$. It is also possible to reduce the CDS problem to an instance of the Dominating Set problem. See the full paper [1] for a discussion of the reductions. Our reduction does not preserve *locality* and so it does not provide any insight into the round complexity of this problem in distributed settings.

2.2 Lower Bound for Paths

We establish two lower bounds for MCDS on a path. First, we argue that, regardless of the preprocessing, the online runtime of every (exact) deterministic distributed algorithm for the MCDS problem must take time $\Omega(D)$ on networks of diameter D. Second, we show that the online runtime of every deterministic distributed approximation algorithm for MCDS with ratio $1+\epsilon$ must require time $\Omega(1/\epsilon)$ on some input. The proofs of these theorems appear in the full paper [1].

▶ **Theorem 2.** *Let $\mathcal{A}$ be a deterministic distributed local algorithm for* CDS *with arbitrary preprocessing. Then there exists some input for which $\mathcal{A}$ requires $\Omega(D)$ time.*

▶ **Theorem 3.** *Let $\mathcal{A}$ be a deterministic distributed local approximation algorithm for* CDS*, with arbitrary preprocessing, whose online runtime on every path and every instance is at most $k = 4\ell + 1$ for some integer $\ell \geq 1$. There exists a network and a set of clients for which the approximation ratio of $\mathcal{A}$ is at least $1 + 1/(k+2)$.*

2.3 A CTAS for Trees

In this section we describe a *constant time approximation scheme* (CTAS) for MCDS on trees, i.e., a $1+\epsilon$ approximation in $O(\epsilon^{-1})$ rounds.

The algorithm for trees is based on a preprocessing stage in which the tree is partitioned into subtrees of depth $O(k)$ for some integer parameter k. Each recurrent instance is then solved by computing a local near-optimal CDS on each subtree, while taking care to ensure that the resulting solutions combine into a $1 + 4/(k-1)$ approximation of the optimal global solution. The "interface" between adjacent subtrees is more difficult to handle, as making a single change in the selection in one subtree (e.g., in one of its leaves) might affect several adjacent subtrees, which makes both the algorithm and its analysis somewhat more complex.

Let us first describe the preprocessing stage, which is applied to the network tree T. The algorithm has an integer parameter $\ell \geq 1$ and sets $k = 4\ell + 1$. Root the tree T at a root vertex r_0, and mark each vertex v by its layer, $layer(v)$, namely, its distance from r_0 (by definition $layer(r_0) = 0$). Partition the tree T into subtrees by taking every vertex v with $layer(v) = pk$ for integer $p \geq 0$ as a root and defining $T[v]$ as the subtree of depth k rooted at v. See Fig. 1(a). For notational convenience, we sometimes use $T[v]$ to denote also the *vertex set* of the subtree $T[v]$. Also, for any subtree $T[v]$ and vertex set $X \subseteq T$, we denote $X[v] = X \cap T[v]$.

The leaves of a subtree $T[v]$ can be classified into *real leaves* and *layer-leaves*, namely, leaves of $T[v]$ that are internal nodes in T. A subtree that has no other subtree below it (namely, all of whose leaves are real) is called a *leaf-subtree* or simply L-tree. Otherwise, it is called an *internal-subtree* or I-tree. (See Fig. 1(a).) We partition the vertices of T into two subsets. Let *lleaves* be the set of all layer-leaves, and *int* be the set of all remainig vertices. This induces a partition of the vertices of each subtree $T[v]$ into $lleaves[v]$ and $int[v]$. (For an L-tree, $lleaves[v] = \emptyset$.)

During the recurrent stage, each instance consists of a set S of clients. This induces additional distinctions on the tree structure. Internal subtrees are classified into two types. The I-tree $T[v]$ is called a *cut I-tree* if on every path from v to a root w hanging from a layer-leaf of $T[v]$ there are two consecutive vertices that do not belong to S. See Fig. 1(b). The figure also illustrates the fact that in a cut I-tree $T[v]$ one can identify a subtree $\check{T}[v]$, referred to as the *peak* of $T[v]$, the minimal sub-tree with the property that for every edge (x, y) connecting a vertex $x \in \check{T}[v]$ to a child $y \notin \check{T}[v]$, both $x, y \notin S$. This implies that nodes *below* $\check{T}[v]$ cannot help in dominating clients in $\check{T}[v]$, namely, taking them into D cannot dominate client vertices in $\check{T}[v]$. $T[v]$ is a *full I-tree* if it is not a cut I-tree, namely, there is at least one path from v to a root w hanging from some layer-leaf of $T[v]$ with no two consecutive vertices of $V \setminus S$.

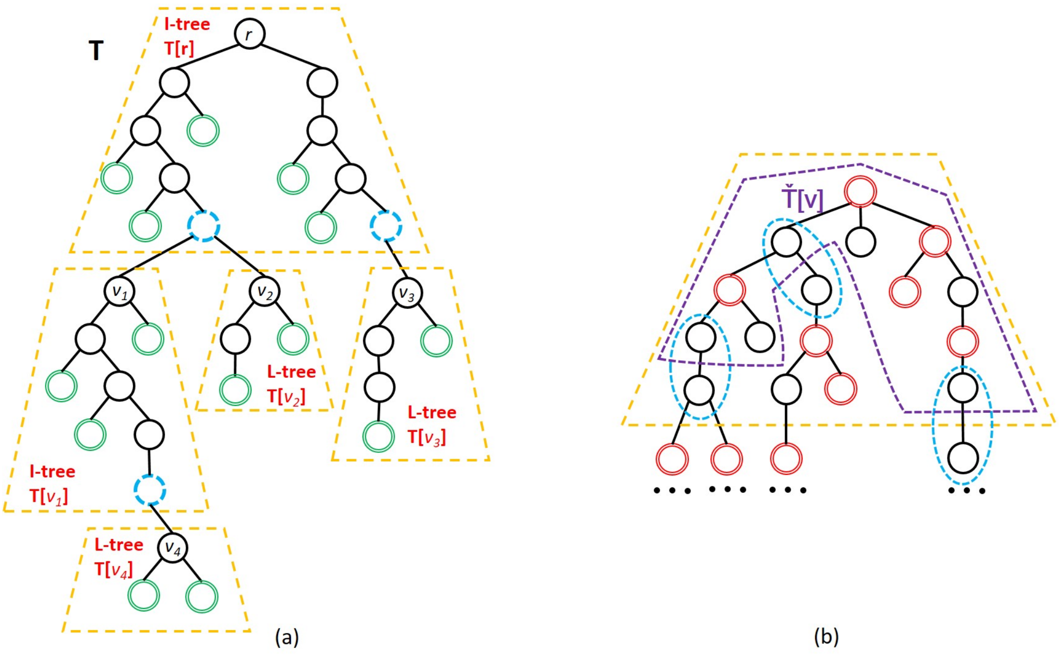

Figure 1 (a) A decomposition of the tree T into subtrees for $\ell = 1$, $k = 5$. Layer-leaves are marked by a blue dashed circle, and real leaves are marked by a green double circle. (b) A cut I-tree, $k = 5$. The client vertices of S are drawn as double red circles. The cuts along root-to-root paths are marked by blue dashed ellipses. The peak-tree $\check{T}[v]$ is marked by a purple dashed curve.

The idea behind the approximation scheme is as follows. Our algorithm solves the CDS problem *separately*, in an optimal manner, on each subtree $T[v]$ of depth at most k for the client set $S[v]$. This can be done in time $O(k)$, but might entail inaccuracies. As illustrated in the lower bound of Sect. 2.2, the main hindrance to the accuracy of a local distributed algorithm for MCDS stems from long paths with a periodic occurrence of client vertices. Such a path, threading its way along some root-to-leaf path in T, might be handled poorly by the local computations. Our goal is to bound the loss by at most 1 per subtree in the decomposition. This is justified for full I-trees, since in a full I-tree the optimum solution D^* must also use $\Omega(k)$ dominators to cover all the clients, so the inaccuracy ratio is just $1/\Omega(k)$.

This approach is made complicated due to the fact that some subtrees are not full, and may require only a small number of dominators. For such subtrees (specifically, L-trees and cut I-trees), we cannot allow the algorithm to "waste" more than the optimum solution. Hence when comparing the number of dominators used by the algorithm to that of the optimum D^*, we must use an accounting method that will equate the costs over L-trees and cut I-trees, and charge all the "waste" to full I-trees.

This is done as follows. In a first phase, we locally solve the problem optimally in each L-tree and cut I-tree, i.e., dominate clients in these sub-trees using only vertices of these sub-trees. This is only used in order to decide, for each such subtree $T[v]$, whether the root's parent, denoted parent(v), must belong to the dominating set. This is important since these vertices cover the "interference layers" between adjacent subtrees. For the full I-trees, an optimal local solution cannot be computed. Therefore, we artificially impose a "waste" in every full I-tree $T[v]$, by selecting the parent of its root, parent(v), as a dominator, whether or not necessary. As explained above, this "waste" is justified by the fact that D^* must also use

$\Omega(k)$ dominators in these subtrees. As a result, when we compute a dominating set for the remaining undominated clients in the second phase of the algorithm, the solution computed by the algorithm on each subtree T' is no greater than the number of D^* dominators in T'.

Optimal procedure P_{up}

A simple procedure P_{up} we use is an optimal algorithm for CDS on rooted trees, which runs in time $O(\mathsf{depth}(T))$ on a tree T. The algorithm starts with an empty set of dominators D and works its way from the leaves up, handling each node w only after it finishes handling all its children. It adds w to the set D in one of the following two cases:

(1) Some of w's children are clients and are not yet dominated, or

(2) w itself is an undominated client and is the root.

It is easy to verify that this algorithm yields a minimum cardinality solution for CDS. It is also easy to implement this greedy algorithm as an $O(\mathsf{depth}(T))$ time distributed protocol.

Modification for subtrees. When applying this procedure to a subtree $T[v]$ of T where v is not the root of T, we make the following small but important modification: When the procedure reaches v itself, if $v \in S$ and v is still non-dominated, then we add $\mathsf{parent}(v)$ instead of v to the solution. (This can be done since $\mathsf{parent}(v)$ belongs to the tree T, although it is outside the subtree $T[v]$.)

The above modification is enough to obtain an approximation scheme. For complete pseudocode and analysis, see the full paper [1]. We get the following result.

▶ **Theorem 4.** *For every positive integer k, there exists a deterministic distributed local approximation algorithm for* CDS*, with preprocessing allowed, whose online runtime on every n-vertex tree and every instance is at most $O(k)$ with approximation ratio of at most $1 + \frac{4}{k-1}$.*

2.4 A CTAS for MCDS on Planar Graphs

2.4.1 Constant Approximation for MCDS on Planar Graphs

The state of the art algorithm for constant round planar dominating set approximation in the LOCAL model achieves an approximation ratio of 20 by a recent work of [21]. Their algorithm and analysis extend to the client dominating set problem with slight modifications. See Algorithm 1 for the pseudocode.

Algorithm 1 Constant Approximation for MCDS in Planar Graphs.

```
1: $C \leftarrow$ client set
2: For every $A \subseteq V(G)$, define $N_C(A) = \{w \mid w \in C$ and $(w,v) \in E(G)$ for some $v \in A\}$
3:                         $N_C[A] = N_C(A) \cup (A \cap C)$
4: $D_1 \leftarrow \{v \in V(G) \mid \forall A \subseteq V(G) \setminus \{v\}, N_C[A] \supseteq N_C(v) \Rightarrow |A| \geq 4\}$
5: For every $v \in V(G)$, compute $B_v = \{w \in V(G) \setminus D_1 \mid |N_C(v) \cup N_C(w)| \geq 10\}$
6: $D_2 \leftarrow \{v \in V(G) \setminus D_1 \mid B_v \neq \emptyset\}$
7: $D_3 \leftarrow C \setminus N_C[D_1 \cup D_2]$
8: Return $D_1 \cup D_2 \cup D_3$
```

▶ **Theorem 5.** *Algorithm 1 provides a 39-approximation for the* MCDS *problem in planar graphs.*

2.4.2 A $1+\epsilon$ approximation

We adapt the distributed $1+\epsilon$-approximation scheme of [14], whose round complexity is $O(\left(\frac{1}{\epsilon}\right)^c \log^* n)$ where $c = \log_{24/23} 3$.

The only non-constant round part of their algorithm is in 3-coloring several pseudo-forests[4] that are obtained as follows. A partition of the vertices of the graph is computed such that each part induces a connected component of diameter $O(\text{poly}(\epsilon^{-1}))$. Each component is then contracted into a single node and a pseudo-forest of the resulting graph is computed by a greedy procedure. The choice of the pseudo-forest and the partition is dictated by the clients, and thus hard to predict without knowledge of the clients.

We first describe the major changes required to adapt this method to the CDS problem. We then discuss the preprocessing that helps to 3-color the pseudo-forests in $O(\text{poly}(\epsilon^{-1}))$ rounds, thus removing the $\log^* n$ factor.

Adapting to CDS. First, we remove the edges that are not incident on any client. These edges do not contribute to the criteria for a set to be a dominating set, and they can be ignored. We then compute a constant approximation $\tilde{D}$ as per Algorithm 1. The initial clustering is obtained by choosing for each client c an arbitrary dominator of c from $\tilde{D}$ and contracting the edge between them. Additionally, every vertex that is neither a client nor a dominator chooses an arbitrary neighboring client and the edge between them is merged. The remaining steps are identical to the previous procedure.

Speeding up using a preprocessing phase. One natural candidate to consider for the preprocessing stage is a proper 4-coloring of the planar graph. Unfortunately, while a coloring of any graph remains valid after the removal of edges or vertices, it does not remain valid after contractions. An arbitrary precomputed coloring might not be of much use in coloring the contracted graphs that arise from repeated contractions. To accommodate contractions, we precompute a *non-repetitive* coloring of G (which is the only output of our preprocessing phase). A *non-repetitive* coloring is a coloring of the graph such that for any odd length simple path, the ordered set of colors in the first half of the path is different from that of the second half. Non-repetitive colorings were first proposed by [2]. The minimum number of colors required to realise a non-repetitive coloring is called the Thue number of the graph and is denoted by $\pi(G)$. [15] showed recently that $\pi(G) \leq 768$ for all planar graphs G.

Suppose we have a pseudo forest F that needs to be 3-colored and suppose F is obtained from G_t, the contracted graph. Let $\mathsf{out}(v)$ denote the other end of the outgoing edge of v in F. In order to 3-color the forest, it is sufficient to choose colors in such a way that $\mathsf{out}(v)$ and v have different colors, for every v. We can associate with each node v of G_t, a connected component (denoted G_v) in the original graph G that contains the ends of all edges that were contracted to v. Choose any edge e that crosses G_v and $G_{\mathsf{out}(v)}$. Construct a spanning tree of G_v and root it at the endpoint $r(v)$ of e that lies in G_v. We now color v with the ordered set of non-repetitive colors traced on the unique path from $r(v)$ to $r(\mathsf{out}(v))$, excluding $r(\mathsf{out}(v))$, in the graph $G_v \cup G_{\mathsf{out}(v)} \cup \{e\}$. We enumerate these colors from 1 to 768^{d+1} where d is the maximum diameter of the clusters. Let the computed path be P_v. Observe that whenever $\mathsf{out}(\mathsf{out}(v)) \neq v$, the paths P_v and $P_{\mathsf{out}(v)}$ can be concatenated to form a simple path in the graph G. If P_v and $P_{\mathsf{out}(v)}$ have different lengths, then the colors assigned to them are different. Otherwise, by the property of a non-repetitive coloring, the

[4] A pseudo-forest is a directed graph wherein every node has exactly one outgoing edge.

ordered set of colors of P_v and $P_{\mathsf{out}(v)}$ must be different. When $\mathsf{out}(\mathsf{out}(v)) = v$, we have a 2-cycle. In this case we color one of the nodes $\{v, \mathsf{out}(v)\}$ (whichever has higher id, say v) with its own non-repetitive color and redefine $P_v = \{\mathsf{out}(v)\}$. Now the paths P_v and $P_{\mathsf{out}(v)}$ may be concatenated to obtain a simple path P. See Algorithm 2 for the pseudo-code.

Algorithm 2 3-coloring pseudo-forest.

```
1:  procedure 3-COLOR
    Input:
    (i) color : $V(G) \to [768]$, A non-repetitve coloring of the given planar graph $G$
    (ii) cluster : $V(G) \to \mathbb{N}$, describes a partitioning of the vertices of $G$ that induce connected
    components of diameter at most $d$
    (iii) $G_t$ : the graph where every cluster is contracted to a single node.
    (iv) out : $V(G_t) \to V(G_t)$, describes a pseudoforest in the graph $G_t$ ▷ out$(v)$ is the other
    end of the unique outgoing edge from $v$
    Output: color$_f$ : $V(G_t) \to [3]$, a proper 3-coloring of the given pseudoforest
2:      for all clusters $v \in V(G_t)$ (in parallel) do
3:          $p \leftarrow$ out$(v)$, the parent of $v$ in pseudo-forest of $G_t$
4:          Let $G_v, G_p$ be the connected components of $G$ that are contracted to $v, p$ in $G_t$
5:          $e_v \leftarrow$ any edge in $G$ that crosses $G_v, G_p$ and $r_v \leftarrow$ the end of $e$ in $G_v$
6:          $T_v \leftarrow$ Any spanning tree of $G_v$, rooted at $r_v$
7:      end for
8:      for all clusters $v \in V(G_t)$ (in parallel) do
9:          if out$(p) \neq v$ or $v < p$ then                    ▷ detect cycles of length 2
10:             path$(v) \leftarrow$ The unique path from $r_v$ to $r_p$ in the graph $T_v \cup T_p \cup \{e\}$
11:         else                        ▷ Treating the case of cycle of length 2 separately
12:             path$(v) \leftarrow \{r_v\}$
13:         end if
14:     end for
15:     color$_f(v) \leftarrow$ the ordered set of colors in path$(v)$
16:     Enumerate color$_f(v)$ using integers from 1 to $768^{d+1}$
17:     Reduce color$_f(c)$ to a 3-coloring using the Cole-Vishkin Algorithm
18:     return color$_f$
19: end procedure
```

We now have a 768^{d+1} coloring of the pseudo-forest F, which can then be reduced to a 3-coloring using the Cole-Vishkin Algorithm. The complexity is $O(d \log^* 768^{d+1}) = O(d \log^* d)$. This leads us to our main lemma:

▶ **Lemma 6.** *Given a clustering of G into connected components of diameter d, let G' denote the graph obtained after contracting each cluster into a single vertex, and let H be a given pseudo-forest subgraph of G', Algorithm 2 provides a 3-coloring of H in $O(d \log^* d)$ rounds.*

Algorithm 2 is the main unique ingredient to our adaptation of [14]'s algorithm. Plugging this component into their algorithm directly leads to an $O(\text{poly}(\epsilon^{-1}))$ round algorithm. For concreteness, the complete clustering procedure is described in Algorithm 3 with some minor changes to account for the clients. Once the clustering is done, we proceed in the same way, i.e., solve the CDS problem optimally and independently within each cluster. Solving CDS exactly requires NP-Hard problems to be solved in the online phase, which may be undesirable. This can be fixed by replacing the optimal solution with a PTAS in planar graphs for the CDS problem by a similar adaptation of Baker's algorithm [4].

Algorithm 3 Clustering for Planar CDS.

```
    Input: Client set $C$, a non-repetitive coloring of $G$ and $\epsilon$.
    Output: A $1+\epsilon$ approximation of the optimal set dominating $C$.
    Phase 1: Finding a good initial clustering.
 1: Remove all edges that do not have a client incident on them.
 2: Remove isolated vertices after previous step.
 3: Compute a constant approximation $D^\star$ for $C$ using Algorithm 1.
 4: for all nodes $v \in V(G) \setminus D^\star$ do
 5:     if $v$ has a neighbor in $D^\star$ then
 6:         $u \leftarrow$ any neighbor in $D^\star$
 7:     else
 8:         $u \leftarrow$ any neighbor in $C$ or $\perp$ (if such a node doesn't exist)
 9:     end if
10:     Contract the edge $e = (u, v)$, if $u$ exists
11: end for
            ▷ Done in parallel and implicitly, i.e., contracted vertices know their neighbors
    Phase 2: Improving the clustering
12: $G_0 \leftarrow$ underlying simple graph obtained at end of Phase 1.
13: Set $\mathsf{wt}(e) \leftarrow 1$ for all $e \in E(G_0)$
14: for all $t = 0, 1, \ldots \lceil \log_{24/23} \frac{234}{\epsilon} \rceil$ do
15:     $\mathsf{out}(u) \leftarrow$ any neighbor $v$ such that $\mathsf{wt}((u, v))$ is maximized
16:     $H \leftarrow$ induced by the edges $\{(\mathsf{out}(u), u) \mid u \in G_t\}$        ▷ Heavy pseudo-forest
17:     $\mathsf{col} \leftarrow$ 3-coloring of $H$ obtained using Algorithm 2.
18:     for all $u \in H$ with $\mathsf{col}(u) = 1$ (in parallel) do
19:         $I_u, O_u \leftarrow \{(u, v) \mid u = \mathsf{out}(v)\}, \{(u, v) \mid v = \mathsf{out}(u)\}$
20:         Remove either $I_u$ or $O_u$ from $H$, whichever has smaller total weight
21:     end for
22:     for all $u \in H$ with $\mathsf{col}(u) = 2$ (in parallel) do
23:         $I_u, O_u \leftarrow \{(u, v) \mid u = \mathsf{out}(v), \mathsf{col}(v) = 3\}, \{(u, v) \mid v = \mathsf{out}(u), \mathsf{col}(v) = 3\}$
24:         Remove either $I_u$ or $O_u$ from $H$, whichever has smaller total weight
25:     end for
26:                    ▷ $H$ now consists of connected components with diameter at most 10.
27:     $F \leftarrow$ rooted spanning forest of $H$
28:     $E_F, O_F \leftarrow$ edges of $F$ at even and odd depths respectively
29:     Remove either $E_F$ or $O_F$, whichever has smaller total weight
30:     For all edges $e \in E(H)$, contract $e$ in $G_t$
31:     $G_{t+1} \leftarrow$ underlying simple graph obtained after contractions.
32:     For all edges $e = (u, v) \in G_{t+1}$, set $\mathsf{wt}(e) \leftarrow$ number of edges between $u, v$ after all
    contractions of edges in $H$.
33: end for
34: return $G_T$
```

▶ **Theorem 7.** *For every planar graph G, $1 + \epsilon$ approximation of* MCDS *can be obtained in $O(\epsilon^{-c} \log^* \epsilon^{-1})$* SUPPORTED *rounds, where $c = \log_{24/23} 3$, using only $O(1)$ additional bits stored in each node as the output of the preprocessing phase.*

The complete pseudocode and proof of completeness appear in the full paper [1].

3 Color Completion

Consider a graph $G(V, E)$ and a coloring $c : V \mapsto \{1, \ldots, k\}$. The vertex v is *properly colored* if each of its neighbors has a different color. The classical vertex coloring problem requires deciding if there exists a coloring for which all vertices are properly colored. When some of the vertices are already assigned a predefined coloring, the resulting recurrent problem is referred to as *color completion* (CC). We use the following measures for evaluating the number of colors used in any valid solution.

- Let $\mathcal{P}_{pc}$ be the set of colors used by the precolored vertices, and denote $\chi_{pc} = |\mathcal{P}_{pc}|$.
- Let $\mathcal{P}_{un}$ be the set of colors used for the uncolored vertices; denote $\chi_{un} = |\mathcal{P}_{un}|$.
- Let $\mathcal{P}_{new} = \mathcal{P}_{un} \setminus \mathcal{P}_{pc}$ be the *new* colors used for the uncolored vertices; denote $\chi_{new} = |\mathcal{P}_{new}|$.
- Let $\mathcal{P}_{all} = \mathcal{P}_{pc} \cup \mathcal{P}_{new}$ be the final set of colors of all vertices; denote $\chi_{all} = |\mathcal{P}_{all}|$.

For a given instance of CC, let χ^*_{un} (respectively, χ^*_{new}, χ^*_{all}) be the smallest possible value of χ_{un} (resp., χ_{new}, χ_{all}) over all possible proper color completions of the precoloring. Additionally, for a given algorithm $\mathcal{A}$, let $\chi^{\mathcal{A}}_{un}$ (respectively, $\chi^{\mathcal{A}}_{new}$, $\chi^{\mathcal{A}}_{all}$) be the value of χ_{un} (resp., χ_{new}, χ_{all}) in the solution computed by $\mathcal{A}$ for the instance.

The efficiency of an algorithm for CC can be measured by two parameters of interest, namely, χ_{new} and χ_{all}. The difference between them becomes noticeable in instances where the colors in $\mathcal{P}_{pc}$ are not contiguous, i.e., when there are colors x such that $x \notin \mathcal{P}_{pc}$ but $x + 1 \in \mathcal{P}_{pc}$. We denote by $\mathsf{CC}_{new}(t)$ (resp. $\mathsf{CC}_{all}(t)$) the problem of color completion such that χ_{new} (resp. χ_{all}) is at most t.

3.1 Single Round Color Completion

We first consider what can be done when the online algorithm is restricted to a single round of communication.

▶ **Theorem 8.** *Consider a graph G with maximum degree $\Delta = \Delta(G)$ and chromatic number $\chi = \chi(G)$ with $\Delta > 0$, then $\mathsf{CC}_{new}(\chi \cdot \Delta)$ can be solved in a single* SUPPORTED *round.*

Proof. The algorithm uses the color palette

$$\mathcal{P} \;=\; \{(i, j) \mid 1 \le i \le \chi,\;\; 1 \le j \le \Delta\}.$$

In the preprocessing stage, compute a proper *default coloring dc* of the graph using the color palette $\mathcal{P}^{def} = \{i \mid 1 \le i \le \chi\}$, and let each vertex v store its default color $dc(v)$ for future use. These values are not used as colors in the final coloring.

In the online stage, we are given an arbitrary precoloring $c(w) \in \mathcal{P}$ for some nodes, and need to complete it to a proper coloring by selecting a color $c(v)$ for each non-precolored node v. (It is assumed that the precoloring itself is proper, i.e., no two precolored neighboring vertices use the same color.)

The algorithm requires a single round of communication. Each precolored node w informs its neighbors about its color $c(w)$. Now consider a non-precolored node v. If all neighbors of v are colored, then v chooses a free color from the color palette. As $\chi \cdot \Delta \ge 2\Delta \ge \Delta + 1$, such a color is guaranteed to exist.

Otherwise, v finds a *free color* of the form $(dc(v), j)$ for $1 \le j \le \Delta$ satisfying $c(w) \neq (i, j)$ for all precolored neighbors w of v. The node v then selects $c(v) \leftarrow (dc(v), j)$.

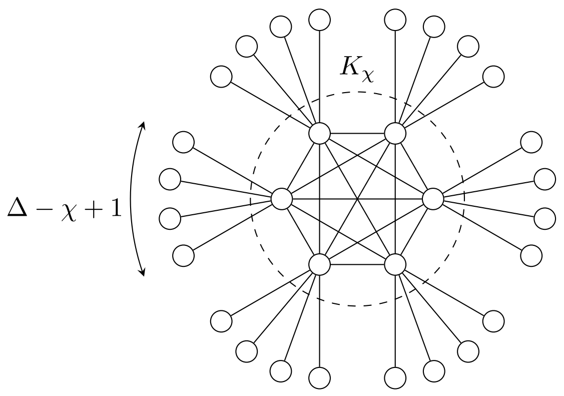

Figure 2 Graph whose single round color completion assigns at least $\chi \cdot (\Delta - \chi + 1)$ different colors across all instances. In this example $\chi = 6, \Delta = 9$.

By this algorithm, the color (i, j) selected by v is different from the color of any precolored neighbor of v. Also, (i, j) cannot be the selected color of any non-precolored neighbor w of v. This is because the default color $dc(w) = i'$ of w satisfies $i' \neq i$, and therefore, the selected color $c(w)$ of w is of the form (i', k) for some k, which must differ from (i, j) at least on the first component. Thus, the coloring c is proper. ◀

▶ Remark 9. In the absence of any preprocessing, [22] showed that we require $\Omega(\log^* n)$ rounds to color the graph even if it is just a path. To complement this, [22] also provides an $O(\log^* n)$ round algorithm that colors the graphs with maximum degree Δ with $O(\Delta^2)$ colors. In the full paper [1] we show that the same algorithm can be adapted to CC with at most $\lceil 23\Delta^2 \log_2 n \rceil$ new colors.

A consequence of the above is that one can readily adapt existing solutions of graph coloring to color completion. For example the results of [23], [8] can be extended to CC, with the number of colors used replaced by χ_{new} and retaining the same round complexities.

We complement the result of Thm. 8 with the following lower bound, see Figure 2 for the graph.

▶ **Theorem 10.** *For every integer χ, Δ, there exists a graph G with chromatic number χ and maximum degree Δ such that for every single round deterministic distributed algorithm $\mathcal{A}$, the total number of colors used by $\mathcal{A}$ over all recurrent instances of* CC *is at least $\chi \cdot (\Delta - \chi + 2)$ even after an arbitrary preprocessing of G.*

The single round algorithm can be extended to multiple round algorithm providing a similar color-round tradeoff as that of Maus' algorithm [23]. See the full paper [1] for the algorithm and discussions.

3.2 CC with fewer than $\Delta + 1$ colors

We next discuss coloring algorithms based on a preprocesing stage which, in many cases, use fewer than $\Delta + 1$ colors.

Our main result is an algorithm that, for a graph G with chromatic number χ, uses preprocessing, and in the recurrent stage solves any instance of CC with at most χ new colors in χ rounds. The algorithm operates as follows.

Preprocessing. The preprocessing stage computes a proper-χ coloring of the graph G. This is stored implicitly, i.e., each node v stores a single color (a positive integer) $dc(v)$. We call this coloring the initial coloring of G.

Online algorithm. We call the algorithm the "priority recoloring" algorithm. The set of nodes with the same initial coloring form an independent set which implies that nodes belonging to this set may be colored independently. We use the standard greedy algorithm to simultaneously color nodes with the same initial color in a single round. The initial colors are only computed to partition the original set of nodes into χ independent sets.
The input of each recurrent instance is a subset S of the nodes that were precolored, i.e., each $v \in S$ has a precolor $c(v)$. For convenience, consider $c(v) = 0$ for all $v \notin S$. The required output is a color completion of the precoloring: each node $v \notin S$ outputs a color $c(v) \in \mathbb{N}$ such that the colors assigned to all vertices form a proper coloring of the graph G.

The online algorithm $\mathcal{A}$ operates as follows.

- For $r = 1, 2, \ldots \chi$ rounds, do
  - If $dc(v) = r$ and $c(v) = 0$, then $c(v) \leftarrow \min(\mathbb{N} \setminus \Gamma(v))$, where $\Gamma(v) = \{c(w) | (w, v) \in E(G)\}$

For a given instance of the problem, χ^*_{un} (respectively, χ^*_{new}, χ^*_{all}) is the smallest possible value of χ_{un} (resp., χ_{new}, χ_{all}) over all possible proper color completions of the precoloring, and $\chi^{\mathcal{A}}_{un}$ (respectively, $\chi^{\mathcal{A}}_{new}$, $\chi^{\mathcal{A}}_{all}$) is the value of χ_{un} (resp., χ_{new}, χ_{all}) in the solution computed by the priority algorithm.

▶ **Observation 11.** *For any coloring, $\chi_{all} = \chi_{pc} + \chi_{new}$. In particular, $\chi^*_{all} = \chi_{pc} + \chi^*_{new}$ and $\chi^{\mathcal{A}}_{all} = \chi_{pc} + \chi^{\mathcal{A}}_{new}$.*

▶ **Lemma 12.** *$\chi^{\mathcal{A}}_{new} \leq \chi$.*

Proof. For every integer $k \geq 1$, let $\mathbb{N}_k = \{1, \ldots, k\}$. Let $M = \max \mathcal{P}_{pc}$, and let $FREE = \mathbb{N}_{M+\chi} \setminus \mathcal{P}_{pc}$ be the set of free colors (not used in the precoloring) up to $M + \chi$. Note that the cardinality of the set $FREE$ is at least χ. Let $\hat{F} = \{f_1, \ldots, f_\chi\}$ consist of the smallest χ integers in the set $FREE$.

By induction on k from 1 to χ, one can verify that during iteration k of the algorithm, the colors the algorithm uses for the uncolored vertices of default color $dc(v) = k$ are taken from $\mathcal{P}_{pc} \cup \{f_1, \ldots, f_k\}$. Hence $\mathcal{P}^{\mathcal{A}}_{un} \subseteq \mathcal{P}_{pc} \cup \hat{F}$, implying that $\chi^{\mathcal{A}}_{new} \leq |\hat{F}| = \chi$. ◀

▶ **Theorem 13.** *Consider a graph G with chromatic number $\chi = \chi(G)$. $\mathsf{CC}_{all}(\chi + \chi^*_{all} - 1)$ and $\mathsf{CC}_{new}(\chi)$ can be solved in χ SUPPORTED rounds.*

In the full paper [1] we discuss how tight these bounds are and prove the following lower bounds.

▶ **Theorem 14.** *(Lower bound for $\chi^{\mathcal{A}}_{new}$). For every deterministic distributed algorithm $\mathcal{A}$, and every integer $D > 3$, that solves CC with the guarantee that $\chi^{\mathcal{A}}_{new} < \chi$, there exists a graph G with diameter D and an instance of CC for which $\mathcal{A}$ takes $\Omega(D)$ SUPPORTED rounds.*

▶ **Theorem 15.** *For every integer $\chi \geq 2$ and deterministic algorithm $\mathcal{A}$ that solves CC with the guarantee that $\chi^{\mathcal{A}}_{new} \leq \chi^*_{new} + 1$, there exists a graph G with chromatic number χ and an instance of CC for which $\mathcal{A}$ takes χ SUPPORTED rounds.*

4 Locally Checkable Labellings

Locally Checkable Labellings (LCL) were first proposed by Naor and Stockmeyer [26].
Formal definitions and some examples can be found in the full paper [1].

4.1 Subgraph LCL's without Input Labels on Paths

In this section we consider a subset of recurrent LCL's, named *subgraph LCL's without input labels*, which were studied by Foerster et al. [16]. In subgraph LCL's, the online instances ask for a valid labelling for some (edge induced) subgraph of the given graph G. This sub-class of LCL's are easier to solve, but already captures several classical problems, such as finding a dominating set, maximal matching, maximal independent set, (k, l)-ruling sets etc.

We consider subgraph LCL on a path P_n. Before getting to the solution, we first remark that one may consider without loss of generality only LCL's with radius 1. Given an LCL problem of radius r, one may construct another LCL with the same round complexity but with radius 1 at the cost of increasing the output label size and the set of rules.

From a prior work (Theorem 3 in Foerster et al. [16]), we may infer that if the round complexity of Π in the LOCAL model is $o(n)$, then it must be $O(1)$ in the SUPPORTED model. This result is non-constructive, i.e., it argues that given a $o(n)$ round distributed algorithm, one can transform it into an $O(1)$ round algorithm. Additionally, it does not help categorize LCL problems that are $\Theta(n)$ in the LOCAL model. Some LCL problems (such as 2-coloring) are $\Theta(n)$ in the LOCAL model, but clearly $O(1)$ in the SUPPORTED model. One can also construct LCL's that remain $\Theta(n)$ in the SUPPORTED model. Furthermore, the proof offers no insight about the additional amount of memory per node that is needed for the preprocessing stage. The following theorem addresses the above questions. Note that as done in prior work, we treat the size of the description of Π as constant in the round complexity (in particular, $|\Sigma_{out}|$ and $|\Sigma_{in}|$ are constants).

▶ **Theorem 16.** *Let Π be a subgraph LCL with $|\Sigma_{in}| = 1$ and let P_n be a path on n vertices, then*

- *The* SUPPORTED *round complexity of Π is either $O(1)$ or $\Omega(n)$*
- *There exists a round optimal* SUPPORTED *algorithm for Π that stores only $O(1)$ bits as the output of the preprocessing phase*
- *The optimal* SUPPORTED *algorithm can be efficiently (polynomial in size of description of Π) computed by a centralized algorithm.*

4.2 Recurrent LCL's on Paths

In this section we look at a broader class of recurrent *in-labeled LCL's*, namely, LCL's augmented with input labels, wherein the online instances specify different input labels. The set of rules $\mathcal{C}$ and the set of output labels Σ_{out} remain the same across instances. The only components of the input that vary across instances are the input labels Γ^{in}. Subgraph LCL's, studied in Sect. 4.1, can be represented in this framework by encoding the adjacent edges that are present in the input labels for each vertex, hence in-labeled LCL's are a generalisation of subgraph LCL's. Problems such as finding a Client Dominating Set, Color Completion, Maximal Matching and in general variants of classical local problems with PFO and / or PCS instances fall into this category.

We show that even for these instances, the optimal round complexity is either $O(1)$ or $\Theta(n)$, thus extending the distributed speed up theorem in Foerster et al. [16]. However, so far we were unable to find a characterization as obtained in the previous subsection. Therefore, we are left with a couple of intriguing open questions. First, we do not know any bounds on the constant of the running time in terms of the size of the LCL Π, namely, $|\Sigma_{in}| + |\Sigma_{out}|$. Second, we do not know if it is possible to decide the online round complexity in polynomial time given the description of the LCL.

▶ **Theorem 17.** *Let Π be a recurrent LCL problem whose online instances differ only in the assignment of input labels, then either Π can be solved in $O(1)$* SUPPORTED *rounds or Π requires $\Omega(n)$* SUPPORTED *rounds. Furthermore, there exists an algorithm with asymptotically optimal online round complexity using only $O(1)$ bits as the output of the preprocessing phase.*

Our proof of the above theorem is almost the same as that of Theorem 6 in [11] (for the LOCAL model) and Theorem 3 in [16] (for the SUPPORTED model). Note that, for paths, the above theorem is stronger than Theorem 3 of [16], which only translates $o(n)$ time algorithms in LOCAL to $O(1)$ time algorithms in SUPPORTED, whereas our argument also translates $o(n)$ time algorithms in SUPPORTED to $O(1)$ time algorithms in SUPPORTED. Theorem 3 of [16] holds for any hereditary graph family (for e.g. graphs with maximum degree Δ).

5 Conclusion and Future Directions

We discussed three recurrent problems and the advantages that a preprocessing phase has on their distributed round complexities. Several open problems remain, in particular:

- Can we extend state of the art algorithms for client dominating set on bounded arboricity graphs or other families? For example, the combinatorial algorithms of Morgan et al. [25] extend readily to the client dominating set problem. Is it possible to improve the round complexity of these algorithms in the SUPPORTED model?
- Can we obtain algorithms with better round complexities for color completion? We can perform color completion with at most χ new colors within χ rounds. Is this optimal?
- Extend the complexity gap theorems for trees - It is known that the distributed round complexity of LCL problems is either $O(1), \Theta(\log^* n), \Theta(\log n)$ or $\Theta(n^{1/k})$ for some integer k. (See [10] and [18]). These methods do not seem to be readily extendable to rooted trees in the SUPPORTED model.

5.1 Maximal Matchings and Maximal Independent sets

Apart from these questions, there are also other local problems of interest for which the round complexities in the SUPPORTED model are yet to be known. For example, consider the sub-graph maximal matching and sub-graph maximal independent set problem. For these problems tight bounds are known in the LOCAL model (see [5]), however these do not translate easily to the SUPPORTED model. We offer some partial results towards this direction. Proofs and detailed discussions appear in the full paper.

▶ **Theorem 18.** *Sub-graph Maximal Matching on graphs of arboricity a can be computed in $O(a)$* SUPPORTED *rounds.*

▶ **Theorem 19.** *If subgraph MM can be solved in $o(\Delta)$* SUPPORTED *rounds for all bipartite graphs, then subgraph-MM can be solved in $o(\Delta)$* SUPPORTED *rounds for all graphs.*

▶ **Theorem 20.** *Sub-graph MIS for a graph G with chromatic number χ can be solved in χ* SUPPORTED *rounds.*

References

1 Akanksha Agrawal, John Augustine, David Peleg, and Srikkanth Ramachandran. Local problems in the supported model. Technical Report 2212.14542, arXiv, 2022. Full version of this paper. doi:10.48550/arXiv.2212.14542.

2 Noga Alon, Jarosław Grytczuk, Mariusz Hałuszczak, and Oliver Riordan. Nonrepetitive colorings of graphs. *Random Structures & Algorithms*, 21(3-4):336–346, 2002. doi:10.1002/RSA.10057.

3 Saeed Akhoondian Amiri, Stefan Schmid, and Sebastian Siebertz. Distributed dominating set approximations beyond planar graphs. *ACM Trans. Algorithms*, 15(3), jun 2019. doi:10.1145/3326170.

4 Brenda S Baker. Approximation algorithms for np-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994. doi:10.1145/174644.174650.

5 Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. *J.ACM*, 68(5):1–30, 2021. doi:10.1145/3461458.

6 Alkida Balliu, Sebastian Brandt, Dennis Olivetti, Jan Studenỳ, Jukka Suomela, and Aleksandr Tereshchenko. Locally checkable problems in rooted trees. In *Proc. 2021 ACM Symp. on Principles of Distributed Computing*, pages 263–272, 2021. doi:10.1145/3465084.3467934.

7 Leonid Barenboim. Deterministic (δ+ 1)-coloring in sublinear (in δ) time in static, dynamic, and faulty networks. *J. ACM*, 63(5):1–22, 2016. doi:10.1145/2979675.

8 Leonid Barenboim, Michael Elkin, and Uri Goldenberg. Locally-iterative distributed (delta + 1): -coloring below szegedy-vishwanathan barrier, and applications to self-stabilization and to restricted-bandwidth models. In *Proc. ACM Symp. on Principles of Distributed Computing (PODC)*, pages 437–446, 2018.

9 Sebastian Brandt, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Patric R. J. Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela, and Przemyslaw Uznanski. LCL problems on grids. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 101–110. ACM, 2017. doi:10.1145/3087801.3087833.

10 Yi-Jun Chang. The complexity landscape of distributed locally checkable problems on trees. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 18:1–18:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.DISC.2020.18.

11 Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the local model. *SIAM J. Computing*, 48(1):122–143, 2019. doi:10.1137/17M1117537.

12 Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the local model. *SIAM J. Computing*, 48(1):33–69, 2019. doi:10.1137/17M1157957.

13 Yi-Jun Chang, Jan Studený, and Jukka Suomela. Distributed graph problems through an automata-theoretic lens. In Tomasz Jurdzinski and Stefan Schmid, editors, *Structural Information and Communication Complexity - 28th International Colloquium, SIROCCO 2021, Wrocław, Poland, June 28 - July 1, 2021, Proceedings*, volume 12810 of *Lecture Notes in Computer Science*, pages 31–49. Springer, 2021. doi:10.1007/978-3-030-79527-6_3.

14 Andrzej Czygrinow, Michal Hańćkowiak, and Wojciech Wawrzyniak. Fast distributed approximations in planar graphs. In Gadi Taubenfeld, editor, *Distributed Computing*, pages 78–92, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. doi:10.1007/978-3-540-87779-0_6.

15 Vida Dujmović , Louis Esperet, Gwenaël Joret, Bartosz Walczak, and David Wood. Planar graphs have bounded nonrepetitive chromatic number. *Advances in Combinatorics*, mar 2020. doi:10.19086/aic.12100.

16 Klaus-Tycho Foerster, Juho Hirvonen, Stefan Schmid, and Jukka Suomela. On the power of preprocessing in decentralized network optimization. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1450–1458. IEEE, 2019. doi:10.1109/INFOCOM.2019.8737382.

17 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *2016 IEEE 57th Symp. on foundations of computer science (FOCS)*, pages 625–634. IEEE, 2016. doi:10.1109/FOCS.2016.73.

18 Christoph Grunau, Václav Rozhon, and Sebastian Brandt. The landscape of distributed complexities on trees and beyond. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 37–47. ACM, 2022. doi:10.1145/3519270.3538452.

19 Chetan Gupta, Juho Hirvonen, Janne H. Korhonen, Jan Studený, and Jukka Suomela. Sparse matrix multiplication in the low-bandwidth model. In *SPAA '22: 34th ACM Symp. on Parallelism in Algorithms and Architectures*, pages 435–444, 2022. doi:10.1145/3490148.3538575.

20 Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1166–1179, 2021. doi:10.1145/3406325.3451081.

21 Ozan Heydt, Sebastian Siebertz, and Alexandre Vigny. Local planar domination revisited. In Merav Parter, editor, *Structural Information and Communication Complexity - 29th International Colloquium, SIROCCO 2022, Paderborn, Germany, June 27-29, 2022, Proceedings*, volume 13298 of *Lecture Notes in Computer Science*, pages 154–173. Springer, 2022. doi:10.1007/978-3-031-09993-9_9.

22 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992. doi:10.1137/0221015.

23 Yannic Maus. Distributed graph coloring made easy. In *Proc. 33rd ACM Symp. on Parallelism in Algorithms and Architectures*, pages 362–372, 2021. doi:10.1145/3409964.3461804.

24 Yannic Maus and Tigran Tonoyan. Local conflict coloring revisited: Linial for lists. *arXiv preprint*, 2020. arXiv:2007.15251.

25 Adir Morgan, Shay Solomon, and Nicole Wein. Algorithms for the minimum dominating set problem in bounded arboricity graphs: Simpler, faster, and combinatorial. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 33:1–33:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.DISC.2021.33.

26 Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

27 Stefan Schmid and Jukka Suomela. Exploiting locality in distributed sdn control. In *Proc. 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 121–126, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2491185.2491198.